PX-8

BASIC REFERENCE MANUAL

FEDERAL COMMUNICATIONS COMMISSION RADIO FREQUENCY INTERFERENCE STATEMENT

"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 away from the receiver
..... plug 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 US Government Printing Office, Washington, D.C., 20402, Stock No. 004-000-00345-4."

Also, only peripherals (computer input/output devices, terminals, printers, etc.) certified to comply with the Class B limits may be attached to this computer. Operation with non-certified peripherals is likely to result in interference to radio and TV reception.

TABLE OF CONTENTS

Trademark Acknowledgments

CP/M® is a registered trademark of Digital Research™.
BASIC (Copyright 1977 – 1983 by Microsoft and Epson) is upward compatible with the BASIC-80 specifications of Microsoft, Inc.
MICROCASSETTE™ is a trademark of OLYMPUS OPTICAL CO., LTD.

NOTICE

- All rights reserved. Reproduction of any part of this manual in any form whatsoever without EPSON's express written permission is forbidden.
- The contents of this manual are subject to change without notice.
- All efforts have been made to ensure the accuracy of the contents of this manual. However, should any errors be detected, EPSON would greatly appreciate being informed of them.
- The above notwithstanding, EPSON can assume no responsibility for any errors in this manual or their consequences.

Introduction

This manual describes EPSON-enhanced Microsoft BASIC for the EPSON PX-8. The manual covers all aspects of PX-8 BASIC, including points which must be considered when using the graphic display, the RS-232C communication functions, disk I/O (including RAM disk and use of the microcassette drive as a disk device), and use of the internal clock to switch the power supply and initiate programs.

Chapter 1 of this manual introduces PX-8 BASIC, the functions of the PX-8's special keys in the BASIC mode, and entry and modification of BASIC programs.

Chapter 2 discusses general concepts which are applicable to programming in BASIC and considerations applicable to programming in EPSON-enhanced PX-8 BASIC in particular.

Chapter 3 describes methods for entering BASIC using extended BASIC commands.

Chapter 4 describes use of the statements and functions of PX-8 BASIC.

Chapter 5 describes disk files (including files in RAM disk and use of the microcassette drive as a disk device) and their handling.

Chapter 6 describes procedures for communication between the PX-8 and other devices using the RS-232C interface.

The appendices describe various subjects such as error codes and messages, features applicable to the PX-8's display, programming, printers, and so forth, together with some sample programs.

This manual is intended to be used together with the PX-8 User's Manual, which describes operation of the PX-8 and basic procedures for use of the PX-8's CP/M operating system.

Finally, there is an index which should be consulted in order to make effective use of the manual as a work of reference.

Chapter 1

GENERAL INFORMATION

A program is a series of instructions which control the operations of a computer. Such instructions must be a part of a predefined set which the computer is designed to understand, and which are combined in accordance with a fixed set of rules. This set of instructions is referred to as the computer's language. The individual instructions (words) used by the language are referred to as commands, statements, or functions, and the rules which govern the manner in which instructions are combined are referred to as the language's syntax.

The programming language supplied with the PX-8 is BASIC (Beginner's All-purpose Symbolic Instruction Code). BASIC for the EPSON PX-8 is an EPSON-enhanced version of Microsoft BASIC which has been expanded by EPSON for use with the PX-8 and which operates under the CP/M operating system of the PX-8 (see the PX-8 User's Manual for further information on the CP/M operating system). BASIC is loaded into the computer's memory from a ROM capsule, by executing the "BASIC" program under CP/M.

Among the enhancements which have been made are:

(i) the addition of a powerful screen editor which vastly increases the ease with which programs are entered, modified and executed

(ii) a variety of graphic statements and functions which take advantage of the PX-8's large 480 by 64 dot display

(iii) statements for selecting different modes of screen operation, such as graphics and split screen

(iv) statements and functions which support communication through the PX-8's RS-232C serial interface

(v) statements which make it possible to use the PX-8's built-in microcassette drive in the same manner as a disk drive

1.1 Installing BASIC

BASIC for the PX-8 is distributed in the form of a ROM capsule which must be installed in the back of the PX-8 before BASIC can be loaded. The location of this socket and procedures for changing ROM capsules are as described in the PX-8 User's Manual. BASIC can be inserted in either ROM socket 1 or ROM socket 2 and would be loaded from the ROM drive which has been allocated to that socket as if it were a program on a conventional floppy disk.

If you are using an applications program ROM as regularly as BASIC, and wish to have the applications ROM as well as BASIC, you may consider using PIP to transfer some of the CP/M utility programs from the utility ROM into the RAM disk area or even onto cassette tape. Details of how to use PIP to do this are given in the User's Manual.

NOTES:

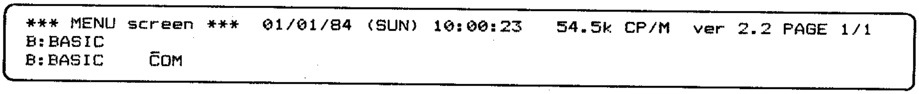
1. *BASIC cannot be started if the RAM disk size is set to 24K with the CONFIG command.*
2. *If a ROM capsule is changed for any reason while in BASIC, execute the RESET command.*

it there are still a number of possibilities, depending on whether BASIC has been used previously and on whether you wish to run a BASIC program directly. The possibilities are as follows:

(i) BASIC.COM is the first file on the MENU.
(ii) BASIC.COM is on the MENU but not the first file.
(iii) BASIC is resident in memory.
(iv) You wish to RUN a stored BASIC program.
(v) You wish to run a BASIC program directly when BASIC is resident.

(i) BASIC.COM is the first file on the MENU

The simplest case is when BASIC has not been used when the PX-8 is switched on. If the MENU has been set up to show the files on the drives allocated to the ROM sockets, the appearance of the screen will be as follows if BASIC.COM is the first file in the main MENU area in the top left hand corner. The command line will have "BASIC" entered by the computer, together with the drive name prefix. BASIC.COM with its drive prefix will be flashing in the main menu area.



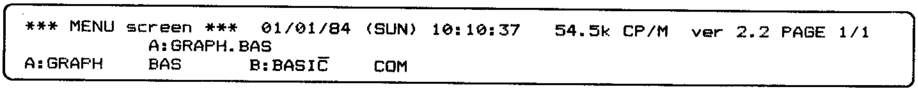
```
*** MENU screen ***  01/01/84 (SUN) 10:00:23   54.5k CP/M  ver 2.2 PAGE 1/1
B:BASIC
B:BASIC    C̄OM
```

Simply pressing the RETURN key will load BASIC into memory from the ROM. BASIC will take a few seconds to load. Section 1.3 describes what to do next.

WARNING:
In loading BASIC into memory, any other programs already there will be destroyed.

(ii) BASIC.COM is not the first file on the MENU

Depending on how the MENU was set up, BASIC.COM may not be the first file on the MENU. For example



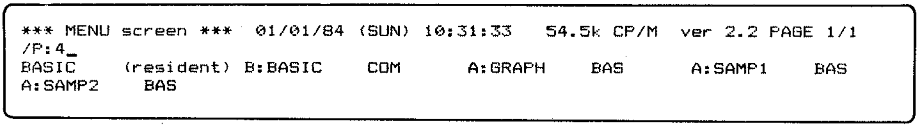
```
*** MENU screen ***  01/01/84 (SUN) 10:10:37   54.5k CP/M  ver 2.2 PAGE 1/1
           A:GRAPH.BAS
A:GRAPH    BAS        B:BASIC̄    COM
```

WARNING:
If a BASIC program is RUN in this way, all the BASIC programs in memory which lie in the BASIC program areas 1-5 WILL BE DESTROYED.

If the MENU is used to run a BASIC program directly when the program is put onto the command line, it should appear as in (iv) above. If it does not, you have made an incorrect entry when setting up the MENU option on the System Display, and should refer to the User's Manual to see the correct way to set up BASIC programs.

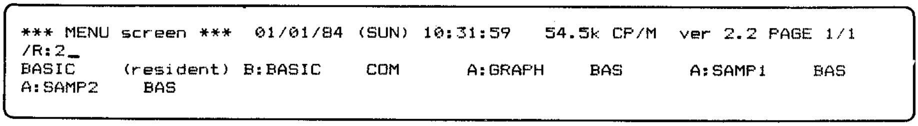
(v) Running a BASIC program directly when BASIC is resident
The situation may occur that the program is in memory when BASIC is resident. Whereas it is possible to go to the BASIC program menu and then run the program as described in the next section, it may be more convenient to run a program directly from the MENU screen. This can be achieved using the following commands which are described fully in Chapter 3.

/P:n
where n is a program area from 1 to 5 (e.g. /P:4 means program area 4), will enter BASIC and login to this area.



```
*** MENU screen ***  01/01/84 (SUN) 10:31:33   54.5k CP/M  ver 2.2 PAGE 1/1
/P:4_
BASIC     (resident) B:BASIC    COM      A:GRAPH    BAS      A:SAMP1    BAS
A:SAMP2    BAS
```

/R:n
where n is the program area, will enter BASIC and run the program in the specified program area.

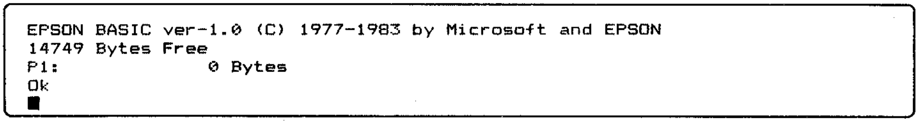


```
*** MENU screen ***  01/01/84 (SUN) 10:31:59   54.5k CP/M  ver 2.2 PAGE 1/1
/R:2_
BASIC     (resident) B:BASIC    COM      A:GRAPH    BAS      A:SAMP1    BAS
A:SAMP2    BAS
```

SPACE   Logs in the program area indicated by the cursor, but does not execute the program in that area.

Since the cursor is shown to the left of "P1:" at this point, pressing the space bar here logs in program area 1. To select another program area, move the cursor up or down with the "up" and "down" arrow keys before pressing the space bar.

If no program exists in any area (shown by the length of program text being 0 bytes), then RETURN simply logs into that area. On logging into such an area the screen appears thus



```
EPSON BASIC ver-1.0 (C) 1977-1983 by Microsoft and EPSON
14749 Bytes Free
P1:             0 Bytes
Ok
■
```

The "Ok" displayed at the end of this message indicates that BASIC is at the command level; in other words, that it is ready to accept commands. At this point the BASIC interpreter may be used in either of two modes: direct (immediate) mode or indirect (execution) mode.

Commands and statements entered in the direct mode are not preceded by program line numbers. Instead, they are executed immediately when the RETURN key is pressed. Since commands/statements entered in this mode are executed as they are input, the results of arithmetic and logical operations are displayed immediately (they can also be assigned to variables for later use). However, the commands themselves are lost once they have disappeared from the screen. This mode is useful for debugging and for using BASIC for simple, non-repetitive arithmetic operations.

The indirect mode is the mode which is used for entering programs. In this mode, commands, statements, and functions are preceded by line numbers which indicate the order in which they are to be executed. However, commands and statements entered in the indirect mode are stored in memory without being executed; execution is deferred until the program is RUN. The indirect mode is used when working with complicated calculations or operations which must be performed many times or stored to be recalled for use at a later date.

1.4 Warm Starts and Cold Starts

There are two methods for starting up BASIC: the cold start and the warm start.

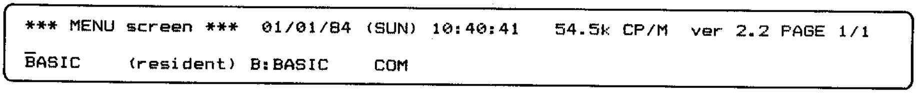
1.4.1 Cold starts

A cold start is made when the BASIC interpreter is loaded into the memory of the PX-8 from ROM capsule. This type of start is made by executing the BASIC command or by selecting the BASIC.COM file from the menu.

NOTE:
The BASIC program areas are cleared whenever a cold start is made. A cold start must be made whenever the PX-8's power switch is turned on unless it is made resident in the PX-8's memory (the area in which utility programs such as the BASIC interpreter are stored while they are being executed). In this case, BASIC can be started up by making a warm start.

1.4.2 Warm starts

A warm start is the procedure by which BASIC is restarted when it is already present in memory. BASIC becomes resident in memory when loaded, and remains there when the power is turned off if the MENU screen function has been turned on. In this situation, the screen appears as shown below when the power switch is turned back on.



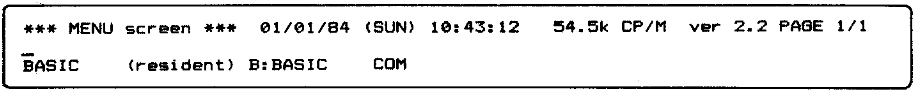
```
*** MENU screen ***   01/01/84 (SUN) 10:40:41    54.5k CP/M  ver 2.2 PAGE 1/1
BASIC     (resident) B:BASIC    COM
```

Procedures for turning on the MENU screen function are as follows.

(1) Turn on the computer's power switch and if the system prompt ("A>" or another drive name) is displayed, proceed as follows:

(2) Press the CTRL and HELP (SYSTEM) keys together; this causes the System Display to appear as shown below.

(5) Pressing CTRL + C or the STOP key at this point causes the system to go immediately to the MENU screen, where BASIC can be loaded and executed simply by pressing the RETURN key. (BASIC can also be loaded and executed at this point by entering "B:BASIC" or "C:BASIC", depending on the ROM socket into which BASIC has been inserted as described in section 1.1 and which drive it has been allocated to as described in section 1.2(a). Log into one of the BASIC program areas by pressing the space bar, then return to the system by entering SYSTEM and pressing the RETURN key.) The MENU screen will now appear as shown below.



```
*** MENU screen ***  01/01/84 (SUN) 10:43:12   54.5k CP/M  ver 2.2 PAGE 1/1

BASIC     (resident) B:BASIC     COM
```

The words "BASIC (resident)" at the left side of the screen indicate that BASIC is resident in memory. At this point, these words should be flashing on and off; this indicates that BASIC execution will be restarted when the RETURN key is pressed.

BASIC will remain resident in memory until one of the following actions is performed:

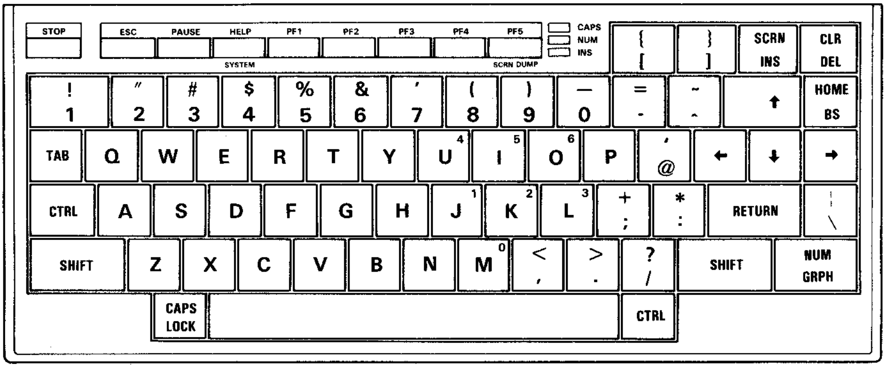
(1) One of the system utilities is executed;
(2) A cold start of the system is made (see the PX-8 User's Manual);
(3) The MENU screen function is turned off from the System Display (see the PX-8 User's Manual);
(4) The MENU screen is terminated by pressing the ESC key.

Once BASIC has been made resident, programs in the BASIC program areas will be retained even if the PX-8's power switch is turned off and back on again. Further, the maximum number of files, upper memory limit, and so forth are maintained from the last time BASIC operation was ended.

WARNING:
If the MENU screen function is switched off via the System Display, BASIC will not be resident and the programs in the BASIC program areas will be lost. It is always good practice to save all programs as well as leaving them in memory.

1.6 Functions of Special Keys in the BASIC Mode

The keyboard of the PX-8 includes a number of special keys as indicated in the figure below.



Functions of these special keys during BASIC operation are as follows.

PF1 to PF5 The Programmable Function Keys

These are the PX-8's programmable function keys. Each key contains two functions, the second of which is accessed by pressing the shift key as well as the function key. In this case the keys are numbered PF6 to PF10, for instance, PF4 when shifted is known as PF9. Any string of up to 15 characters can be assigned to each of these keys with the KEY command of BASIC or the CONFIG command of CP/M; afterwards, that string of characters can be entered simply by pressing the applicable programmable function key. This can greatly reduce the need to type in commands one character at a time from the keyboard. See the explanation of the KEY command in Chapter 4 and the discussion of the CONFIG command in the PX-8 User's Manual.

There is a third function of each key which is accessed by pressing the CTRL key at the same time as the programmable function key. This causes a machine code subroutine to be executed. In the case of the PF5 key this has been set to dump the screen to a printer. The others can be set by the user. Procedures for doing this are described in the PX-8 OS Reference Manual.

together). The PAUSE condition is then released by pressing any key other than STOP or CTRL and C . Pressing either of these will terminate either the listing or the program execution.

SCRN DUMP (CTRL + PF5)

The SCRN DUMP key outputs the contents of the screen to the printer. The screen contents are output to the printer as ASCII character codes when screen modes 0, 1, or 2 (the text modes) are selected, and in bit image format when the screen mode 3 (the graphic mode) is selected. The same result is obtained by executing the COPY statement in BASIC. (A detailed explanation of the screen modes are given in Chapter 2).

STOP

This key interrupts execution of BASIC programs and returns the BASIC interpreter to the command mode. (The same result is obtained by pressing the CTRL and C keys together). Execution of the interrupted program can then be continued by entering the CONT command. This key is also used to terminate automatic program line number generation initiated by the AUTO command.

The STOP KEY command can be used to disable or re-enable the STOP key (see the section on STOP KEY in Chapter 4).

CTRL + STOP

Pressing CTRL and STOP together during write or read access to the microcassette drive forcibly terminates the access operation and generates an error. (This function cannot be disabled by the STOP KEY command in BASIC).
However, if a search is being performed (if the tape is being wound to a specific counter reading to prepare for access) CTRL + STOP will not stop the drive until the search is completed.

CTRL + HELP

Pressing the CTRL and HELP keys together switches operation back to the System Display mode. This means that options such as setting the MENU screen, and setting the alarm and wake independently of BASIC can be carried out just as at the CP/M command level.

Chapter 2

PROGRAMMING CONCEPTS

This chapter discusses a variety of concepts which are applicable to programming in BASIC for the PX-8. These are presented in logical order, with the most fundamental concepts presented first. Mastery of the information included in this chapter is essential to realizing the full potential of the capabilities of BASIC.

2.1 Program Lines and Statements

All BASIC programs are composed of one or more lines, each of which begins with a line number, ends with a carriage return (or RETURN), and includes one or more commands, statements, or functions. The line numbers indicate the order in which the lines are stored in memory and executed. Lines numbers are referenced when the program is edited or when the flow of execution is switched from one point in a program to another. (See the descriptions of the GOTO and GOSUB statements in Chapter 4). Line numbers must be within the range 0 to 65529.

Commands and statements are words in the BASIC language which instruct the computer to perform specific operations. Each line of a program may consist of a single statement, or several statements which may be included on one line; in the latter case, each statement must be separated from the one following by a colon, and the total length of each program line cannot exceed 255 characters.

2.3 Screen Editor

The screen editor is a feature of EPSON BASIC which makes it easy to enter and edit the lines of BASIC programs. This ability is central to programming the PX-8 in BASIC.

The screen editor uses the concept of logical lines for display of commands and statements (in the direct mode) or program lines (in the indirect mode). A logical line is a collection of characters which is handled by the screen editor as one logical unit, regardless of the number of physical screen lines which it may occupy. Normally, a logical line is terminated by pressing the RETURN key.

During typing, logical lines are automatically continued when the cursor moves from the right side of the screen to the beginning of the following physical line. This applies regardless of whether the cursor is moved by typing characters or spaces, or by the TAB key.

There are several methods of editing lines of BASIC programs which have previously been stored in memory. The most primitive is simply to retype the entire line using the same line number. The BASIC interpreter automatically replaces the old line with the new one when the RETURN key is pressed.

However, the screen editor makes it possible to edit a logical line (after displaying it, if necessary, with the EDIT or LIST commands; these commands are explained in detail in Chapter 4) by moving the cursor to that line with the cursor controls, then making changes using the screen editor's control keys. A variety of keys are provided for use with the screen editor; these keys are described below.

↑ → ↓ ←

At the command level, these keys move the cursor (the flashing square which appears on the screen during BASIC operation) in the directions indicated by the arrows on the key tops. These keys are equipped with a repeat function which moves the cursor continuously at a steady rate when any of the keys is held down.

TAB

The TAB key moves the cursor to the right from its current location to the next tab position on the screen. The liquid crystal display screen of the PX-8 has 10 tab positions, starting with the column on the far left side of the screen. (A column consists of one character width in the same position on each line of

DEL

Pressing the DEL key deletes the character which is located at the position of the cursor and moves the remainder of the logical line to the left by one character position. No characters are deleted if this key is pressed while the cursor is at the end of a logical line.

CTRL + E

Pressing these keys together deletes all characters from the cursor position to the end of that logical line.

CTRL + Z

Pressing these keys together deletes all characters from the cursor position to the end of the screen.

CLR (SHIFT + DEL)

This key clears the entire screen and moves the cursor to the home position (the upper left corner of the screen). The same effect is achieved by pressing CTRL and L

INS

Pressing this key once places the screen editor in the insert mode; pressing it again (or pressing any of the cursor control keys or the RETURN key) restores normal operation. In the insert mode, the cursor and characters from the cursor to the end of the logical line are moved to the right by one position when any character key is pressed; the character typed is then inserted at the cursor's former position. The red INS LED built into the keyboard lights when the screen editor is in the insert mode, and the cursor changes from block form to underline form.

The screen editor can also be placed in the insert mode by pressing CTRL and R together.

RETURN

Pressing this key executes direct commands in the logical line in which the cursor is located or stores program lines in the computer's program text area. Operation is the same no matter where the cursor is located in the logical line. The same effect is achieved by pressing the CTRL and M keys together.

STOP

The principal function of this key is to halt program execution. Pressing this key in the command mode moves the cursor from the logical line in which it is currently positioned to the beginning of the next logical line. This key is also

2.4 EDIT Mode

In addition to the screen editor, PX-8 BASIC features an edit mode which increases the efficiency of program editing by making it possible to scroll to any point in a program. The edit mode is entered by executing the EDIT command in the direct mode (see the explanation of the EDIT command in Chapter 4), and is terminated by pressing the CLR or ESC keys.

The keys used for scrolling, cursor movement and program editing in the edit mode are basically the same as with the normal screen editor. However, functions of certain keys differ as follows:

(1) Cursor control keys
In the edit mode, the cursor control keys (← , → , ↑ and ↓) move the cursor in the same manner as when the screen editor is used in the normal direct mode. However, when the cursor is moved to the logical line at the top or bottom of the screen, that line is automatically scrolled as necessary to bring it completely inside the real screen. (This operation is performed when part of the program has been moved beyond the outside of the screen by previous scrolling).

(2) SHIFT + ↑
When only part of a logical line is displayed at the top of the screen, pressing SHIFT and ↑ together scrolls the screen so that the entire logical line is displayed, then moves the cursor so that it is positioned at the beginning of that logical line. Otherwise, the screen window is scrolled as necessary to display the logical line preceding that displayed at the top of the screen.

(3) SHIFT + ↓
When only part of a logical line is displayed at the bottom of the screen, pressing SHIFT and ↓ together scrolls the screen so that the entire logical line is displayed, then moves the cursor so that it is positioned at the beginning of that logical line. Otherwise, the screen window is scrolled as necessary to display the logical line following that displayed at the bottom of the screen.

(4) CTRL + ↑
Pressing CTRL and ↑ together clears the screen, displays the program's first line, and moves the cursor to the beginning of that line.

2.5 Using the Screen Editor and EDIT

The use of the edit and screen editor modes are best seen with the help of an example:

Use an empty program area or clear the memory in the area currently logged in by typing NEW and pressing RETURN, then type in the following line of a BASIC program:

10 REM This is a remark statement

When you press RETURN, but not until you do, the line will be stored as a BASIC program line. This line can be edited simply by using the screen editor. Use the cursor keys to move up the screen so that the cursor lies over one of the characters of the line which has just been typed in.

With the CTRL key held down, press the X key. The cursor now jumps to the end of the line so that you can add more characters to it.

With the CTRL key held down, press the A key and see that the cursor returns to the beginning of the line. Type a "2" so that the line becomes:

20 REM This is a remark statement

Now holding down the SHIFT key, press the right cursor key five times. Each time it is pressed the cursor jumps to the first character of the next word. With five jumps, it will be on the "r" of "remark". Use the shifted left cursor key to move the cursor back in a similar way.

Reposition the cursor at the beginning of "remark" and with the unshifted left cursor key place the cursor in the space before the "r". Now press the INS key. The LED next to INS will light to show that the PX-8 is in insert mode, and the cursor will become a flashing underline character. Type a word such as "new " and press the RETURN key.

Now if you LIST the program you will find that it consists of the following two lines:

Now press CTRL + E . The line will clear from the cursor onwards and by pressing RETURN the truncated line can be entered, as can be seen by listing the program again.

If you wish to clear the rest of the virtual screen from the cursor onwards CTRL + Z can be used instead.

Add line 40 to the program as follows:

```
40 PRINT “This is line 40”
```

Now move the cursor up to the “4” of line 40 and make it into line 60 by overtyping the “4” with a “6”. Move along the line using the shifted right cursor key or the alternative CTRL + F . Alter the “4” to a “6” here also.

When line 60 has been added to the program by pressing the RETURN key, add a further line 70 in the same way. It may be faster to use CTRL + X to move to the end of the line then the shifted left cursor key to move the cursor back to the number “6” in the string. Just as CTRL + F can be substituted for the right cursor key, so CTRL + B can be used to move back instead of the left cursor key.

Now add the lines:

```
80 FOR J = 1 TO 10
90 PRINT J * J
100 NEXT J
```

Until now only the normal screen editor has been used. In most cases this is all that is required, since a program can be listed and edited by moving around the screen. The important point to remember is that the line is only altered in the stored program IF THE RETURN KEY IS PRESSED.

The EDIT mode has some advantages if a number of lines are to be edited and if the editing is to be carried out on a screen with a limited number of lines, e.g. in screen mode 3.

To illustrate the use of the edit mode, type

```
SCREEN 3,0,0 : LIST
```

Now type in a line 110 to complete the program:

110 PRINT A$

There is one other aspect of using the EDIT mode, which applies to any screen mode, which makes it preferential to using LIST and the screen editor. This may be illustrated as follows:

Edit line 10 to read as shown so that it runs over more than one line of the screen:

10 REM This is a remark statement which has now been extended to run over on to the next line of the screen

List the program to line 60. Note that the screen shows only half of line 10. Move the cursor up to this line and try altering a character. When you press RETURN ,a "syntax error" message will be shown because you tried to enter a line which was not logically correct for BASIC. The screen couldn't "see" the first part of the line so it rejected the line as nonsense.

Clear the screen and list line 10. With the full line present on the screen move the cursor to the second part of the line and make an alteration. Pressing the RETURN key will cause the change to be accepted. The EDIT mode only allows complete lines to appear on the screen, and so prevents the syntax error obtained using list.

To illustrate this type EDIT and when line 5 has been displayed use the SHIFT and ↓ keys to display successive lines of the program until line 80 is reached. At this point, only the part of line 10 which overruns will be displayed on the screen. Now move the cursor to the top of the screen. As soon as the cursor reaches the top line, the whole of line 10 is displayed. This would not happen in the normal screen editor mode.

2.6.2 Control characters

The ASCII character set includes a number of special codes which can be used in programs to control various devices. These control codes have different functions when used to control the LCD screen or a printer. Also they have further special functions when used with the screen editor. The following table outlines the functions of the control characters when acting on just the LCD screen. They are used in a PRINT statement together with the CHR$ function. For example:

PRINT CHR$ (12)

clears the screen.
There are also extended control sequences which consist of a group of character codes following the ESC code (ASCII 27 decimal, 1B hexadecimal). These are described in Appendix C.

Dec. code	Hex. code	Function
5	05	Deletes characters to the end of the line.
7	07	Sounds the speaker (at about 440 Hz).
8	08	Moves the cursor to the left.
9	09	Moves the cursor to the next tab postion.
10	0A	Moves the cursor down one line.
11	0B	Moves the cursor to the home position.
12	0C	Clears the virtual screen.
13	0D	Moves the cursor to the beginning of the line.
16	10	In mode 0/1/2, moves the screen window upward.
17	11	In mode 0/1/2, moves the screen window downward.
26	1A	Deletes all characters to the end of the screen.
27	1B	Escape code.
28	1C	Moves the cursor to the right.
29	1D	Moves the cursor to the left.
30	1E	Moves the cursor upward.
31	1F	Moves the cursor downward.

Ordinarily, decimal notation is used for display of the contents of memory or the results of calculations. However, it is also possible to specify display of integer values in hexadecimal or octal notation.

Example

235.988E-Ø7 = 235.988×10^{-7} = .0000235988
2.359E09 = 2.359×10^{9} = 2359000000

(With double precision floating point constants, the letter "D" is used to indicate the implicit base (10) instead of "E". See below for a discussion of single and double precision numeric constants).

2.7.3 Single and double precision numeric constants

PX-8 BASIC allows use of both single and double precision numbers. Single precision numbers are handled internally as seven significant digits, and are rounded to 6 digits for display or printout. Double precision numbers are handled internally as 16 significant digits, and are also printed or displayed as 16 digits (with leading zeroes suppressed).

A single precision constant is any numeric constant that fulfills one of the following conditions:

(1) Consists of seven or fewer digits;
(2) Is represented in exponential form with "E"; or
(3) Has a trailing exclamation point ("!").

A double precision constant is any numeric constant that fulfils one of the following conditions:

(1) Consists of eight or more digits;
(2) Is represented in exponential form with "D"; or
(3) Has a trailing number sign (" # ").

NOTE:
The " # " character can be assigned different values, depending on the country in which the computer is being used. Consequently it will correspond to whatever character is assigned to CHR$(35).

Examples

Single Precision Constants	Double Precision Constants
46.8	345692811
– 7.09E-06	– 1.09432D-06
3489.0	3489.0 #
22.5!	7654321.1234

2.8 Variables

Variables are named locations in memory which are used to hold values during execution of BASIC programs. Names are assigned to variables by the programmer, and the values stored in variables are either assigned by the user during program execution or assigned as a result of progam execution itself.

The two general types of variables used with BASIC are numeric variables and string variables. The former are used to store numeric values, and the latter are used to store character strings. Until a variable is defined it has the value of zero if it is a numeric variable, and the value of null or empty string if it is a string variable.

2.8.1 Variable names and type declaration characters

Variable names may consist of up to 40 characters (including all letters, the decimal point, and all numerals), followed by a type declaration character; however, the first character of each name must be a letter. Reserved words may not be used as variable names. (Reserved words are the keywords used in entering BASIC commands, statements, and functions). Further, the letters "FN" must not be used at the beginning of any variable name (BASIC interprets words beginning with the letters "FN" as calls to a user-defined function).

The names of string variables must end with a dollar sign ($); this is the type declaration character which indicates that a variable is used to hold string data.

Numeric variable names may end with type declaration characters which indicate the type of numeric data which they contain. The type declaration characters for numeric variables are as follows:

%	Integer variable type declaration character
!	Single precision variable type declaration character
#	Double precision variable type declaration character

A single precision numeric variable is assumed if no type declaration character is specified.

2.9 Type Conversion of Numeric Values

BASIC automatically converts numeric values from one type to another as necessary. This section describes the rules governing numeric type conversion for various kinds of operations.

(1) Type conversion upon storage of values in variables
If a numeric constant of one type is assigned to a numeric variable of another type, it is stored after being converted to the type declared for that variable name. For example, if an integer-type numeric constant is assigned to a single precision variable, it is automatically converted to a single precision value at the time it is stored. Note that a certain amount of error may be introduced by the process of conversion.

Example 1

```
10 A%=12.34          :'Assigns single precision number
20 '                 :'12.34 to integer variable A%.
30 '
40 PRINT A%          :'DISPLAYS CONTENTS OF VARIABLE A%
Ok
run
 12
Ok
```

Example 2

```
10 A#=12.34          :'Assigns single precision number
15                   :'12.34 to double precision variable A#.
20 '
25 B#=23.45#         :'Assigns double precision number
30                   :'12.34# to double precision variable B#.
35 '
40 PRINT A#          :'Displays contents of variable A#.
45                   :'Extra digits are result of conversion
50                   :'error.
55 '
60 PRINT B#          :'Displays contents of variable B#.

Ok
run
 12.34000015258789
 23.45
Ok
```

(3) Conversion for logical operations
During logical operations, non-integer operands are converted to integers and the result is returned as an integer. The operands of logical operations must be in the range from −32768 to 32767; otherwise an "Overflow" error will occur.

Example

```
10 PRINT 6.34 OR 15     :'Converts single precision
20                      :'number 6.34 to an integer
30                      :'value (6), then displays the
40                      :'logical sum of 6 and 15.
Ok
run
 15
Ok
```

See section 2.10.3 for an explanation of logical operations.

(4) Type conversion of floating point numbers to integers
When a floating point number is converted to an integer, the decimal fraction is rounded to the nearest whole number.

Example

```
10 C%=55.88         :'Converts single precision number 55.88
20                  :'to integer by rounding to 56, then
30                  :'stores 56 in integer variable C%.
40 '
50 PRINT C%         :'Displays contents of variable C%.

run
 56
Ok
```

(5) Conversion of single precision numbers to double precision
If a single precision number is assigned to a double precision variable, only the first seven digits of the converted number are significant. This is because only six digits of accuracy are provided by single precision numbers.

2.10 Expressions and Operations

An expression is any notation within a program which represents a value. Thus variables, numeric constants and string constants constitute expressions, either when they appear alone or when combined by operators with other constants or variables.

Operators are symbols which indicate mathematical or logical operations which are to be performed on given values. The types of operations which are performed by BASIC can be divided into four categories as follows:

(1) Arithmetic operations
(2) Relational operations
(3) Logical operations
(4) Functional operations

2.10.1 Arithmetic operations

The arithmetic operations performed by BASIC include exponentiation, negation, multiplication, division, addition and subtraction. The precedence of these operations (the order in which they are performed when included in a single arithmetic expression) is as shown below.

Operator expression	Operation	Sample
∧	Exponentiation	X∧Y
−	Negation (conversion of the sign of a value)	(−Y)
*,/	Multiplication, division	X*Y, X/Y
MOD	Modulus arithmetic	X MOD Y
\	Integer division	X\Y
+ , −	Addition, subtraction	X+Y, X−Y

The concepts of integer division and modulus are explained in (1) and (2) below.

The order in which operations are performed can be changed by including parts of expressions in parentheses according to the normal rules of algebra. When this is done, the operations within parentheses are performed first according to the normal rules of precedence.

Examples

```
PRINT 10 MOD 4
 2
Ok
PRINT 25.68 MOD 6.99
 5
Ok
```

(3) If division by zero is encountered during evaluation of an expression, the "Division by zero" message is displayed, machine infinity (the value of greatest magnitude which can be displayed by the computer) is displayed as the result, then execution continues.

Example

```
10 B=7/0                   :'Generates "Division by zero"
20                         :'error, then stores machine
30                         :'infinity in variable B.
40 '
50 PRINT"PROGRAM LINE 30" :'Displays "PROGRAM LINE 30".

run
Division by zero
PROGRAM LINE 30
Ok
```

The "Division by zero" message is also displayed and machine infinity returned when zero is raised to a negative power.

(4) An overflow error ıs tne condıtion which occurs when the magnitude of the result of an operation exceeds the maximum value which can be displayed by the machine or when one of the operands of an operation such as integer division exceeds the maximum allowed value. Whether or not execution continues depends on which situation is encountered.

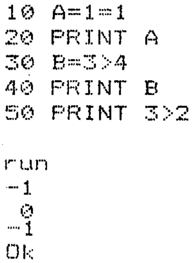
Example

```
10 A#=666^666               :'Generates "Overflow"
20 '                        :'error and stores machine
30 '                        :'infinity in variable A#.
40 '
50 PRINT A#                 :'Displays contents of A#.
60 '
70 PRINT "Program line 30" :'Displays "Program line 30".
80 '
```

When arithmetic and relational operators are combined in one expression, the arithmetic is always performed first. For example, the following expression is true if the value of X plus Y is less than the value of T – 1 divided by Z.

$$X+Y<(T-1)/Z$$

Example



```
10 A=1=1
20 PRINT A
30 B=3>4
40 PRINT B
50 PRINT 3>2

run
-1
 0
-1
Ok
```

In the example above, line 10 tests for equality between the first and second operands of the relational expression "1=1", then stores the result (–1, or true) in variable A. Line 20 then displays the contents of A. Line 30 tests whether the first operand of the relational expression "3>4" is greater than the second, then stores the result (0, or false) in variable B. The result is then displayed by the statement on line 40. Line 50 evaluates and displays the result of the relational expression "3>2" (–1, or true).

In a logical operation, the operands are converted to signed 16-bit two's complement integers† in the range from −32768 to 32767 before their logical connection is checked. (An error will result if any operand is not within this range.)

The specified operation is then performed for each bit of each operand (that is, for bits which are in the same position in each operand) and the result is returned as a two's complement integer which represents the results for all bits. Some examples of this are shown below.

Example 1

```
10 PRINT 63 AND 16
Ok
run
 16
Ok
```

In binary notation, the two's complement integer 63 is 111111B and the two's complement integer 16 is 010000B. Since 1 AND 0 yields 0 and 1 AND 1 yields 1, the result is 010000B, or 16.

Example 2

```
10 PRINT 21 XOR 17
Ok
run
 4
Ok
```

The two's complement integer 21 is expressed in binary as 10101, while the two's complement integer 17 is expressed as 10001; since 1 XOR 1 and 0 XOR 0 yield 0, while 1 XOR 0 yields 1, the result is 00100, or 4.

Example 3

```
10 PRINT -1 OR -2
Ok
run
-1
Ok
```

† The first bit of a two's complement integer indicates whether the integer is positive or negative. In binary notation, the two's complement integers from 0 to 32767 are expressed as 0000000000000000B to 0111111111111111B. The integers from −1 to −32788 are expressed as 1111111111111111B (−1) to 1000000000000000B (−32768). The value 1111111111111111B is obtained by adding 1 to the complement of 0000000000000001B (i.e., 1111111111111110B+1B=1111111111111111B). The binary representations of other negative two's complement integers can be obtained in the same manner.

2.10.4 String operations

String operations involve manipulation of character strings with operators. For example, the "+" operator makes it possible to concatenate (link) strings as shown in the example below.

Example

```
10 A$="File":B$="name"  :'Assigns string "File" to A$
20                      :'and string "name" to B$.
30 '
40 PRINT A$+B$          :'Concatenates string
50                      :'expressions A$ and B$ and
60                      :'displays the result.
70 '
80 PRINT "New "+A$+B$   :'Concatenates string
90                      :'expressions "New ", A$,
100                     :'and B$ and displays the
110                     :'result.

run
Filename
New Filename
Ok
```

Character strings can also be compared using the same relational operators as are used with numeric values.

```
10 A$="ALPHA":B$="BETA"
20       :'Assigns string "ALPHA" to A$ and
30       :'string "BETA" to b$.
40 '
50 IF A$<B$ THEN 100 ELSE 120
60       :'Jumps to line 100 if value of A$ is less
70       :'than that of B$; otherwise, jumps to
80       :'line 120.
90 '
100 PRINT A$;" IS LOWER THAN ";B$
110 END  :'Stops program execution.
120 PRINT A$;" IS NOT LOWER THAN ";B$

run
ALPHA IS LOWER THAN BETA
Ok
```

Strings are compared by taking one character at a time from each string and comparing their ASCII codes. The strings are equal if all codes are the same; if the codes differ, the character with the lower ASCII code is regarded as lower.

2.11 Functions

Functions are operations which return a specific value for a single operand. For example, the function SIN(X) returns the sine of the numeric value stored in variable X when the value in X is in radians. A variety of functions are built into PX-8 BASIC; these are referred to as intrinsic functions, and are described in Chapter 4.

PX-8 BASIC also allows the programmer to define his own functions; for details, see the explanation of the DEF FN statement in Chapter 4.

2.11.1 Integer functions

The CINT, FIX, and INT functions all return an integer value for an argument consisting of a numeric expression. These functions are described in detail in Chapter 4.

(1) CINT
The CINT function rounds the argument to the nearest integer value.

Examples:
```
CINT(1.1)  = 1
CINT(0.9)  = 1
CINT(-5.4)= -5
CINT(-5.7)= -6
```

(2) FIX
The FIX function truncates the argument; that is, it discards the decimal portion.

Examples:
```
FIX(1.1)  = 1
FIX(0.9)  = 0
FIX(-5.4) = -5
FIX(-5.7) = -5
```

(3) INT
The INT function returns the largest integer which is less than or equal to the argument.

2.12 Files

In general, a file is any set of data records which is output to or input from an external device (such as a disk drive) under a common identifier. This includes text files containing the program lines of BASIC programs, machine language program files and data files. Files can be stored in the RAM disk, microcassette tape or on floppy disks; however, they may also be input from and output to other devices. (See Chapter 5 for information on the types of file organizations used with BASIC for the PX-8.)

2.12.1 File descriptors

With PX-8 BASIC, files are identified by means of descriptors which consist of a device name and a file name. Together, these are referred to as the "file descriptor" and are specified as follows.

<device name> <option> <file name>

(1) Device name
PX-8 BASIC supports the concept of general device I/O. This means that input and output access to all devices can be handled in the same manner, regardless of whether the device accessed is a floppy disk drive, printer, the microcassette drive, or the RS-232C interface.

The format of all input and output commands is the same regardless of the type of I/O device. I/O devices are distinguished from one another by means of device names; the devices which can be addressed for I/O operations in this manner are as follows.

Name	Device	Modes
KYBD:	Keyboard	Input only
SCRN:	LCD screen	Output only
LPTØ:	Printer	Output only
COMØ:	RS-232C interface	Input and output
A: to H:	Disk devices	Input and output (Input only for ROM capsule)

Device names may be omitted when specifying file descriptors; however, the current CP/M default device is assumed if the device name is omitted.

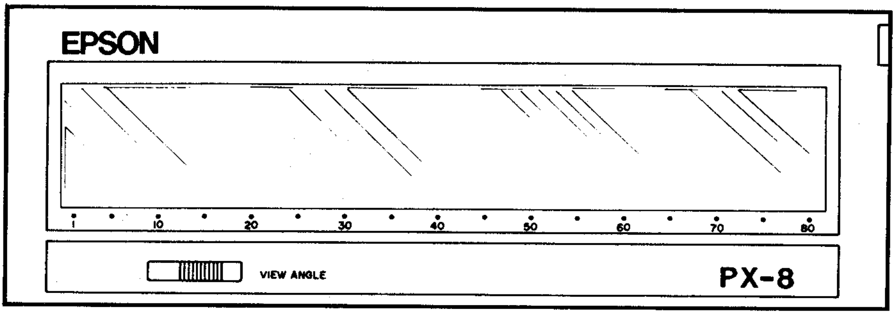
2.13 Display Screen

This section discusses the types of screens used with PX-8 BASIC, describes the various screen modes and the manner in which they are used, and explains the systems of coordinates which are used in specifying the locations of characters and graphics on the screen.

2.13.1 Real screen, virtual screen, and virtual screen window

With PX-8 BASIC, several different screen concepts are used for display. These include the real screen, virtual screens and the virtual screen window .

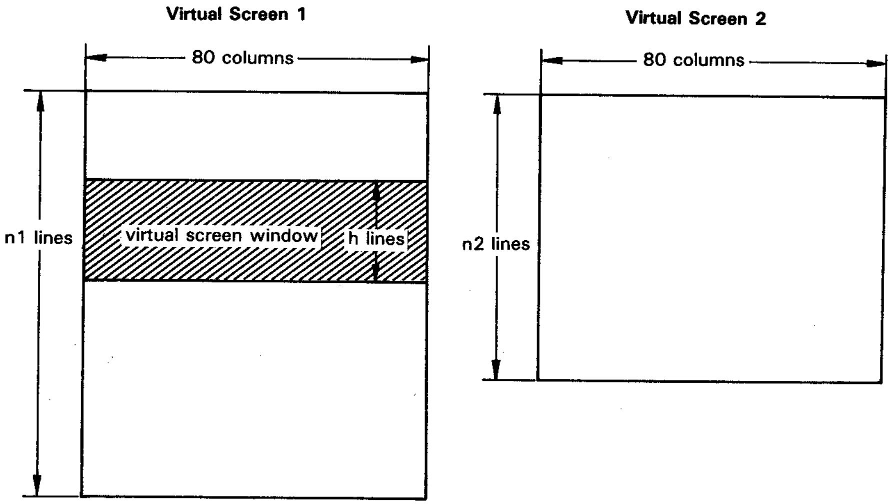
(1) Real screen
The PX-8 LCD screen permits display of up to 8 lines of 80 characters each or 480 by 64 dots of graphics. The LCD device on which characters and graphics are physically displayed is referred to as the real screen.



(2) Virtual screen
Although the LCD screen of the PX-8 allows display of up to 8 lines of 80 characters each, the concept of virtual screens has been introduced to make it possible to use application programs which require screens with even larger capacities. A virtual screen is not a physical device like the real screen, but exists in an area in memory which is referred to as VRAM (video random access memory) and is displayed through the "window" provided by the real screen. This VRAM is connected to the PX-8's 6301 slave CPU, and cannot be accessed by the PEEK or POKE commands of BASIC. Except in the graphics screen mode there are two virtual screens. The two virtual screens are independent of one another and either can be displayed under program control. In the split screen modes, it is possible to display both virtual screens on the two halves of the real screen.

When display is in the graphic screen mode, all of VRAM is used for storing the settings (on or off) of all the dots in the LCD screen, rather than codes representing character data.

(1) Screen mode 0 (the 80-column text screen mode)
In this screen mode, the two virtual screens each have a width of 80 columns (characters per line). The number of lines in each of the virtual screens can be set as desired by the user, provided that the total number of lines in the two screens does not exceed 48 and that each screen contains at least as many lines as the real screen. The virtual screen window can be switched back and forth between the two virtual screens, and can be scrolled up or down in the currently selected virtual screen to display its contents.



Conditions: $n1 \geq 8$ $h = 7$ or 8
$n2 \geq 8$
$n1 + n2 \leq 48$

(2) Screen mode 1 (the 39-column text screen mode)
In this screen mode, the virtual screen window is split vertically into two parts, each with a width of 39 columns. The remaining two columns of the 80 making up the real screen are used to display characters which represent a boundary between the two halves.

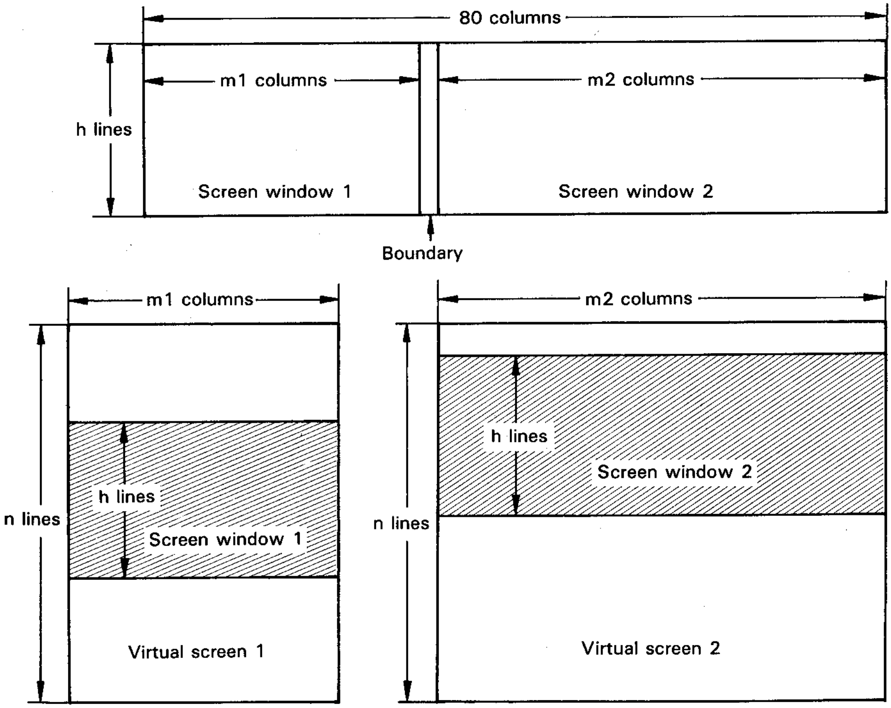
Both sides of the virtual screen window are positioned over the same virtual screen, and the display at the bottom of the left half of the virtual screen window is continued at the top of the right half.

(3) Screen mode 2 (the dual screen mode)

In this screen mode, the virtual screen window is vertically divided into two parts, each of which displays the contents of one of the virtual screens and which can be scrolled independently of the other. This makes it possible to display the contents of both virtual screens at the same time.

The width of each part of the screen window can be set as desired by the user as long as the total number of columns in both parts is equal to 79. The remaining column of the real screen is used to display a boundary character between the two screen windows.

In this screen mode, the two virtual screens each have a column width which is equal to the width of the screen window which displays its contents. The number of lines in each virtual screen can be set as desired by the user in the range from 8 to 48; however, both virtual screens must have the same number of lines.



Conditions: $m1+m2=79$
$h=7$ or 8
$8 \leq n \leq 24$

2.13.4 Scrolling control

The screen window has two scrolling modes, either of which can be selected by pressing the SCRN key (SHIFT + INS) or by escape sequence. These two modes are referred to as the tracking mode and the non-tracking mode.

(1) Tracking mode
In this mode, the screen window is scrolled along with the cursor (it "tracks" the cursor). If the cursor is outside the screen window when this mode is selected from the non-tracking mode, the screen window immediately moves to the position of the cursor in the virtual screen.

(2) Non-tracking mode
In this mode, the screen window does not follow the cursor.

The following keys are used for scrolling.

(1) SHIFT + ↑ (scroll up)
Pressing these keys together scrolls the screen window upward.

(2) SHIFT + ↓ (scroll down)
Pressing these keys together scrolls the screen window downward.

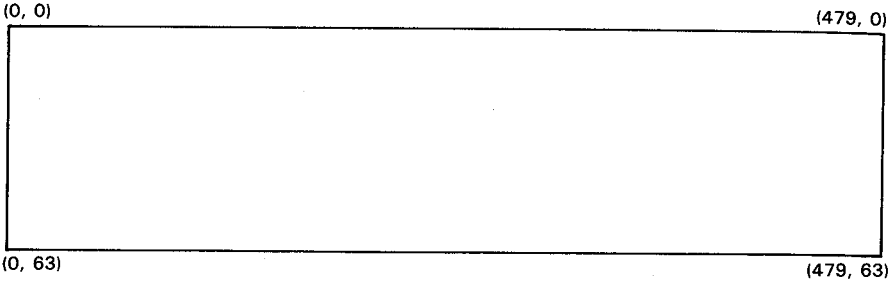
(3) CTRL + ↑ (top-of-screen)
Pressing these keys together moves the screen window to the top of the selected virtual screen.

(4) CTRL + ↓ (end of screen)
Pressing these keys together moves the screen window to the bottom of the selected virtual screen.

When the screen window is scrolled using the four keys described above, the cursor does not move from its current position in the virtual screen. This makes it possible to keep the cursor in one position while the screen is moved, even if scrolling is done in the tracking mode.

(5) SHIFT + INS (change scroll mode)
When the same virtual screen is used both as the write screen and the display screen, this key switches scrolling back and forth between in the tracking mode and the non-tracking mode. The scrolling mode used is switched each time this key is pressed.

With graphic coordinates, dots are numbered horizontally from 0 on the left side to 479 on the right, and vertically from 0 at the top of the screen to 63 at the bottom.



When the positions of individual dots on the screen are specified directly, the absolute form (<horizontal position>, <vertical position>) is used, for example PSET (19, 29) would switch on the twentieth dot across on the thirtieth row. This type of coordinate specification can be used with all graphic statements and functions.

It is also possible to specify the positions of screen dots in relation to previously specified dots; in this case, the coordinate specification takes the form STEP (<horizontal position>, <vertical position>). Here, STEP indicates that the values specified for (<horizontal position>, <vertical position>) are to be added to the values contained in a pointer called the last reference pointer or LRP which indicates the absolute coordinates of a previously specified dot. Relative coordinate specification can be used with the PSET, PRESET and LINE statements, and the last reference pointer (LRP) is updated by execution of these statements. Thus the following example will plot a row of ten dots at intervals of ten horizontal positions. Line 20 sets the position of the first point absolutely and so when line 40 is executed for the first time the LRP is the position (20, 10).

```
10 SCREEN 3
20 CLS
30 PSET (20,10)
40 FOR J=1 TO 9
50 PSET STEP (10,0)
60 NEXT J

Ok
     . . . . . . . . . .
```

The display on the LCD screen is a window on one or both of the virtual text screens. In mode 0, 1, or 3 only one of the virtual screens is displayed at a particular time. In mode 2 they can be displayed side by side.

If the screen mode is changed, the real and virtual screens will be cleared. However, if only the virtual screen is being changed, the window is changed and there is no clearing of either virtual screen. In illustrating this and successive operations, it is necessary to see the effect in each screen mode.

(1) MODE 0

The following sequence of operations shows how switching the display between the virtual screens leaves the virtual screens intact, and then goes on to illustrate how to move about the screens.

Type MENU and press RETURN, or use the shifted function key PF4.

From the BASIC menu login to a program area.

You will now be in screen mode 0 and see a portion of the first virtual screen. Without clearing the screen, type SCREEN 0,1 and press the RETURN key; the screen will clear but for "Ok" and a flashing cursor (this is the second virtual screen).

Now type SCREEN 0,0 and press RETURN; the first virtual screen will be redisplayed as you left it, except that the cursor will have moved down and another "Ok" will have been written onto the screen.

It is possible to switch screens in the direct mode using the CTRL and ← or → cursor keys. If you hold down the CTRL key and press the ← cursor key the display will switch to the first virtual screen. If the first virtual screen is being displayed, then no change will be apparent. Similarly, pressing the CTRL and → cursor key will show the second virtual screen. This can only be used in the direct mode and not in programs, even in INPUT statements.

The ↑ and ↓ cursor keys also have a function when pressed together with the CTRL key. They cause the first and last displayable lines respectively of the current virtual screen to be displayed. Depending on whether the screen is showing the function key assignments, 7 or 8 lines will be shown.

If the cursor is not on these lines it will not be displayed. Set the screen to the last lines of the virtual screen and type another key. If the key is not CTRL, This sets the first virtual screen to 40 lines and the second to 8. It also switches to the first virtual screen if you were not already displaying it, and turns off the function key assignments display on the bottom line. Note that there is no reference to boundary character, because there is no division of the screen in this screen mode, but the comma as separator must be inserted or an error will occur. You can see the extent of the first virtual screen by using the cursor keys. Now press the CTRL and → keys to display the second virtual screen. Now if you fill the screen with text, e.g. a listing, you can see that the cursor keys only allow movement on the real screen (because the virtual screen is the same size as the real screen).

(2) MODE 1

In changing to screen mode 1, use the SCREEN command to show the first virtual screen and turn the function key assignment display off by typing the following SCREEN command and pressing RETURN

```
SCREEN 1,Ø,Ø
```

The screen will clear to give a display with a boundary of two characters width in the centre of the creen. This boundary marker cannot be changed either in position or as the character displayed.

Use the cursor keys to move down the screen. You will see that when the cursor reaches the base of the left-hand side of the screen, instead of disappearing off the bottom and causing the screen to scroll up, it moves to the top right-hand part of the screen. Only when the cursor reaches the base of the right-hand side of the screen does the top line scroll off the left-hand side of the screen. Moreover, the right-hand side of the screen scrolls up into the left-hand side. This is because the two halves of the screen are displaying 16 lines of the virtual screen 39 columns wide but in two blocks of 8 lines side by side. This can be seen better if the following program is run to fill the screen with numbers on each line.

```
1Ø CLS
2Ø FOR J = 1 TO 2Ø
3Ø PRINT J
4Ø NEXT
```

Try the following to see how it differs from screen mode 0.

The boundary character can be reset using an ASCII character code with the CHR$( ) function. For example type the following to change the character to the vertical line graphics character (ASCII code 134):

SCREEN ,,,CHR$(134)

Note how both of the virtual screens are cleared when this command is executed.

(4) MODE 3

This screen mode is the mixed graphics and text mode. The dots are individually addressable, using the various graphics commands. This means that the video memory is largely devoted to storing the graphics information. Consequently text is limited to one virtual screen. This virtual screen is the same size as the real screen (80 columns and 8 lines) and so the various combinations of CTRL and cursor keys do not function in this screen mode.

Type SCREEN 3 and press RETURN. You can see you have entered graphics mode, because there is no longer a flashing cursor. It has changed to an underline character, which is the same as the insert cursor in the other screen modes. Thus the only indication of the PX-8 being in the insert mode is that the INS LED above the keyboard is lit.

As an example of the use of the graphics screen mode, the following command will draw a line from the top left-hand corner to the bottom right.

LINE (Ø, Ø) – (48Ø, 64)

With this type of device, the GET and PUT statements cannot be used concurrently when a file is opened in the random access mode. Further, record numbers must be used in sequence each time the GET or PUT statement is executed. (See the explanations of the GET and PUT statements in Chapter 4.)

When the PX-8's power is first turned on, the device names of the random access devices are as shown below. The name assignments can be changed with the CONFIG command of CP/M; see the PX-8 User's Manual for procedures for doing this. These device names are included in the file descriptor with the file name as described in section 2.12.

RAM disk	A:	(in main memory or optional RAM disk unit)
ROM capsule 1	B:	
ROM capsule 2	C:	
Floppy disk drives	D:, E:, F:, G:	
Microcassette drive	H:	

2.15.2 Sequential access devices

Sequential access devices are devices which can be open as files for input (the "I" mode) and/or output (the "O" mode). Sequential access devices supported by PX-8 BASIC and the modes in which they can be opened are as follows.

KYBD:	Keyboard, input only
SCRN:	LCD screen, output only
LPTØ:	Printer, output only
COMØ:	RS-232C interface, both input and output

Note that the colon is a part of each of these device names, and must be specified in the file descriptor whenever one of these devices is prepared for input/output by execution of an OPEN statement. However, no file name is specified following the device name when one of these devices is opened as a file.

Statements and functions which can be used with sequential access devices are as shown in section 2.15.7 below.

NOTE:
The format for file specification with the RS-232C interface is slightly different than that of other sequential access devices. See Chapter 6 for details.

2.15.6 Other devices

Communication with a number of other devices can be achieved if they have an RS-232C serial port. Besides transmitting data of a conventional nature (e.g. text sent over a modem or acoustic coupler) data can also be sent to control other equipment. Please consult your dealer for further information.

2.16 Error Messages

Error messages are displayed when errors are detected during execution of BASIC commands, statements or functions. If errors occur during program execution, execution stops and BASIC returns to the command level. However, it is possible to prevent this by including error processing routines which use the ON ERROR statement and ERR and ERL functions in programs; see the next section and corresponding explanations in Chapter 4 for details.

A complete list of the BASIC error codes, error messages, and causes of errors is shown in Appendix A.

Chapter 3

ENTERING BASIC WITH EXTENDED FORMAT COMMANDS

It is possible to enter BASIC with various commands appended which can set up the number of files allowed, upper memory limit etc. This chapter summarises these options.
The full syntax of the BASIC command is as follows:

BASIC [<filename>][/F:<no. of files>][/M:<upper memory limit>] [/S:<maximum record size>][/P:<program area no.>] [/R:<program area no.>]

All command operands indicated in brackets ([ ]) above are optional; the functions of each of the operands are as described below. The brackets are inserted to show the separation of the options and should not be typed in.

(1) BASIC <filename>

Example **BASIC TEST1**

When the name of a BASIC program file is specified following BASIC, that program is loaded and executed upon completion of the BASIC command in the same manner as if RUN "<filename>" had been entered immediately after start-up. If no file name extension is specified, .BAS is assumed. If neither the /R: nor /P: options are specified, BASIC starts operation using program area 1. See section 2.12 for details on <filename>. The above example would enter BASIC and proceed to RUN the program which was named TEST1 on the currently logged in disk drive. It would be placed in program area 1.

(4) BASIC /S:<maximum record length>

Example **BASIC /S:256**

The /S:<maximum record length> option sets the maximum record length which can be used with random access files to the value specified in < >; the maximum record length can be specified in decimal, hexadecimal (&H) or octal (&O) notation. When an OPEN "R" statement is executed after starting BASIC, the record size specified in that statement cannot be larger than the value specified with this option. If this option is not specified when the BASIC command is executed, the maximum record size is set to 128 bytes.

(5) BASIC /P:<program area no.>

Example **BASIC /P:3**

The /P:<program area no.> option specifies the program area which is selected at the time BASIC is started and automatically logs into that area. The value of <program area no.> must be specified as a number from 1 to 5.

(6) BASIC /R:<program area no.>

Example **BASIC /R:3**

As with the /P: option, the /R:<program area no.> option starts BASIC and selects and logs in the specified <program area no.>; in addition, this option immediately executes any BASIC program which is present in the specified program area. As with the /P: option, <program area no.> must be specified as a number from 1 to 5.

If both the /P: and /R: options are omitted, the BASIC program menu described in section 1.3 is displayed when BASIC is started.

If any errors are made while entering the BASIC command, an error messsge is displayed and the MENU screen or system prompt is redisplayed (depending on whether or not the MENU screen function is turned on). This also occurs if sufficient memory is not available for the BASIC working area (either because the upper memory limit or the starting address of BDOS is too low).

Chapter 5
MICROCASSETTE AND DISK FILES

This chapter describes procedures for creating and accessing files on microcassettes and floppy disks (including the RAM disk) with BASIC. The types of file covered include program files, random access files and sequential access files. In reading this chapter, keep in mind that file management is a process which involves a number of interrelated commands and statements, each of which must be prepared with consideration for the others. Also be sure to specify file descriptors in accordance with the rules described in the "2.12 Files" section of Chapter 2.

A summary of procedures for handling errors occurring during disk access is included at the end of this chapter, together with a review of general precautions to be observed in using microcassettes and floppy disks.

5.1 Program Files

This section reviews the commands and statements used to manipulate program files. In specifying these commands/statements, remember that the disk drive which is currently logged in is assumed unless otherwise specified in <file descriptor>. Thus if BASIC was entered from the command line with the logged in drive A:, files would be saved to the RAM disk. If you had entered BASIC from the menu with BASIC resident you may not know which is the currently logged in drive. Also remember that the CP/M operating system will automatically assume that the file name extension is explicitly specified.

```
SAVE <file descriptor>[ | ,A | ]
                        | ,P |
```

This command writes the program in memory to the disk or microcassette under the file name specified in <file descriptor>. If neither the A nor P options are specified the program is written to the disk or microcassette in compressed binary format. If the A (ASCII) option is specified, the file is written as a series of ASCII characters. If the P (PROTECT) option is specified, the file is saved in encoded binary format. A program saved using the P (PROTECT) option

NAME <old filename> AS <new filename>

This command is used to change the name of a file. Specify the current file name in <old filename> and the new file name which is to be assigned to the file in <new filename>. This command can be used to rename any type of file.

5.2 Sequential Files

This section describes procedures for creating, accessing and updating sequential data files. Sequential files are easier to create than random files, but they are not as easy to update and take longer to access. As the name implies, the items included in a sequential file are stored in the file in the order in which they are written, and must be read back in the same order. Because of these characteristics, sequential files are often used for address books, dictionaries or other files which are searched from the beginning when they are used and relatively rarely updated.

The statements and functions used to write to or read from sequential files are as follows:

OPEN, CLOSE
PRINT #, PRINT # USING, WRITE #
INPUT #, LINE INPUT #
EOF, LOC

Continue in this manner so that the following items are input to the file.

NAME	SECTION	DATE OF BIRTH
JOE SOAP	ACCOUNTS	08/05/49
FRED BLOGGS	ENTERTAINMENT	01/02/60
BETTY JONES	ACCOUNTS	09/09/55
GLORIA SMITH	CUSTOMER SERVICE	03/04/62

Note that starting the program again after it has been terminated (that is, after the file has been closed) erases all the data entered previously and creates a new file to which the same name has been assigned.

After the program has been executed, if the command FILES "A: is typed a file "EMPLOYEE.DAT" should have been added to the directory.

5.2.2 Accessing sequential files

The procedures for accessing sequential files are as follows:

(1) Execute an OPEN statement to open the file in the "I" mode.
(2) Read data from the file into variables in memory by executing either the INPUT# or the LINE INPUT# statement.
(3) Close the file after input has been completed by executing a CLOSE statement.

Notes:
1) Data is read from the beginning of the file each time the file is opened.
2) If all data included in the file is to be read at once, a DIM statement must be executed to dimension one or more variable arrays of the appropriate size.
3) An "Input past end" error will occur if an attempt is made to read data from a sequential file after the end of that file has been reached.

(4) After all data included in the original file has been written to the second file, close the original file and delete it with the KILL command.
(5) Write the new information to the second file.
(6) Rename the second file using the name which was assigned to the original file, then close the file.

The result is a sequential file which has the same file name as the original file, and which includes both the original data and the new data. A sample program illustrating this technique is shown below.

```
10 ON ERROR GOTO 310                     :'If file not found,
20 '                                       jump to 310.
30 '                                       If file exists, write
40 '                                       it to A:TEMP.
50 OPEN "I",#1,"A:EMPLOYEE.DAT"          :'Open file EMPLOYEE.
60 '                                       DAT for input.
70 OPEN "O",#2,"A:TEMP"                  :'Open temporary file
80 '                                       A:TEMP for output.
90 IF EOF(1) THEN 180                    :'If EOF is encountered
100 '                                      jump to 180.
110 LINE INPUT#1,A$                      :'Read data into A$.
150 PRINT#2,A$                           :'Write data in A$ to
160 '                                      A:TEMP.
170 GOTO 90                              :'Next data
180 CLOSE#1                              :'After all data has
190 '                                      been read, original
200 '                                      file is closed.
210 KILL "A:EMPLOYEE.DAT"                :'Kill original file
220 INPUT "NAME";N$                      :'Add new file entries.
230 IF N$="XX" THEN 280                  :'If XX is typed, close
240 LINE INPUT "SECTION? ";S$            :'file and end program.
250 LINE INPUT "DATE OF BIRTH? ";D$      :'
260 PRINT#2,N$;",";S$;",";D$             :'Write data to A:TEMP
270 PRINT:GOTO 220                       :'Next data
280 CLOSE                                :'Close A:TEMP.
290 NAME "A:TEMP" AS "A:EMPLOYEE.DAT":'Change filename back
300 '                                    to "EMPLOYEE.DAT"
310 IF ERR=53 AND ERL=30 THEN PRINT "File not found"
320 '                                     If A:EMPLOYEE.DAT not
330 '                                     exists, display "File
340 '                                     not found".
350 CLOSE:END                            :'
```

5.3 Random Files

More program steps are required to create and access random files than is the case with sequential files; however, random files have two advantages which make them more useful when there are large quantities of data which must be frequently updated. The first is that random files require less disk space for storage because data is recorded using a packed binary format, whereas sequential files are written as series of ASCII characters. The second advantage is that random files allow data to be accessed anywhere on the disk; it is not necessary to read through each data item in sequence, as is the case with sequential files. Random access is made possible by storing and accessing data in distinct, numbered units called records.

The statements and functions which are used with random files are as follows.

OPEN, CLOSE
FIELD, LSET/RSET
GET, PUT
LOC, LOF
MKI$, CVI
MKS$, CVS
MKD$, CVD

5.3.1 Creating random access files

The steps required to create random files are as follows:

(1) Open the file in the "R" mode.

For example

```
OPEN "R", #2, "STOCKLST.DAT", 50
```

opens file number 2 as a random access file named "STOCKLST.DAT". The record length is 50 bytes. If the record length is omitted, records of 128 bytes are assumed.

(2) Next, allocate space in the file buffer for each of the variables which are to be written to the random access file. This is done using the FIELD command, for example:

The following program example allows data to be input from the keyboard for storage in a random access file. In this example, one record is written to the file output buffer each time the PUT statement on line 100 is executed. The record number which is used by the PUT statement is that which is input at line 30.

```
10 OPEN"R",#1,"A:STOCKLST.DAT",36
                   :'Open file                         -(1)
20 FIELD#1,2 AS S$,30 AS N$,4 AS C$
                   :'FIELD data to variables           -(2)
30 INPUT "ENTER STOCK NO.";S%
                   :'Input data items.
40 IF S%=0 THEN CLOSE:PRINT"END":END
                   :'Enter 0 to finish.
50 INPUT "ENTER ITEM NAME";A$
60 INPUT "QUANTITY";C%
70 LSET S$=MKI$(S%)
                   :'LSET data to buffer(file buffer -(3)
80 LSET N$=A$
90 LSET C$=MKS$(C%)
100 PUT#1,S%       :'Write data to file.               -(4)
110 GOTO 30        :'Next entry
```

NOTE:
Once a variable name is specified in a FIELD command, do not use that name in an INPUT or LET statement. The FIELD statement assigns variable names to specific positions in the random file buffer, and using an INPUT or LET statement to store values in a variable specified in the FIELD statement will cancel this assignment and reassign the name to normal string space instead of to the random file buffer.

With random files, the LOC function returns the current record number; that is, the record number which is one greater than the number of the record last accessed by a GET or PUT statement. This function can be used to control the flow of program execution according to the total number of records which have been written to the file. For example, the following statement ends program execution if the current record number for file # 1 is greater than 50:

```
IF LOC(1)>50 THEN END
```

5.3.3 Hints for increased performance

When BASIC is started, memory is automatically reserved for use as random file buffers. The amount of memory reserved equals the number of bytes specified in the /S: option (the maximum record length of random files) times the number of files specified in the /F: option (the maximum number of files which can be opened at one time). Specify 0 in the /S: option to conserve memory if random access files are not to be used. Also, specify /F: <number of files> in the BASIC command if fewer than three files (the default value) are to be used.

The "s" option specifies whether the stop mode or the non-stop mode is to be used for reading from and writing to the file as follows:

S — Stop mode
N — Non-stop mode

Data is saved to the tape in blocks of 256 bytes. These blocks are duplicated on the tape. In order to achieve greater accuracy in reading and writing data these options allow the tape to be stopped between writing the duplicated blocks. In a normal BASIC SAVE operation data is always written in non-stop mode. It is possible to LOAD files in the stop mode, for instance:

```
LOAD "H:(S)PROGRAM"
```

will load the program named "PROGRAM" from the tape, stopping between each block. The tape will physically stop and start during this operation. It takes longer than normal operation and should only be used for loading BASIC programs if difficulty is experienced in loading a particular program.

In opening a data file for input it is often possible for a "Disk read" error to occur if the file was originally written in the non-stop mode and is read in the same mode. Greater reliability can thus be achieved by using the stop mode for reading back data.

If the option is omitted when the file is opened in the "I" mode, the mode actually used is that specified when the file was created. When a new file is created in the "R" mode, data is written in the stop mode unless otherwise specified.

The "v" option specifies whether data is to be automatically verified after being written, as follows:

V — Data is automatically verified (with a Cyclic Redundancy Check) from the beginning of the file through to its close after write access has been completed.
N — Data is not verified.

When omitted, the value set in the system display is assumed.

(2) Errors occurring during output

CLOSE the applicable file immediately if any of the following errors occur during output with statements such as PRINT # or PUT. The contents of the file may be destroyed if output is continued.

Device unavailable
Disk write protected
Disk read error
Disk write error

(3) Errors occuring when a file is closed

Although a file will be closed if an error occurs when a CLOSE statement is executed, the contents of the file are not assured. Further, if an error occurs when an attempt is made to close more than one file with a single CLOSE statement there is a possibility that some files will not be closed. If an error occurs in this situation, additional CLOSE statements must be executed until no further errors occur.

(4) "Disk write protected" error

This error will occur if a disk is changed without having CLOSEd the files on the original disk. All files should be closed before a disk is changed, then the RESET command should be executed.

If the disk is changed without closing the files, the RESET command will close the files. However, it will be necessary to execute the RESET command a number of times until no further errors occur before the new disk can be accessed.

Chapter 6

SEQUENTIAL ACCESS USING DEVICE FILES

This Chapter describes procedures for sequential access to the RS-232C serial communications interface and other external I/O devices such as the keyboard, screen and printer.

6.1 Using the RS-232C Interface

The PX-8's RS-232C interface makes it possible to connect the PX-8 to RS-232C compatible devices such as printers, acoustic couplers, or other PX-8s. Support for RS-232C interface access is a standard feature of BASIC for the PX-8. The RS-232C port is handled as a sequential input/output device, and is identified by the device name "COMØ:".

6.1.1 Opening the RS-232C interface

The RS-232C communications interface is opened for data communication by executing an OPEN statement. The parameters of this statement specify the mode in which the interface is to be opened (input or output), the file number whicl: is to be assigned to the device and the communication protocol and control options. The communication protocol and control options are specified as a string of up to seven characters, each of which determines the setting of one of seven communication options. Devices which are connected to the RS-232C port must be compatible with the communication protocol under which the interface is opened.

The format for specification of the OPEN statement and the meanings of the various communication options are described in detail below.

(1) OPEN statement

The general format of the OPEN statement for opening the RS-232C interface port is the same as that used when opening sequential access files on ting of one option. The general format of the string specifying these options is (blpscxh), where "b" specifies the bit rate, "l" the word length, "p" the type of parity check to be made, and "s" the number of stop bits between characters. Option "c" specifies which of the interface's four control lines are to be checked when the interface is opened and during data communication. The "x" option specifies whether or not communication is to be controlled according to XON/XOFF protocol (that is, whether the PX-8 is to issue or respond to "wait until I catch up" requests during communication with the device on the other end of the RS-232C line), and the "h" option indicates whether Shift-In/Shift-Out sequences are to be used to indicate whether characters are upper case or lower case (applicable only when the data word length is 7 bits and when it is necessary to send characters whose codes are 128 or greater). These options are summarized in the table on page 6-4.

WARNING:
When using the XON/XOFF or Shift-In/Shift-Out options, make sure they are reset to off on exiting from the program. It is only possible to set them from BASIC and not from the CONFIG program of the CP/M utility ROM.

If care is not taken the XON/XOFF or SI/SO options may be set when they are not required, this can result in the following problem when machine code is being received. The SI, SO, XON or XOFF characters may be part of the machine code. If the receiving computer has either or both of the XON/XOFF or SI/SO options set, when these characters are received they will be intercepted by the RS-232C software and acted upon. This means they will be lost as part of the machine code file. Moreover, if an SO character is received, the data following will be changed. It will have the high bit set. This will further corrupt the file. Similarly XON/XOFF characters can be intercepted and cause the transmisssion to hang indefinitely.

When using communications programs other than BASIC ones, a warm start should be made before setting the RS-232C parameters using the CONFIG program. This ensures that the SI/SO and XON/XOFF parameters are set to be off.

Meanings of each position in the "blpscxh" string are as follows.

b — A hexadecimal integer from &H0 to &HF which determines the bit rate as follows:

0 — Send bit rate = 1200 bps, receive bit rate = 75 bps
1 — Send bit rate = 75 bps, receive bit rate = 1200 bps
2 — 110 bps
3 — Not specifiable
4 — 150 bps
5 — Not specifiable
6 — 300 bps
7 — Not specifiable
8 — 600 bps
9 — Not specifiable
A— 1200 bps
B— Not specifiable
C— 2400 bps
D— 4800 bps
E— 9600 bps
F— 19200 bps

l — A number from 6 to 8 which determines the number of bits per character (the data word length):

6: 6 bits/character
7: 7 bits/character
8: 8 bits/character

p — A letter which determines the type of parity check to be made:

N: No parity check
E: Even parity
O: Odd parity

s — A number which determines the number of stop bits to be included between each character:

1: 1 bit
2: 1.5 bits
3: 2 bits

the line. This prevents data from being lost due to receive buffer overflow when the speed of data transmission is greater than the speed with which data received can be unloaded from the buffer.

X: XON/XOFF protocol used for send control.
N: XON/XOFF protocol not used for send control.

h — A letter which determines whether the shift-in/shift-out (SI/SO) control sequences are to be used. The SI/SO control sequences are used when sending 8-bit data with a data word length of 7 bits. Shift-in/shift-out control can be used only when the data word length is 7 bits:

S: Shift-in/shift-out control used.
N: Shift-in/shift-out control not used.

The (blpscxh) options can be completely or partially omitted when the communications interface is opened; however, spaces must be specified for options which are omitted if there are any following options. If b, l, p or s are omitted, the default values are those which have been set with the CONFIG command. If c is omitted, "F" is used, and if x and/or h are omitted, "N" is used.

(2) OPEN modes

The RS-232C interface can be opened in either the "I" or "O" modes. The "I" mode is specified for input and the "O" mode is specified for output. If both input and output are to be performed simultaneously, the interface must be opened as two files (one for input and one for output). In this case, the communication protocol and control options used are those specified in the first OPEN statement executed: the options will be ignored if they are included in the file descriptor of the second OPEN statement.

Example

```
10 OPEN"I",#1,"COM0:(68N3FXN)"
20 OPEN"O",#2,"COM0:(48E2F)"
.
.
.
100 PRINT#2,A$
110 INPUT#1,B
```

In the example above, the option specification (48E2F) on line 20 is ignored and the output file (#2) is opened using the protocol specified on line 10 (68N3FXN).

/

6.1.2 Output to the RS-232C Interface

Statements and functions used for access to the RS-232C interface are as follows:

Statements

OPEN, CLOSE, INPUT #, LINE INPUT #, PRINT #,
PRINT # USING, LOAD, LIST, RUN, MERGE

Functions

EOF, LOC, LOF, INPUT$

The following statements can be used to output data to the RS-232C port.

PRINT #
PRINT # USING

When data is output using these statements, the data format is the same as when data is output to a disk drive.

(1) Control line checks for the "O" mode

(a) CTS (Clear To Send)
Output to the RS-232C port becomes possible when the level on this line becomes HIGH.

(b) DSR (Data Set Ready)
When the DSR send check bit is OFF (when bit 2 of option "c" is 1), data is output to the RS-232C port regardless of the level on the DSR line. When the DSR check bit is ON ("0"), data is output after checking the level on the DSR line and waiting for it to become HIGH.

(2) Errors applicable to the "O" mode

(a) Device unavailable
The RS-232C interface cannot be used.

(b) Device time out
The level on the DSR line did not become HIGH within a certain period of time when output to the RS-232C port was attempted after opening it in the "O" mode with the DSR send check bit (bit 2 of option "c") set to ON ("0"). This error also occurs if the STOP key is pressed while transmission is being deferred for some reason.

(b) Device time out
This error occurs if the level on the DSR line does not become HIGH within a certain period of time after an attempt is made to open the RS-232C interface in the "I" mode with the DSR receive check bit set to ON (with bit 1 of option "c" set to 0). The same is true if the level on the DCD line does not become HIGH within a certain period of time after an OPEN"I" statement is executed with the DCD check bit set to ON (with bit 0 of option "c" set to 0)

(c) Device fault
This error occurs if the RS-232C port is opened for input with the DSR receive or DCD check bits set to ON (with bits 1 or 0 of option "c" set to 0), and the level of a corresponding line becomes LOW during input.

(d) Device I/O error
This error occurs if a parity error, overrun error or framing error occurs during input. Although this error is reset if input is continued, there is no assurance that data received into the receive buffer at that time will be correct.

(e) Input past end
This error occurs if the STOP key is pressed during input from the RS-232C interface with INPUT#, LINE INPUT# or INPUT$.

6.1.4 RS-232C functions

The four functions used with the RS-232C interface are as follows.

EOF
LOC
LOF
INPUT$

(1) EOF (<file no.>)
This function returns −1 (true) when the receive buffer is empty, and 0 (false) when the buffer is not empty.

(2) LOC (<file no.>)
This function returns the number of bytes of data remaining in the receive buffer.

6.2 Printer and Display Screen

With PX-8 BASIC, a printer and the LCD screen are supported as sequential output devices. This means that data can be output to the printer or screen using the file output statements (PRINT # and PRINT # USING), as well as the dedicated printer/display statements LIST/LLIST and PRINT/LPRINT.

The device name used to open the printer as a device file is "LPTØ:", and that used to open the display screen is "SCRN:".

(1) Statements
Statements which can be used for output to the display screen or printer when it is handled as a device file are as follows:

OPEN, PRINT #, PRINT # USING, CLOSE, LIST "<file descriptor>"

(2) Errors
The "Device time out" error will occur if the printer is not ready for output (because it is offline or out of paper, for instance).

Appendix A ERROR CODES AND ERROR MESSAGES

When an error occurs in a BASIC program, it is detected by the interpreter and a message is printed. In most cases the error stops the program and will not allow it to continue. BASIC will return to the direct mode and present the error message. It will not always be obvious what exactly has caused the error. It may be something as simple as a mistyped command which BASIC does not recognise, an error of logic or any one of a series of programming faults. This appendix is an attempt to help the user/programmer to find out what exactly he has done wrong. It is not easy to cover each and every cause of an error, because some errors are particular to the logic of a program and simply cannot be predicted. However, many are due to definite reasons, and these are described below.

Each error has a code associated with it, which is useful for trapping errors and also simulating them. See ERROR, ON..ERROR, ERR and ERL in Chapter 4 for details of their use. A list of errors in numerical order is given at the end of this section. However, as the error is normally encountered as a message, the details of each error are given in alphabetical order. The number at the left of each error is the error code.

54 Bad file mode

A statement or function was used with a file of the wrong type.

Possible causes:
(i) An attempt was made to use PUT, GET or LOF with a sequential file.
(ii) A non BASIC program file was specified in a LOAD command.
(iii) A file mode other than I, O, or R was specified in an OPEN statement.
(iv) An attempt was made to MERGE a file that was not saved in ASCII format.

64 Bad file descriptor

An illegal file name was specified in a LOAD, SAVE or KILL command or an OPEN statement (for example, a file name with too many characters).

52 Bad file number

A statement or command references a file that has not been opened, or the file number specified in an OPEN statement is outside of the range of file numbers that was specified when BASIC was started.

Appendix A ERROR CODES AND ERROR MESSAGES

When an error occurs in a BASIC program, it is detected by the interpreter and a message is printed. In most cases the error stops the program and will not allow it to continue. BASIC will return to the direct mode and present the error message. It will not always be obvious what exactly has caused the error. It may be something as simple as a mistyped command which BASIC does not recognise, an error of logic or any one of a series of programming faults. This appendix is an attempt to help the user/programmer to find out what exactly he has done wrong. It is not easy to cover each and every cause of an error, because some errors are particular to the logic of a program and simply cannot be predicted. However, many are due to definite reasons, and these are described below.

Each error has a code associated with it, which is useful for trapping errors and also simulating them. See ERROR, ON..ERROR, ERR and ERL in Chapter 4 for details of their use. A list of errors in numerical order is given at the end of this section. However, as the error is normally encountered as a message, the details of each error are given in alphabetical order. The number at the left of each error is the error code.

54 Bad file mode

A statement or function was used with a file of the wrong type.

Possible causes:
(i) An attempt was made to use PUT, GET or LOF with a sequential file.
(ii) A non BASIC program file was specified in a LOAD command.
(iii) A file mode other than I, O, or R was specified in an OPEN statement.
(iv) An attempt was made to MERGE a file that was not saved in ASCII format.

64 Bad file descriptor

An illegal file name was specified in a LOAD, SAVE or KILL command or an OPEN statement (for example, a file name with too many characters).

52 Bad file number

A statement or command references a file that has not been opened, or the file number specified in an OPEN statement is outside of the range of file numbers that was specified when BASIC was started.

(i) Transmission via the RS-232C interface was not enabled within a certain period of time after an OPEN"O" statement was executed with the DSR send check set to ON by option "c" of the communications format specification.

(ii) The STOP key was pressed while output to the RS-232C interface was being deferred for some reason.

(iii) The DSR or DCD line did not become high within a certain period of time after an OPEN"I" statement was executed with the DSR receive check or DCD check set to ON by option "c" of the communications format specification.

(iv) The printer was not ready when output to the printer was attempted.

68 Device unavailable

An attempt was made to access a drive which did not contain a floppy disk or the RS-232C interface was not available.

66 Direct statement in file

A program line without a line number was encountered during execution of a LOAD or MERGE command, or an attempt was made to LOAD a data file or machine language program.

61 Disk full

Either the disk directory or the disk itself has no space left.

7Ø Disk read error

An error occurred while data was being read from a disk device.

71 Disk write error

An error occurred while data was being written to a disk device.

69 Disk write protect

Possible causes:

(i) An attempt was made to write data to a disk which is protected by a write protect tab.

(ii) An attempt was made to write data to a disk drive without executing the RESET command after replacing the disk in that drive.

(iii) An attempt was made to write data to a ROM capsule.

12 Illegal direct

A statement that is illegal in the direct mode (such as DEF FN) was entered as a direct mode command.

5 Illegal function call

A statement or function was incorrectly specified.

Possible causes:

(i) Specification of a negative number or a number which is too large as an array variable subscript.
(ii) Specification of zero or a negative number as the argument in the LOG function.
(iii) Specification of a negative number as the argument of the SQR function.
(iv) Specification of a non-integer exponent with a negative mantissa.
(v) A call to a USR function for which the starting address has not yet been defined.
(vi) An incorrectly specified argument in any of the following functions or statements:
ALARM, ALARM$, ASC, CSRLIN, INP, INSTR, LEFT$, LOCATE, MID$, ON...GOSUB, ON...GOTO, OUT, PEEK, POKE, POWER, PRESET, PSET, RIGHT$, SCREEN, SPACE$, SPC, STRING$, TAB, VARPTR, WAIT, WIND.
(vii) Specification of a non-existent line number in a DELETE statement.
(viii) Attempting to erase a non-existent variable array with an ERASE statement.
(ix) Specification of a number other than 1 to 5 as the parameter of a LOGIN, PCOPY or STAT statement.
(x) Execution of a RENUM command with parameters which do not conform to the rules for specifying such commands.
(xi) Specification of an undefined array variable or a variable whose value has not yet been defined in a SWAP statement.
(xii) Execution of the EDIT command when the virtual screen window was less than 38 columns wide.
(xiii) Specification of a number other than 1 to 10 as the parameter of a KEY command.

62 Input past end

Possible causes:

(i) An INPUT statement was executed for a file which was empty or one from which all data had been read. To avoid this error, use the EOF function to detect the end of the file.

7 Out of memory

Memory available is insufficient for processing required.

Possible causes:
(i) Program is too long.
(ii) The program uses too many variables.
(iii) The subscript range specified in a DIM statement is too large.
(iv) An expression has too many levels of parentheses.
(v) FOR...NEXT loops or GOSUB...RETURN sequences are nested to too many levels.
(vi) The stack area size or machine language area specified in a CLEAR statement is too large.
(vii) Insufficient memory was available to allow a program to be copied with the PCOPY statement.

14 Out of string space

Insufficient memory space is available for storage of characters in string variables.

6 Overflow

A numeric value was encountered whose magnitude exceeds the limits prescribed by PX-8 BASIC. If underflow occurs, zero is assumed and execution continues without error.

Possible causes:
(i) The result of an integer calculation was outside the range from -32768 to 32767.
(ii) The result of a single or double precision number calculation was outside the range from 1.70141E38 to – 1.70141E38.
(iii) One of the operands of a logical operation was not in the range from – 32768 to 32767.
(iv) The argument specified for the CINT function or POINT statement was outside the range from – 32768 to 32767.
(v) The argument specified for the HEX$ or OCT$ function was outside the range from – 32768 to 65535.
(vi) The number specified as one of the parameters of the LOCATE or WIND statements was outside the prescribed range.

20 RESUME without error

A RESUME statement was encountered outside an error processing routine.

(iii) Unmatched parentheses.
(iv) Wrong delimiting punctuation (commas, full stops, colons or semicolons) used between statements, expressions or arguments.
(v) Variable name beginning with a character other than a letter.
(vi) Keyword used as the first letters of a variable name.
(vii) Wrong number or type of arguments specified in a function or statement.
(viii)Type of value included in a DATA statement did not match the corresponding variable in the list of variables specified in a READ statement.

72 Tape access error

Possible causes:
(i) An attempt was made to access an access-inhibited microcassette file.
(ii) An attempt was made to mount a tape without executing the REMOVE command to unmount the previous tape.
(iii) The REMOVE command was executed while the tape in the microcassette drive was in the unmounted condition.
(iv) An attempt was made to change the setting of the tape counter while the tape in the microcassette drive was in the mounted condition.

67 Too many files

An attempt was made to create a new disk file after all directory entries were full.

13 Type mismatch

A string expression was used where a numeric expression is required, or vice versa.

Possible causes:
(i) An attempt was made to assign a numeric value to a string variable.
(ii) An attempt was made to assign a string to a numeric variable.
(iii) The wrong type of value was specified as the argument of a function.

8 Undefined line number

A non-existent line number was specified in one of the following commands or statements — EDIT, GOTO, GOSUB, RESTORE, RUN, RENUM — or when attempting to delete a non-existent line by typing a number and pressing the RETURN key.

18 Undefined user function

A call was made to an undefined user function.

Possible causes:

17	Can't continue
18	Undefined user function
19	No RESUME
20	RESUME without error
21	Unprintable error
22	Missing operand
23	Line buffer overflow
24	Device time out
25	Device fault
26	FOR without NEXT
28	Communication buffer overflow
29	WHILE without WEND
30	WEND without WHILE
50	FIELD overflow
51	Internal error
52	Bad file number
53	File not found
54	Bad file mode
55	File already open
57	Device I/O error
58	File already exists
61	Disk full
62	Input past end
63	Bad record number
64	Bad file descriptor
6	Direct statement in file
67	Too many files
68	Device unavailable
69	Disk write protect
70	Disk read error
71	Disk write error
72	Tape access error

Appendix C *PX-8 BASIC CONSOLE ESCAPE SEQUENCES*

Whereas BASIC as a high level language has a large number of commands and functions, it is also possible to print sequences of characters which will allow further or additional commands which affect output to the screen. This appendix deals with the use of these commands. Some of them are not additonal to BASIC but duplicate BASIC commands in a way which can make programming easier for advanced programmers in some circumstances.

The sequences involve the printing of the ESCAPE character, ASCII code 27 decimal (1B in hexadecimal), followed by one or more characters, the values of which determine the command to be carried out. In the remainder of this appendix the ESCAPE character is denoted by the letters "ESC". The User's Manual contains further information on using the sequences under CP/M or in machine code programs. Not all the commands are supported in BASIC, for example because they interact with the screen editor.

The following table lists the character sequences for the commands alphabetically to make them easy to find. Notes on the use of the commands and parameters are given in numerical order following the table. The numerical values are given in decimal notation in the table and headings.

Control Code	Function
ESC "%"	Access CGROM directly
ESC 243	Arrow key code
ESC 246	Buffer clear key
ESC "C"	Character table
ESC 246	Clear keyboard buffer
ESC "*"	Clear screen
ESC 245	CTRL key code
ESC 215	Cursor find
ESC 243	Cursor key code
ESC "="	Cursor position set
ESC 214	Cursor type select
ESC "P"	Dump screen
ESC "T"	Erase to end of line
ESC "Y"	Erase to end of screen
ESC 210	Display characters on real screen
ESC 208	Display mode set
ESC 198	Dot line write

Control Code	Function
ESC 213	End locate
ESC 215	Find cursor
ESC 177	Function key code returned
ESC 176	Function key string returned
ESC 211	Function key display select
ESC "C"	International Character Sets
ESC 161	INS LED off
ESC 160	INS LED on
ESC 242	Key repeat interval time
ESC 240	Key repeat on/off
ESC 241	Key repeat start time
ESC 244	Key code scroll
ESC 247	Key shift set
ESC "T"	Line erase
ESC 198	Line dot draw
ESC 213	Locate end of screen
ESC 212	Locate top of screen

ESC "%"

Reads the character corresponding to the specified code from the character generator ROM and displays it at the present cursor position in the currently selected screen (in the virtual screen for modes 0, 1, and 2, and in the real screen for mode 3). The sequence is as follows:

```
PRINT CHR$(27);"%";CHR$(n)
```

The value of n is the ASCII code corresponding to the character to be displayed.

ESC "*"

Clears the currently selected screen and moves the cursor to the home position.

ESC "="

Moves the cursor to the specified position in the screen being written. In the tracking mode, the screen window is moved so that the cursor is positioned at screen centre if the position specified is outside the screen window. The tracking mode is turned on and off by pressing the SHIFT and SCRN keys together. The sequence for moving the cursor is as follows:

```
PRINT CHR$(27);" = ";CHR$(m+31);CHR$(n+31);
```

Here, m specifies the vertical cursor position and n specifies the horizontal position. The value of n should be greater than 1 and less than the screen width in the particular screen mode being used. The value of m should be greater than 1 and less than the number of lines in the virtual screen.

The ESC "=" sequence duplicates the LOCATE command with its first two parameters.

ESC "C" <character>

Used to select one of the nine international character sets as follows:

The <character> is a letter which corresponds to the character sets of one of the following countries. It must be an uppercase character.

US ASCII	PRINT CHR$(27);"CU"
France	PRINT CHR$(27);"CF"
Germany	PRINT CHRS(27);"CG"
England	PRINT CHR$(27);"CE"
Denmark	PRINT CHR$(27);"CD"

blank. This is done as follows:

PRINT CHR$(27);CHR$(144);CHR$(n – 1);CHR$(m);

Numbers specified for n and m must satisfy all of the following conditions.

$$\emptyset \leqq (n-1) \leqq (R-1)$$
$$1 \leqq m \leqq R$$
$$(n-1+m-1) \leqq R$$

Here, R is the number of virtual screen lines in mode 0, 1, or 2 and is the number of screen window lines in mode 3.

ESC CHR$(145)

Scrolls (n – 1) lines down starting at line n so that line n becomes blank. This is done as follows:

PRINT CHR$(27);CHR$(145);CHR$(n-1);CHR$(m);

Numbers specified for n and m must satisfy all of the following conditions:

$$\emptyset \leqq (n-1) \leqq (R-1)$$
$$1 \leqq m \leqq R$$
$$(n-1+m-1) \leqq R$$

Here, R is the number of virtual screen lines in mode 0, 1, or 2 and is the number of screen window lines in mode 3.

ESC CHR$(148)

In modes 0, 1, and 2 this escape sequence sets the number of lines n which are moved by one scrolling operation. The actual scrolling is carried out by printing an ESC 150 sequence. The number of lines are set up using the following sequence:

PRINT CHR$(27);CHR$(148);CHR$(n);

The number specified for n must be greater than 1 and less than the number of lines in the screen window.

This escape sequence does nothing in mode 3.

ESC CHR$(149)

In modes 0, 1, and 2 this escape sequence determines whether scrolling is performed automatically. The automatic scrolling mode is referred to as the track-

ESC CHR$(164)

Lights the NUM LED, but does not select the numeric keypad.

ESC CHR$(165)

Turns off the NUM LED.

ESC CHR$(176)

This ESC code is used to disable the string printed by a programmable function key. However, with input from the command line or from an INPUT statement the PX-8 will be returned to the normal string printing mode of the programmable function keys. If you wish to determine if any of the programmable function keys have been pressed, use the ESC CHR$(176) mode in combination with INPUT$ or with INKEY$. An example of this is shown below.

```
10 PRINT CHR$(&H1B);CHR$(&HB0);
20 PRINT HEX$(ASC(INPUT$(1)));" ";
30 GOTO 20

run
E0  E1  E2  E3  E4  E5  E6  E7  E8  E9
↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑
PF1 PF2 PF3 PF4 PF5 PF6 PF7 PF8 PF9 PF10
```

ESC CHR$(177)

This ESC code re-enables the programmable function keys so that a string is printed when they are pressed.

ESC CHR$(198)

In mode 3, this escape sequence draws a line on the graphic screen using the dot pattern specified by the user. No operation is performed when this sequence is executed in modes 0, 1, or 2. The elements of the sequence are as follows:

Byte 1:	CHR$(27)
Byte 2:	CHR$(198)
Byte 3:	High byte of horizontal starting position
Byte 4:	Low byte of horizontal starting position
Byte 5:	High byte of vertical starting position
Byte 6:	Low byte of vertical starting position
Byte 7:	High byte of horizontal ending position
Byte 8:	Low byte of horizontal ending position
Byte 9:	High byte of vertical ending position

Byte 1;	CHR$(27)	
Byte 2:	CHR$(199)	
Byte 3:	Function code (1:PSET, 0: PRESET)	
Byte 4:	Vertical dot position — n1	
Byte 5:	High byte of horizontal dot position	n2
Byte 6:	Low byte of horizontal dot position	

Numbers specified for n1 and n2 must be in the following ranges:

Ø ≦ n1 ≦ 63, Ø ≦ n2 ≦ 479

ESC CHR$(2Ø8)

Switches the display mode. Mode specification is as follows:

Mode 0		Mode 1	
Byte 1:	CHR$(27)	Byte 1:	CHR$(27)
Byte 2:	CHR$(208)	Byte 2:	CHR$(208)
Byte 3:	CHR$(0)	Byte 3:	CHR$(1)
Byte 4:	CHR$(n1)	Byte 4:	CHR$(n1)
Byte 5:	CHR$(n2)		

Mode 2		Mode 3	
Byte 1:	CHR$(27)	Byte 1:	CHR$(27)
Byte 2:	CHR$(208)	Byte 2:	CHR$(208)
Byte 3:	CHR$(2)	Byte 3:	CHR$(3)
Byte 4:	CHR$(n1)		
Byte 5:	CHR$(n2)		
Byte 6:	CHR$(p)		

The meanings of n1, n2, m, and p are as follows:

n1 Number of lines in virtual screen 1

n2 Number of lines in virtual screen 2

m Number of columns in virtual screen 1

p ASCII code corresponding to desired boundary character

ESC CHR$(211)

Turns on or off display of function key definitions. This is done as follows:

PRINT CHR$(27);CHR$(211);CHR$(n)

Function key definitions are displayed when 0 is specified for n, and are not displayed when 1 is specified. The default value is 1.

ESC CHR$(212)

In modes 0, 1, and 2 this escape sequence moves the screen window to the top of the virtual screen containing the cursor. No operation is performed if this sequence is executed in mode 3. The position of the cursor remains unchanged.

ESC CHR$(213)

In modes 0, 1, and 2 this escape sequence moves the screen window to the end of the virtual screen containing the cursor. No operation is performed if this sequence is executed in mode 3. The position of the cursor remains unchanged.

ESC CHR$(214)

In modes 0, 1, and 2 this escape sequence selects the type of cursor to be displayed. This sequence does nothing if executed in mode 3. The sequence consists of three bytes as follows:

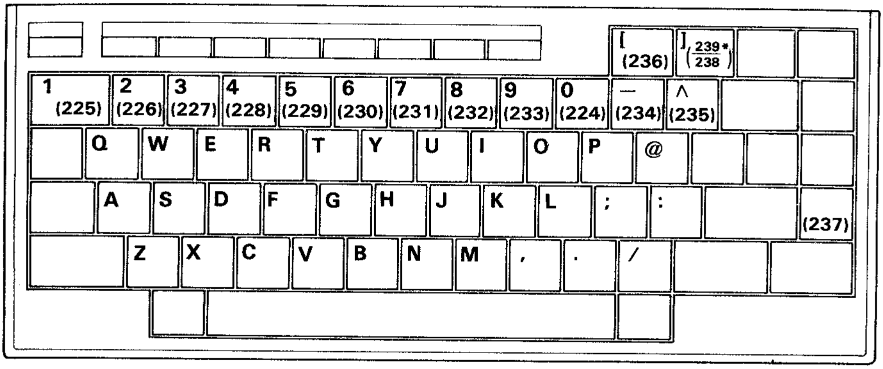
Byte 1: CHR$(27)
Byte 2: CHR$(214)
Byte 3: CHR$(n)

Here, n specifies the type of cursor displayed as follows:

0	Block cursor, flashing
1	Block cursor, non-flashing
2	Underline cursor, flashing
3	Underline cursor, non-flashing

The cursor will be set to the normal flashing block cursor if the return key or one of the cursor keys is pressed.

Note:
User character definitions for codes 224 to 239 can be displayed by pressing the graph key together with certain other keys on the keyboard. Keys pressed for each code are as shown in the figure below.



* Press together with SHIFT key.

A sample definition for character code 230 is shown below:

**PRINT CHR$(27);CHR$(224);CHR$(23Ø);CHR$(12);
CHR$(12);CHR$(3Ø);CHR$(63);CHR$(12);CHR$(18);
CHR$(Ø);CHR$(Ø);**

After executing this sequence, the character corresponding to code 230 can be displayed by pressing GRAPH and the key marked "230" in the figure above.

ESC CHR$(24Ø)

Controls the key repeat function. This sequence consists of three bytes as follows:

Byte 1:	CHR$(27)
Byte 2:	CHR$(240)
Byte 3:	CHR$(n)

If 0 is specified for n, the repeat function is turned off. If 1 is specified, it is turned on.

ESC CHR$(241)

Sets the starting time for the key repeat function. The sequence consists of three bytes as follows:

ESC CHR$(245)

Sets the CTRL + arrow key codes. This sequence consists of six bytes as follows:

Byte 1:	CHR$(27)
Byte 2:	CHR$(245)
Byte 3:	Code for CTRL + →
Byte 4:	Code for CTRL + ←
Byte 5:	Code for CTRL + ↑
Byte 6:	Code for CTRL + ↓

ESC CHR$(246)

Clears the keyboard buffer of all unprocessed input characters.

ESC CHR$(247)

The ESC 247 code allows the programmer to switch the various shift keys on and off. Thus the numeric key pad can be set on, or the shift key 'held down'. The key state is set to normal by the user pressing the appropriate key, so it is advisable to program with the possiblity in mind that the key may be reset outside program control.

The sequence of characters is as follows:

Byte 1:	CHR$(27)
Byte 2:	CHR$(247)
Byte 3:	CHR$(n)

Numbers which may be specified for n and their meanings are as follows:

n (Decimal)	Shift state
0	Normal
2	SHIFT
4	CAPS LOCK
6	CAPS LOCK SHIFT
16	NUM
18	Numeric SHIFT
32	GRPH
34	GRPH SHIFT
64	CTRL
66	CTRL SHIFT

This sequence does nothing if numbers other than those above are specified for n.

Here, <digit> is a number from 0 to 9 and the argument specified is any numeric or string expression. <digit> specifies which of the USR routines is being called, and corresponds to the digit specified in the DEF USR statement for that routine. If <digit> is omitted USR0 is assumed. The address specified in the DEF USR statement determines the starting address of the subroutine.

When a USR function call is made a value is placed in CPU register A which specifies the type of argument specified. The value placed in register A may be any of the following.

Value	Type of argument
2	Two-byte integer (two's complement)
3	String
4	Single precision floating point number
8	Double precision floating point number

If the argument is an integer

FAC+0 contains the lower 8 bits of the argument
FAC+1 contains the upper 8 bits of the argument.

If the argument is a single precision floating point number
FAC+0 contains the lowest 8 bits of the mantissa
FAC+1 contains the middle 8 bits of the mantissa
FAC+2 contains the highest 7 bits of the mantissa (with leading 1 suppressed). Bit 7 is the sign of the number (0 for positive and 1 for negative)
FAC+3 is the exponent minus 128. The binary point is the bit to the left of the most significant bit of the mantissa.

If the argument is a double precision floating point number, FAC-4 to FAC-1 contain four more bytes of the mantissa with the lowest 8 bits in FAC-4.

If the argument is a string, the DE register pair points to three bytes called the “string descriptor.” Byte 0 of the string descriptor contains the length of the string (0 to 255); and bytes 1 and 2 are the lower and upper 8 bits of the starting address of the string in string space.

CAUTION:
If the argument is a string literal in the program, the string descriptor will point to program text. Be careful not to alter or destroy your program in this way. To avoid unpredictable results, add +“ ” to the string literal in the program.

4. Interrupts

Machine language subroutines can be written to handle interrupts. All interrupt handling routines should save the stack, registers A to L and the PSW. Since an interrupt received automatically disables all further interrupts, they should always be re-enabled before returning from the subroutine.

Appendix F ASCII CHARACTER CODES

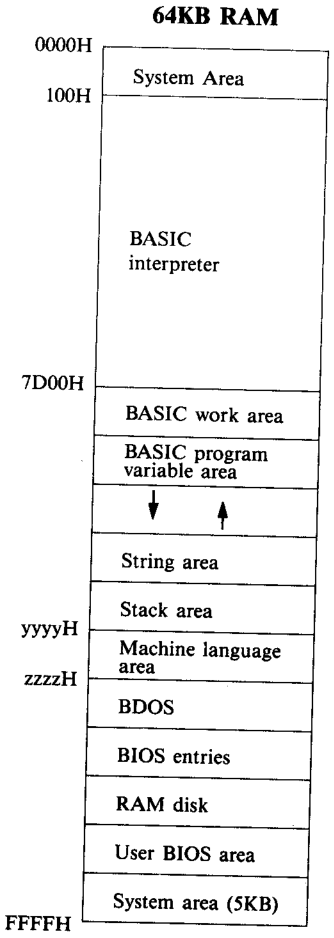
	Hex. No.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Hex. No.	Binary No.	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	0	16	SP 32	0 48	@ 64	P 80	` 96	p 112	┼ 128	○ 144	160	176	192	208	⌂ 224	240
1	0001	1	17	! 33	1 49	A 65	Q 81	a 97	q 113	┴ 129	♣ 145	161	177	193	209	↵ 225	241
2	0010	2	18	" 34	2 50	B 66	R 82	b 98	r 114	┬ 130	♥ 146	162	178	194	210	226	242
3	0011	3	19	# 35	3 51	C 67	S 83	c 99	s 115	┤ 131	♦ 147	163	179	195	211	227	243
4	0100	4	20	$ 36	4 52	D 68	T 84	d 100	t 116	├ 132	♠ 148	164	180	196	212	228	244
5	0101	5	21	% 37	5 53	E 69	U 85	e 101	u 117	─ 133	♪ 149	165	181	197	213	229	245
6	0110	6	22	& 38	6 54	F 70	V 86	f 102	v 118	│ 134	☎ 150	166	182	198	214	230	246
7	0111	7	23	' 39	7 55	G 71	W 87	g 103	w 119	┌ 135	✈ 151	167	183	199	215	231	247
8	1000	8	24	(40	8 56	H 72	X 88	h 104	x 120	┐ 136	🚗 152	168	184	200	216	232	248
9	1001	9	25	) 41	9 57	I 73	Y 89	i 105	y 121	└ 137	🐱 153	169	185	201	217	232	249
A	1010	10	26	* 42	: 58	J 74	Z 90	j 106	z 122	┘ 138	🚶 154	170	186	202	218	234	250
B	1011	11	27	+ 43	; 59	K 75	[91	k 107	{ 123	▒ 139	↑ 155	171	187	203	219	235	251
C	1100	12	28	, 44	< 60	L 76	\ 92	l 108	¦ 124	█ 140	↓ 156	172	188	204	220	236	252
D	1101	13	29	- 45	= 61	M 77	] 93	m 109	} 125	▬ 141	× 157	173	189	205	221	237	253
E	1110	14	30	. 46	> 62	N 78	^ 94	n 110	~ 126	▌ 142	÷ 158	174	190	206	222	238	254
F	1111	15	31	/ 47	? 63	O 79	_ 95	o 111	⌂ 127	● 143	± 159	175	191	207	223	239	255

ASCII

NOTES:

1. *$(0)_D$ through $(31)_D$ are control characters.*
2. *$(32)_D$ through $(127)_D$ are ASCII characters.*
3. *Characters displayed for codes E0 to FF can be defined by the user. For further details see section 2.6.2, "Control Characters," and the User's Manual.*

Appendix G *MEMORY MAP*



yyyyH .... Can be found at memory addresses 7D38H and 7D39H. Address 7D38H contains the lower byte of yyyyH and address 7D39 contains the higher byte.

zzzzH......... Varies according to the size of RAM disk.
zzzzH can be found in locations 6 and 7 in page zero. From BASIC,

PEEK(6)+PEEK(7)*256

will return the present value of zzzzH.

The pattern which makes up each row is specified by the bit settings of the number, sent as an ASCII code. It is easiest to design the characters on squared paper and translate it into numbers. Dots in the pattern which are turned on correspond to a "1" in the binary number and dots which are turned off correspond to a "0". The design which gives the pattern must be converted from binary into a decimal or hexadecimal number. For those not familiar with converting the binary numbers to decimal the procedure is best explained with an example.

128	64	32	16	8	4	2	1
		*		*		*	*

The pattern in the diagram shows which dots of the line will be set. This particular pattern would correspond to the binary number "00101011". To convert the number to decimal, add the numbers above the bòxes where there is dot to be set. Thus 32+8+2+1 gives a total of 43. For a whole character a pattern would be produced as follows. The numbers at the right are the ones which would define the row in a program.

128	64	32	16	8	4	2	1	
				*	*			= 12
			*	*	*	*		= 30
		*					*	= 33
		*					*	= 33
			*			*		= 18
				*	*			= 12
		*	*	*	*	*	*	= 63
								= 0

Characters are six dots wide by eight high. The two left-hand positions are always ignored. The numbers used to define the rows will thus be in the range 0 to 63 (0 to 3F hexadecimal). Any attempt to use the left-hand two positions of the full eight bits of the byte will be ignored.

If the bottom row of the character is filled in there will be no space between the character printed and the character on the next row of the screen. If you want the two characters to be contiguous, dots on this row should be set; otherwise the row should be left blank.

```
 100 PRINT CHR$(X);CHR$(X+1);
 110 NEXT:PRINT
 120 FOR X=&HE2 TO &HFB STEP 4
 130 PRINT CHR$(X);CHR$(X+1);
 140 NEXT:PRINT
1000 DATA &H02,&H02,&H02,&H02,&H02,&H03,&H0F,&H00
1010 DATA &H00,&H30,&H00,&H18,&H24,&H04,&H3C,&H00
1020 DATA &H00,&H00,&H00,&H00,&H00,&H00,&H00,&H00
1030 DATA &H00,&H00,&H00,&H00,&H00,&H00,&H00,&H00
2000 DATA &H00,&H03,&H00,&H07,&H04,&H02,&H01,&H02
2010 DATA &H00,&H00,&H00,&H30,&H10,&H20,&H3C,&H00
2020 DATA &H04,&H08,&H08,&H08,&H07,&H00,&H00,&H00
2030 DATA &H00,&H00,&H00,&H08,&H30,&H00,&H00,&H00
3000 DATA &H00,&H00,&H00,&H00,&H01,&H08,&H08,&H08
3010 DATA &H00,&H00,&H00,&H00,&H20,&H04,&H04,&H04
3020 DATA &H04,&H03,&H00,&H00,&H00,&H00,&H00,&H00
3030 DATA &H08,&H30,&H00,&H00,&H00,&H00,&H00,&H00
4000 DATA &H00,&H00,&H00,&H00,&H00,&H01,&H01,&H00
4010 DATA &H00,&H00,&H00,&H1C,&H24,&H04,&H3C,&H04
4020 DATA &H00,&H00,&H00,&H07,&H00,&H00,&H00,&H00
4030 DATA &H08,&H10,&H20,&H00,&H00,&H00,&H00,&H00
5000 DATA &H00,&H00,&H00,&H00,&H04,&H04,&H0F,&H00
5010 DATA &H00,&H00,&H00,&H00,&H24,&H24,&H3C,&H00
5020 DATA &H00,&H00,&H00,&H00,&H00,&H00,&H00,&H00
5030 DATA &H00,&H00,&H00,&H00,&H00,&H00,&H00,&H00
6000 DATA &H00,&H00,&H00,&H00,&H00,&H00,&H0F,&H00
6010 DATA &H00,&H00,&H00,&H00,&H04,&H04,&H3C,&H00
6020 DATA &H00,&H00,&H00,&H00,&H00,&H00,&H00,&H00
6030 DATA &H08,&H08,&H00,&H00,&H00,&H00,&H00,&H00
7000 DATA &H01,&H01,&H01,&H01,&H01,&H01,&H01,&H01
7010 DATA &H00,&H00,&H00,&H00,&H00,&H00,&H00,&H00
7020 DATA &H00,&H00,&H00,&H00,&H00,&H00,&H00,&H00
7030 DATA &H00,&H00,&H00,&H00,&H00,&H00,&H00,&H00
```

Lines 1000 onwards contain the data for successive characters. Each line contains the data for successive rows of each character. All characters and data have been entered in hexadecimal notation. See HEX$ in Chapter 4 for conversion between decimal and hexadecimal numbers. The four individual characters making up each large character are defined in the order top left, top right, bottom left, and bottom right.

Lines 20 to 80 read this data from the DATA statements and download each character in the same manner as the previous program.

Lines 90 to 140 print the top halves and then the bottom halves of the characters. Since four user-defined characters are used together to make one large screen character, the STEP to find the next character in the loops is four.

User-defined characters can only be printed from screen mode 3 — by bit image mode printing of a screen dump — unless they have also been down-

The program disables the STOP key, then initialises variables XP and YP which are used to position the first character.

The loop forming lines 40 to 70 defines the characters which will be printed if the keys "A" to "G" are pressed. They are stored in the array C$( ). The characters are built up as follows: the first two characters are the top pair of the block of user defined characters. Next, two backspace characters are added (the ASCII code for a backspace is 8) because the position of the cursor is one to the right of the characters when they have been printed. This leaves the cursor in the position of the first character of the pair. By adding a linefeed character (ASCII code 10) the cursor is moved down one line to the bottom left of the block of four. The bottom row of the block is now added as the next two characters. If the block is printed at this stage the cursor will be placed to the right of the bottom of the block of four user-defined characters. The next group of characters must be printed two places to the left on the top of the first group. This is achieved by adding a 'cursoı up' character (ASCII code 30, or 1E in hexadecimal), and then four backspace characters. This gives a total string length of twelve characters. When this string is printed, the sequence appearing on the screen will be as follows: the top two of the block will appear; the cursor will then move back two positions and down one, print the bottom two characters, then move up, back four, and be ready to print again.

The program as a whole prints a cursor which is defined as two horizontal lines using the predefined graphics character whose ASCII code is 133. This together with the backspaces to move to the left is defined as the variable CSR$ in line 80. To allow a space to be printed, line 90 defines SP$ as two normal spaces with backspaces added to allow printing from left to right.

Lines 100 to 190 form the main part of the program. The cursor is placed on the second line of the screen at the extreme right. It cannot be placed directly on the edge. When one of the groups of characters from the array C$( ) is printed the LOCATE command tests to see if the string length added to the horizontal position of the LOCATE command is greater than 81. If it is, the complete string will be printed on the next line of the screen. The strings of the array are twelve characters long — it does not matter that some are cursor movement control codes. The maximum position the strings of C$( ) can be printed from is thus 69. The cursor is printed on the second line so that when the screen scrolls up on reaching the bottom line the top half of the character groups are not lost.

of characters are on the line.

Lines 50 to 80 check for input from the keyboard and exclude unwanted characters. If the space bar is pressed line 70 calculates the position in which to place two spaces so that the cursor is erased.

Line 90 calculates the position for the first character of the block to be printed whenever a key in the range "A" to "G" is pressed.

Lines 100 and 110 determine which of the the blocks of characters to print from the ASCII code of the key pressed. The corresponding user-defined characters are printed in positions calculated according to the number of blocks already on the line. Line 120 prints the upper pair of the block, and line 140 the lower pair.

Line 150 then updates the counter for the number of blocks of characters printed. As with the previous program, when the line is full the cursor is moved to the next line.

This program is best understood by working through what will actually happen when the program runs. You can do this by calculating the values of XP and YP, or having the PX-8 print them to a file or to an external printer. It is rather difficult to write a program such as this because if the program has to be altered the recalculation may not be easy. The previous program is easier to re-program and understand.

```
330 IF T=1 AND X=9 THEN ZX=46:ZY=1:GOTO 350
340 ZX=T*9+X+28:ZY=2
350 LOCATE ZX,ZY,0:IF SS=0 THEN PRINT CHR$(143); ELSE PRINT
CHR$(144);
360 LOCATE XP(X),YP(X),0
370 H1=HR(X):M1=ME(X):S1=SD(X)
380 IF AS$(X)="-" THEN GOSUB 2000 ELSE GOSUB 1000
390 PRINT USING "& &";FNS$(HRS,MIN,SEC,":");FNS$(FNP(L(1,X))
,FNP(L(2,X)),FNP(L(3,X)),"/");
400 NEXT
410 GOTO 220
420 STOP KEY ON:CLS:END
1000 SEC=S+S1
1010 MIN=M+M1+SEC\60:SEC=SEC MOD 60
1020 HRS=H+H1+MIN\60:MIN=MIN MOD 60
1030 IF HRS<24 THEN 1080
1040 DY=DY+HRS\24:HRS=HRS MOD 24
1050 ND=FNN(MN,YR)
1060 MN=MN+DY\ND:DY=DY MOD ND
1070 IF MN>12 THEN YR=YR+1:MN=MN-12
1080 RETURN
2000 SEC=S-S1:IF SEC<0 THEN SEC=SEC+60:M1=M1+1
2010 MIN=M-M1:IF MIN<0 THEN MIN=MIN+60:H1=H1+1
2020 HRS=H-H1:IF HRS>=0 THEN 2060
2030 DY=DY-1+HRS\24:HRS=HRS+(HRS\24+1)*24
2040 ND=FNN(MN-1,YR):IF DY<1 THEN MN=MN-1:DY=DY+ND
2050 IF MN<1 THEN YR=YR-1:MN=MN+12:DY=FNN(MN,YR)
2060 RETURN
3000 DATA 9
3010 DATA 05,04,"+","00:00:00",2,1,3
3020 DATA 30,04,"+","01:00:00",2,1,3
3030 DATA 55,04,"-","05:00:00",1,2,3
3040 DATA 05,06,"+","09:00:00",3,1,2
3050 DATA 30,06,"+","10:00:00",2,1,3
3060 DATA 55,06,"+","01:00:00",2,1,3
3070 DATA 05,08,"+","07:30:00",3,1,2
3080 DATA 30,08,"+","03:00:00",2,1,3
3090 DATA 55,08,"-","04:00:00",1,2,3
```

number of the days in the month and year required. In evaluating the function P1 will correspond to the number of the month and P2 to the number of the year. The algorithm is as follows:

If the number of the month is greater than 7 and is even then the number of days is equal to 31, and if it is odd then the number of days is 30. This is achieved with the part of the logical statement which reads as follows:

```
(P1>7)*(((P1 MOD 2)=Ø)*31+((P1 MOD 2)=1)*3Ø)
```

If the month has an even number the expression ((P1 MOD 2)=0) is true and a value of −1 will be returned; when multiplied by 31 this will give a value of −31. The expression ((P1 MOD 2)=1) will be false and so return a value of 0. If the number of the month is greater than 7 the expression (P1>7) will be true and hence return a value of −1. Thus the total value returned will be 31. This is easier to see if the values are placed under the expressions as follows :

```
(P1>7)*(((P1 MOD 2)=Ø)*31+((P1 MOD 2)=1)*3Ø)
 -1   *((     -1      )*31+(      Ø      )*3Ø)=31
```

If the month has a value greater than 7 and is an odd month the expression ((P1 MOD 2)=1) will evaluate as true and the expression ((P1 MOD 2)=0) will be false. Thus the total value for the complete expression above will be 30.

If the month is less than 7 the expression (P1>7) will be false and by returning a value of 0 for false makes the whole expression have a value of 0.

A second expression evaluates the part of the algorithm which deals with the remaining months. This reads as follows:

```
(P1<8)*(((P1 MOD 2)=Ø)*3Ø+((P1 MOD 2)=1)*31
  +(P1=2)*(((P2 MOD 4)=Ø)+((P2 MOD 4)< >Ø)*2))
```

This is built up from a similar expression to that for the months from August onwards. However, it also has to take account of the month of February and the fact that it has a different number of days in a leap year. Apart from February, note that even months have 30 days and odd months have 31 days in contrast to the other part of the year. The part involved with February is

```
(P1=2)*(((P2 MOD 4)=Ø)+((P2 MOD 4)< >Ø)*2)
```

If P1 corresponds to February, the expression (P1=2) will be true and thus return a value of −1. The rest of the expression involves deciding on whether the year is a leap year or not. A leap year occurs if the year is divisible by 4. If this is the case ((P2 MOD 4)=0) will be true, otherwise ((P2 MOD 4)< >0) will be variable is converted to a string the leading space is added to the string. Thus STR$(15) would give a string of length 3, the string " 15". The function MID$(STR$(P),2) returns the string starting with the second character. Thus if P has the value 5, STR$(P) has the value " 5" and the value returned by MID$(STR$(P),2) would be "5", whereas if it were 15 it would be "15". The leading zero is added in each case to give "05" and "015" respectively. By taking the two characters at the right of this string, the zero is lost if the number had two digits before being converted to a string, but is retained if there was only one. The string returned by the function is of the form "HH:MM:SS" in the example described.

Line 60 is a function which is used for manipulating the order of the day, month and year in the date to cope with the fact that different countries display the order of these values in their own way. The order is determined by three items in the DATA statement associated with that particular country. This makes it easy to add more countries or change the ones already displayed.

Each DATA statement (lines 3000 – 3090) ends with the three numbers 1,2 and 3. These are in a different order depending on the country. By using this number with the function defined in line 60, the month, date or year can be found as follows. The number is passed to the function as the variable P1 will be either 1, 2 or 3, and the value determines which of the statements (P1=1), (P1=2) or (P1=3) is true. Suppose P1 has the value 1. The statement (P1=1) will be TRUE and thus return a value of –1, whereas all the other logical statements will be false and so return a value of zero. The value returned by the function will be the value of the variable MN when the signs have been taken into account.

Line 70 initializes the variables used temporarily at various stages in the program. The variable OH is used to see if the hour has changed in line 300 and if it has, a sound is made. Variables T and SS are used to change the display around the heading as the time changes.

Line 80 reads the number of countries from the first DATA statement in line 3000. This makes it easy to alter the number of countries without having to alter the program. On the basis of the number read, the following arrays are dimensioned in line 90:

The arrays XP and YP hold the position to print the time and date for each country;

In line 270 the day, month and year which were determined in line 240 are transferred into variables DY, MN and YR which are used for the subtraction and addition subroutines in lines 1000 to 1080 and 2000 to 2060 respectively.

Line 290 prints the current time and the current date on either side of the main heading. The date is printed in the format used in Great Britain since the program as it stands has the values set for GMT (Greenwich Mean Time). The order is found by using the function FNP of line 60, with the values set to 2, then 1 and then 3 to give day, month and then year. The leading zeros and separator are inserted using the function FNS$ of line 50.

Line 300 sounds the hour.

Line 310 determines the current time, using the function FNT of line 20 to split the string returned by TIME$ into hours, minutes and seconds.

Lines 320 to 350 form a routine to change the graphics characters around the heading.

Line 360 takes the location of the position to print the time and date from the arrays XP and YP using the value of X from the loop as index.

Line 370 determines the time offset for the particular country by indexing the arrays where they were stored when read from the DATA statements.

Line 380 checks to see if the time offset is positive or negative, and goes to the appropriate subroutine to determine the time in that country.

Line 390 formats the printout using the function FNS$ of line 80. The order of printing the date is determined from the array L.

When all the countries have been updated line 410 redirects execution to line 220 to repeat the process.

The addition and subtraction of time is carried out in the subroutines which precede the DATA statements.

Lines 1000 to 1080 form the addition subroutine. These routines are straightforward arithmetic additions. The seconds are converted to minutes and seconds, and the process is repeated for hours and minutes. The minutes offset is added as variable M1, and the hours offset as H1. When the hours have been deter-

Index

EPSON OVERSEAS MARKETING LOCATIONS

EPSON AMERICA, INC.

3415 Kashiwa Street
Torrance, CA 90505 U.S.A.
Phone: (213) 539-9140
Telex: 182412

EPSON UK LTD.

Dorland House
388 High Road,
Wembley, Middlesex, HA9 6UH, U.K.
Phone: (01) 902-8892
Telex: 8814169

EPSON DEUTSCHLAND GmbH

Am Seestern 24
4000 Düsseldorf 11
F.R. Germany
Phone: (0211) 5952-0
Telex: 8584786

EPSON ELECTRONICS (SINGAPORE) PTE. LTD.

No.1 Maritime Square, #02-19
World Trade Centre
Singapore 0409
Phone: 2786071/2
Telex: 39536

EPSON ELECTRONICS TRADING LTD.

Room 411, Tsimshatsui Centre,
East Wing, 66, Mody Road
Tsimshatsui Kowloon, Hong Kong
Phone: 3-694343/4
3-7213427
3-7214331/3
Telex: 34714

EPSON ELECTRONICS TRADING LTD. TAIWAN BRANCH

1,8F K.Y. Wealthy Bldg. 206, Nanking
E. Road, Sec, 2, Taipei, Taiwan, R.O.C.
Phone: 536-4339
536-3567
Telex: 24444

EPSON FRANCE S. A.

114, Rue Marius Aufan
92300 Levallois-perret
France
Phone: (1) 758-77-00
Telex: 614169

EPSON AUSTRALIA PTY. LTD.

Unit 3, 17 Rodborough Road
Frenchs Forest, NSW 2086
Australia
Phone: (02) 452-5222
Telex: (71) 75052

(vi) and statements which make it possible to control the computer's power supply under program instruction.

Other features of this BASIC are as follows.

- BASIC can be made resident in RAM after loading it from ROM capsule, allowing it to be started up almost instantly whenever the PX-8's power is turned on.

- The BASIC program area in memory is divided into five parts, allowing up to five different programs to be held simultaneously. This facilitates development of programs and makes it possible to use multiple program applications.

- The LCD screen can be switched between any of four different screen modes, which are (1) a full screen text mode in which the screen consists of 7 or 8 lines (depending on whether function key definitions are displayed) of 80 columns each; (2) a split screen mode, in which the screen is divided into two consecutive halves of 39 characters each; (3) a twin screen mode, in which two separate areas in display memory are displayed simultaneously; and (4) a graphic mode which is used for drawing figures and diagrams with PX-8 BASIC's graphic statements.

- The microcassette drive built into the PX-8 is supported as a disk device. This means it can be used in almost exactly the same manner as if it were a disk drive. This also applies to auxiliary storage devices such as ROM capsules (which because they are ROMs can only be read from and not written to), and an area in random access memory which is referred to as RAM disk. Floppy disk drives connected to the PX-8's high speed serial port are treated as normal disk drives. The ability to use the PX-8's random access memory in the same manner as if it were a disk drive (RAM disk) is particularly useful because it allows utilization of disk-based functions which ordinarily would require a disk drive. The microcassette drive has some limitations. The main differences are the speed of access and the fact that tape access is sequential, so that random access file handling is not possible.

Procedures for installing and starting up BASIC are described below. Before following these procedures, you may wish to use the CONFIG command of CP/M to select the printer output port (high speed serial or RS-232C) and to specify the default parameters for communication through the RS-232C interface. See the PX-8 User's Manual for details.

1.2 Starting BASIC

When you switch on the PX-8, there are a number of possible states in which the computer can be. If it is in the middle of an applications program (either because the power has been switched off with CTRL held down, or the power has automatically switched off because there was no input), then it is necessary to exit from the program before loading BASIC. The other possiblities are that the CP/M command line is displayed, or the MENU screen has been set.

(a) USING BASIC FROM THE CP/M COMMAND LINE

Entering BASIC from the CP/M command line is achieved by treating the BASIC ROM as a program on a disk drive. Thus if the system prompt says "A>", you would need to type "B:BASIC" or "C:BASIC" followed by the RETURN key in each case, depending on which socket the ROM has been placed in, and which ROM has been allocated to which drive. (Allocation of drive names is carried out with the CONFIG program which is described in the User's Manual. If you wish to find out in which drive BASIC is located, use the DIR command of CP/M rather than using the CONFIG program). On pressing RETURN, BASIC will be loaded into RAM. If you wish to run a BASIC program directly you can do so by adding the name of the program and its drive location, as for example running the program "NAME" which is located in drive A: when BASIC is in drive C:

```
A>C:BASIC   A:NAME.BAS
```

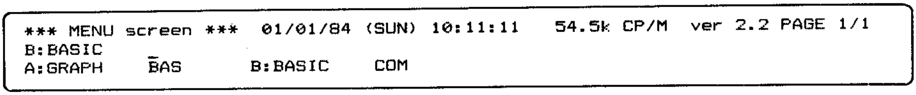
In this case there MUST be a space between the word BASIC and the drive name in which the program sits. The extension ".BAS" showing that "NAME" is a BASIC program is not necessary. However, if a BASIC program has been named with a different extension (e.g. ".GPH" so that all graphics programs are identifiable), then this extension MUST be used; otherwise the computer cannot find the program. Also in this particular case, since the default drive name is A: it is not necessary to type it in before the name of the BASIC program.

(b) ENTERING BASIC FROM THE MENU

The PX-8 has a mode which allows easy loading of programs which are set up on a menu. A description of how to use the MENU is given in the User's Manual, and details of setting up the menu for use with BASIC is given in section 1.4.2 (Warm starts) of this manual.

If you have set up the MENU so that BASIC is one of the files to appear on

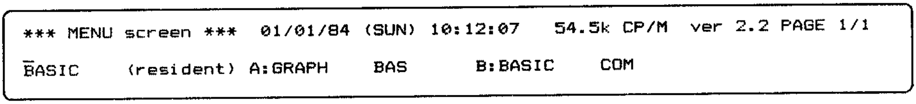
The MENU screen will automatically put the first program in the top left hand position onto the command line. If, as in the screen above, the first program is not BASIC.COM, then you must put BASIC onto the command line using the cursor keys to select it. The screen display will then change as follows



```
*** MENU screen ***   01/01/84  (SUN)  10:11:11    54.5k CP/M   ver 2.2 PAGE 1/1
B:BASIC
A:GRAPH     BAS        B:BASIC     COM
```

and BASIC.COM will be flashing in the main MENU area. Now BASIC can be loaded by pressing the RETURN key, as in the previous example.

(iii) BASIC is resident in memory
If BASIC had been used when the PX-8 was switched off, instead of the MENU screen showing BASIC or indeed another program on the command line, the following display would come up



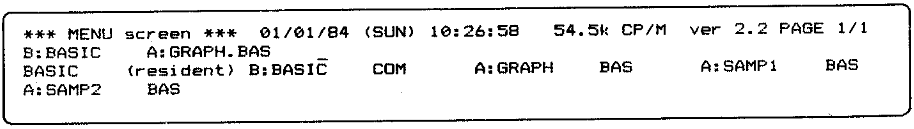
```
*** MENU screen ***   01/01/84  (SUN)  10:12:07    54.5k CP/M   ver 2.2 PAGE 1/1

BASIC     (resident) A:GRAPH     BAS        B:BASIC     COM
```

and the first program position showing "BASIC (resident)" would be flashing. The command line is empty when this occurs.

Simply pressing RETURN will enter BASIC without loading it.

(iv) Running a BASIC program directly
The MENU can also be used to select and RUN a BASIC program directly. If the program is selected by means of the cursor keys, the screen will appear as follows.

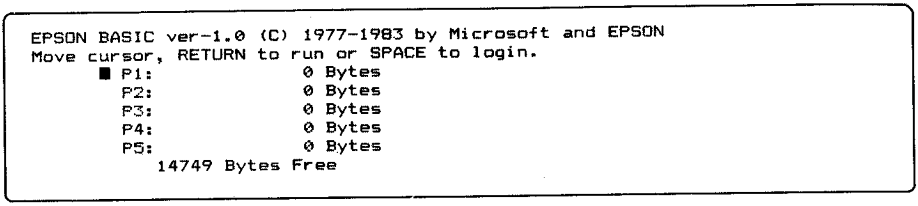


```
*** MENU screen ***   01/01/84  (SUN)  10:26:58    54.5k CP/M   ver 2.2 PAGE 1/1
B:BASIC     A:GRAPH.BAS
BASIC     (resident) B:BASIC     COM        A:GRAPH     BAS        A:SAMP1     BAS
A:SAMP2     BAS
```

Note the appearance of the command line. This means that BASIC will be loaded first, and then the program in order to RUN it.

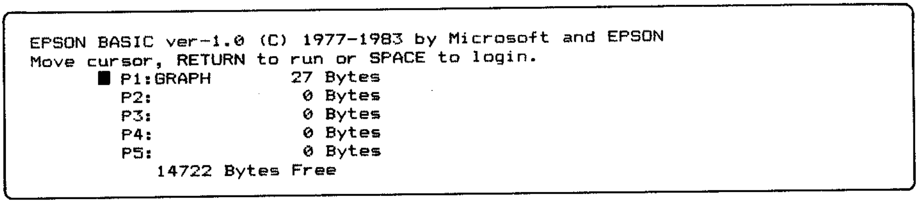
1.3 The BASIC Program Menu

Once BASIC has been started, the BASIC program menu is displayed as shown below.



```
EPSON BASIC ver-1.0 (C) 1977-1983 by Microsoft and EPSON
Move cursor, RETURN to run or SPACE to login.
      ■ P1:              0 Bytes
        P2:              0 Bytes
        P3:              0 Bytes
        P4:              0 Bytes
        P5:              0 Bytes
           14749 Bytes Free
```

This might look slightly different if BASIC was already resident, in which case a display such as the following could appear.



```
EPSON BASIC ver-1.0 (C) 1977-1983 by Microsoft and EPSON
Move cursor, RETURN to run or SPACE to login.
      ■ P1:GRAPH        27 Bytes
        P2:              0 Bytes
        P3:              0 Bytes
        P4:              0 Bytes
        P5:              0 Bytes
           14722 Bytes Free
```

Again, if the command line was of the form shown in (iv) of section 1.2 or the entry from the CP/M command line was to run a BASIC program directly, the above menu would only flash up briefly before the program began running.

The BASIC program menu shows the number of bytes of program text contained in each of BASIC's five program areas (P1 to P5) and the number of free bytes of memory which are available for use as string area, variables, or additional program text. (The BASIC interpreter automatically handles allocation of memory between the three of these as appropriate.)

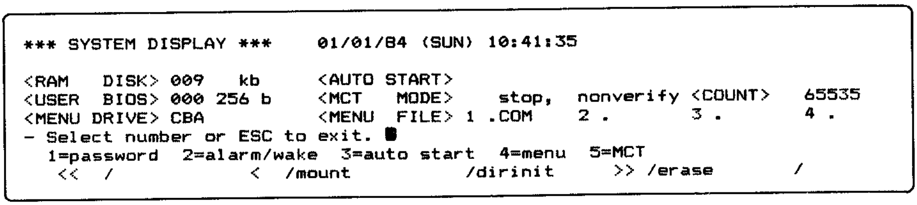
The following keys can be used while the BASIC program menu is displayed.

↑ Moves the cursor upward.

↓ Moves the cursor downward.

RETURN Logs in (selects) the program area indicated by the cursor and executes the program which is present in that area.

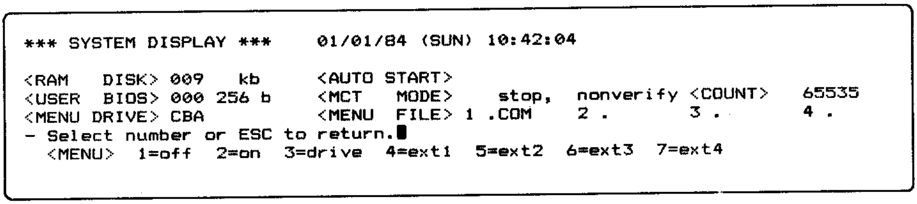
From the point of view of the user, the only difference between the direct mode and the indirect mode is that commands and statements entered in the indirect mode must be preceded by line numbers. The computer automatically switches from one mode to the other according to whether commands/statements are preceded by line numbers.



```
*** SYSTEM DISPLAY ***    01/01/84 (SUN) 10:41:35

<RAM   DISK> 009   kb      <AUTO START>
<USER  BIOS> 000 256 b     <MCT   MODE>    stop,  nonverify <COUNT>   65535
<MENU DRIVE> CBA           <MENU  FILE> 1 .COM    2 .        3 .       4 .
- Select number or ESC to exit. █
  1=password  2=alarm/wake  3=auto start  4=menu  5=MCT
   <<  /            <  /mount          /dirinit      >> /erase         /
```

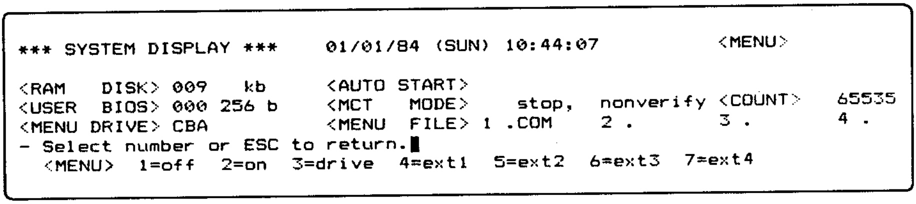
(3) Now press the 4 key to select the menu specification mode; this causes the screen to change as follows.



```
*** SYSTEM DISPLAY ***    01/01/84 (SUN) 10:42:04

<RAM   DISK> 009   kb      <AUTO START>
<USER  BIOS> 000 256 b     <MCT   MODE>    stop,  nonverify <COUNT>   65535
<MENU DRIVE> CBA           <MENU  FILE> 1 .COM    2 .        3 .       4 .
- Select number or ESC to return.█
  <MENU>  1=off  2=on  3=drive  4=ext1  5=ext2  6=ext3  7=ext4
```

(4) When the screen changes as shown above, press the 2 key; the display will also show "<MENU>" towards the top right of the screen to show that the menu option is set.



```
*** SYSTEM DISPLAY ***    01/01/84 (SUN) 10:44:07          <MENU>

<RAM   DISK> 009   kb      <AUTO START>
<USER  BIOS> 000 256 b     <MCT   MODE>    stop,  nonverify <COUNT>   65535
<MENU DRIVE> CBA           <MENU  FILE> 1 .COM    2 .        3 .       4 .
- Select number or ESC to return.█
  <MENU>  1=off  2=on  3=drive  4=ext1  5=ext2  6=ext3  7=ext4
```

This causes BASIC to become resident in memory the next time it is loaded, as well as causing the PX-8's MENU screen to be displayed the next time BASIC operation is ended. Finish by pressing the ESC key once to return to the System Display, then once again to redisplay the system prompt ("A>").

1.5 Ending BASIC Operation

BASIC operation is ended and control returned to the CP/M operating system (or the MENU screen) by typing SYSTEM and pressing the RETURN key. (This command can also be executed from within a program.)

Example

```
10 ...
20 ...
30 ...
SYSTEM    RETURN
A>
```

Note:
In the example above, the symbol“ RETURN ”indicates that the operator hits the RETURN key. This also applies to other examples throughout this manual.

BASIC operation is also terminated when the PX-8's power goes off. This occurs under the following circumstances.

(1) When the PX-8's power switch is turned off;

(2) When there are no entries typed in from the keyboard for the amount of time specified with the POWER <duration> command while the PX-8 is standing by for input (either at the command level or during execution of an INPUT statement);

(3) When the POWER OFF command is executed.

If the POWER OFF command has been executed, when the PX-8 is turned on again, the MENU screen will be the resumption point. It is possible to have BASIC operating as it was at the point at which the power was switched off if one of the following conditions is met:

(1) The power switch is turned off while pressing the CTRL key;
(2) The power switch is turned off during execution of a command or program; or
(3) The power goes off because no entries have been typed in from the keyboard while the PX-8 is standing by for input.

When BASIC operation is started, the first seven characters of the character string assigned to each of the programmable function keys is displayed at the bottom of the screen. Definitions for the unshifted and shifted functions are displayed together (separated by a slash). Display of these definitions can be turned on or off as required with the SCREEN command (see Chapter 4) or by using a control sequence as follows:

PRINT CHR$(27); CHR$(&HD3); CHR$(1)	will turn the display off
PRINT CHR$(27); CHR$(&HD3); CHR$(0)	will turn the display on again.

The default settings of the function keys are as follows:

PF1	auto	PF6	(shifted PF1)	load"
PF2	list	PF7	(shifted PF2)	save"
PF3	edit	PF8	(shifted PF3)	system
PF4	stat	PF9	(shifted PF4)	menu^M
PF5	run ^M	PF10	(shifted PF5)	login

With function keys such as PF5 where there is a carriage return in the command, the command will be executed as soon as the key is pressed. Some (such as PF8) have been defined without a carriage return to ensure that the user makes the final decision on their execution, since if a mistake were made the results could destroy a program. Others (such as PF6) must have further input or an error message will be generated. There are also commands such as LIST which can have optional characters added. Thus typing the letter "L" followed by PF1, will result in Llist being generated, and a listing will be printed on an external printer when RETURN is pressed. Similarly, pressing PF1 and then the characters " – 100" will generate LIST-100 and the lines of the program in the current logged in area will be listed to the screen as far as line 100.

If you are typing many lines of a program using a particular command frequently, it may be useful to change one of the function keys temporarily using the KEY function of BASIC. However, the value you use will be lost when BASIC is cold started and reset to the default values.

PAUSE

The PAUSE key makes it possible to temporarily suspend listing of a BASIC program with the LIST command, or to temporarily stop execution of a BASIC program. (The same result is obtained by pressing the CTRL and S keys

2.2 Multiple Program Areas

In PX-8 BASIC the BASIC program area is divided up into five parts, making it possible for up to five separate BASIC programs to be present in memory simultaneously. The program area selected is determined by the /R: or /P: options when BASIC is started. Once BASIC has been started, program areas can be switched by executing the LOGIN command (either in the direct mode or from within a program); this makes it possible to chain execution of programs between areas.

Further, the TITLE command can be used to assign names to the program areas displayed in the BASIC program menu; this command includes an optional parameter which can be specified to prevent the program in the applicable area from being edited or accidentally erased.

In addition the STAT command makes it possible to determine the status of the currently selected program area or other program areas.

The BASIC program areas are managed on a dynamic basis; that is, BASIC allocates memory to each of the areas according to the size of the program that area contains.

the screen). Subsequent tab positions are located in every eighth column. The same effect is achieved by pressing the CTRL and I keys together.

BS

The BS (backspace) key deletes the character located immediately to the left of the cursor and moves the remainder of that logical line to the left by one character position. This key does nothing if pressed while the cursor is at the beginning of a logical line. The same effect is achieved by pressing the CTRL and H keys together.

CTRL + A

Pressing these keys together moves the cursor to the beginning of the logical line in which it is currently located.

CTRL + B

Pressing these keys together moves the cursor to the first character of the word preceding its current position. For the purpose of this function, a word is any group of letters which is separated from other letters by a space or special character. The same result is achieved by pressing the SHIFT and ← keys together.

CTRL + F

Pressing these keys together moves the cursor to the first character of the word following its current position. The same result is achieved by pressing the SHIFT and → keys together.

CTRL + J

Pressing these keys divides the logical line into two parts at the current position of the cursor. When the cursor is already located at the beginning of a logical line, it inserts a logical line consisting only of spaces. If the cursor is positioned to the right of the last character in a logical line, it inserts a logical line consisting entirely of spaces between the current logical line (the line in which the cursor is located) and the following one.

CTRL + X

Pressing these keys together moves the cursor to the position following the last character in the current logical line.

HOME (SHIFT + BS)

Pressing these keys together moves the cursor to the home position without clearing the screen. The same result is achieved by pressing the CTRL and K keys together.

used to terminate automatic program line number generation by the AUTO command. The same result is achieved by pressing the CTRL and C keys together.

CTRL + →

Pressing these keys together switches the cursor to virtual screen 2. This key is effective only while BASIC is in the program input mode, and cannot be used during execution of an INPUT or LINE INPUT statement. (See section 2.13 below for an explanation of the PX-8's virtual screens.)

CTRL + ←

Pressing these keys together switches the cursor to virtual screen 1. This key is effective only while BASIC is in the program input mode, and cannot be used during execution of an INPUT or LINE INPUT statement.

(5) CTRL + ↓
Pressing CTRL and ↓ together clears the screen, displays the program's last line, and moves the cursor to the beginning of that line.

(6) CLR
Pressing this key clears the screen and terminates operation in the edit mode.

(7) ESC
Pressing the ESC key terminates operation in the edit mode without clearing the screen.

When editing programs, remember that changes made on the screen are not reflected in the program in memory until the RETURN key has been pressed with the cursor located in that line. This applies both in the edit mode and to changes made using the screen editor in the normal mode.

Direct mode commands can be executed in the edit mode; however, BASIC returns to the normal command level after execution is completed (unless the command executed is the EDIT command).

```
10 REM   This is a remark statement
20 REM   This is a new remark statement
```

This means of duplicating lines with the same or similar statements but with different line numbers can be very useful in saving a great deal of typing, especially when a program has a number of similar subroutines or loops.

Sometimes it is necessary to split one line into two lines. For example add line 30 to the program to read as follows:

```
30 FOR N = 1 TO 20 : NEXT N
```

Move the cursor onto line 30 again and move the cursor to the colon using the shifted right cursor key. Pressing the DEL key will remove the colon if the cursor is directly on top of it. If the cursor is to the right of the colon it can be removed by using either CTRL + H or the BS key. Now press the INS key and add “ 50 ” so that the line appears as follows:

```
30 FOR N = 1 TO 20  50 NEXT N
```

This is an incorrect BASIC line. Move the cursor to the space before the “5” and press the J key while holding down the CTRL key. The characters from “50” onwards will be moved to the next line but will not be added to the BASIC program. However, if the RETURN key is pressed a line 50 will be added to the program. The program will now have the following lines if listed:

```
10 REM   This is a remark statement
20 REM   This is a new remark statement
30 FOR N = 1 TO 20 : NEXT N
50 NEXT N
```

Note how line 30 has remained unchanged, because in exiting from it the RETURN key was not used. Since in moving the “NEXT N” to line 50, line 30 has been left with a surplus statement, it has to be removed. Move the cursor to line 30 and use CTRL + X to move the cursor to the end of the line, then the shifted left cursor key to move it back to the “N” of “NEXT”. The unshifted left cursor key can be used to bring it on to the colon. If the shifted cursor key is used, the cursor will move back too far.

The screen will clear and show lines of the program from 50 onwards, since the previous ones have scrolled off the virtual screen; in this screen mode it is limited to 8 lines, the same as the window. The cursor will be seen as a non-flashing underline character on the bottom line of the screen. Using LIST would be slow if it was necessary to edit a number of lines in this screen mode. However, EDIT makes a considerable difference.

Type EDIT and press the RETURN key.

The screen will clear and display line 10. This can be edited normally with the screen editor. Since the screen displayed is screen mode 3, moving the cursor to the bottom of the screen with the cursor key will not cause line 10 to scroll off the top of the screen. However, it is possible to scroll through the program by using the shifted ↑ and ↓ cursor keys. The screen editor can be used as described above when a particular line needs to be altered.

In programming, it is often necessary to know which is the first or last line of a program, because a new subroutine needs to be added or a constant inserted at the beginning of the program. If the PX-8 is in EDIT mode, pressing CTRL and the ↑ cursor key will place the first line on the top of a blank screen ready for editing. If the CTRL and ↓ keys are pressed, the last line of the program is displayed instead.

To illustrate the use of these keys, type edit 50, so that line 50 is displayed on the screen. Now press CTRL and the ↑ key. The screen will clear and line 10 will be displayed. Press the SHIFT and ↑ keys. Because there is no lower line number, the cursor will be placed on a blank line, above line 10. Type in a line 5 as follows, remembering to press RETURN to enter it as a line of the BASIC program.

```
5 A$="THE END"
```

Now press CTRL and the ↓ key. You can see that the last line of the program is line 100. Whereas the SHIFT and ↓ keys can be used to move to a new line, it is better simply to press RETURN because the contents of line 100 will still be visible. If the CTRL and ↓ keys are used the screen will be clear because line 100 will be scrolled off the top.

2.6 Types of Data

2.6.1 Text data — The ASCII character set

The ASCII character set is a set of characters which are internally represented in the form of 1-byte† numeric codes and converted to characters for display by the PX-8's character generator.

The character generator of the PX-8 includes character sets for the following nine countries.

1. Denmark
2. England
3. France
4. Germany
5. Italy
6. Norway
7. Spain
8. Sweden
9. United States

Any of these character sets can be selected with the OPTION COUNTRY command.

The ASCII character code table is shown in Appendix F, together with a list of differences between the U.S. ASCII character set and those of the other eight countries.

† The byte is the unit in which data is handled by the PX-8's central processing unit (CPU); one byte consists of eight bits, or binary digits. In the binary numbering system, which uses the numerals 0 and 1, it is possible to represent all numbers from 0 to 255 as numbers of up to 8 digits. This is the range of numbers which is used for representing characters in the ASCII character code system.

2.6.3 Numeric data

All numeric data is converted to binary form for storage in memory or calculation by the PX-8's CPU. However, BASIC allows numeric data to be entered in any of three number bases. These are decimal (base 10), octal (base 8), and hexadecimal (base 16).

Decimal notation is the familiar numbering system we use in everyday life, with numerals which range from 0 to 9. With this system, the number of digits required to express numbers increases by one each time the magnitude of the number being expressed increases by a factor of ten (1, 10, 100, and so forth.) Decimal notation can be used to represent both integers and numbers which include decimal fractions.

Octal notation (also referred to as base 8 numeration) uses only the digits from 0 to 7. With this system, the number of digits required to express numbers increases by one each time the magnitude of the number being expressed increases by a factor of eight. Octal numbers are indicated by affixing the characters "&O" or "&" to the beginning of the number. The decimal equivalents of octal numbers can be calculated as shown below.

$$\&O347 = 3\times8^2 + 4\times8^1 + 7\times8^0 = 231$$
$$\&1234 = 1\times8^3 + 2\times8^2 + 3\times8^1 + 4\times8^0 = 668$$

Numbers entered in octal notation may not include a decimal point; therefore, octal notation can only be used to represent integer values.

Hexadecimal notation (also referred to as base 16 numeration) uses the digits from 0 to 9 and the characters from A to F to represent the values from 10 to 15. With this system, the number of digits required to express numbers increases by one each time the magnitude of the number being expressed increases by a factor of 16. Hexadecimal numbers are indicated by affixing the characters "&H" to the beginning of the number. The decimal equivalents of hexadecimal numbers can be calculated as follows.

$$\&H76 = 7\times16^1 + 6\times16^0 = 118$$
$$\&H32F = 3\times16^2 + 2\times16^1 + 15\times16^0 = 815$$

As with octal notation, hexadecimal notation can only be used to represent integer values.

2.7 Constants — String Constants and Numeric Constants

Constants are fixed values which are writen into a program and used by that program during its execution. These values may consist of either characters or numbers; in the former case they are referred to as string constants, and in the latter case as numeric constants.

2.7.1 String constants

A string constant is any sequence of alphanumeric characters which is enclosed in quotation marks. Some examples of string constants are shown below.

"EPSON PX-8"
"John Jones"
"$10,000.00"
"The quick brown fox jumped over the lazy yellow dog."

The length of a string constant cannot exceed the maximum length of a program line (255 characters).

2.7.2 Numeric constants

Numeric constants are positive or negative numbers. There are five types of numeric constants, as follows:

(1) Integer constants
Integer constants are whole numbers in the range from −32768 to +32767. Such constants can be expressed in either decimal, hexadecimal, or octal form.

(2) Fixed point constants
Fixed point constants are positive or negative numbers which include a decimal fraction.

(3) Floating point constants
Floating point constants are positive or negative numbers which are represented in exponential form. A floating point constant consists of an integer or fixed point constant, followed by the letter E (denoting an implicit base of 10) and an exponent. Either the fixed-point part or the exponent may be preceded by a minus ("−") or plus ("+") sign to indicate that it is positive or negative; if no sign is present, it is assumed that that portion of the constant is positive. The exponent must be in the range from −38 to +38.

NOTE:
When a BASIC program is written, care should be taken to declare the constants correctly. The BASIC interpreter may list numbers in a different form than that in which they are typed in. BASIC may also insert the trailing signs in the listing. For example the following lines:

```
10 PRINT A*1234567
20 PRINT B*123456789
```

would be listed as:

```
10 PRINT A*1.23457E+06
20 PRINT B*123456789#
```

Examples of variable names are shown below.

PI #	Double precision numeric variable
MINIMUM!	Single precision numeric variable
LIMIT%	Integer variable
CATEGORY$	String variable

Variables may also be defined in advance as string, integer, single precision, or double precision with the DEFSTR, DEFINT, DEFSNG, and DEFDBL statements. When variable types are specified in this manner, type declaration characters are not required. See the explanations of these statements in Chapter 4 for details.

NOTE:
As with constants, BASIC may add the trailing declaration character when the program is listed. It is important to be aware that A # is a different variable from the variable A, unless A has been declared as a double precision variable with the DEFDBL statement. Also, statements declaring variable types do so for all variables beginning with a particular character and it is not possible to specify variable names consisting of more than one character in such statements.

2.8.2 Array variables

An array is a group of variables which is referred to by a common name. Each variable of an array is identified by one or more subscripts, each of which is specified as an integer value. The number of subscripts corresponds to the number of variables in a one-dimensional array (for instance, P(x) where "x" is the integer expression which identifies the individual variable); P(x, y) refers to a specific variable in a two-dimensional array, and can be thought of as a table containing a certain number of rows and columns; the number of rows depends on the maximum value of x, and the number of columns depends on the maximum value of y. Theoretically, an array can have any number of dimensions; however, in practice the number of dimensions and the size of the array are limited by the amount of memory which is available. The DIM statement is used to define the number of dimensions of an array and the size of each dimension. See the explanation of the DIM statement in Chapter 4 for details.

(2) Conversion in arithmetic and relational operations

If an arithmetic or relational expression includes numeric opèrands of different types, all operands are converted to the same degree of precision (that of the operand with the highest degree of precision). Further, the results of arithmetic expressions are returned to the degree of precision of the most precise operand. Note that error may be introduced when constants are converted from one precision to another. (Note: Relational operations are described in section 2.10.2 below.)

Example

```
10 A#=6#/7.1#          :'Assigns result of arithmetic
20                     :'operation 6#/7.1# (double precision)
30                     :'to double precision variable A#.
40 '
50 PRINT A#            :'Displays contents of variable A#.

Ok
run
 .8450704225352113
Ok
```

In the example above, arithmetic is performed using double precision numbers and the result is returned in double precision.

Example

```
10 A#=6#/7.1    :'Assigns result of 6#/7.1 to variable A#.
20 '
30 PRINT A#     :'Displays contents of variable A#.
Ok
run
 .8450704338862249
Ok
```

Here, the single precision value 7.1 is converted to double precision for arithmetic and the result is returned as a double precision number. The difference between the result returned in this example and that returned in the preceding example is due to conversion error.

When evaluating a function care must be taken to ensure that the value of the expression being evaluated is declared to the precision required. In the following example, only the first seven characters of C# are correct because the square root of a single precision number is being converted to a double precision number. D# gives a more accurate value because the square root of a double precision number is being assigned to a double precision variable. Lines 80 and 90 show the result of printing a function without and then with double precision declarations.

Example

```
10 A=2
20 B#=2
30 C#=SQR(A)
40 D#=SQR(B#)
50 PRINT "C# =";C#
60 PRINT "D# =";D#
70 PRINT "The square root of 2 is";SQR(2#)
80 PRINT SQR(1+1)
90 PRINT SQR (1#+1#)

run
C# = 1.414213538169861
D# = 1.414213562373095
The square root of 2 is 1.414213562373095
 1.41421
 1.414213562373095
Ok
```

Sample algebraic expressions and their equivalents in BASIC are shown below.

Algebraic Expression	BASIC Expression
$X+2Y$	X+2*Y
$X-Y\div Z$	X−Y/Z
$X\times Y\div Z$	X*Y/Z
$(X^{Y})^{2}$	(X∧Y)∧2
$X^{Y^{2}}$	X∧(Y∧2)
$X\times(-Y)$	X*(−Y)

When two consecutive operators are included in an expression, they should be separated by enclosure in parentheses as shown in the last example above.

(1) Integer division

With integer division, the operands of an expression are rounded to integers, then division is performed and the integer portion of the quotient is returned. The operator for integer division is the backslash (\). This should not be confused with standard division for which the operator is the slash (/).

NOTE:

Regardless of the International character set used, internal code CHR$(92) is used as the operator for integer division.

The following example of integer division compares the same division taking the integral part of the result after the division:

```
10 A=33.55:B=4.62
20 ID=A\B                :'Integer division
30 ND%=A/B               :'Normal division
40 PRINT ID,ND%
run
 6             7
Ok
```

When integer division is performed, both operands must be within the range −32768 to 32767.

(2) Modulus arithmetic

Modulus arithmetic is the arithmetic operation which returns the remainder of integer division as an integer. Rounding up can occur as the sécond example shows. The operator for modulus arithmetic is MOD. The precedence of modulus arithmetic is just below that of integer division.

```
90 A%=66666!\25                    :'Generates an "Overflow" error
100                                :'because dividend (66666) is
110                                :'outside of permitted range
120                                :'for integer division.
125
130 PRINT A%                       :'Not executed because
140                                :'program execution is
150                                :'aborted by error in
160                                :'line 90.

run
Overflow
 1.701411733192645D+38
Program line 30
Overflow in 90
Ok
```

2.10.2 Relational operations

Operations in which two values are compared are referred to as relational operations. The result returned by such a comparison is either "true" (– 1) or "false" (0), and is then used to make a decision regarding subsequent program flow. (See the discussion of the IF...THEN...ELSE and IF...GOTO statements in Chapter 4.)

The relational operators and their meanings are listed below.

Operator	Relation tested	Example expression
=	Equality	X=Y
< >, > <	Inequality	X< >Y, X> <Y
<	Less than	X<Y
>	Greater than	X>Y
< =, = <	Less than or equal to	X< =Y, X= <Y
> =, = >	Greater than or equal to	X> =Y, X= >Y

NOTE:
The "=" sign is used both for testing equality in relational expressions and in LET statements for assigning values to variables. However, its meaning is not the same in both cases. See the discussion of the LET statement in Chapter 4 for details on assigning values to variables.

2.10.3 Logical operations

A logical operation uses Boolean arithmetic to define the logical connection between the results (– 1 or 0) of relational operations. In any given expression, logical operations are always performed after arithmetic and relational operations. The results of operators are listed in the table below according to the order of precedence.

NOT (Negation)

X	NOT X
1	0
0	1

AND (Logical product)

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR (Logical sum)

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

XOR (Exclusive – OR)

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

IMP (Inclusion)

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1

EQV (Equivalence)

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

Since relational operations can be used to make decisions concerning program flow, logical operators can be used to connect two or more relational operations. This allows decisions to be based on multiple conditions. (See the discussion of the IF...THEN...ELSE and IF...GOTO statements in Chapter 4).

Examples

1) IF D<200 AND F<4 THEN 80
   This statement causes program execution to branch to line 80 if the contents of variable D are less than 200 and the contents of variable F are less than 4.
2) IF I<10 OR K<0 THEN 50
   This statement causes program execution to branch to line 50 if the contents of variable I are less than 10 or the contents of variable K are less than 0.

The two's complement integer −1 is expressed in binary as 1111111111111111B, while the two's complement integer −2 is expressed as 1111111111111110B. Since both 1 OR 1 and 1 OR 0 yield 1, the result is 1111111111111111B, or − 1.

Logical operators can be used to test data bytes for a particular bit pattern. For instance, the AND operator can be used to mask all but one bit of a status byte to obtain the status of a device I/O port; or, the OR operator can be used to merge two data bytes to obtain a particular value.

Example

```
10 FOR I=97 TO 122            :'Displays characters
20 PRINT CHR$(I);             :'from "a" to "z".
30 NEXT                       :'
40 '
50 PRINT                      :'Moves cursor down to
60                            :'next line on display.
70 '
80 '       Following lines convert lowercase
90 '       character codes to uppercase.
100 FOR I=97 TO 122           :'Binary   nnnnnnnn
110 PRINT CHR$(I AND 223);:'AND      11011111
120 NEXT                      :'yields:  nn0nnnnn

run
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Ok
```

Comparison ends when different characters are encountered in the two strings or when the end of one of the strings is reached; in the former case, the string in which the lesser code is encountered is regarded as smaller, and in the latter case the shorter string is regarded as smaller.

Spaces included in strings are also significant; for example:

"ALPHA" is smaller than "ALPHA   "
"ALPHA" is greater than "   BETA"

Further examples are:

"AA" is less than "AB"
"FILENAME" is equal to "FILENAME"
"X&" is greater than "CL"
"SMYTH" is less than "SMYTHE"

Thus, string comparisons can be made to test string values and to sort strings into alphabetical order. All string constants used in relational expressions must be enclosed in quotation marks.

Examples: INT(1.1) = 1
INT(0.9) = 0
INT(−5.4) = −6
INT(−5.7) = −6

2.11.2 Trigonometric functions

PX-8 BASIC supports the following trigonometric functions.

Function	Argument	Value returned
ATN	Tangent of an angle	Angle in radians
COS	Angle in radians	Cosine of an angle
SIN	Angle in radians	Sine of an angle
TAN	Angle in radians	Tangent of an angle

If you want to work with angular measurements in degrees, remember that you will have to convert the arguments of these functions from degrees into radians (with the ATN function, you will have to convert the result from radians into degrees). Since 180 degrees is equal to π radians, there are $180/\pi$ degrees in a radian. Thus, you can convert degrees to radians by dividing by $180/\pi$. Conversely, radians can be converted to degrees by multiplying by $180/\pi$. Single and double precision values corresponding to $180/\pi$ are as follows.

Single precision
180/3.1416=57.2958

Double precision
180/3.141592653589795=57.29577951308228

Other trigonometric functions must be derived from these four built-in functions. For example, the secant of an angle is equal to 1 divided by the angle's cosine. See Appendix E for derivation of other trigonometric functions.

NOTE:
Device name LPTØ: can be assigned to either the RS-232C port,or the serial port with the CONFIG program of CP/M to indicate the printer connected to that port. Initially, LPTØ: is assigned to the RS-232C port.

(2) Option
<option> is used to set the baud rate and communication format for the RS-232C interface, the write mode for the microcassette drive, and so forth. The format of <option> varies according to device.

(3) File name
File names are used to distinguish files within a device from one another. Specification of file names is mandatory when accessing files in disk devices; however, file names are meaningless in the case of device files such as the keyboard and RS-232C interface, and will be ignored if specified.
A <file name> is composed of a primary name of up to 8 alphanumeric characters, and an extension consisting of up to 3 alphanumeric characters. The primary file name is separated from the extension by a full stop.

Example SAMP1.BAS

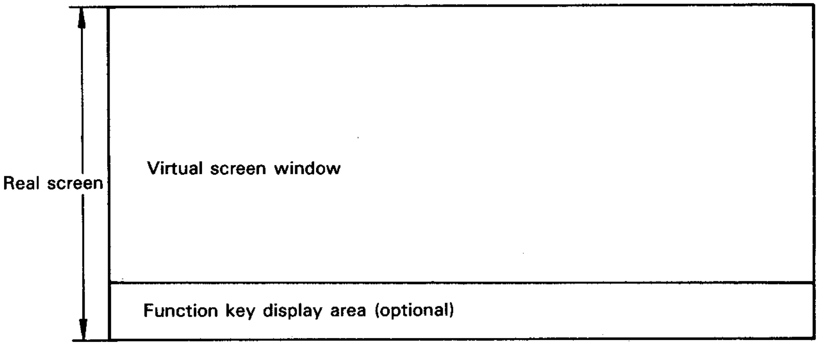
With the LOAD, MERGE, RUN, LIST and SAVE commands, the extension ".BAS" is assumed only if the primary name is specified in the command's operand. The FILES, KILL or NAME commands require extensions to be specified.

2.12.2 File numbers

A file number is a number which is assigned to a device file as an identifier when that device is opened for input and/or output. The number specified as the file number must be in the range from 1 to the maximum specified in the /F: option when BASIC operation is started.
With PX-8 BASIC, a logical file number must be assigned to each file which is read or written by a program (except when a program text file is accessed using the LOAD, MERGE, RUN, LIST or SAVE commands). This is done with the OPEN statement, which links a specific logical file number to the physical file defined in the file descriptor. Unless otherwise specified with the /F: option when BASIC is started, the maximum number of files which can be open at one time is 3. See the explanation of the OPEN statement in Chapter 4 for the procedure for assigning file numbers.

(3) Virtual screen window

Since the real screen is limited to seven or eight lines depending on whether the function key definitions are displayed, it acts as a window on the virtual screens. However, scrolling can only be performed in the vertical direction, either by means of the cursor keys or under program control.



When function key definitions are not displayed, the number of lines displayed by the virtual screen window (also referred to as the screen window) is the same as the number of lines in the real screen (8). When function key definitions are displayed, the number of lines displayed is reduced by one (the line used to display function key definitions).

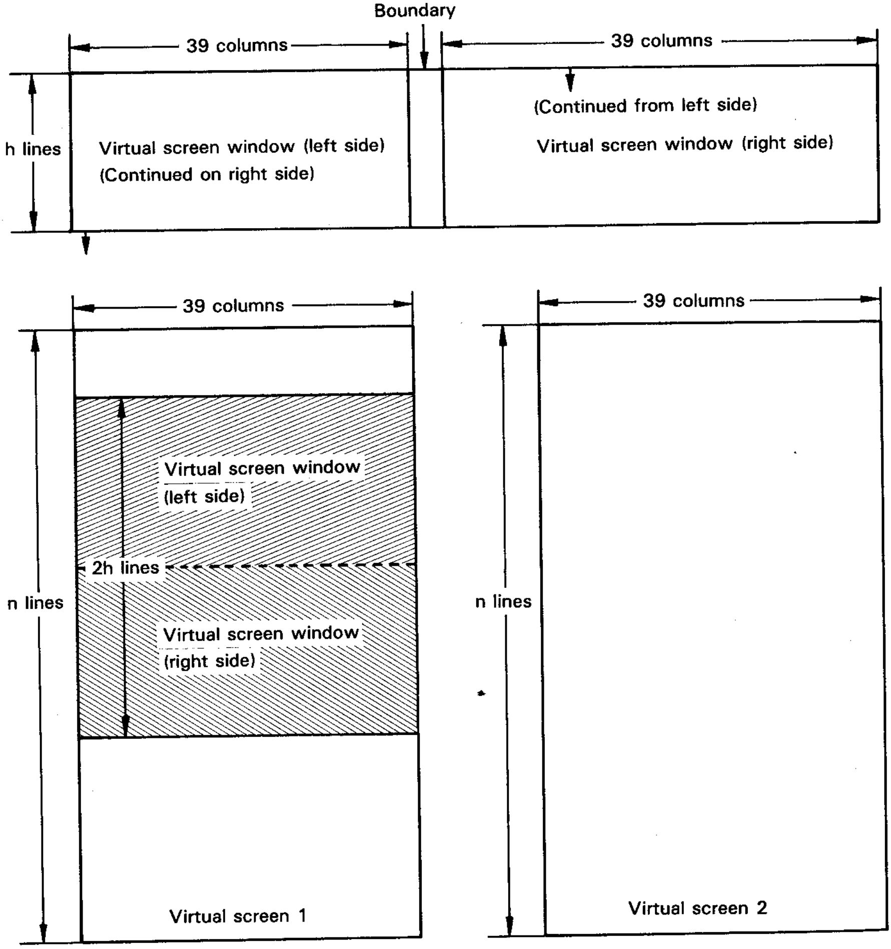
2.13.2 Screen modes

The PX-8's LCD screen has four modes of operation. These are referred to as screen modes 0, 1, 2, and 3. Screen modes 0, 1, and 2 are solely text screen modes; screen mode 3 is the graphic display mode; and allows text to be mixed with graphics.

When the display is in one of the text modes, the PX-8's VRAM is divided into two sections which are used as two independent virtual screens. In the text modes, character data consisting of one-byte ASCII codes is written into VRAM; these codes are converted to character images for display by the PX-8's display controller.

In this screen mode, the size of the virtual screen window is 39 characters × 2h lines (where h is the number of lines in each half of the screen window).

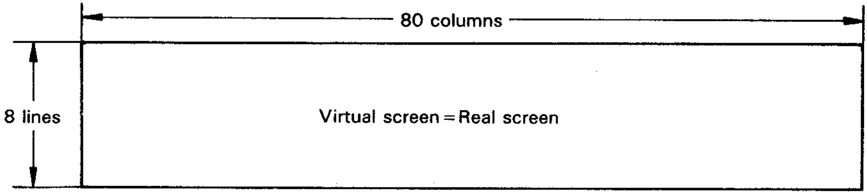
As with screen mode 0, the virtual screen window can be switched back and forth between the two virtual screens, both of which have a width of 39 columns. The number of lines in the two virtual screens can be set as desired by the user within the range from 16 to 48, and both virtual screens must have the same number of lines.



Conditions: 16≤2n≤48
h=7 or 8

(4) Screen mode 3 (the graphic screen mode)
This is the screen mode which is used for displaying graphics. In this screen mode, the PX-8's display controller works on a bit image basis, rather than using character codes, making it possible to display a full range of graphics. Although text can be displayed with the graphics, the size of the virtual screen in this screen mode is the same as the size of the real screen, that is, the virtual screen size is 80 columns × 8 lines. When the function key assignments are displayed, there are only 7 usable lines.

Although the screen editor can be used in the same manner in this screen mode as in the text screen modes, the virtual screen (i.e., bit image data in VRAM) is displayed directly to the real screen and there is no screen window. Because the real and virtual screens are the same, scrolling of the virtual screen cannot occur. Graphic statements such as PSET, PRESET, LINE and POINT can only be used in this screen mode.



2.13.3 Selection and display of virtual screens

Display screen modes 0 to 2 each include two virtual screens, only one of which can be selected at any given time. The screen selected is that in which characters are displayed when keys are pressed or when PRINT and similar statements are executed. The SCREEN command in BASIC is used to select the virtual screen.

(1) Screen modes 0 and 1
In these screen modes, only one virtual screen can be displayed at a time. The virtual screen selected is displayed in the screen window, and characters typed or output are displayed in the selected virtual screen. The scrolling control keys and scrolling escape sequences move the virtual screen window through the virtual screen which is currently selected.

(2) Screen mode 2
In this screen mode, both virtual screens are displayed at the same time. However, the scrolling control keys and scrolling escape sequences only move the virtual screen window through the virtual screen which is currently selected for output (the screen in which typed characters are displayed).

(6) CTRL + INS (find cursor)

When the cursor is not displayed in the screen window, pressing CTRL + INS moves the screen window to the current position of the cursor.

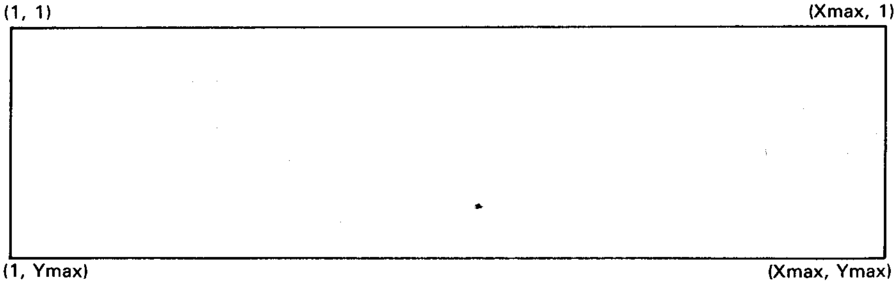
2.13.5 Screen coordinates

(1) Character coordinates

Character coordinates are the coordinates which are used to specify the position in which characters are displayed on the screen. These coordinates are used with the LOCATE statement and the SCREEN, POS and CSRLIN functions.

When using character coordinates, the column on the left side of the screen is numbered 1 and that on the right side of the screen is numbered according to screen mode or the maximum screen width specified by the user. In screen modes 0 and 3, the column on the right side of the screen is numbered 80 and in screen mode 1 it is numbered 39. In screen mode 2, the number of the right hand column is the same as the column width specified for the selected virtual screen by the user.

In the vertical direction, the top line is numbered 1 and the bottom line is numbered according to screen mode or the maximum number of screen lines specified by the user.



Xmax: Number of columns in selected virtual screen
Ymax: Number of lines in selected virtual screen

(2) Graphic coordinates

Graphic coordinates are used to specify the positions of individual dots on the screen. The graphic coordinate system is used with statements such as PSET, PRESET and LINE, and with graphic functions such as POINT.

2.14 A Practical Guide to the Screen Modes

Whereas the above description summarises the possibilities for the various screen modes, it is difficult to appreciate the full facilities without seeing them in action. This section is thus meant to be followed actually using the PX-8.

The various screen modes are accessed by using the SCREEN command. The screen size is set by the WIDTH command. This can be attached to the SCREEN command to give the full format of the screen command as follows:

```
SCREEN M,VS,FKS,BC WIDTH C,NL1,NL2
```

Where M is screen mode 0, 1, 2 or 3

VS is the virtual screen to be displayed.

FKS allows the function key assignments at the base of the screen to be switched on and off to give the full 8 lines of the LCD screen.

BC sets the boundary character for split screen display in screen mode 2.

WIDTH is a separate command which is used to set the number of columns and lines of the virtual screens. It would normally be used on its own, but since in many cases use of the SCREEN command will involve setting the size of the virtual screens, it has been added to the syntax of the SCREEN command. The addition does not require a comma to separate the WIDTH command from the boundary character, but it does require a space as separator. In WIDTH, C sets the number of columns (in screen modes 1 and 2) and NL1 and NL2 the number of lines of the two virtual screens. The range of these options varies according to the modes and thus will be discussed under each mode.

If one of the parameters is to be changed, other than the mode, the correct number of commas must be inserted to denote which parameter is being changed. To illustrate this the function key assignments can be switched off using the command

```
SCREEN,,Ø
```

because the function key switch is the third parameter.

If a screen mode or virtual screen is not specified, the current values are used.

SHIFT, CAPS LOCK or NUM/GRAPH or a key which cannot function (e.g., the BS key if the cursor is at the beginning of a line) the window will be returned to the part where the cursor is located. The character pressed will be printed beside the cursor, or the function of the key carried out. You can also return to the cursor position by pressing the CTRL and INS keys together.

The cursor can be moved anywhere on the virtual screen using the cursor keys. Note how the screen moves with the cursor if you move to a line above or below the real screen. This is the normal tracking mode. It can be changed to the non-tracking mode, where the cursor can be moved anywhere on the virtual screen, leaving the window fixed. Pressing the SCRN key (shifted INS) will switch between the two modes. Move the cursor off the real screen with the cursor keys in the non-tracking mode, and then restore the part of the screen containing the cursor to the window as before using the CTRL and INS keys. It is possible to set the cursor to the first character position of the virtual screen with the HOME key (shifted BS) but this does not display the cursor. In combination with the CTRL and INS keys as a sequence it can be used to set the display to the top of the virtual screen with the cursor on that position.

It is still possible to scroll the screen even if the cursor is not in the window, by using the SHIFT and ↑ and ↓ cursor keys. This only moves the screen up or down one line at a time.

The boundary character is not used in screen mode 0.

The WIDTH command is used to set the number of columns and number of lines of each virtual screen. However, in screen mode 0 it is not possible to alter the number of columns. It is only possible to display an 80 column screen, with either of the two virtual screens being displayed at any one time. If using screen mode 0 a value of 80 must be used for the column width or no value used at all.

The number of lines in each virtual screen can be altered provided the sum of the lines specified is less than or equal to 48, and that neither screen is made less than 8. If your program uses only one virtual screen, it is beneficial to increase its size to the maximum, (i.e., 40 since the other screen must be 8 and the total 48). It is also useful to do this if you are writing a BASIC program since it is possible to scroll back without having to relist the program. Type

SCREEN Ø,Ø,Ø, WIDTH 8Ø,4Ø,8

or the equally valid

SCREEN Ø,Ø,Ø, WIDTH ,4Ø,8

(i) Change the window between the virtual screens, using CTRL and the → and ← cursor keys.
(ii) Change the tracking mode, using the SCRN key.
(iii) Scroll using the shifted ↑ and ↓ keys.
(iv) Look at the first and last displayable lines using the CTRL and ↑ and ↓ keys.

Only one parameter can be altered in the WIDTH statement in screen mode 1. The screen is set to display two 39-column halves on each side of the screen. It is not possible to alter this so the WIDTH statement must specify 39 as the column width, if any value is used, or an error will occur. The number of lines in each virtual screen is the same in screen mode 1. The number of lines must be specified in the range 16 to 48 and setting the number of lines for the first screen also sets the number for the second. Any attempt to set the size of the second virtual screen will be ignored and no error generated. Thus to set the number of lines on each screen to 20, the following are valid commands:

```
WIDTH 39, 2Ø
WIDTH ,2Ø
WIDTH 39, 2Ø, 8
```

(3) MODE 2

This is the most versatile screen mode. The width of each half of the screen and boundary character can be set. The number of lines on each virtual screen is the same as with screen mode 1. For example type

```
SCREEN 2, Ø, Ø, "*" WIDTH 2Ø, 4Ø
```

This will switch to screen mode 2, placing the cursor on the left-hand side of the screen and removing the function key assignments from the base of the screen. It also sets the width of the left-hand side of the screen to 20 columns. The right-hand side is thus set as 59 since one of the 80 columns is used for the boundary characters.

Now use the CTRL and → keys to change the virtual screen. Note how the cursor moves to the right half of the screen. This is because the two halves of the screen correspond to the two virtual screens. Use the cursor keys in combination with the SHIFT and CTRL keys to explore this mode as with modes 1 and 2.

2.15 Input/Output Device Support

PX-8 BASIC supports data input and output (I/O) to and from a variety of peripheral devices. These include random access devices such as external floppy disk drives and RAM disk (a user-specified area in memory which is used in the same manner as an external disk drive), and sequential access devices such as the RS-232C interface, printer and LCD screen.

Devices which have full random access capability allow the records of files to be read or written in any order. All external storage devices used with the PX-8 have some degree of random access capability.

Sequential access devices are devices in which each item of data in a set is input from or output to the device in the order in which it occurs in that set. With PX-8 BASIC, sequential file organization may be used for storage of information in external access devices such as floppy disk drives, or for input/output of information when I/O devices such as the keyboard and RS-232C interface are handled as device files.

2.15.1 Random access devices

Random access devices are devices which can be open in either the random ("R") or sequential ("I" or "O") modes. Random access devices supported by PX-8 BASIC include external floppy disk drives, RAM disk, and the PX-8's built-in microcassette drive. Collectively, these are referred to as disk devices.

Disk devices can be classified into three categories (types I, II, and III) as follows.

(I) Type I disk devices are external storage devices which can be both read and written to on a random access basis. Devices included in this category are RAM disk and external floppy disk drives. All disk I/O statements and functions can be used with type I devices.

(II) Type II disk devices are devices which can be read (but not written to) on a random access basis. Devices included in this category are ROM capsules in ROM socket 1 and 2. Only read statements/functions can be used with this type of device.

(III) Type III disk devices are devices which can both be read and written, but for which random access support is limited. At present, the only device which is included in this category is the PX-8's built-in microcassette drive.

2.15.3 Speaker

PX-8 BASIC also supports output to a speaker (either the built-in speaker or an external speaker connected to the SP OUT jack on the back of the PX-8). No device name is assigned to this device; however, output control is possible using the BEEP and SOUND statements.

The speaker can also be used to check for the presence of recorded signals on the tape. This is done by executing the WIND ON statement. (See the explanation of the WIND statement in Chapter 4.)

2.15.4 The analog converter

On the rear of the PX-8 is a socket marked A/D IN. This is the analog-to-digital converter. It takes an analog voltage and converts it into a number. This number is proportional to the voltage. The analog-to-digital converter is not supported by BASIC, but it can be used if a short machine code subroutine is added to a BASIC program. This machine code subroutine is required to make the necessary BIOS call to return the digital value corresponding to the analog voltage. An example of its use is given in the User's Manual, where the voltage across a variable resistor is determined. This is then used as a paddle in a simple game. The program shows how the machine code routine to read the A/D port can be used with a BASIC program.

2.15.5 The bar code reader

Many products are marked with a machine readable coded series of bars. This allows a numerical value to be read simply by moving a bar code wand across the bars. A socket for insertion of the wand can be found on the rear of the PX-8 marked BRCD. Further details are included in the User's Manual.
Examples of the use of a bar code reader include identifying suitably coded products, then using the information in a program to count the number of different items. Special software is required to perform the task of reading the data. Such software must be written in machine code, but can be linked to BASIC. Use of the bar code reader requires the purchase of a separate software package, and details of how to use the software with BASIC are included with the package. Please consult your dealer for more information.

2.15.7 Commands, statements, and functions usable with I/O devices

Commands, statements, and functions which can be used with the various I/O devices are as indicated in the table below.

Device	KYBD	SCRN	LPT0	COM0	Disk-I	Disk-II	Disk-III
CLOSE	○	○	○	○	○	○	○
DSKF	×	×	×	×	○	○	†2
EOF	—	×	×	○	○	○	○
GET	×	×	×	×	○	○	†3
INPUT #	○	×	×	○	○	○	○
INPUT$	○	×	×	○	○	○	○
LINE INPUT #	○	×	×	○	○	○	○
LIST	×	○	○	○	○	×	○
LOAD	○	×	×	○	○	○	○
LOC	—	×	×	○	○	○	○
LOF	—	×	×	○	○	○	○
OPEN "I"	○	×	×	○	○	○	†3
OPEN "O"	×	○	○	○	○	×	†3
OPEN "R"	×	×	×	×	○	○	†3
POS	×	○	○	○	○	○	○
PRINT #	×	○	○	○	○	×	○
PRINT USING #	×	○	○	○	○	×	○
PUT	×	×	×	×	○	×	†3
SAVE	×	†1	†1	†1	○	×	○
WIDTH	×	×	○	○	×	×	×
WRITE	×	○	○	○	○	×	○

Disk-I RAM disk, floppy disk drives
Disk-II ROM capsules
Disk-III Microcassette drive

○: Usable ×: Not usable —: Meaningless

†1 Output is in ASCII format even if binary save or protect save is specified.
†2 Value returned is not trustworthy due to the nature of cassette tape.
†3 Use is subject to restrictions.

2.17 Error Processing Routines

When it is possible to anticipate that a certain line of a program will result in an error or that an error of a particular type will occur, programs can be designed to include routines which are referred to as error traps, or error processing routines. The purpose of an error processing routine is either to evaluate and correct an error or to allow the user of the program to input corrective data; afterwards, the error processing routine either terminates program execution or causes it to resume at a particular point, depending on what conditions are written into the routine.

The ON ERROR GOTO <line no.> statement must be executed in order to define the starting line number of the error processing routine. After execution of this statement, occurrence of any error at any line in the program will cause execution to jump immediately to the line number specified following GOTO. Afterwards, the ERR and ERL functions can be used to evaluate the type of error and the point at which it occurred in the program, and the RESUME statement can be used to transfer execution out of the routine and back to a specific point in the main program.

Errors can also be simulated using the ERROR statement.

See the explanations of the ERROR, ON ERROR GOTO, and RESUME statements and the ERR/ERL functions in Chapter 4 for further information.

(2) BASIC /F: <no. of files>

Example **BASIC /F:5**

The /F: <no. of files> option sets the number of files which can be open simultaneously. In the example the number would be 5. If this operand is omitted, the maximum number of files which can be open simultaneously is set to 3 (the system default value). The maximum value which can be specified in <no. of files> is 15. In this statement "files" refers to data files, either for communication or saving and loading to a disk drive. For example

OPEN "O", #3,"FILENAME"

If a file number is used which is greater than the number specified in the option, a

Bad file number in<line number>

error will occur.

(3) BASIC /M: <upper memory limit>

Example **BASIC /M:&HC000**

The /M: <upper memory limit> option specifies the highest address in RAM which can be used as program or variable area by the BASIC interpreter. The value specified for <upper memory limit> must be smaller than the starting address of the basic disk operating system (BDOS). The memory area starting at the address from <upper memory limit> to the beginning of BDOS can then be used for storage of machine language programs. Naturally, this reduces the amount of memory which can be used for variables or storage of BASIC programs. However, it must be noted that the BDOS starting address will vary according to the number of bytes of memory reserved for use as the RAM disk. The starting address of BDOS can be found by looking in page zero. Locations 5, 6 and 7 contain a jump to BDOS. Therefore locations 6 and 7 contain the starting address of BDOS, in the order LSB, MSB. To obtain this from BASIC use:

PEEK(6) + PEEK(7) * 256

In the example above, the upper memory limit is specified as a hexadecimal number; however, it can also be specified as a decimal number. Full details of this are described in the section on the CLEAR command in Chapter 4.

cannot be listed or edited when it is later reloaded; therefore it is recommended that you also save an unprotected copy of the program for future listing or editing.

LOAD <file descriptor> [,R]

This command loads the program specified in <file descriptor> into memory from the disk. If the R option is specified, the program will be automatically executed as soon as loading is completed. Executing this command without the R option closes all files which are currently open; however, files will not be closed if the R option is specified. This makes it possible to chain programs which access the same data files.

RUN <file descriptor>[,R]

If <file descriptor> is omitted, this command executes the program which is currently in memory. Specifying <file descriptor> causes the specified program to be loaded into memory from the disk or microcassette (deleting any program currently in memory) and to be immediately executed. As with the LOAD command, all open files are closed upon execution of this command unless the R option is specified.

MERGE <file descriptor>

The MERGE command loads the specified program into memory from the disk or microcassette and merges it with the program in memory. The program merged must have been stored in ASCII format. If any lines of the program loaded have the same line numbers as those of the program in memory, those program lines in memory are replaced with those of the program from the disk or microcassette. BASIC always returns to the command level after execution of a MERGE command.

KILL <file descriptor>

This command deletes the specified file from the disk. The function of this command is the same regardless of whether the file specified is a system file, a program file, or a random or sequential access data file; therefore, great care should be exercised in using it.

5.2.1 Creating sequential files

The steps involved in creating and accessing a sequential file are as follows:

(1) Execute an OPEN statement to assign a file number to the file and to open it in the "O" (output) mode.
(2) Write data to the file using the PRINT# or WRITE# statement.
(3) Close the file by executing a CLOSE statement. This must be done before the file can be reopened in the "I" mode for input.

Example:
The following is a short program which creates a sequential file of employee data. The file it creates will be used with subseauent programs. To simplify programming, type in all the data with the CAPS LOCK key set to SHIFT ON.

```
10 OPEN "O",#1,"A:EMPLOYEE.DAT":'Open file for output.
20 INPUT "NAME";N$             :'Assign name to N$.
30 IF N$="XX" THEN CLOSE:END   :'If XX is typed, the
40 '                             program ends.
50 INPUT "SECTION";S$          :'Assign section to S$.
60 INPUT "DATE OF BIRTH";D$    :'Assign date of birth to D$
70 PRINT #1,N$;",";S$;",";D$   :'Write data to file.
80 PRINT:GOTO 20               :'Moves cursor down.
```

Try running the program. The following prompt appears on the LCD screen:

NAME?

Type a name of employee (e.g. JOE SOAP) and press the RETURN key: the following message appears:

SECTION?

Type the name of section where the employee works (e.g. ACCOUNTS) and press the RETURN key: the following message appears:

DATE OF BIRTH?

Type the date of birth of the employee (e.g. 08/05/49) and press the RETURN key: the first message appears again. The above steps are repeated until "XX" is typed in response to the message "NAME?".

Example:
The following program reads data from the file created by the sample program in Section 5.2.1 and displays the names of all employees working in the ACCOUNTS section.

```
10 OPEN"I",#1,"A:EMPLOYEE.DAT"      :'Open file for input.
20 IF EOF(1) THEN GOTO 110          :'If EOF is encountered,
30 '                                  file closed and program
40 '                                  ends.
50 INPUT#1,N$,S$,D$                 :'Read data and assign
60 '                                  items to N$,S$ and D$
70 IF S$="ACCOUNTS" THEN PRINT N$   :'When the section is
80 '                                  ACCOUNTS, name is
90 '                                  displayed on screen.
100 GOTO 20                         :'Next data record
110 CLOSE:END                       :'Close file and end
120 '                                 program
```

The screen will show the result as follows:

```
RUN
JOE SOAP
BETTY JONES
```

The comparison for the "ACCOUNTS" string requires the string not only to be spelt the same, but that it be all in upper case. This was why the suggestion was made to set the CAPS LOCK in the program to write the data, otherwise the program would have had to check for all possibilities, for instance "Accounts", "accounts", etc.

5.2.3 Updating sequential files.

After a sequential file has been written to a disk or microcassette, it is not possible to add data to that file once it has been closed. The reason for this is that the contents of a sequential disk file are destroyed whenever that file is opened in the "O" mode. To overcome this, the following procedures can be used:

(1) Open the existing file in the "I" mode.
(2) Open a second file on the disk or microcassette in the "O" mode under a different file name.
(3) Read in data from the original file and write it to the second file, adding the new data.

The contents of the original file could be changed by replacing the contents of variables before writing them to the second file. This could be done by adding the following sequence between lines 110 and 150.

```
111 PRINT A$                      :'Display contents of A$.
112 PRINT "Change entry(Y/N)?"
113 YN$=INPUT$(1)                 :'Wait for character input.
114 IF YN$="Y" THEN 117           :'Go to 117 if Y typed.
115 IF YN$="N" THEN 150           :'Go to 150 if N typed.
116 GOTO 112                      :'Go to .112.
117 INPUT "Enter new name";NN$    :'Input new entries.
118 INPUT "Enter new section";SS$:'
119 INPUT "Enter new birthdate";DD$
120 A$=NN$+","+DD$+","+DD$        :'Assign new entries.
```

FIELD #2 , 1Ø AS S$, 3Ø AS N$, 1Ø AS C$

This allocates the fields for file number 2. The example distributes the total length of the record among the variables as follows:

1Ø bytes for S$, 3Ø$_3$ for N$, and 1Ø for C$

Note that all fields are allocated as strings. Even numeric variables are stored as strings and must be converted as shown in step (3). Make sure the total equals the declared record length. If this length is exceeded, a

FIELD overflow in <line number>

error will be generated.

(3) Data is then placed into the random file buffer using the LSET and RSET commands, depending on whether they require left or right justification. Only strings can be placed·into the buffer so any numeric values must be converted to strings first; this is done using the MKI$, MKS$, and MKD$ functions. For example:

LSET S$ = MKI$(S%)	converts and sets an integer.
LSET N$ = MKS$(Q!)	converts a single precision number.
LSET R$ = MKD$(R#)	converts a double precision number.
LSET N$ = A$	sets a string.

(4) Write data to the file from the random file buffer with the PUT statement. Records can be written in any order (in contrast to sequential files) and thus to add or change records it is only necessary to reopen the file and write further records. HOWEVER, with MICROCASSETTE files, the files MUST be written in sequential order.

(5) When all the data has been written to the file, the file must be closed using the CLOSE command:

e.g. **CLOSE**	closes all files.
CLOSE #2	simply closes file #2.

Failure to close a file may render the file impossible to be read from or written to at a later date.

5.3.2 Accessing random files

The following steps are required to retrieve data from a random access file:

(1) Open the file in the "R" mode.

(2) Using the FIELD statement, allocate space in the random file buffer for variables which are to be read in from the random file.

Note:
If the same program both writes data to a file and reads data from it, it is often possible to use just one OPEN statement and one FIELD statement.

(3) Move the desired record into the random file buffer with the GET statement. Any record number can be accessed without reading the whole file into memory as is the case with sequential files. HOWEVER, with microcassette files, the data MUST be accessed sequentially.

(4) Data in the random file buffer can now be used by the program. Be sure that numbers which are converted to ASCII strings for storage in the file are converted back into numeric values for use by the program; that is done using the CVI, CVS and CVD functions.

The following sample program accesses random file "STOCKLST.DAT" created using the program example shown in paragraph 5.3.1 above. Data records are read in and displayed by entering the stock number (record number) from the keyboard.

```
10 OPEN "R",#1,"A:STOCKLST.DAT",36
                     :'Open file in R mode.          -- (1)
20 FIELD#1,2 AS S$,30 AS N$,4 AS C$
                     :'Allocate space for variables. -- (2)
30 INPUT "ENTER STOCK NO.";S%
                     :'Input stock No.
40 IF S%=0 THEN CLOSE:PRINT"END":END
                     :'End program.
50 GET#1,S%
                     :'Read record into buffer.      -- (3)
60 PRINT USING "###";CVI(S$);:PRINT"    ";
                     :'Convert strings to numeric values
                       and display them.
70 PRINT USING "&";N$;:PRINT"    ";
80 PRINT USING "#####";CVS(C$)
90 GOTO 30
                     :'Next record
```

5.4 Microcassettes

The microcassette drive is supported as a disk device, and can be used in generally the same manner as a disk drive. It is, however, intended as a storage device which adds to the portability of the PX-8, and not as a device which would substitute for a disk drive in normal day to day use. It is obviously slower than a disk drive and has some further limitations. This section summarises the differences between the use of the microcassette as a disk drive and that of a conventional disk drive.

5.4.1 Restrictions on use

Since the microcassette is essentially a sequential access device, there are a number of restrictions on its use as a disk device.

(a) Only one microcassette file can be opened at once. If two files are opened in the input mode, only the second one to be opened can be accessed. When a file is opened in the output mode no other file can be opened until it has been closed.

(b) When a microcassette file is opened in the random ("R") mode it can be either read from or written to, but not both.

(c) When the GET statement is executed during random access, records must be read in sequence starting with record number 1. True random access is not possible.

(d) As with the GET statement, records must be accessed in sequence starting with record number 1 when the PUT statement is executed in the random mode. Further, the file must be one which has been opened for the first time; it is not possible to write data to a file which was previously created in the random mode. If an attempt is made to write another record to a file previously stored on tape a "Tape access error" will be generated.

5.4.2 Opening options

The options for opening a file with the microcassette are the same as for any other file with two additions. The full syntax is:

```
OPEN "|O|",#n,"h:(sv)filename.ext"
      |I|
      |R|
```

5.5 Errors

5.5.1 Error messages and causes

(1) Disk read error
An error occurred while data was being read from a disk.

(2) Disk write error
An error occurred while data was being written to a disk.

(3) Device unavailable
Access was attempted to a drive which did not contain a diskette, or the specified drive was not connected.

(4) Disk write protected
An attempt was made to write data to a disk which was protected with a write protect tab.
An attempt was made to write data without executing the RESET command after the diskette in that drive had been replaced.
An attempt was made to write data to a file for which the write protect attribute was set.
An attempt was made to write data to a ROM device.

(5) Tape access error
An attempt was made to access an access-inhibited microcassette file.
An attempt was made to MOUNT a tape without REMOVEing the previous tape.
An attempt was made to REMOVE a tape which has not been MOUNTed.

5.5.2 Error processing

(1) Errors occurring when a file is opened
Identify and eliminate the cause of the error, then re-execute the OPEN statement.

5.6 Precautions On Changing Floppy Disks

This section may be skipped if you do not use an optional floppy disk drive unit.

Before removing a floppy disk from the drive, be sure to CLOSE all files currently open on that drive. The reason for this is as follows:

Assume that a file on the disk being replaced is open in the "O" or "R" mode and that data has been output to that file with the PRINT # or PUT statements. Write operations to the disk by these statements are not necessarily actually made until the file is closed. Therefore, if the floppy disk is replaced without executing the CLOSE statement the contents of the file on that disk are not assured. Further, if another disk is inserted in place of the one on which the file was opened, the contents of the disk on that drive may be destroyed when an attempt is made to CLOSE the file.

To avoid the destruction of data to the maximum extent possible, the CP/M operating system is designed so that disks are automatically write protected on replacement. If an attempt is made to write data to a disk while it is this condition a "Disk write protected" error will occur. The write protected condition can be cleared and write access to the new floppy disk enabled by executing the RESET command.

For these reasons, the following procedures should be observed when replacing floppy disks:

In the direct mode:

CLOSE all files;
Replace disk;
Execute RESET.

During program execution:

```
100 CLOSE
110 PRINT "Change the disk!"
(Change the disk and press any key)
120 A$ = INPUT$(1)
130 RESET
```

floppy disks or in the RAM disk. However, the format of the file descriptor differs slightly.

For disk files the format allows an optional device name but the file must be given a name, for example:

```
OPEN “O”, # 3,“A:TEST FILE”
OPEN “O”, # 1,“NAME”
```

For the RS-232C interface port the format is as follows:

```
OPEN “<mode>”, # <file number>, “COMØ:[(<options>)]”
```

“COMØ:” must always be specified when opening the port, and no file name is required. However, the options for determining the communication mode and protocol need not be specified; if they are omitted, the default values of CP/M are used. After the PX-8 is initialized, these values are as follows; they remain effective until changed with the CONFIG program of CP/M or the OPEN statement of BASIC.

Data transfer rate:	4800 bits per second
Word length (bits/character):	8
Parity:	None
No. of stop bits:	2
DSR send check:	OFF
DSR receive check:	OFF
DCD check:	OFF
SI/SO control:	OFF
XON/XOFF:	OFF

Opening an RS-232 port for output would thus take forms such as:

```
OPEN “O”, # 3,“COMØ:”
OPEN “I”, # 1,“COMØ:(68E3F)”
OPEN “O”, # 2,“COMØ:(68E3AXN)”
```

- Options for Protocol and Control

The <options> specification in the OPEN statement determines the data communication protocol and the control options. These are specified as a character string of up to seven characters, each of which determines the set-

blpscxh protocol format

<table>
<tr><td rowspan="2">b</td><td>BIT RATE</td></tr>
<tr><td><table>
<tr><td>0:</td><td colspan="3">1200 send, 75 receive</td></tr>
<tr><td>1:</td><td colspan="3">75 send, 1200 receive</td></tr>
<tr><td>2:</td><td>110</td><td>3:</td><td>Not specifiable</td></tr>
<tr><td>4:</td><td>150</td><td>5:</td><td>Not specifiable</td></tr>
<tr><td>6:</td><td>300</td><td>7:</td><td>Not specifiable</td></tr>
<tr><td>8:</td><td>600</td><td>9:</td><td>Not specifiable</td></tr>
<tr><td>A:</td><td>1200</td><td>B:</td><td>Not specifiable</td></tr>
<tr><td>C:</td><td>2400</td><td>D:</td><td>4800</td></tr>
<tr><td>E:</td><td>9600</td><td>F:</td><td>19200</td></tr>
</table></td></tr>
<tr><td rowspan="2">l</td><td>WORD LENGTH</td></tr>
<tr><td>6: 6 bits
7: 7 bits
8: 8 bits</td></tr>
<tr><td rowspan="2">p</td><td>PARITY</td></tr>
<tr><td>N: None
E: Even
O: Odd</td></tr>
<tr><td rowspan="2">s</td><td>STOP BITS</td></tr>
<tr><td>1: 1 stop bit
2: 1.5 stop bits
3: 2 stop bits</td></tr>
<tr><td rowspan="2">c</td><td>ACTIVE CONTROL LINES</td></tr>
<tr><td><table>
<tr><th>Value</th><th>DSR send check</th><th>DSR receive check</th><th>DCD check</th></tr>
<tr><td>0 or 8</td><td>ON</td><td>ON</td><td>ON</td></tr>
<tr><td>1 or 9</td><td>ON</td><td>ON</td><td>OFF</td></tr>
<tr><td>2 or A</td><td>ON</td><td>OFF</td><td>ON</td></tr>
<tr><td>3 or B</td><td>ON</td><td>OFF</td><td>OFF</td></tr>
<tr><td>4 or C</td><td>OFF</td><td>ON</td><td>ON</td></tr>
<tr><td>5 or D</td><td>OFF</td><td>ON</td><td>OFF</td></tr>
<tr><td>6 or E</td><td>OFF</td><td>OFF</td><td>ON</td></tr>
<tr><td>7 or F</td><td>OFF</td><td>OFF</td><td>OFF</td></tr>
</table></td></tr>
<tr><td rowspan="2">x</td><td>XON/XOFF</td></tr>
<tr><td>X: On
N: Off</td></tr>
<tr><td rowspan="2">h</td><td>SHIFT-IN/SHIFT-OUT</td></tr>
<tr><td>S: On
N: Off</td></tr>
</table>

c — A hexadecimal digit from 00H to 0FH which determines which of the four control lines are checked. Correspondence between the settings of each of the four bits and the control lines to be checked is as follows:

Bit 3 — No meaning

Bit 2 — Data Set Ready (DSR) send check
1: OFF
0: ON

Bit 1 — Data Set Ready (DSR) receive check
1: OFF
0: ON

Bit 0 — Data Carrier Detect (DCD) check
1: OFF
0: ON

Combinations of settings for each hexadecimal digit are as follows:

	DSR send check	DSR receive check	DCD check
0 or 8	ON	ON	ON
1 or 9	ON	ON	OFF
2 or A	ON	OFF	ON
3 or B	ON	OFF	OFF
4 or C	OFF	ON	ON
5 or D	OFF	ON	OFF
6 or E	OFF	OFF	ON
7 or F	OFF	OFF	OFF

x — A letter which determines whether XON/XOFF (send ON/send OFF) protocol is to be used for communication control. When XON is specified and the interface is opened in the “I” mode, the PX-8 automatically outputs control code 19 (13H) during output via the RS-232C interface in the “O” mode, and automatically interrupts output until control code 17 (11H) is received from the device at the other end of

(3) Control lines used for communication through the RS-232C interface

(a) DTR (Data Terminal Ready)

DTR is a signal which is output by the PX-8 to indicate that it is ready for data communications. The level on this line becomes HIGH when the communications interface is opened in either the "I" or "O" mode, and becomes LOW when the interface is closed (when it is no longer open in any mode).

(b) RTS (Request To Send)

RTS is a signal which controls operation of a communication device (modem or acoustic coupler) connected to the PX-8. The signal on this line becomes HIGH when the interface is opened in the "O" mode, and LOW when the interface is closed.

(c) DSR (Data Set Ready)

DSR is a signal which indicates whether the communication device connected to the RS-232C port is ready for operation. When HIGH, the device connected to the interface port is ready to accept signals controlling data transmission/reception. When the interface is opened in the "I" mode with the DSR receive check bit (bit 1 of option "c") set to "0" (ON), OPEN statement execution is not completed until the level on the DSR line becomes HIGH.

(d) DCD (Data Carrier Detect)

This line is used for detecting the data carrier signal from the device connected to the RS-232C port. When the interface is opened in the "I" mode with the DCD check bit set to "0" (ON), the OPEN statement is not completed until the level on the DCD line becomes HIGH.

6.1.3 Input from the RS-232C interface

The following statements and functions are used to input data via the RS-232C interface.

Statements	Functions
INPUT #	**INPUT$**
LINE INPUT #	

The format in which data is input from the interface by these statements is exactly the same as in the case of input from disk files.

The INPUT # and LINE INPUT # statements do not allow full freedom of data format during input because they require pre-determined delimiters and termination symbols. However, the INPUT$ function permits input without regard for delimiters or terminators; thus it can be used with functions such as EOF and LOF to provide full freedom of format.

(1) Control line checks for the "I" mode

(a) DSR (Data Set Ready)
If the DSR receive check bit is set to ON (if bit 1 of option "c" is set to 0), the DSR line is monitored during input and an error is generated if it drops to LOW.

(b) DCD (Data Carrier Detect)
When the interface is opened for input with the DCD check bit set to ON (with bit 0 of option "c" set to 0), the level of the DCD line is checked at the time of execution of an OPEN "I" statement for the RS-232C interface and the port is not opened until the DCD line becomes HIGH. An error is generated if the level on this line becomes LOW during input.

(2) Errors applicable to the "I" mode

(a) Device unavailable
This error occurs when the RS-232C interface cannot be used for some reason.

(3) LOF (<file no.>)
This function returns the number of free bytes remaining in the receive buffer.

NOTE:
The size of the receive buffer is 262 bytes.

(4) INPUT$(<no. of characters>,<file no.>)
This function inputs the specified <no. of characters> from the RS-232C interface and returns them as a character string.

6.1`5 Using the LOAD, SAVE and LIST commands with the RS-232C interface

Programs can be output via the RS-232C interface in ASCII format by using LIST "COMØ:",A. When this is done, CTRL-Z (code 26, an end mark) is output after transmission of the program has been completed. However, when SAVE "COMØ:" is executed, the program saved is output in ASCII format regardless of whether the A option or the P option is specified.

When an ASCII program is loaded via the RS-232C interface with LOAD "COMØ:", loading is terminated when CTRL-Z is received. The same applies when the program is loaded using RUN "COMØ:".

If CTRL-Z is not received after receiving a program through the RS-232C interface, loading can be terminated by pressing CTRL together with the STOP key.

NOTE:
When transferring BASIC programs via the RS-232C interface, either specify a data word length of 8 bits or use Shift-in/Shift out control with a word length of 7 bits.

6.3 Keyboard

PX-8 BASIC also allows the keyboard to be handled as a sequential access input device. When the keyboard is opened as a file, data input is assigned to variables using INPUT#, LINE INPUT# and INPUT$ (X,<file no.>) instead of the corresponding dedicated keyboard input statements. This makes it possible to use common routines for input of data from the keyboard, disk device files and the RS-232C interface.

The device name used to OPEN the keyboard as a device file is "KYBD:".

(1) Statements
Statements which can be used for input from the keyboard when it is handled as a device file are as follows:

CLOSE, INPUT#, INPUT$(X,<file no.>),
LINE INPUT#, LOAD, OPEN "I"

(2) Errors
A "Bad file descriptor" error will occur if an attempt is made to open the keyboard in the "O" mode.

6.3 Keyboard

PX-8 BASIC also allows the keyboard to be handled as a sequential access input device. When the keyboard is opened as a file, data input is assigned to variables using INPUT#, LINE INPUT# and INPUT$ (X,<file no.>) instead of the corresponding dedicated keyboard input statements. This makes it possible to use common routines for input of data from the keyboard, disk device files and the RS-232C interface.

The device name used to OPEN the keyboard as a device file is "KYBD:".

(1) Statements
Statements which can be used for input from the keyboard when it is handled as a device file are as follows:

CLOSE, INPUT#, INPUT$(X,<file no.>),
LINE INPUT#, LOAD, OPEN "I"

(2) Errors
A "Bad file descriptor" error will occur if an attempt is made to open the keyboard in the "O" mode.

63 Bad record number

The record number specified in a PUT or GET statement was either zero or greater than the maximum allowed.

17 Can't continue

An attempt was made to resume execution of a program when continuation was not possible.

Possible causes:
(i) Program execution was terminated due to an error.
(ii) The program was modified while execution was suspended.
(iii) The STOP key was pressed during execution of an INPUT statement.
(iv) The program had not yet been executed.

28 Communication buffer overflow

The receive buffer overflowed during receipt of data via the RS-232C interface. This error is likely to occur when the speed with which receive processing is performed is lower than that at which data is being received, but is unlikely if the communication rate is set to 1200 bps or less.

25 Device fault

The level of the signal on the DSR or DCD line became low during input from the RS-232C interface after the DSR receive check or DCD check had been set to ON (by option "c" of the communications format specification in the OPEN"I" statement executed to open the interface).

57 Device I/O error

An error occurred involving input or output to a peripheral device.

Possible causes:
(i) An I/O error occurred during access to a disk device. This is a fatal error; that is, one from which the operating system cannot recover.
(ii) A parity error, overrun error or framing error occurred during input from the RS-232C interface. In this case, the error condition will be reset if input is continued, but there is no assurance that data received will be correct.
(iii) The printer power was off or a fault occurred when data was output to the printer.

24 Device time out

Possible causes:

11 Division by zero

An operation was encountered which included division by zero.

Possible causes:
(i) Zero was used as a divisor possibly because a variable or expression was zero at that point in the program.
(ii) Division was attempted using an undefined variable as a divisor.

1Ø Duplicate Definition

A variable array was defined more than once.

Possible causes:
(i) A second DIM statement was executed for an array without erasing that array with an ERASE statement.
(ii) An undeclared array was used, then an attempt was made to re-dimension that array with a DIM statement.
(iii) The OPTION BASE statement was executed more than once, or was executed after an array had already been dimensioned, either by a DIM statement or implicitly by assignment of a value to a variable with a subscripted name.

5Ø FIELD overflow

A FIELD statement attempted to allocate more bytes in a random file buffer than were specified for that buffer when the file was opened.

58 File already exists

The new file name specified in a NAME statement is already being used with another file on the disk.

55 File already open

An OPEN "O" statement was executed for a file which was already open, or a KILL command was executed for a file that was open.

53 File not found

The file name specified in a LOAD, KILL, NAME or OPEN statement does not exist on the disk in the accessed drive.

26 FOR without NEXT

A FOR statement was encountered without a corresponding NEXT.

(ii) The STOP key was pressed while input from the RS-232C interface was pending with INPUT#, INPUT$ or a similar command.

51 Internal error

An internal malfunction occured in BASIC.

23 Line buffer overflow

An attempt was made to input a line that contains too many characters.

22 Missing operand

Possible causes:
(i) An expression contains an operator without a following operand.
(ii) A required parameter is missing from the AUTO START or LOCATE commands.

1 NEXT without FOR

A NEXT statement was encountered without a corresponding FOR statement. Possible causes:
(i) Improperly nested FOR/NEXT loops or variables specified in the wrong order in a common NEXT statement for loops that end at the same point.
(ii) The variable in a NEXT statement does not correspond to any previously executed FOR statement variable.
(iii) More than one NEXT statement was specified for one FOR statement.
(iv) Execution branched to a point within a FOR/NEXT loop from elsewhere in the program.

19 No RESUME

No RESUME statement was included in an error processing routine. All error processing routines must conclude with an END or RESUME statement.

4 Out of DATA

A READ statement was executed when there was no unread data remaining in the program's DATA statements.

Possible causes:
(i) Insufficient number of data items in DATA statement(s).
(ii) Incorrect specification of a RESTORE statement.
(iii) Incorrect delimiting punctuation used in a DATA statement.

Possible causes:
(i) Transfer of execution to an error processing routine by a GOTO or GOSUB statement.
(ii) Lack of an END statement at the end of the main routine to keep execution from moving into an error processing routine.

3 RETURN without GOSUB

A RETURN statement was encountered which did not correspond to a previously executed GOSUB statement.

Possible causes:
(i) Execution was transferred to a subroutine by a GOTO statement.
(ii) The line number specified in a RUN command was a line in a subroutine.
(iii) No END statement was included following a main routine to keep execution from moving into a subroutine.

16 String formula too complex

The complexity of a string operation is too great.

15 String too long

An attempt was made to create a string whose length exceeds 255 characters.

9 Subscript out of range

The subscript specified in a statement referencing an array element is outside the range permitted for that array.

Possible causes:
(i) Subscript specified was greater than the maximum specified in the DIM statement defining that array.
(ii) Wrong number of subscripts specified in a statement referencing an array variable.
(iii) A subscript greater than 10 was used without executing a DIM statement to define that array.
(iv) Zero was used as a subscript after executing OPTION BASE 1.

2 Syntax error

A statement does not conform to the syntax rules of PX-8 BASIC.

Possible causes:
(i) A space was not left between a command and a parameter, e.g. LIST10.
(ii) Incorrectly typed keywords.

(i) The letters FN were used at the beginning of a variable name.
(ii) The function name was specified incorrectly in the DEF FN statement or when the function was called.
(iii) The user function was called before the corresponding DEF FN statement was executed.

21 Unprintable error

No error message has been assigned to the error condition which exists. This message is also issued for error codes 27, 31-49, 56, 59, 60, 65 and 73-255, usually due to execution of an ERROR statement specifying one of these codes.

3Ø WEND without WHILE

WEND statement was encountered without a corresponding WHILE.

29 WHILE without WEND

A WHILE statement was encountered without a corresponding WEND.

TABLE OF ERROR CODES AND ERROR MESSAGES

1	NEXT without FOR
2	Syntax error
3	RETURN without GOSUB
4	Out of DATA
5	Illegal function call
6	Overflow
7	Out of memory
8	Undefined line number
9	Subscript out of range
10	Duplicate Definition
11	Division by zero
12	Illegal direct
13	Type mismatch
14	Out of string space
15	String too long
16	String formula too complex

Appendix B TABLE OF RESERVED WORDS

ABS
ALARM
ALARM$
AND
ASC
ATN
AUTO
BEEP
CALL
CDBL
CHAIN
CHR$
CINT
CLEAR
CLOSE
CLS
COMMON
CONT
COPY
COS
CSNG
CSRLIN
CVD
CVI
CVS
DATA
DATE
DATE$
DAY
DEF
DEFDBL
DEFINT
DEFSNG
DEFSTR
DELETE
DIM
DSKF
EDIT
ELSE
END
EOF
EQV
ERASE
ERL
ERR
ERROR
EXP
FIELD
FILES
FIX
FN
FOR
FRE
GET
GO
GOSUB
GOTO
HEX$
IF
IMP
INKEY$
INP
INPUT
INPUT #
INPUT$
INSTR
INT
KEY
KILL
LEFT$
LEN
LET
LINE
LIST
LLIST
LOAD
LOC
LOCATE
LOF
LOG
LOGIN
LPOS
LPRINT
LSET
MENU
MERGE
MID$
MKD$
MKI$
MKS$
MOD
MOUNT
NAME
NEW
NEXT
NOT
OCT$
OFF
ON
OPEN
OPTION
OR
OUT
PCOPY
PEEK
POINT
POKE
POS
POWER
PRESET
PRINT
PRINT #
PSET
PUT
RANDOMIZE
READ
REM
REMOVE
RENUM
RESET
RESTORE
RESUME
RETURN
RIGHT$
RND
RSET
RUN
SAVE
SCREEN
SGN
SIN
SOUND
SPACE$
SPC
SQR
STAT
STEP
STOP
STR$
STRING$
SUB
SWAP
SYSTEM
TAB
TAN
TAPCNT
THEN
TIME
TIME$
TITLE
TO
TROFF
TRON
USING
USR
VAL
VARPTR
WAIT
WEND
WHILE
WIDTH
WIND
WRITE
WRITE #
XOR

Control Code	Function
ESC 125	Non secret
ESC 165	NUM LED off
ESC 164	NUM LED on
ESC 199	PSET/PRESET
ESC 242	Repeat interval time for keys
ESC 240	Repeat on/off for keys
ESC 241	Repeat start time for keys
ESC "*"	Screen clear
ESC 209	Screen display select
ESC "P"	Screen dump
ESC 213	Screen window end
ESC "Y"	Screen erase
ESC 212	Screen window top
ESC 145	Scroll down
ESC 244	Scroll key code

Control Code	Function
ESC 148	Scroll step
ESC 149	Scroll mode
ESC 144	Scroll up
ESC 151	Screen down n lines
ESC 150	Scroll up n lines
ESC 123	Secret mode
ESC 125	Secret mode cancel
ESC 214	Select cursor type
ESC 209	Select virtual screen
ESC 211	Select function key display
ESC 247	Shift key set
ESC 163	CAPS LED off
ESC 162	CAPS LED on
ESC 212	Top locate
ESC 224	User defined character

Use of the ESCAPE Code control sequences

Sweden	PRINT CHR$(27);"CW"
Italy	PRINT CHR$(27);"CI"
Spain	PRINT CHR$(27);"CS"
Norway	PRINT CHR$(27);"CN"

This code sequence is equivalent to the BASIC OPTION COUNTRY command.

ESC "P"

In modes 0, 1, and 2 this escape sequence outputs the contents of the screen window currently being displayed to a printer in ASCII format. In mode 3 it outputs the contents of the entire physical screen in bit image format. It duplicates the COPY or screen dump function obtained by pressing the CTRL and PF5 key.

ESC "T"

Clears the line currently containing the cursor from its present position to the end of that logical line.

ESC "Y"

Clears the screen from the current position of the cursor to the end of the screen.

ESC CHR$(123)

Causes all characters to be displayed on the screen as blanks (the secret mode). The secret mode is not active in the System Display.

WARNING:
You should make sure that a program returns the user to normal non-secret mode, for example with an error handling routine. If the user is placed in immediate mode and the secret mode is still active, it is impossible to know what is happening. Also the reset button on the left of the PX-8 must be pressed in order to see any printed output except for the clock on the MENU screen and the System Display.

ESC CHR$(125)

Terminates the secret mode.

ESC CHR$(144)

Scrolls (n – 1) lines up, starting at line (n+1) so that line (n+m – 1) becomes ing mode, and the mode in which automatic scrolling is not performed is referred to as the non-tracking mode. The tracking mode is used unless otherwise specified. The escape sequence for determining the tracking mode is as follows:

PRINT CHR$(27);CHR$(149);CHR$(<mode>);

In this sequence, <mode> is specified as either 0 or 1. The tracking mode is selected when 0 is specified, and the non-tracking mode is selected when 1 is specified.

ESC CHR$(150)

In modes 0, 1, and 2 this escape sequence displays the contents of the virtual screen containing the cursor after moving the screen window up n lines where n is the value specified by ESC CHR$(148), or 1 if ESC CHR$(148) has not been executed. If scrolling the screen up n lines would move the screen window beyond the home position, the virtual screen is displayed starting at the home positon. The cursor remains in its original position in the virtual screen.

ESC CHR$(151)

In modes 0, 1, and 2 this escape sequence displays the contents of the virtual screen containing the cursor after moving the screen window down n lines, where n is the value specified by ESC CHR$(148), or 1 if ESC CHR$(148) has not been executed. If scrolling the screen down n lines would move the screen window beyond the end of the virtual screen, the screen window is positioned so that the virtual screen's last line is displayed in the last line of the screen window. The cursor remains in its original position in the virtual screen.

ESC CHR$(160)

Lights the INS LED. It does not put the user in the insert mode.

ESC CHR$(161)

Turns off the INS LED.

ESC CHR$(162)

Lights the CAPS LED. It does not set the CAPS LOCK key to the on position.

ESC CHR$(163)

Turns off the CAPS LED.

Byte 10:	Low byte of vertical ending position
Byte 11:	First byte of mask pattern
Byte 12:	Second byte of mask pattern
Byte 13:	Function

The starting and ending positions are specified as two-byte hexadecimal numbers which indicate coordinates in the graphic screen. For example, starting coordinates of 400,20 (&H0190,&H0014) would be specified as follows:

Byte 3:	1 (&H01)
Byte 4:	144 (&H90)
Byte 5:	0 (&H00)
Byte 6:	20 (&H14)

The mask pattern used for drawing the line is specified in bit image format as described in the explanation of the LINE statement in Chapter 4. Calculations for diagonal lines are performed automatically. Function is specified as a number from 1 to 3 with the following meanings:

1:	OFF
2:	ON
3:	Complement

Dot positions corresponding to "1" bits in the mask pattern are reset (turned off) when 1 is specified for the function and are set (turned on) when 2 is specified. When 3 is specified, the complements of dots corresponding to "1" bits are displayed (ON dots corresponding to "1" bits are turned off, and OFF dots are turned on).

An example of specification of this sequence as follows draws a line from point (400,18) of the screen to point (18,18):

```
PRINT CHR$(27);CHR$(198);CHR$(1);CHR$(144);CHR$(Ø);
CHR$(18);CHR$(Ø);CHR$(18);CHR$(Ø);CHR$(18);
CHR$(&HAA); CHR$(&HAA);CHR$(2);
```

This command duplicates the LINE command of BASIC, but also allows the dots to be inverted (i.e. switch them on if they are off and vice versa), which LINE does not.

ESC CHR$(199)

This escape sequence sets or resets the specified points of the graphic screen. No operation is performed if this sequence is executed in modes 0, 1, or 2. The sequence consists of six bytes as follows:

The following sequence selects screen mode 2, sets the number of lines in virtual screen 1 to 10, the number of columns to 20 and "#" as the boundary character.

```
PRINT CHR$(27);CHR$(2Ø8);CHR$(2);CHR$(1Ø);CHR$(2Ø); "#";
```

ESC CHR$(2Ø9)

In modes 0, 1, or 2 this escape sequence specifies which of the two virtual screens is to be displayed. The operation is performed if this sequence is executed in mode 3. This is done as follows:

```
PRINT CHR$(27);CHR$(2Ø9);CHR$(n);
```

The first virtual screen is selected when 0 is specified for n, and the second virtual screen is selected when 1 is specified for n. If the third byte is not specified the default is 1.

ESC CHR$(21Ø)

Displays the specified character in the specified position on the real screen. This is done as follows:

```
PRINT CHR$(27);CHR$(21Ø);CHR$(x);CHR$(y);CHR$(p)
```

The meanings of x, y and p are as follows:

x	Vertical position (1 to 8)
y	Horizontal position (1 to 80)
p	ASCII character code

This sequence makes it possible to output characters to any location in the real screen, regardless of the position of the cursor or number of lines in the screen window.

ESC CHR$(215)

In modes 0, 1, and 2 this escape sequence moves the screen window to the position occupied by the cursor. This sequence does nothing if executed in mode 3. The screen window is positioned so that the cursor is located near its centre.

ESC CHR$(224)

This escape sequence defines those characters corresponding to ASCII codes 224 (&HE0) to 254 (&HFE). This sequence consists of eleven bytes as follows:

Byte 1:	CHR$(27)
Byte 2:	CHR$(224)
Byte 3:	Character code
Byte 4:	Pattern for dot row 1
Byte 5:	Pattern for dot row 2
Byte 6:	Pattern for dot row 3
Byte 7:	Pattern for dot row 4
Byte 8:	Pattern for dot row 5
Byte 9:	Pattern for dot row 6
Byte 10:	Pattern for dot row 7
Byte 11:	Pattern for dot row 8

The pattern making up each dot row is specified as the ASCII code equivalent of the binary number whose "1" bits correspond to dots which are turned on, and whose "0" bits correspond to dots which are turned off. For example, specifying CHR$(63) (where 63 is the decimal equivalent of 1111111B) for byte 1 causes all dots in dot row one to be turned on when the character code specified in byte 3 is displayed; conversely, specifying CHR$(0) (i.e.,00000000B) causes all dots in the applicable row to be turned off.

Byte 1:	CHR$(27)
Byte 2:	CHR$(241)
Byte 3:	CHR$(n)

The keyboard repeat function starting time is equal to n/64 seconds where n is a number from 1 to 127.

ESC CHR$(242)

Sets the duration of the key repeat interval. This sequence consists of three bytes as follows:

Byte 1:	CHR$(27)
Byte 2:	CHR$(242)
Byte 3:	CHR$(n)

The key repeat interval is equal to n/256 seconds, where n is a number from 1 to 127.

ESC CHR$(243)

Sets the arrow key codes. This sequence consists of six bytes as follows:

Byte 1:	CHR$(27)
Byte 2:	CHR$(243)
Byte 3:	Code for →
Byte 4:	Code for ←
Byte 5:	Code for ↑
Byte 6:	Code for ↓

This sequence only changes the arrow key codes during program execution. Normal code assignments are restored automatically when BASIC returns to the command mode.

ESC CHR$(244)

Sets the scroll key codes. This sequence consists of six bytes as follows:

Byte 1:	CHR$(27)
Byte 2:	CHR$(244)
Byte 3:	Code for SHIFT + →
Byte 4:	Code for SHIFT + ←
Byte 5:	Code for SHIFT + ↑
Byte 6:	Code for SHIFT + ↓

Appendix D MACHINE LANGUAGE SUBROUTINES

The CALL and USR statements of BASIC make it possible to execute machine language subroutines from programs written in BASIC. Such subroutines must be written into memory in machine language with the POKE statement before they can be called. It is also possible to use an assembler such as the MACRO-80 assembler and LINK-80 linker/loader to assemble and load routines written in assembly language; however these programs are not included in the transient program ROM capsule provided with the PX-8, and must be loaded from a flexible disk which is compatible with CP/M and the PX-8. See any of the various handbooks available on the Z80 microcomputer or the Z80 assembly language for the Z80 instruction code set.

When preparing machine language subroutines, remember that the presence of even a single error in the machine code is likely to result in destruction of all data included in the PX-8's memory (including BASIC itself). Therefore, be sure to back up all data and programs in memory on a disk before attempting to test or debug such routines.

1. Memory Allocation

Memory space must be reserved for storage of the instruction codes of machine language subroutines before they can be written into memory with the POKE statement. This is done using the CLEAR statement of BASIC or the /M: option of the BASIC command. When using the /M: option, the starting address of the machine language area is the address specified, and the ending address is that immediately preceding the starting address of BDOS. Locations 6 and 7 in page zero hold the current BDOS starting address. This will change depending on the USER BIOS and RAM disk sizes.

When a machine language subroutine is called, the stack pointer is set up for 8 levels (16 bytes) of stack storage. If more stack space is required, BASIC's stack can be saved and a new stack set up for use by the machine language subroutine. BASIC's stack must be restored, however, before returning from the subroutine.

2. USR Function Calls

With BASIC, the format used for calling USR functions is as follows.

USR[< digit >](argument)

Example A$=”STRING CHARS” + “”

This will copy the string literal into string space and prevent alteration of program text during a subroutine call.

3. CALL Statement

BASIC user function calls may also be made with the CALL statement. A CALL statement with no arguments generates a simple “CALL” instruction. The corresponding subroutine should return to the BASIC program via a simple “RET” instruction (“CALL” and “RET” are Z80 opcodes; see a Z80 reference manual for details.)

A subroutine CALL with arguments results in a somewhat more complex calling sequence. For each argument in the CALL argument list, a parameter is passed to the subroutine. That parameter is the address of the low byte of the argument. Therefore, parameters always occupy two bytes each, regardless of type.

The method of passing parameters depends on the number of parameters to be passed as follows:

(1) If the number of parameters is less than or equal to 3, they are passed in the registers. Parameter 1 will be in HL, 2 (if present) in DE, and 3 (if present) in BC.

(2) If the number of parameters is greater than 3, they are passed as follows:
   (a) Parameter 1 in HL.
   (b) Parameter 2 in DE.
   (c) Parameters 3 through n in a contiguous data block. Register pair BC will point to the low byte of this data block (i.e., to the low byte of parameter 3).

Note that with this scheme the subroutine must know how many parameters to expect in order to find them. Conversely, the calling program is responsible for passing the correct number of parameters. There are no checks for correct number or type of parameters.

When accessing parameters in a subroutine, don’t forget that they are pointers to the actual arguments passed.

NOTE:
It is entirely up to the programmer to ensure that arguments in the calling program match those expected by the subroutine in number, type and length. This applies to BASIC subroutines as well as those written in machine language.

Appendix E DERIVED FUNCTIONS

Functions that are not intrinsic to PX-8 BASIC may be calculated as follows.

Function	BASIC Equivalent
SECANT	SEC(X)=1/COS(X)
COSECANT	CSC(X)=1/SIN(X)
COTANGENT	COT(X)=1/TAN(X)
INVERSE SINE	ARCSIN(X)=ATN(X/SQR(1-X*X+1))
INVERSE COSINE	ARCCOS(X)= -ATN(X/SQR(1-X*X)) +1.570796326794897
INVERSE SECANT	ARCSEC(X)=ATN(SQR(X*X-1)) +(SGN(X)-1)*1.570796326794897
INVERSE COSECANT	ARCCSC(X)=ATN(1/SQR(X*X-1)) +(SGN(X)-1)*1.570796326794897
INVERSE COTANGENT	ARCCOT(X)= -ATN(X)+1.570796326794897
HYPERBOLIC SINE	SINH(X)=(EXP(X)-EXP(-X))/2
HYPERBOLIC COSINE	COSH(X)=(EXP(X)+EXP(-X))/2
HYPERBOLIC TANGENT	TANH(X)=(EXP(X)-EXP(-X))/(EXP(X) +EXP(-X))
HYPERBOLIC SECANT	SECH(X)=2/(EXP(X)+EXP(-X))
HYPERBOLIC COSECANT	CSCH(X)=2/(EXP(X)-EXP(-X))
HYPERBOLIC COTANGENT	COTH(X)=(EXP(X)+EXP(-X))/(EXP(X) -EXP(-X))
INVERSE HYPERBOLIC SINE	ARCSINH(X)=LOG(X+SQR(X*X+1))
INVERSE HYPERBOLIC COSINE	ARCCOSH(X)=LOG(X+SQR(X*X-1)
INVERSE HYPERBOLIC TANGENT	ARCTANH(X)=LOG((1+X)/(1-X))/2
INVERSE HYPERBOLIC SECANT	ARCSECH(X)=LOG((SQR(1-X*X)+1)/X)
INVERSE HYPERBOLIC COSECANT	ARCCSCH(X)=LOG((1+SGN(X)* SQR(1+X*X))/X
INVERSE HYPERBOLIC COTANGENT	ARCCOTH(X)=LOG((X+1)/(X-1))/2

Any of these functions can easily be used in a program by defining it with a DEF FN statement. This is illustrated in the example below.

Example

Function definition: DEF FN SINH(X)-(EXP(X)-(EXP(X))/2
Function call: A=FNSINH(Y)

Differences between the USASCII character set and the character sets of other countries are as shown below.

Country / Dec. Code	United States	France	Germany	England	Denmark	Sweden	Italy	Spain	Norway
35	#	#	#	£	#	#	#	₧	#
36	$	$	$	$	$	¤	$	$	¤
64	@	à	§	@	É	É	@	@	É
91	[	°	Ä	[	Æ	Ä	°	¡	Æ
92	\	ç	Ö	\	Ø	Ö	\	Ñ	Ø
93	]	§	Ü	]	Å	Å	é	¿	Å
94	^	^	^	^	Ü	Ü	^	^	Ü
96	`	`	`	`	é	é	ù	`	é
123	{	é	ä	{	æ	ä	à	¨	æ
124	¦	ù	ö	¦	ø	ö	ò	ñ	ø
125	}	è	ü	}	å	å	è	}	å
126	~	¨	ß	~	ü	ü	ì	~	ü

Appendix H SOME EXAMPLE PROGRAMS

This manual is not meant to be a tutorial manual to teach BASIC — there are many books which teach the use of MICROSOFT BASIC. However, some aspects of programming are specific to the PX-8. This appendix is meant to illustrate some of these specific points and provide examples of how the computer can be programmed in ways which exploit the features of the machine.

1. Use of the User-Defined Characters

Appendix F shows the character set of the PX-8. It is normally only possible to program the characters which have an ASCII code of 22 and above. It is possible with a machine code routine to alter the VRAM and reconfigure it to allow the characters from ASCII code 160 to 254 to be programmed. This is beyond the scope of this manual. The downloading of a character from software is outlined in Appendix C, under the escape sequence ESC CHR$(224). The following programs and descriptions extend this information by showing practical examples.

(i) A simple program to illustrate the definition of a character and printing it to the screen.

A character is defined by sending the sequence:

BYTE 1:	CHR$(27)	The ESC character
BYTE 2:	CHR$(224)	The code to download
BYTE 3:	CHR$(n)	The code for the character to be changed
BYTE 4:	CHR$(r1)	The pattern for the top row
BYTE 5:	CHR$(r2)	The pattern for row 2
BYTE 6:	CHR$(r3)	The pattern for row 3
BYTE 7:	CHR$(r4)	The pattern for row 4
BYTE 8:	CHR$(r5)	The pattern for row 5
BYTE 9:	CHR$(r6)	The pattern for row 6
BYTE 10:	CHR$(r7)	The pattern for row 7
BYTE 11:	CHR$(r8)	The pattern for the bottom row

The first program defines the character shown in the diagram and prints it on the screen. It is downloaded into the user-defined character area as the character with ASCII code 231 (or E7 in hexadecimal notation).

```
10 CLS
20 PRINT CHR$(27);CHR$(224);CHR$(231);
30 FOR Y=1 TO 8
40 READ A
50 PRINT CHR$(A);
60 NEXT
70 PRINT
80 PRINT CHR$(231)
90 PRINT
100 DATA 12,30,33,33,18,12,63,0
```

```
\run
 ⍜
 Ok
```

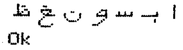
In line 20 it it very important that the semi-colon is placed at the end of the line. Without this the first two bytes of the row will be interpreted as the carriage return and line feed, which would normally cause the cursor to move to the beginning of the next line. Carriage return is ASCII code 13 and line feed is ASCII code 10 in decimal notation. Try leaving the semi-colon out and note the change in the character. Because these two extra characters are inserted the bottom two rows defined are lost, with the two extra rows being inserted at the top to correspond to the line feed and carriage return.

The data for the character is read in from the series of DATA statements. If a series of characters are being defined, they are best arranged in sets of eight DATA statements on different lines. This makes it easier to find out which data byte corresponds to which row of which character when you wish to change the character or are debugging a program.

(ii) The next program shows how blocks of graphics characters can be used to make larger characters. The example shows a set of ARABIC characters where each character is made up of a block of four user-defined characters.

```
10 'User defined graphics
20 FOR X=&HE0 TO &HFB
30 PRINT CHR$(27);CHR$(&HE0);CHR$(X);
40 FOR Y=1 TO 8
50 READ A
60 PRINT CHR$(A);
70 NEXT Y
80 NEXT X
85 CLS
90 FOR X=&HE0 TO &HFB STEP 4
```

loaded into a suitable printer which is capable of receiving characters in a downloadable form. Such a screen dump of the output of the screen appears as follows when the program has been run:

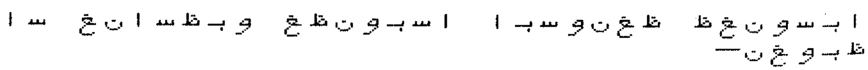


```
Ok
```

NOTE:
When using these characters in the following programs you should LOGIN to another program area before typing them in. If the screen is changed by either the SCREEN command or WIDTH command, or by going via the menu, the first two user-defined characters, and possibly more will be altered. Simply using LOGIN will not reset them.

(iii) When combinations of characters are used in this way, it is often more convenient if the characters are grouped as a variable, so that simply saying PRINT A$ for example prints the block as a whole. This can be achieved using string concatenation. The following program shows how the ARABIC characters of the previous program can be defined as variables, and how they can be printed as one character by typing a key. The characters have been designed so that the cursor moves from right to left to illustrate ways of using control codes.

```
10 STOP KEY OFF
20 SCREEN 3,0,0 : CLS
30 XP = 1: YP = 1
40 FOR N = 0 TO 6
50 J = 4 * N
60 C$(N) = CHR$(&HE0+J)+ CHR$(&HE0+J+1)+CHR$(8)+CHR$(8)+CHR$
(10)+CHR$(&HE0+J+2)+CHR$(&HE0+J+3)+CHR$(&H1E)+STRING$(4,8)
70 NEXT N
80 CSR$ = CHR$(133)+CHR$(133)+STRING$(2,8)
90 SP$ = "  " + STRING$(4,8)
100 LOCATE XP+68,YP*2,0
110 PRINT CSR$;
120 A$ = INKEY$ : IF A$ = "" THEN 120
130 IF A$ = CHR$(27) THEN CLS : STOP KEY ON : END
140 IF A$ = " " THEN PRINT SP$;: GOTO 180
150 IF A$ < "A" OR A$ > "G" THEN 120
160 V = 71 - ASC(A$)
170 PRINT C$(V);
180 XP = XP + 1 : IF XP > 34 THEN XP = 1 : YP = YP + 1: IF Y
P > 4 THEN CLS : YP = 1 : GOTO 100 ELSE GOTO 100
190 GOTO 110
```



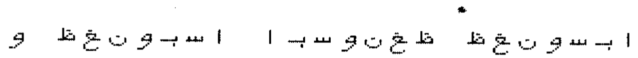
Line 110 prints the cursor and line 120 waits for a key to be pressed. Lines 130 to 150 test which key has been pressed. The [ESC] key (ASCII code 27) allows the user to exit from the program. Line 140 prints a space and line 150 eliminates all characters other than those in the range "A" to "G".

When a key in the permitted range is pressed, the ASCII code is subtracted from the constant 71 to index the array C$( ), and the corresponding character is printed. The counter XP is then incremented so that a check can be made on the number of characters per line. When this is exceeded the line counter YP is incremented and the cursor moved to the right hand position on the next line. When the screen is full it is cleared.

The program loops back to line 110 to print the cursor and wait for another character, until the user exits by pressing the [ESC] key.

(iv) The restrictions of the LOCATE command in the previous program can be overcome if the position of the characters is calculated instead of being printed as a block having previously been defined in a string. This allows the characters to go up the edge of the screen, but does requires some sophisticated numerical computation.

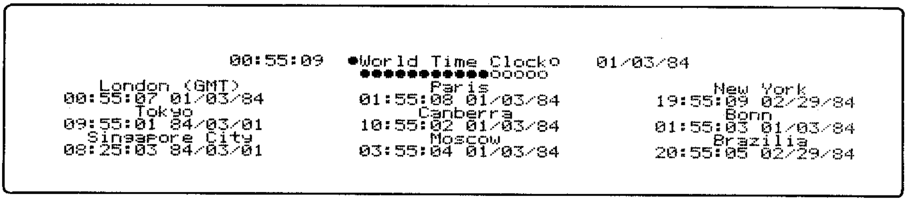
```
10 STOP KEY OFF
20 SCREEN 3,,0:CLS
30 XP=1:YP=1
40 LOCATE 81-XP*2,YP*2+1:PRINT CHR$(133);CHR$(133);
50 A$=INKEY$:IF A$="" THEN 50
60 IF A$=CHR$(27) THEN CLS:STOP KEY ON:END
70 IF A$=" " THEN LOCATE 81-XP*2,YP*2+1,0:PRINT"  ";:GOTO 150
80 IF A$<"A" OR A$>"G" THEN 50
90 LOCATE 81-XP*2,YP*2,0
100 V=71-ASC(A$)
110 C=&HE0+V*4
120 PRINT CHR$(C);CHR$(C+1);
130 LOCATE 81-XP*2,YP*2+1,0
140 PRINT CHR$(C+2);CHR$(C+3);
150 XP=XP+1:IF XP>40 THEN XP=1:YP=YP+1:IF YP>4 THEN CLS:YP=1
160 GOTO 40
```



This program is a modification of the previous one.

After initialising the variables in lines 10 to 30, the main body of the program begins at line 40 by printing the cursor, using the same characters as in the previous program. The position is calculated by means of the counter XP which is used later in the program to determine how many blocks

2. A Clock Program



```
                    00:55:09  ●World Time Clock○   01/03/84
                               ●●●●●●●●●●●○○○○○
      London (GMT)                  Paris                      New York
00:55:07 01/03/84          01:55:08 01/03/84           19:55:09 02/29/84
        Tokyo                     Canberra                       Bonn
09:55:01 84/03/01          10:55:02 01/03/84           01:55:03 01/03/84
   Singapore City                  Moscow                      Brazilia
08:25:03 84/03/01          03:55:04 01/03/84           20:55:05 02/29/84
```

```
10 STOP KEY OFF
20 DEF FNT(P1$,P2$)=VAL(MID$(P1$,INSTR("HMS",P2$)*3-2))
30 DEF FND(P1$,P2$)=VAL(MID$(P1$,INSTR("MDY",P2$)*3-2))
40 DEF FNN(P1,P2) = (P1>7)*(((P1 MOD 2)=0)*31+((P1 MOD 2)=1)
*30) + (P1<8)*(((P1 MOD 2)=0)*30+((P1 MOD 2)=1)*31+(P1=2)*((
(P2 MOD 4)=0) + ((P2 MOD 4)<>0)*2))
50 DEF FNS$(P1,P2,P3,P4$)=RIGHT$("0"+MID$(STR$(P1),2),2)+P4$
+RIGHT$("0"+MID$(STR$(P2),2),2)+P4$+RIGHT$("0"+MID$(STR$(P3)
,2),2)
60 DEF FNP(P1)=-MN*(P1=1)-DY*(P1=2)-YR*(P1=3)
70 OH=FNT(TIME$,"H"):T=0:SS=0
80 READ MAX
90 DIM XP(MAX),YP(MAX),AS$(MAX),HR(MAX),ME(MAX),SD(MAX),L(3,
MAX)
100 FOR X=1 TO MAX
110 READ XP(X),YP(X),AS$(X),NT$,L(1,X),L(2,X),L(3,X)
120 HR(X)=FNT(NT$,"H")
130 ME(X)=FNT(NT$,"M")
140 SD(X)=FNT(NT$,"S")
150 NEXT
160 SCREEN 0,0,0:CLS
170 LOCATE 29,1:PRINT CHR$(143);"World Time Clock";CHR$(143)
180 LOCATE 30,2:PRINT STRING$(16,143)
190 PRINT"      London (GMT)                Paris
     New York"
200 PRINT:PRINT"          Tokyo                       Canberra
          Bonn"
210 PRINT:PRINT"      Singapore City                 Moscow
         Brazilia"
220 A=FRE(0):A=FRE(A$)
230 H=FNT(TIME$,"H"):M=FNT(TIME$,"M"):S=FNT(TIME$,"S")
240 TD=FND(DATE$,"D"):TM=FND(DATE$,"M"):TY=FND(DATE$,"Y")
250 T=1-T:IF T=0 THEN SS=1-SS
260 FOR X=1 TO MAX
270 DY=TD:MN=TM:YR=TY
280 IN$=INKEY$:IF IN$=CHR$(3) THEN 420
290 LOCATE 19,1:PRINT TIME$:LOCATE 50,1:PRINT FNS$(FNP(2),FN
P(1),FNP(3),"/")
300 IF OH<>H THEN SOUND 1200,5:OH=H
310 H=FNT(TIME$,"H"):M=FNT(TIME$,"M"):S=FNT(TIME$,"S")
320 IF T=0 AND X=1 THEN ZX=29:ZY=1:GOTO 350
```

This program, while being useful, is meant to illustrate the use of string handling and other commands in BASIC in a practical program. It also contains subroutines which handle the time and date. These will be of use in programming the PX-8 in combination with the ALARM command. When combined with the type of program given as an example in the ALARM section of Chapter 4 these subroutines allow a wide range of time based applications for the PX-8.

The program shows the time at various places around the world relative to GMT.

Line 10 switches off the STOP key while the program is running, allowing lines 280 and 420 to end the program in an orderly manner by re-enabling the key only when the user wishes to cease program execution.

Lines 20 to 60 define a number of functions which allow values to be determined related to the date and time.

Line 20 defines a function which returns the hours, minutes or seconds from the TIME$ string. Line 30 performs the same function on the DATE$ string. They are used in lines 230 and 240 and elsewhere in the program. As an example of their operation, consider the problem of returning the minutes from the time in the second statement in line 310:

M=FNT(TIME$,"M")

The two strings TIME$ and "M" are substituted for the values of P1$ and P2$ respectively in function FNT. The INSTR function is then used to return a value of 1, 2 or 3 depending on whether string P2$ is "H", "M" or "S" since P2$ is being searched for in the string "HMS". In this example it is "M", so INSTR returns a value of 2. The characters corresponding to the value of the hour begin at position 1 in the TIME$ string, the minutes at position 4 and the seconds at position 7. By multiplying the value returned by the INSTR function by 3 and subtracting two, the value returned will correspond to the correct position. The function MID$ is then used to extract the string beginning with this position, and the numerical value of the minutes is found using the VAL function. This value is then stored in the variable M.

Line 40 defines a function which returns the value of the number of days in the month. It is used in the two subroutines which add or subtract the time difference, for example in line 1050. In this example, MN is a variable which contains the number of the month and YR is a variable containing the last two digits of the year. These values are passed to the variables P1 and P2 in the function FNN. The function uses an algorithm involving logical operations to return the true. This means that the expression will return a total value of 1 if the year is a leap year and 2 if it is not. The way this is built up is again easier to see if the values are placed under the expressions:

```
                 (P1=2)*(((P2 MOD 4)=Ø)+((P2 MOD 4)< >Ø)*2)
leap        -1      *((        -1       )+(        Ø          )*2)=1
non-leap  -1      *((         Ø        )+(       -1          )*2)=2
```

If the month is not February the expression (P1=2) is false and therefore returns 0 and so the whole expression evaluates to 0.

If the rest of the expression is broken down so that the part concerned with February is marked FEB, the decision as to the days of the months becomes:

```
(P1<8)*(((P1 MOD 2)=Ø)*3Ø+((P1 MOD 2)=1)*31+FEB)
```

Apart from allowing for February the logic is the same as in the other months of the year except that even months have 30 days and odd months 31.

Suppose that the month is February. A value of -30 will be returned by the expression ((P1 MOD 2)=0)*30 since February is an even month, and also FEB will return a value of 1 for a leap year, and 2 for a non leap year.' The total expression thus gives 28 days for February in a normal year and 29 in a leap year, as follows:

```
               (P1<8)*(((P1 MOD 2)=Ø)*3Ø+((P1 MOD 2)=1)*31+FEB)
leap        -1      *((     -1         )*3Ø+(       Ø          )*31+ 1   )=29
non leap  -1      *((     -1         )*3Ø+(       Ø          )*31+ 1   )=3Ø
```

Although this might seem complicated to begin with, it is extremely compact, and therefore the program runs faster. Try writing the same algorithm as a series of IF..THEN statements and see how many lines it involves.

Line 50 defines a function to cope with the case where the number returned for any part of the date or time is not a two digit number. It adds the leading "0" if required. It is used in line 390 to print the time and date. For example FNS$(HRS,MIN,SEC, " : ") passes the values for the hour, the minutes, the seconds and the separator ":" to the variables P1,P2,P3 and P4$ of the function. Each numerical value is then converted to a string, using the string expression:

```
RIGHT$ ("Ø"+MID$(STR$(P),2),2)
```

A number is always printed with a leading and trailing space, so that an expression such as PRINT "PROFIT";PR; "percent" does not print "PROFIT20 percent" but a legible "PROFIT 20 percent". When a number or a numerical

The array AS$ is used to hold the variable which denotes whether the time is ahead of or behind the local time;

The three arrays HR, ME and SD hold the values by which the time is different, and L the order in which the date is displayed for each country as used in the function FNP in line 60.

The loop 100 to 150 reads the values into the arrays from the data statements, using the function FNT to convert the time difference from a string into the appropriate numerical value.

Lines 160 to 210 set up the screen. The graphics character of ASCII code 143 is printed around the title, and is used later to give a visual indication of the seconds ticking by.

Line 220 is an important line in helping the program run without delays. A large number of variables are used and they are constantly changing. Initially BASIC stores them until it begins to run out of space, then it has to clear out the unwanted old values to make space to work. This is known as "garbage collection" and when it occurs with a large number of variables, it can cause the program to appear to have stopped working for a time. By executing the functions A=FRE(0) and A=FRE(A$), both the areas used for numerical and for string variables are cleaned up. Although the program will actually stop whilst this process is carried out, many small stops are invisible to the user because they are forced to happen. This line is the start of the main program loop and so happens before the screen is changed each time.

Lines 230 and 240 use the functions FNT and FND to determine the current hour and day, date and year from the internal clock in the PX-8. The hour is used in line 300 to sound the hour if there has been a change. If the hour is not set in line 230, the hour will be sounded when the program starts because H will have the value zero and will be seen as not equal to the variable OH in line 300.

Line 250 sets the variables T and SS which are used in lines 320 to 350 to define the position of the changing graphics character around the heading.

The loop in lines 260 to 400 prints out each time and date. The particular country is indexed by the loop counter X. Line 280 checks whether the STOP key or CTRL + C has been pressed. If it has an orderly exit is made in line 420.

mined a correction is made for the day and if necessary the year. The function FNN of line 40 is used in line 2040 to determine the number of days in the month.

Lines 2000 to 2060 form a similar subroutine to subtract the time.