A COURSE IN BASIC PROGRAMMING

by
Hugo Davenport

CONTENTS

Chapter 1 INTRODUCTION
Chapter 2 GETTING STARTED
Chapter 3 NOT SO BASIC
Chapter 4 TALKING TO THE ZX-80
Chapter 5 SO YOU'VE GOT PROBLEMS
Chapter 6 FINDING THE ANSWERS
Chapter 7 DECISIONS, DECISIONS
Chapter 8 BRANCHING OUT
Chapter 9 ITER-WHAT?
Chapter 10 LOOPING THE LOOP
Chapter 11 HOW TO PRINT
Chapter 12 COPING WITH CHARACTERS
Chapter 13 HELP! OR WHAT TO DO WHEN DESPERATE
Chapter 14 A RAGBAG OF FUNCTIONS
Chapter 15 OVER TO YOU
Appendix I ERROR CODES
Appendix II 4K BASIC FOR ZX-80
Appendix III SYSTEM VARIABLES
Index

© 1980 by Science of Cambridge Ltd.
Second edition.

ClibPDF - www.fastio.com

CHAPTER 1

INTRODUCTION

ClibPDF - www.fastio.com

INTRODUCTION — READ CHAPTER TWO FIRST!

This book is the user's manual for the ZX-80 personal computer. In it is everything you need to know in order to set up the ZX-80 hardware and to use the ZX-80 BASIC language for writing your own programs.

For most of the book the chapters are grouped in pairs. The first (odd-numbered) chapter of a pair is purely descriptive and prepares the reader for what is covered in the second (even-numbered) chapter of the pair. The second chapter of a pair is intended to take you through the use of ZX-80 BASIC as painlessly as possible while you're actually sitting at the keyboard.

If you are an experienced BASIC user, you won't need to read very much — probably only chapter 2, the ZX-80 BASIC language summary, and the Index.

If you're less experienced and patient, it's probably best to read through the whole book, chapter by chapter. If you're less experienced but impatient you can skip the odd numbered chapters 3, 5, 7, and 9 (and probably will!).

OK, NOW READ CHAPTER TWO — AGAIN!

ClibPDF - www.fastio.com

CHAPTER 2

GETTING STARTED

ClibPDF - www.fastio.com

GETTING STARTED

This is the most important chapter in the book. It tells you how to connect up and switch on your ZX-80.

The ZX-80 consists of two units:

(1) The computer

(2) The power supply

The power supply used should be 9 volts DC @ 600 mA unregulated, terminated with a 3.5 mm jack plug. The tip of the jack plug must be positive.

DON'T PLUG IN TO THE MAINS SUPPLY YET

Look at the back of the ZX-80 computer. You will see four sockets. Three of these are 3.5 mm jack sockets marked MIC, EAR and POWER. The fourth is a phono socket for the video cable.

The connection diagram shows how to connect up the ZX-80 to the power supply and to a domestic television which will be used to display the output for the ZX-80.

A video cable with a coaxial plug at each end is provided to connect the TV to the computer.

A twin cable with 3.5 mm jack plugs at both ends is provided to connect the ZX-80 to a cassette recorder. Don't worry about this for the moment.

Connect the power supply to the ZX-80 by plugging the 3.5 mm jack plug into the socket marked 9V DC IN (if you do get it in the wrong jack socket you won't damage your ZX-80 even if you switch on the power. It won't work until you get the plug in the right socket, though!)

SEE DIAGRAM

Connect the TV using the coax lead provided. You will have to tune the TV to the ZX-80 frequency, approximately channel 36 on UHF, so check that you can do this – most TV sets either have a continuously variable tuning control or, if they select channels with push-buttons, have separate tuning controls for each channel. If your set is a push-button set select an unused channel (ITV 2, perhaps). Turn down the volume control.

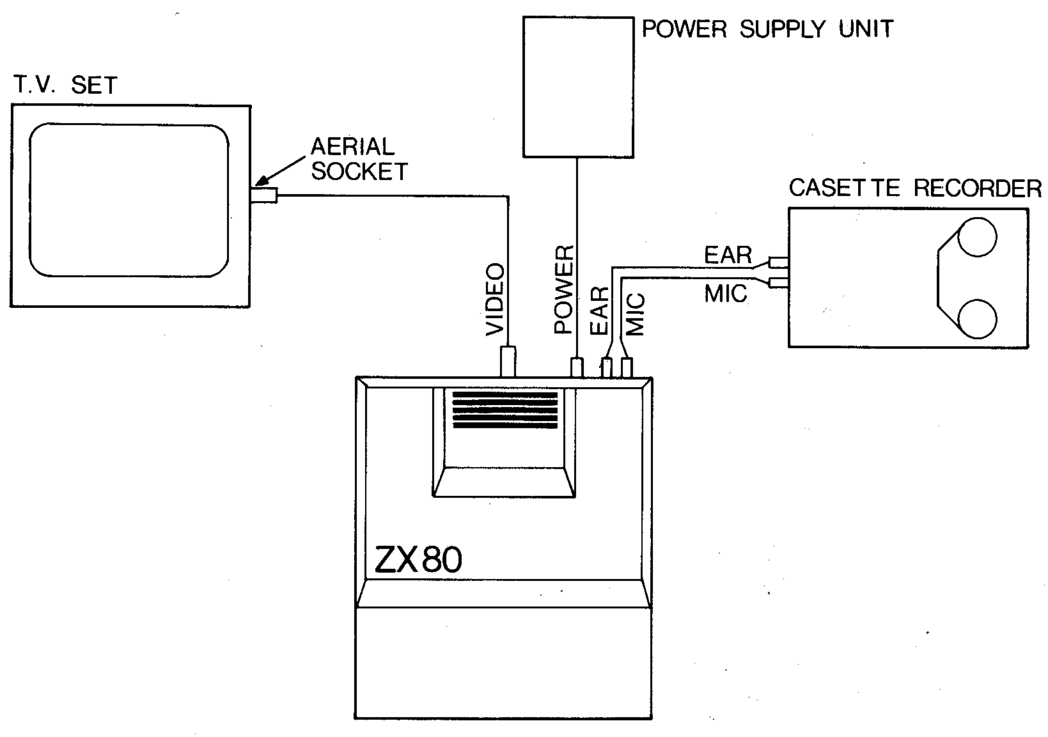
OK – we can't put off the moment of truth any longer!

Switch on the TV.

When it has warmed up . . .

SWITCH ON THE ZX-80.

ClibPDF - www.fastio.com



Connecting up the ZX80

ClibPDF - www.fastio.com

You will be disappointed to see a horrible grey mess on the screen!!

Try tuning in the TV set. At some position of the tuning control the screen will suddenly clear. At the bottom left hand corner of the screen you will see a curious symbol – a black square with a white letter K in it.

If you can't see the K, turn the brightness control up until you can see it; you may find that adjusting the contrast control improves legibility. You may be planning to use your ZX-80 a lot. If so it may be worth considering buying a second-hand black-and-white TV to use with the ZX-80. They can be bought cheaply – in the UK, at least.

STORING PROGRAMS ON TAPE

Before you can store programs on tape you'll have to write a program. The rest of this book is all about writing programs, so the first thing to do is to read on until you've got a program to store. . .
. . . and now that you have we can start.

The twin coax cable with 3.5 mm jack plugs is used to connect the tape recorder to the ZX-80. Most cassette recorders have 3.5 mm jack sockets for MICROPHONE and EARPIECE. If you have one of those, just connect the 3.5 mm jack plugs as shown in the diagram. On other recorders DIN sockets are used. DIN plug to 3.5 mm jack plug connecting leads are available from most Hi-Fi shops. Consult the handbook to find out how to connect up the plug or plugs.

Once you have done this, connect up the cassette recorder to the ZX-80. Set the tone control (if any) to MAXIMUM. Some recorders have separate TREBLE and BASS control. In this case set TREBLE to MAXIMUM, BASS to MINIMUM. Set the volume control to MAXIMUM.

Have you got that program?

Unplug the MIC plug or DIN plug from the recorder and use the recorder's microphone to record the program's title; if you haven't recorded your voice saying the program's title, you will have trouble in finding programs when you've got a lot of them on tape. Stop the recorder. Reconnect the plug to the recorder. Get into command mode. Start recording and <u>then</u> type SAVE (E key) and NEWLINE.

The screen will go grey for about 5 seconds, then you'll see a series of horizontal streaks

ClibPDF - www.fastio.com

across the screen. After a few seconds the screen will clear and will show the program listing.

You will want to check that the program has been SAVED. Unplug the EAR plug. Rewind the tape until you get back to your voice. Now play back. You will hear your title, then you may (or may not) hear a short buzz followed by about 5 seconds of silence. You will then hear a peculiar sound, rather like a supercharged bumble bee. This is the program being played back via the loudspeaker. After a while the sound will change to a loud buzz. Rewind until you are at the beginning of the 5 second silence. Reconnect the EAR plug. Start playing back and then type LOAD (W key) NEWLINE. The screen will go grey or black and after a few seconds it may start looking grey but agitated. After a few more seconds the screen should clear to show a listing of the program.

Note: when you SAVE a program you also SAVE all the data and variables. You can avoid deleting these (RUN clears them) by using the command GO TO 1.

If it doesn't, use BREAK (SHIFT SPACE) to stop LOADING. If this doesn't work, unplug the power from the ZX-80 for a few seconds. Try repeating the procedure using different volume control settings. If this doesn't work, are you using the cassette recorder on mains power? Try again using battery power. If this fails, check the wiring to MIC and EAR.

On the ZX-80 some cassette recorders will not work properly with both jacks plugged in at the same time. You can usually tell if yours is one of these by observing that there is a signal (self oscillation) during the 5 second "silence". To SAVE on these recorders you should connect only from the computer to the cassette recorder external MIC socket. To LOAD connect only from the computer to the cassette recorder EAR socket.

Good luck!

ClibPDF - www.fastio.com

CHAPTER 3

NOT SO BASIC

ClibPDF - www.fastio.com

NOT SO BASIC

Nearly all digital computers (such as the ZX-80) talk to themselves in binary codes. We talk to each other in English. This means that either we have to learn binary codes, or we have to teach the computer English. Binary codes are about as difficult to learn as Chinese, but it can be done. Forty years ago you had to learn them if you wanted to use a digital computer at all. But why should we do the work when we've got a computer to do it for us?

So we teach the computer English. There is a snag, though. Computers, so far, aren't bright enough to learn English, something which a child of two does relatively easily. This is mainly because we have so many words and ways of saying things in English. A computer would have to learn every single one. We have to compromise half-way between binary and English. Sometimes we compromise at a low level by using assembly codes which are more like binary than English. Sometimes we compromise at a high level by using languages such as ALGOL, PL/1, PASCAL, FORTRAN, and BASIC. These languages are much closer to English than binary. The ZX-80 uses BASIC because it is easy to learn and more than adequate for most purposes. BASIC stands for Beginner's All-purpose Symbolic Instruction Code and was devised at Dartmouth College, New Hampshire in 1964 as a simple beginners' programming language. Although it was intended just for beginners, it has since become one of the most widespread and popular high level languages. This isn't very surprising when you think about it, because even scientists and engineers prefer to concentrate on science and engineering instead of trying to talk to computers in complex and difficult languages.

Like English, BASIC has a variety of dialects, depending on which computer is being used. The ZX-80 BASIC differs from other BASIC's in some respects. These differences are listed in the appendix "Summary of ZX-80 BASIC" at the end of this book.

As we said before, computers aren't that bright. We have to tell them exactly what to do by giving them a step-by-step list of instructions. This is called a program. Each instruction must be given in a clear and un-ambiguous way — the syntax must be right (in ZX-80 BASIC the computer will tell you if it thinks that the syntax is wrong before you can run the program. Some BASIC's wait until you've tried to run the program before they say "nuts!"). Even if the

ClibPDF - www.fastio.com

syntax is right, the computer will be confused if you give it a badly thought out list of instructions. Chapters 5 and 7 give some ideas for making sure that you're telling the computer to do the right things in the right order.

STATEMENTS AND COMMANDS

ZX-80 BASIC allows you to use 22 instructions or statements.

These are divided into several categories.

1. System commands:

NEW –	Clears ZX-80 ready for a new program
RUN –	Runs the current program
LIST –	Lists the current program
LOAD –	Loads a program from tape
SAVE –	Saves a program on tape

2. Control statements:

GOTO

IF . . . THEN . . .

GOSUB

STOP

RETURN

FOR . . . TO

NEXT

CONTINUE

These statements allow the programmer to control the order in which instructions are carried out. They are described in detail in chapters 7–10.

3. Input/Output statements;

PRINT –	allows computer to output data
INPUT –	allows user to enter data

I/O statements provide a means for getting data into and out from BASIC programs.

4. Assignment statement;

LET . . . = . . .

This versatile statement is used whenever an arithmetic operation is to be performed.

5. Other statements:

CLEAR –	clears the stored values of variables
CLS –	clears the screen
DIM –	sets the size of an array
REM –	indicates that what follows is a remark

ClibPDF - www.fastio.com

RANDOMISE — sets up random number generator

POKE — allows user to talk to the computer in binary code

There are some normal BASIC statements which are not included in ZX-80 BASIC. These are:

READ

DATA

RESTORE

END

ON

Nearly everything that can be done by using these statements can be done in ZX-80 BASIC by using other statements. (The END statement is not needed at all in ZX-80 BASIC).

In a BASIC program all statements or lines are preceded by a statement number or line number which labels that particular line. Line numbers can vary between 1 and 9999. The computer carries out, or executes, statements in the order in which they are numbered. It makes life easier by always displaying programs in order of increasing statement numbers, so that the order in which the program is listed is also the order in which it would be executed.

VARIABLES

All the pieces of information stored in the computer for use in a program are labelled so that the computer can keep track of them. Each piece of information, or variable, has a name.

There are two sorts of variable:

1. Numbers (integer variables) which can take any whole number value from –32768 to 32767 inclusive. Example: 142
2. Strings which can be any sequence of any characters (except ") of any length. Example: "AROUND THE WORLD IN 80 DAYS"

Integer variables have names which must always start with a letter and only contain letters and digits.

Integer variable names can be any length, so if you wish the name could be a mnemonic for the variable.

Some allowable variable names for integer variables are:

A A2 AB AB3 ANSWER A4X

Y Y8 YZ YZ9 FREDBLOGGS (N.B. no space).

AAAA Z123ABC QTOTAL

ClibPDF - www.fastio.com

Some illegal names are:

4 4 4AD	(name must start with a letter)
FRED BLOGGS	(space not allowed)
A.B	(other characters not allowed)
FRED-BLOGGS	(not a variable, but two variables, one subtracted from the other)

String variables have names of the form: letter $

The dollar sign tells you (and the computer) that the variable is a string variable. Because only one letter is allowed, this means that you can only have up to 26 string variables in a program.

Some allowable names for string variables:

A$ P$ X$

Some illegal names:

2$	(name must start with letter)
A2$	(too many characters)
AC$	(too many characters)

All this may seem a little daunting at first. The good news is that, like so many things, it's easier than it sounds.

Notes: (1) Arrays must have a name consisting of a single letter.

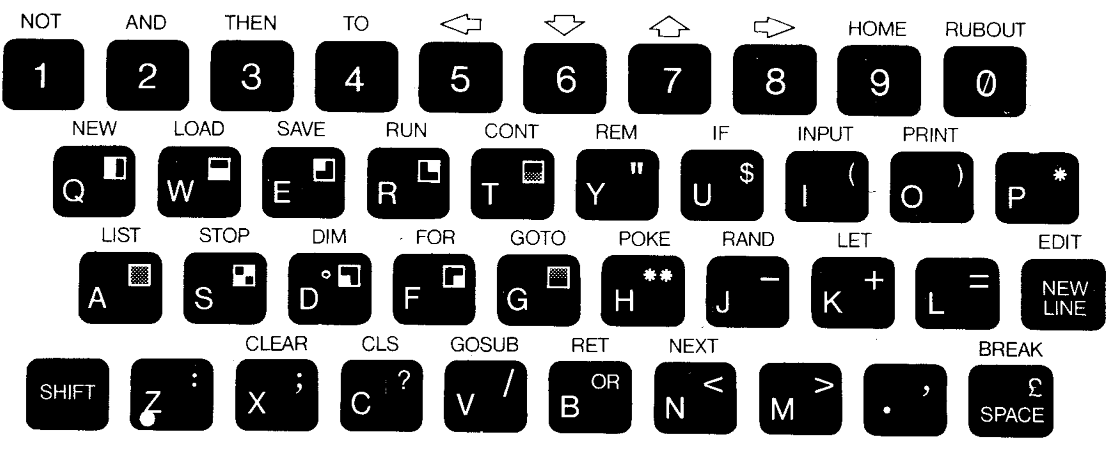
(2) Control variables (see Chapter 10) must have a name consisting of a single letter.

ClibPDF - www.fastio.com

CHAPTER 4

TALKING TO THE ZX-80

ClibPDF - www.fastio.com



The Keyboard

ClibPDF - www.fastio.com

LET'S TALK

Set up the computer as described in chapter 2 and switch on. You will see a [K] sign appear at the bottom left hand corner of the screen. This is the cursor and shows you where you are on the screen.

THE KEYBOARD

If you look at the keyboard you will see that it looks quite like a typewriter keyboard but there are some differences. There is no carriage return key (instead there's a key called NEW LINE) and the space bar is replaced by a key labelled SPACE. All the letters are upper case (capitals) so that if you use the SHIFT key you get the symbol printed on the top right hand corner of the key. You'll also notice that quite a lot of keys have labels above them – such as NEW above the Q key. These will save you a lot of time from now on.

One vital point: it is very important to distinguish between the letter O and figure 0 or Zero. In the book we will use O to mean letter O and Ø to mean zero. On the screen the computer uses a square O [□] for letter O and a hexagonal 0 [0] for zero. (The reason that we've used Ø in the book is because it is easier for printing and because all other books on programming use it).

SINGLE-KEY KEYWORD SYSTEM

Hit the Q key. Amazing! The computer writes NEW on the screen. Now hit NEWLINE. You have cleared the computer ready to accept a new program. This illustrates the single-key keyword system. Most of the words you'll have to use with ZX-80 BASIC can be typed with only one keystroke in this way. In all this may save you up to 40% of the typing you'd otherwise have to do. Some BASIC's allow you to use single key-strokes to input words like NEW, PRINT, RUN etc but only the ZX-80 prints out the word in full. This makes it much easier to keep track of what's going on in programs. You'll see how it works as you go along.

The labels above the numeral keys – NOT, AND etc – are not keywords. They are called tokens and, when needed, are obtained by using the SHIFT key. This also applies to EDIT on the NEWLINE key.

ClibPDF - www.fastio.com

Getting data in and out of the ZX-80 computer.

The PRINT statement allows you to get data out of the computer. This can be done while a program is running — or afterwards.

Type exactly: 1ØO"THIS IS A STRING"

using SHIFT Y for "

That should have come out as

1Ø PRINT "THIS IS A STRING"

The computer provided the word PRINT, as well as the space between 1Ø and PRINT. Did you notice what happened to the [K] cursor? First of all, it followed what you were typing in, appearing just after the last character typed. After PRINT it changed to a [L] sign — this showed that the computer was not expecting any more single-key keywords just then. After you typed " you may have noticed that another character appeared as well — a [S] sign. What was actually on the screen was

1Ø PRINT "[S]

This [S] is the syntax error marker and occurs if there is a syntax error in the line being typed in. In this case it came up because the line needs the closing quotation marks if it is to be correct. The [S] followed the cursor as you typed in THIS IS A STRING and vanished when you typed the second ". Now erase the " by using the RUBOUT key. The [S] reappears. Try hitting the NEWLINE key — nothing happens because the computer won't accept a line containing a syntax error.

Type " and hit the NEWLINE key.

The computer accepts the line and displays it at the top of the screen together with a [>] cursor which shows that it is the last (and in this case the first) line entered. The [>] points to the current line.

Although there is only one line entered it is a program consisting of only one instruction. Type R (RUN) and NEWLINE and the program will run — after a brief flicker on the screen the words

THIS IS A STRING

appear at the top of the screen. At the bottom you'll see Ø/1Ø which tells you that the program ran successfully with no errors.

Type any key to get back into command mode.

What the computer did was to print all the characters between the quotation marks. Any

ClibPDF - www.fastio.com

character (except for quotation marks) is legal when PRINT is used in this way.

In general this form of the PRINT statement is useful for things like titles which only have to be printed once in a program. There are other forms which allow more complicated things to be done; you'll come across these the whole way through this book.

Now let's try getting some data into the computer. This is done by using the INPUT statement, which is basically of the form:

INPUT variable name.

Hit NEW (key Q) and NEWLINE. This tells the computer that a new program is being entered.

Type in the following program, using NEWLINE at the end of each line. (If you make a mistake, use SHIFT Ø (RUBOUT) to delete the mistake; every time you hit RUBOUT the character, keyword, or token to the left of the cursor is deleted).

1Ø PRINT "ENTER YOUR STRING"	
2Ø INPUT A$	(A$ is the name of a string variable)
3Ø PRINT A$	(You can use PRINT to print out a variable)

Now run the program by typing RUN (key R) and NEWLINE.

The computer responds with

ENTER YOUR STRING

"[L]"

and waits. The "L" means that it is waiting for you to do something; in this case the quotation marks show that is is expecting you to enter a string variable.

Type in a short message or random string of letters (don't use quote marks) and press NEWLINE. The computer prints out what you have just put in, which was a string variable called A$.

Get back into command mode by pressing any key. Try running the program several times using more and more characters in your string. Keep an eye on the cursor. On the 15th line it will vanish. RUBOUT (shift Ø) — the cursor will reappear. Now carry on adding characters — the display will gradually get SMALLER as you use up the storage capacity of the processor. Now hit NEWLINE—a 4/2Ø error message appears. The 4 tells you that the ZX-80 ran out of space to put the string variable into, and the 2Ø tells you that this happened at line 2Ø of the program.

Get back into command mode.

ClibPDF - www.fastio.com

You may have wondered why the line numbers have been 1∅, 2∅, 3∅ instead of (say) 1, 2, 3.

This is because you may want to add bits to the middle of your program.

For instance, enter

```
22 PRINT "ENTER A NUMBER"
24 INPUT A
4∅ PRINT A
```

The program displayed at the top of the screen now reads

```
1∅ PRINT "ENTER YOUR STRING"
2∅ INPUT A$
22 PRINT "ENTER A NUMBER"
24 INPUT A
3∅ PRINT A$
4∅ PRINT A
```

because the computer always sorts out the lines in numerical order. By leaving gaps between line numbers you can always do this. It also makes it easy to see how you've altered the program since the original lines will have line numbers in multiples of 1∅ (unless the lines were added at the end of the program, like statement 4∅). Now try running the program again.

FIRST it asks you for a string. Enter a string.

THEN it asks you for a number. Notice that when the ZX-80 is waiting for you to enter a number it displays [L][S] and not "[L]". Enter a number.

FINALLY it prints out what you put in.

In chapter 3 the difference between string variables and integer variables is mentioned. The program has two variables in it – a string variable A$ and an integer variable A. How does the computer react when you give it a different sort of variable to the one that it's expecting?

Easy enough to find out – just run the program again. When the computer asks you for a string, give it a number. What happened? It was accepted – numbers are acceptable inside strings. Now – when it asks you for a number, type in a letter. Nothing much happened except that 2/24 appeared at the lower left hand corner of the screen. This is another error code. The 2 stands for the type of error – VARIABLE NAME NOT FOUND – because the computer knew that the letter was the name of an integer variable and you hadn't assigned a value to it. The 24 stands for the line number at which the error was found.

ClibPDF - www.fastio.com

EDITING PROGRAMS

Suppose that you want to alter something in a program that you've already entered. (This is called editing a program.)

It's easy on a ZX-80.

Get into command mode.

Look at the listing at the top of the screen. The ⊳ cursor (current line pointer) is at line 4Ø. Now see what happens when you hit SHIFT 7 (⇧) a few times — the cursor moves up line by line. SHIFT 6 (⇩) moves it down line by line. These keys allow you to select any line in your program to edit.

A further key, HOME (SHIFT 9) sets the line pointer to line Ø. Because there isn't a line Ø the cursor will vanish if you use HOME — but when you hit SHIFT 6 the cursor will jump to the next line — in this case line 1Ø — and reappear.

Select a line — line 22 for example.

Now hit SHIFT NEWLINE. Line 22 (or any other one you might have selected) appears at the bottom of the screen.

Try SHIFT 8 (⇨) — you'll see the cursor jump past PRINT. SHIFT 5 (⇦) moves the cursor left, to where it was before. Try moving the cursor right a few times — say to the position after ENTER. Now type a few A's — they are inserted into the string to the immediate left of the cursor. A few RUBOUTS and you can erase them again.

This editing facility is very useful if you want to alter individual characters within lines. However, if you want to delete a whole line it would be rather tedious to go through the procedure mentioned above. There is a simpler way to delete lines, though.

Type the line number, then NEWLINE — and you'll see the line vanish from the listing at the top of the screen. The cursor ⊳ will also vanish. It will not reappear until you type a new line, use the control keys ⇧ and ⇩, or use the LIST command.

THE LIST COMMAND

As you get more experienced with the ZX-80 you'll want to write programs longer than the 24 lines which will fit onto the screen. This could pose a problem — or could it? We can find out by writing a really long program.

Type NEW then NEWLINE.

ClibPDF - www.fastio.com

Enter the following program (which prints a large number of blank lines).

```
10  PRINT
20  PRINT
30  PRINT
40  PRINT
```

. . . and so on to . . .

```
230  PRINT    (keep an eye on the listing in
              the upper part of the screen)
240  PRINT
250  PRINT
260  PRINT
```

We seem to have lost the first few lines. Or have we?

Hit LIST (key A) followed by NEWLINE – the program is now displayed from line 1Ø down.

Now try LIST 2ØØ – the program is displayed from line 2ØØ. You can list the program starting from any line you want in this way. Notice that the current line cursor is altered to the line you selected.

Try using the up/down cursor control keys – go on hitting the ⇧ key and see what happens.

In this way you can display any part of the program you want.

By now you will have realised that the ZX-80 has some powerful (and very convenient) facilities to help you write programs.

From now on we can concentrate on how to write programs which actually do something – like solving arithmetic problems, for instance.

ClibPDF - www.fastio.com

CHAPTER 5

SO YOU'VE GOT PROBLEMS

ClibPDF - www.fastio.com

SO YOU'VE GOT PROBLEMS

People are very good at disentangling confused instructions and solving complex problems. A computer isn't – all it can do is to follow a list of instructions and carry out the instructions as it comes to them.

As an example, take the following instructions which a mother might give to her child:

"Could you run down to the shop and buy some bread? Take 50p which is in my purse on the kitchen table. And for goodness sake get dressed!"

It all seems quite clear; the child knows where to go, what to buy and where the money is.

A computer controlled robot would take the first instruction:

run down the shop, and do that.

The second instruction: buy some bread. It can't do that (no money) and at this stage would probably just stop, baffled.

The third instruction: take 50p from the purse, it can't do that.

The fourth instruction: go to the kitchen table.

It returns home and goes to the kitchen table.

Then the fifth instruction: get dressed.

It gets dressed.

A child would instinctively carry out the instructions in the following order:

1. Get dressed
2. Go to the kitchen table
3. Take 50p from the purse
4. Go to the shop
5. Buy some bread

He or she would come back home!

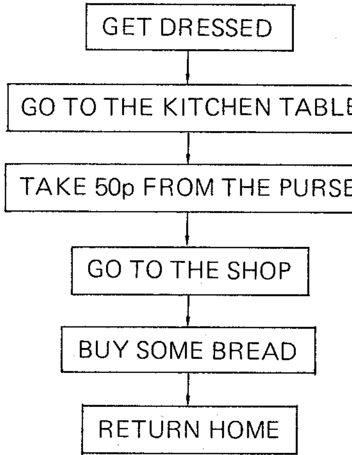
The mother in this example missed out one vital instruction ("come home") as well as putting the instructions in an illogical order. Children use commonsense to interpret complex instructions, but computers can't do this. You have to do all the thinking in advance when you use a computer. So remember:

THINK STRAIGHT!

One way of making sure that you are thinking straight is by drawing a flow diagram before you start to do any actual programming. This helps you get things in the right order to begin with.

ClibPDF - www.fastio.com

The flow diagram for the example above would look like this:



Each box contains an instruction, and the arrows show the order in which the instructions are executed. Well, that looks pretty simple and neat. Now that the flow diagram is drawn on paper we can check it by going through the boxes one by one and asking ourselves: Can the child (or the computer) do this?

Starting with the first box (get dressed) we might think that it's pretty basic. On the other hand, the child may not be able to find his clothes because his naughty sister has thrown them out of the bedroom window. Perhaps we should add an instruction (find clothes) at the beginning of the program.

Go to the kitchen table: well, that looks OK.

Take 50p from purse: what happens if the purse is empty?

Go to the shop: does the child know where the shop is?

Buy some bread: what sort?

Return home: nothing much wrong with that!

Maybe the flow diagram wasn't as good as we thought. Even simple tasks can turn out to be more complicated than we think.

Most of the problems which we've spotted are due to a lack of information and this can be coped with by adding extra instructions at the beginning of the program. These could include:

1. Ask where the shop is
2. Find clothes

When you are writing programs for the ZX-80 it is particularly important to give the computer all the information it needs before it has to carry out a task. For instance, if you ask it to PRINT a variable the computer must know the value of the variable.

Now for some arithmetic.

ClibPDF - www.fastio.com

CHAPTER 6

FINDING THE ANSWERS

ClibPDF - www.fastio.com

FINDING THE ANSWERS

Up to now you've just been getting used to the feel of the computer. Now we'll actually use the computer to do a few sums.

THE LET STATEMENT

Nearly all arithmetic operations are done by using the LET statement. It is of the form

LET variable = expression.

In this case the variable (which can have any name you assign to it) is what you want to find out. The expression describes how you want to define the variable.

For instance

LET A = 2+4

variable expression

means "add 2 to 4 and set A equal to the result (6)".

You can use other variables:

LET A = 2 + B

or

LET A = B + C

One condition: the computer must already know what the values of B and C are before it comes to the LET statement.

The equal sign (=) isn't used in quite the same way it is in ordinary arithmetic or algebra. For instance:

LET J = J + 1

Obviously J isn't equal to J + 1. What the LET statement means is:

"Take the existing value of J, add one to it and then set J to this value."

The + sign is called an <u>operator</u> and defines the operation you wish to be performed. The other arithmetic operators are:

–	(minus)
*	(multiply – there is no x sign to avoid confusion with letter x)
/	(divide)
**	(raise to the power)

For the moment we'll leave / and ** out of it.

Here is a program for multiplying two numbers together

```
1Ø PRINT "MULTIPLICATION"
2Ø PRINT "ENTER FIRST NUMBER"
```

ClibPDF - www.fastio.com

```
3Ø INPUT A
4Ø PRINT "ENTER SECOND NUMBER"
5Ø INPUT B
6Ø LET C = A *B
7Ø PRINT "THE ANSWER IS", C
```

Notice line 7Ø. It's a PRINT statement with a string between inverted commas, but it's got a C added to it. This form allows you to economize on PRINT statements.

Enter the program and try running it, using two quite small numbers — such as 17 and 25.

The computer will come up with the answer:

THE ANSWER IS 425

Run the program a few more times, increasing the size of the input variables. At about 255 x 129 or just a bit larger you will get another error message — 6/6Ø. The 6 is the code for arithmetic overflow because the answer is larger than 32767, which is the largest number the computer can hold as an integer variable.

Now try editing the program so that it does addition instead — just change the * to a + in line 6Ø. You can also try subtraction and check that you get negative numbers when A is less than B.

All fairly straightforward so far.

Now let's try division. Alter line 6Ø to

6Ø LET C = A/B

Run the program for A (first number) = 24
B (second number) = 12

The computer will come up with:

THE ANSWER IS 2

Run the program again, this time for A=18 and B=12

THE ANSWER IS 1

??!!

Shouldn't that be 1.5? Well, yes, but the ZX-80 uses integer arithmetic — only whole numbers can be expressed. What the computer does is to do a division normally and the truncate the result towards ZERO. As examples:

ANSWER		
2.58	would become	2
–Ø.Ø1	" "	Ø
212.1	" "	212
–5941.98	" "	–5941

This means that you may have to be careful when you use division — it's always fairly accurate when you're dividing a large number by a small number, but may be less accurate when the two numbers are closer together in size.

ClibPDF - www.fastio.com

Luckily there are ways of getting round this problem. For instance, here is a program which will give quotients to 3 decimal places using integer arithmetic only.

```
10  PRINT "DIVISION PROGRAM"
20  PRINT "DIVIDEND = ?"
30  INPUT X
40  PRINT "DIVISOR = ?"
50  INPUT Y
60  LET Z = X/Y                  Divides X by Y
70  LET R1 = X – Z * Y           Calculates
                                 remainder
80  LET D1 = 10 * R1/Y           Divides 10 * re-
                                 mainder by Y to
                                 give 1st decimal
90  LET R2 = 10 * R1–D1*Y        Calculates 2nd
                                 remainder –
100 LET D2 = 10*R2/Y             Divides to give
                                 2nd decimal
                                 place
110 LET R3 = 10* R2–D2 * Y       Calculates 3rd
                                 remainder
120 LET D3 = 10 * R3/Y           Last decimal
                                 place
130 PRINT "THE ANSWER IS ";Z;".";
D1;D2; D3
```

This program does a long division in exactly the same way as we would do it on paper.

Try running it!

Notice the PRINT list in line 130 – it contains several literal strings and variables separated by semicolons. The semicolons tell the ZX-80 that each thing to be printed must be printed immediately after the preceding item, without spaces between them.

Also notice the LET statements from line 70 on. There's no fiddle – you can use more than one operator per statement. There is a snag, though.

Operations are not necessarily carried out in order from left-to-right.

The order in which they are carried out depends on what sort of operations are present in the statement.

Here is a list of priorities for arithmetic operations:

First	A**B	(A to the power B)
Second	–A	(Negation, or multiplication by –1)
Third	A*B	
Fourth	A/B	
Fifth	A+B or A–B	

ClibPDF - www.fastio.com

Now for some experiments

```
1Ø   REM TEST PROGRAM    (REM is a key-
                          word – key Y)
2Ø   PRINT "ENTER A"
3Ø   INPUT A
4Ø   PRINT "ENTER B"
5Ø   INPUT B
6Ø   PRINT "ENTER C"
7Ø   INPUT C
8Ø   LET Z = A + B * C
9Ø   PRINT Z
```

Try running this with A = 1, B = 2, C = 3 but before you start, use the priority rules to predict what answer the ZX-80 will give: 7 or 9?

O.K. – run it now.

The answer was 7. First the computer multiplied 2 by 3 to get 6, then it added 1 to get the final answer.

Try a few other combinations by editing statement 8Ø – like

A*B/C

A/B*C (watch out for truncation!)

A**B (A to the power B – try using small numbers with C = Ø)

–A**B

A*–B

Here's a golden rule: when in doubt either use more than one LET statement and build up that way, or use brackets.

Brackets make life much easier when it comes to complex arithmetic operations.

Take example Z = A/B*C

The normal sequence would be for the computer to evaluate B times C and then divide A by the result.

If we put brackets round A/B thus:

8Ø LET Z = (A/B)*C

then the computer will carry out the operation within the brackets first, even though it has a lower priority.

A more subtle example is:

Z = A*(B/C)

and

Z = A*B/C

At first sight you would think that these would give the same answer.

Well, they do – sometimes!

ClibPDF - www.fastio.com

Try them both using A = 100, B = 25, C = 5 (the answer is 5ØØ).

Now try it both ways using A = 100, B = 3, C = 5 (the answer should be 6Ø).

What happened when you used the brackets?

Think about it.

The phantom truncator has struck again!

The computer evaluated 3/5 first and truncated it to ZERO – then it multiplied 1ØØ by ZERO and naturally got ZERO as a result.

All this emphasizes that you have to be very much on the alert when you use integer arithmetic for multiplication and division. Multiplications can cause arithmetic overflow problems (which will cause the program to stop), whilst small numbers used in division may give rise to funny answers which don't give error codes.

If you think that you're going to run into truncation then do multiplications first – on the other hand it may be better to do divisions first to avoid overflow.

That's probably enough arithmetic for now. Let's move on to more interesting things.

ClibPDF - www.fastio.com

CHAPTER 7

DECISIONS, DECISIONS

ClibPDF - www.fastio.com

DECISIONS, DECISIONS

So far we have only considered problems which can be solved by carrying out a list of instructions, starting at the beginning of the list and working steadily down the list until the last instruction is carried out.

The next four chapters deal with ways in which you can make the ZX-80 (or any computer using BASIC) carry out much more complex programs which can perform many tedious tasks.

In chapter 5 we talked briefly of the use of flowcharts for checking that the program steps were in a logical sequence. This can be useful even for simple programs. When it comes to complex programs flowcharts are vital.

Let's consider a problem we face every day — getting up in the morning. There is a flowchart for getting up on the next page.

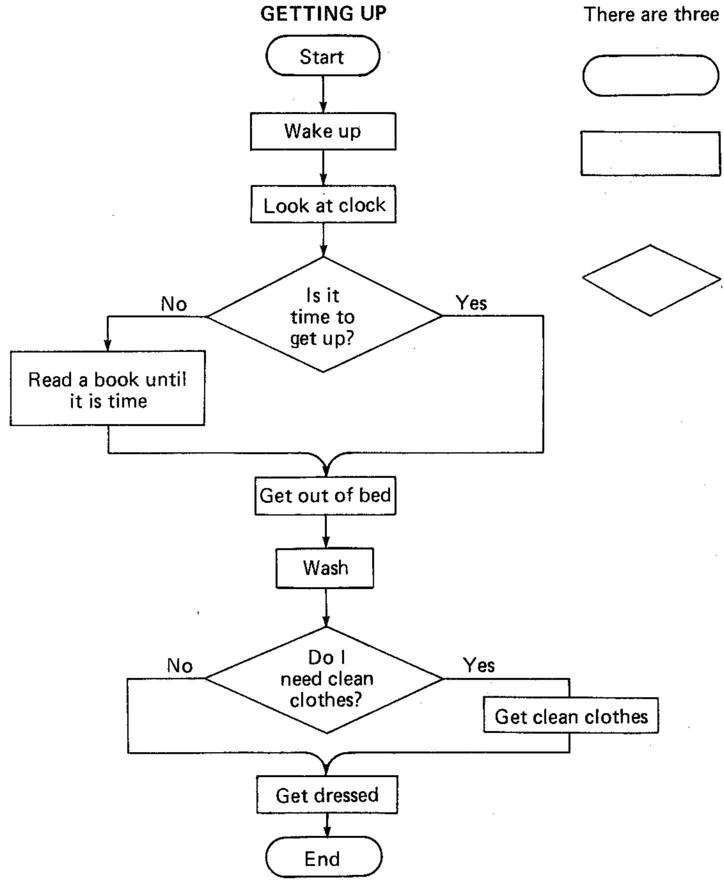
The program is said to branch at each decision diamond. As you can see, one feature of branches is that they allow some parts of the program to be skipped if they are not neccessary.

Programs can get difficult to understand if there are a lot of branches, and this is where flow diagrams help.

It is a good idea to keep a pad or notebook handy when you are writing programs and an even better one to draw a flow diagram of your program before you start writing it!

Turn to the next chapter to find out how the ZX-80 can take decisions.

ClibPDF - www.fastio.com



There are three sorts of symbols (or boxes).

Begin or end.

Processing block —
this encloses anything which
can be done without making any decision.

Decision diamond —
encloses a decision with branches
to either side depending on
whether the decision is yes
or no.

ClibPDF - www.fastio.com

CHAPTER 8

BRANCHING OUT

ClibPDF - www.fastio.com

BRANCHING OUT

Before we go on to real decisions lets have a look at the GO TO statement.

This is of the form

GO TO n where n is normally a line number. It can be a variable, though only the ZX-80 BASIC has the power to do this.

When the ZX-80 comes to a GO TO statement it jumps straight to statement number n and executes that statement. If n is a variable – X, say – the computer will look at the value of X and jump to the statement with that number – assuming that there is a statement with that number in your program.

A further refinement of the GO TO statement is that n can be any integer expression such as A/1Ø. Here again, this is possible with the ZX-80.

Here is a program which illustrates the use of GO TO.

```
1Ø    PRINT "THIS IS STATEMENT 1Ø"
2Ø    GO TO 4Ø
3Ø    PRINT "THIS IS STATEMENT 3Ø"
4Ø    PRINT "END OF PROGRAM"
```

Now run the program.

The computer responded with

THIS IS STATEMENT 1Ø

END OF PROGRAM

It skipped statement 3Ø. "That's not much use", you must be saying, "what's the point of including a perfectly good statement which will never be executed?"

OK

Get back into command mode.

Now, type GO TO 3Ø, and then NEWLINE.

The ZX-80 came back with

THIS IS STATEMENT 3Ø

END OF PROGRAM

This demonstrates two uses of GO TO – first to jump unconditionally to an arbitary statement (in this case statement 4Ø); second as a command which causes the computer to jump to the desired statement (in this case 3Ø) and start executing the program from there.

ClibPDF - www.fastio.com

THE IF STATEMENT

This is the most powerful control statement in BASIC. It allows the user to incorporate decision-making into his programs.

Here is a program for calculating square roots approximately. It does this by multiplying, a number by itself, starting with Ø and comparing the result with the number whose square root is to be found. If the product is less than the number to be rooted it increases the number by 1 and tries again.

```
10   PRINT "SQUARE ROOT ROUTINE"
20   PRINT "ENTER THE NUMBER TO BE
     ROOTED"
30   INPUT X
40   LET J = Ø
50   LET K = J*J
60   LET D = X-K
70   IF D = Ø THEN GO      (Use SHIFT 3 to
     TO 110                get THEN)
80   IF D < Ø THEN GO      (< is SHIFT N)
     TO 130
90   LET J = J+1
100  GO TO 50
110  PRINT "THE ROOT IS" ; J
120  GO TO 140
130  PRINT " THE ROOT LIES BETWEEN ";
     (J-1); " AND "; J
140  STOP
```

See the diagram on the next page.

The form of the IF statement is:

IF (CONDITION TO BE MET) THEN (DO THIS)

7Ø IF D = Ø THEN GO TO 11Ø

In line 7Ø the condition to be met was <u>D = Ø</u> and if this is so then J is <u>exactly</u> the square root of X, so we want to print THE ROOT IS J. In this case (DO THIS) is GO TO 11Ø which is the number of the PRINT statement we want.

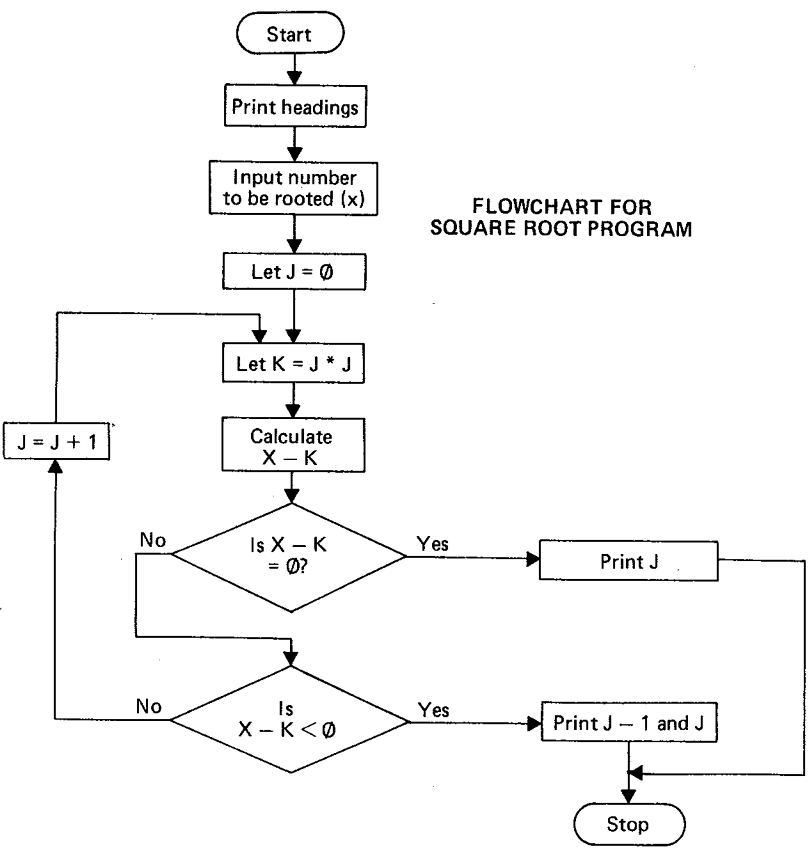
In most BASIC's the only thing you can do in a IF statement is to GO TO a line number. The ZX-80 lets you do other things as well.

In fact it lets you do almost anything!

Some examples:

```
70   IF D = Ø THEN PRINT "NUTS"
70   IF D = Ø THEN INPUT G
70   IF D = Ø THEN LET A = 10
```


ClibPDF - www.fastio.com



ClibPDF - www.fastio.com

Any keyword can follow THEN in an IF statement – even things like LIST or RUN. The results can be rather peculiar if you use commands such as LIST. The examples listed above can be useful, though.

The condition to be met is (in its simplest version) of the form:

(Expression) (relational operator) (expression)
This sounds rather a mouthful!

A relational operator can be:

= equal to

< less than

> greater than

Examples

A = 2 A equal to 2

A<1Ø A less than 1Ø

A+B>C+D A+B greater than C+D

In these cases, A, 2, 1Ø, A+B and C+D are the expressions.

The expressions need not be a simple integer or integer variable.

The statement IF A=Z – (Z/B) * B THEN GO TO 1Ø (Z – (Z/B) * B: expression)

is quite valid, and so is IF A+B > C+D THEN GO TO 2Ø

The condition to be met can be even more complex because logical operators can be used.

For example

1Ø IF NOT A=2 THEN GO TO 1ØØ (NOT: logical operator)

(use SHIFT 1 for NOT)

This NOT operator negates the succeeding condition, so that the ZX-80 will jump to 1ØØ if A does not equal 2.

This program illustrates the use of the AND operator.

```
1Ø    INPUT A
2Ø    INPUT B
3Ø    INPUT C
4Ø    IF A=1 AND B=1 AND C=1 THEN PRINT
      "OK"
```

(AND: Another logical operator)

(Use SHIFT 2 for AND)

The condition to be met is that ALL three variables must be 1; if this is so it will print OK. You can chain together as many conditions as you like in this way. Try running this program

ClibPDF - www.fastio.com

and see if you can make it print OK in any way other than by entering three 1's.

The other logical operator is the OR operator.

Edit the program by changing the AND's to OR's (OR is SHIFT B).

When you run the program now you will find that it prints OK as long as one of the numbers entered is a 1.

You can use brackets to group things together. Consider the following statements:

(a) 40 IF A=1 OR B=1 AND C=1 THEN PRINT "OK"

(b) 40 IF (A=1 OR B=1) AND C=1 THEN PRINT "OK"

If you remember, in chapter 6 we said that some operators had priority over others — multiplications were carried out before divisions or additions, for instance. The same applies to logical operators.

Their priority (in descending order) is

NOT

AND

OR

Thus, in (a) B=1 AND C=1 is taken first, so that the computer will print OK if B AND C are 1 but will also print OK if A=1. The brackets in (b) work in the same way as they did for arithmetic operators. Thus the ZX-80 will print OK if C=1 and either B=1 or A=1.

The operators NOT, AND and OR also allow you to produce conditional expressions. For example, if you require

X=3 if A>B

X=Q+R if A=B

X=P if A<B

The obvious way is:

```
110  IF A>B THEN LET X=3
120  IF A=B THEN LET X=Q+R
130  IF A<B THEN LET X=P
```

However you could instead write:

```
110  LET X=A>B AND 3 or A=B AND Q+R
     OR A<B AND P
```

To round off this chapter on branching, here is a program which throws a die. The program uses a useful facility, the function

RND(X)

This generates a random number in the range 1

ClibPDF - www.fastio.com

to X. It occurs in line 12Ø of the program — in this case X = 6

```
10  PRINT "DIE THROWING"
2Ø  LET A$ = "■. . . ■"          ■ is SHIFT A
3Ø  LET B$ = ". . ■ . ."
4Ø  LET C$ = "■. . . ."
5Ø  LET D$ = ". . . . ■"
6Ø  LET E$ = ". . . . . "
12Ø LET X = RND (6)
13Ø PRINT "YOU THREW . . ."
14Ø IF X = 1 THEN GO TO 2ØØ
15Ø IF X = 2 THEN GO TO 3ØØ
16Ø IF X = 3 THEN GO TO 4ØØ
17Ø IF X = 4 THEN GO TO 5ØØ
18Ø IF X = 5 THEN GO TO 6ØØ
19Ø IF X = 6 THEN GO TO 7ØØ
195 GO TO 1ØØØ

2ØØ PRINT E$
2Ø5 PRINT E$
21Ø PRINT  B$
215 PRINT E$
22Ø PRINT E$
23Ø GO TO 1ØØØ
3ØØ PRINT C$
3Ø5 PRINT E$
31Ø PRINT E$
315 PRINT E$
32Ø PRINT D$
33Ø GO TO 1ØØØ

4ØØ PRINT D$
4Ø5 PRINT E$
41Ø PRINT B$
415 PRINT E$
42Ø PRINT C$
43Ø GO TO 1ØØØ

5ØØ PRINT A$
5Ø5 PRINT E$
51Ø PRINT E$
515 PRINT E$
52Ø PRINT A$
53Ø GO TO 1ØØØ

6ØØ PRINT A$
6Ø5 PRINT E$
```

ClibPDF - www.fastio.com

```
61Ø  PRINT B$
615  PRINT E$
62Ø  PRINT A$
63Ø  GO TO 1ØØØ

7ØØ  PRINT A$
7Ø5  PRINT E$
71Ø  PRINT A$
715  PRINT E$
72Ø  PRINT A$
1ØØØ STOP
```

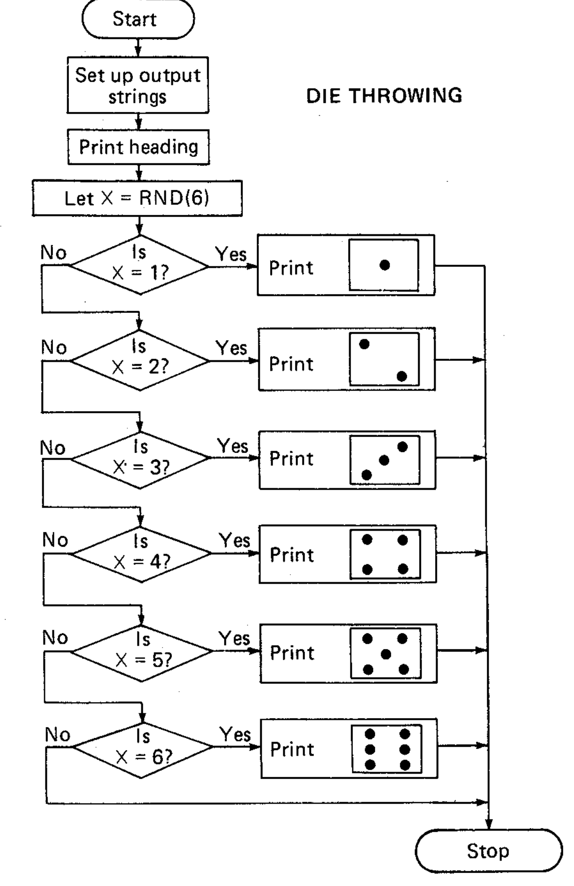
It's a bit of a strain to enter this program but it's quite fun to run.

Note that lines 14Ø to 19Ø could have been replaced by

```
14Ø  GO TO (X+1) * 1ØØ
```

If you want to make the program restart itself try editing the program:

```
1ØØØ PRINT "HIT NEWLINE TO THROW
     AGAIN"
11ØØ INPUT X$
12ØØ CLS                (CLS is on key C)
13ØØ IF X$ = "" THEN GO TO 12Ø
```



ClibPDF - www.fastio.com

Two points: CLS clears the screen. If you didn't put this in the screen would fill up after 3 goes and the program would stop, giving an error message such as 5/3Ø5 (run out of screen at line 3Ø5).

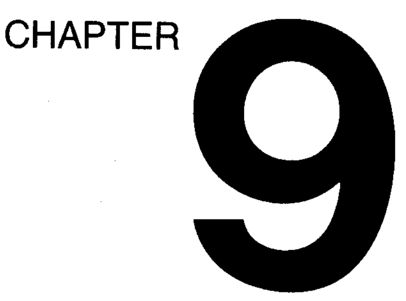
Secondly, the use of a INPUT statement to halt program execution is a useful trick. The string variable X$ is used in statement 13ØØ when it is tested to see that only NEWLINE has been pressed; any other entry will stop the program (There is nothing between the quotation marks in 13ØØ.)

The reason that there is a IF statement at 13ØØ rather than a GO TO statement is that, while the ZX-80 is waiting for a string variable to be entered, the BREAK key (SHIFT SPACE) doesn't work. As a result it is difficult to get out of the program except by entering a very long string variable, a variable so long, in fact, that it makes the computer run out of storage space. This is very tedious, so it's better to use the IF statement. Then if you enter any character the program will stop.

(If you do ever get into a situation when nothing you do seems to have any useful effect on the computer, try the BREAK key first. If this has no effect, give in and switch the ZX-80 off for a few seconds. You lose the current program, but this may be inevitable if you really have got yourself into a fix.)

ClibPDF - www.fastio.com

CHAPTER



ITER – WHAT?

ClibPDF - www.fastio.com

ITER – WHAT?

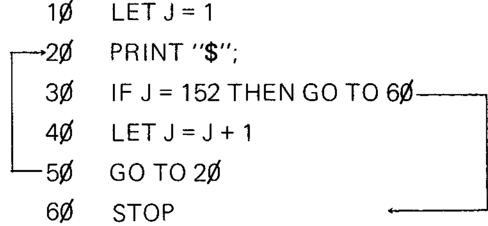
Very often we come up against problems which involve repeated operations. The square rooting program in chapter 8 was one example of such a problem, and the program was written so that the computer jumped back to line 5Ø, if the test conditions in statements 7Ø or 8Ø were not met, and tried again – and again – until the conditions were met.

Programs of this sort are called iterative. By using iterative techniques it's possible to get the computer to do a lot of work with a very short program.

It's easy to make a program iterative – all you have to do is to put a GO TO statement at the end of the program to jump the computer back to the first statement (or any other suitable statement). One snag is that such iterative programs can get stuck in endless loops.

One way to avoid this is to allow the computer to go round only a set number of times. One way of doing this would be to use IF and GO TO statements.

Consider the following program which prints **$** 152 times.



First the program sets J (the loop control variable) to 1. Then it prints **$**. Then it tests to see if J = 152. It isn't and so the computer goes on to line 4Ø which increments the value of J by 1 to 2. The computer passes on to 5Ø which tells it to jump back to 2Ø. This process goes on until J = 152 when the IF statement is satisfied and the computer jumps out of the loop to line 6Ø.

The arrows on the program help you to see where the program is going; for short programs this is quicker than drawing a flow diagram and does make it easier to see what the program is doing.

The program took 5 lines (if you don't count the STOP statement, which doesn't actually do anything – it just serves as a target for the GO TO in line 3Ø.

ClibPDF - www.fastio.com

Loops are so useful that you'd have thought that there was an easier way of writing programs using them. Well there is. Turn to chapter 10.

ClibPDF - www.fastio.com

LOOPING THE LOOP

Because loops are so useful some special statements have been devised to make it easy to use them.

Program for printing $:

```
1Ø   FOR J = 1 TO 152      (use SHIFT 4 for TO)
2Ø   PRINT "$";
6Ø   NEXT J
```

In chapter 9 it took 5 lines of program to get the same result using IF and GO TO statements.

Note the ; after the PRINT statement. This tells the computer to print each $ immediately after its predecessor. Try running this program. It's a pity that one can't have a dollar bill for each $ sign printed.

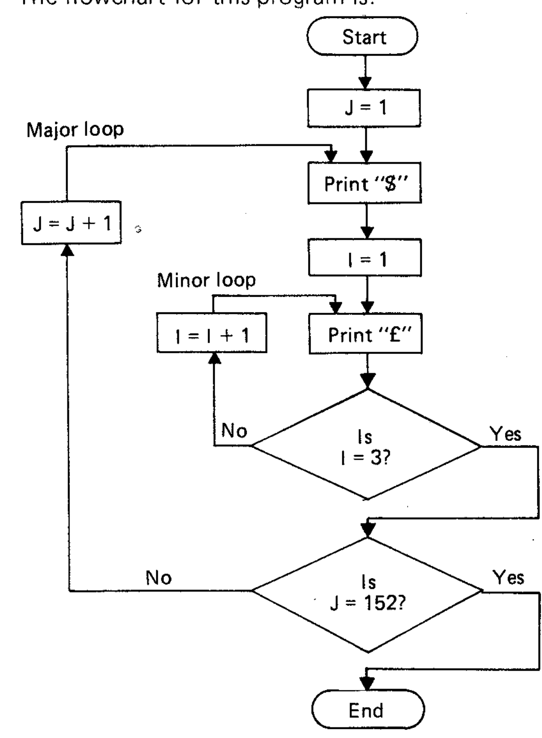
In BASIC these loops are normally called FOR loops. They are also widely known as DO loops (because other languages, such as FORTRAN, use DO instead of FOR).

There is nothing to stop us from putting a loop inside another loop — or putting several loops in:

Add the following lines to the program:

```
3Ø   FOR I = 1 TO 3
4Ø   PRINT "£"
5Ø   NEXT I
```

The flowchart for this program is:



ClibPDF - www.fastio.com

You can jump out of a loop at any time. If you do this the loop control variable will remain at whatever value if had got to when you jumped out. In our case the loop control variables were J for the major loop and I for the minor loop.

Try jumping out of the major loop at J = 1ØØ add:

15   IF J = 1ØØ THEN GO TO 1ØØØ

1ØØØ STOP

Run this. You'll see that you get fewer $'s and £'s. Get into command mode.

Now type PRINT J NEWLINE

The computer responds with 1ØØ

This shows two things

1. J was 1ØØ when the program jumped out of the loop and stopped.
2. You can use PRINT to find out what state the control variables (or other variable) were in when the program stopped. This is especially useful when you are debugging programs.

If the loop is completed successfully for all values of J up to 152 (in this case) the control variable will be left at 153. This is because the NEXT J statement not only tests the value of J but also increments J by 1. Some BASIC's allow you to increment the control variable by other amounts, but this isn't possible on the ZX-80 (It's not a very useful feature anyway.)

All this is quite straightforward.

FOR J = N to M

control variable · starting value   finishing value

defines what variable you want to use, what value it should start at and at what value of control variable you want to leave the loop.

NEXT J reminds the computer that it is in a loop and tells it what variable it should be testing and (in fact) where it should jump back to if the loop isn't finished.

You can start at any value you like — try starting at J = 1ØØ. The computer will go round the loop M — N + 1 times.

DON'T EVER jump into a loop unless you have just jumped out of it — if you do the computer will skip the FOR statement and will get very confused because it won't know what the control variable is, what it should start at and what it should finish at. It will do the best it can but it will probably stop when it gets to the NEXT statement. The error message will be of the form

ClibPDF - www.fastio.com

1/LINE No. where 1 means "no FOR statement to match this NEXT statement", or 2/LINE No. where 2 means "variable name not found".

You can try this by adding to the program:

5 GO TO 2Ø

and running, and then adding

4 LET J = 4

and running (you must say RUN, not GO TO 4).

This will jump the ZX-80 into the major loop and it will stop when it gets to statement 6Ø (NEXT J) giving the error message 2/6Ø or 1/6Ø.

In chapter 6 there was a program to divide one number by another to give a quotient to 3 decimal places.

Here is a program to give any number of decimal places. Statement 1Ø is a remark. The computer disregards REM statements when it executes programs. It's often useful to put comments and remarks into programs so that you can remind yourself what the program does when you look at it, often months later!

```
1Ø   REM HIGH PRECISION DIVISION
2Ø   PRINT "HOW MANY DECIMAL
     PLACES?"
3Ø   INPUT D
4Ø   PRINT "DIVIDEND?"
5Ø   INPUT R
6Ø   PRINT "DIVISOR?"
7Ø   INPUT Y
8Ø   LET Z = R/Y
9Ø   LET R = R – Z*Y
1ØØ  PRINT "QUOTIENT IS "; Z; ".";
11Ø  FOR J = 1 TO D          (11Ø to 15Ø evalu-
12Ø  LET Z = 1Ø*R/Y          ate successive
13Ø  LET R = 1Ø*R–Z*Y        decimal places un-
14Ø  PRINT Z;                til the program has
15Ø  NEXT J                  produced D decimal
                             places)
```

Try running this, first of all with dividend = 1, divisor = 3, D = 1ØØ (say).

<u>Note line 11Ø</u>

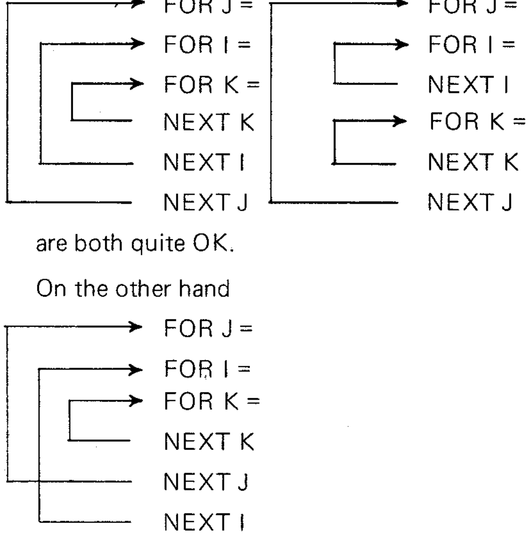
11Ø FOR J = 1 to D

The starting and finishing values can be variables. This means that you can control the starting value and finishing value of the loop variable as you wish. The starting and finishing values can also be <u>expressions</u> of the form D/2, (D+1)*5 or indeed any other arithmetic expression.

ClibPDF - www.fastio.com

11Ø FOR J = D*1Ø TO D*2ØØ would be quite legal. However, the name of the control variable must be a single letter.

If you do decide to use several FOR loops, one inside the other, be very careful in the way you go about it.



are both quite OK.

On the other hand

is definitely not all right and could cause major problems. Another good reason for using flowcharts!

One final point: because the control variable is effectively tested and incremented by the NEXT statement the main part of the loop will always be executed at least once regardless of the value of the control variable even if you have jumped into the loop.

SUBROUTINES

A subroutine is a sub-program which may be used once or many times by the program (or main program).

Example:

```
  1Ø FOR J = 1 TO 1Ø
  2Ø GO SUB 1ØØØ        (GO SUB is on key V)
  3Ø NEXT J
  4Ø PRINT "END"
 9ØØ GO TO 12ØØ
1ØØØ PRINT "SUBROUTINE EXECUTED"
11ØØ RETURN             (RETURN is on key B)
12ØØ STOP
```

Statement 2Ø tells the computer to GO TO the subroutine at line 1ØØØ.

ClibPDF - www.fastio.com

This prints:

SUBROUTINE EXECUTED

Statement 11ØØ tells the computer that the subroutine is finished and that it should return to the main program. The computer then jumps to the line immediately following the GO SUB statement and executes that (in this case: NEXT J).

An example of the use of subroutines is given below: The Chinese Ring Puzzle.

As you will see, one subroutine can call another, or even call itself (this is called recursion).

THE CHINESE RINGS PUZZLE

This is a program which will say what moves are required to remove N rings from the T-shaped loop.

The mechanics of this wire puzzle are not important — roughly speaking one manipulates the rings until they all come off the loop.

For an arbitrary number of rings the rules are as follows:

1. Each ring can be either on the loop or off it.
2. Only one ring may be moved (from on to off or vice versa) at a time.
3. The first ring may be moved at any time.
4. The ith ring (i>1) may be moved if and only if:
   (a) All the rings numbered i-2 or lower are off.
   (b) Ring i-1 is on.

. . . the rings higher up the loop (number >i) may be in any state, so:

To remove the first i rings:

1. Remove the first i-2 rings
2. Remove the ith ring
3. Replace the first i-2 rings
4. Remove the first i-1 rings

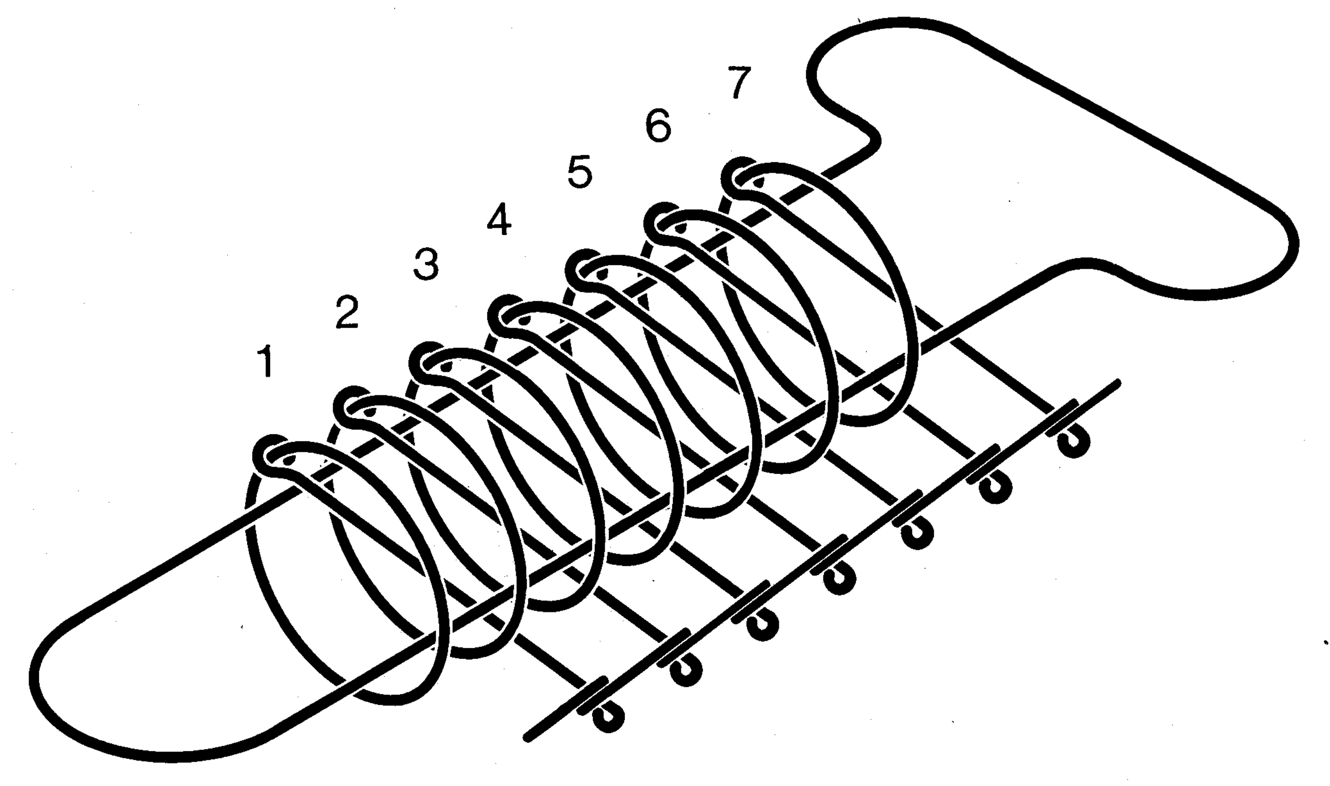
To replace the first i rings:

1. Replace the first i-1 rings
2. Remove the first i-2 rings
3. Replace the ith ring
4. Replace the first 1-2 rings

CHINESE RINGS (Recursive procedure)

```
1Ø    INPUT N
2Ø    GO SUB 1ØØ
3Ø    STOP
```

ClibPDF - www.fastio.com



Chinese Ring Puzzle

ClibPDF - www.fastio.com

```
100  IF N<1 THEN RETURN
120  LET N = N -2
130  GO SUB 100
140  PRINT N+2; " OFF",
150  GO SUB 500
160  LET N = N+1
170  GO SUB 100
180  LET N = N+1
190  RETURN

500  IF N<1 THEN RETURN
520  LET N = N-1
530  GO SUB 500
540  LET N = N-1
550  GO SUB 100
560  PRINT N + 2; " ON",
570  GO SUB 500
575  LET N = N+2
580  RETURN
```

Your output for the case N = 4 should look like:

```
2 OFF   1 OFF   4 OFF   1 ON
2 ON    1 OFF   3 OFF   1 ON
2 OFF   1 OFF
```

The GO SUB statement can be used in conjunction with a variable, e.g.

```
10   GO SUB G
```

whereupon the computer will jump to the subroutine at line whose number is the value of G (if there is such a line).

As before, an expression can also be used.

In general it is not a very good idea to use variables or expressions in this way because the program may alter the variable in a way you hadn't foreseen and thus cause problems.

ClibPDF - www.fastio.com

CHAPTER 11

HOW TO PRINT

ClibPDF - www.fastio.com

HOW TO PRINT

Up to now we have used the PRINT statement as and when it was needed to print out headings and variables. In fact PRINT is a very versatile statement indeed.

The general form of the PRINT statement is

PRINT expression ch expression ch
expression ch . . . and so on.

ch stands for control character and this can be , or ; or nothing at all (at the end of the statement only).

Expressions can be literal strings in quotes, e.g.

"THIS IS A LITERAL STRING"

or string variables such as A$

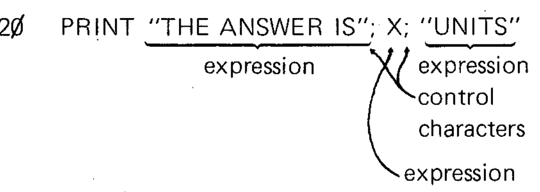
Alternatively they can be integer variables or arithmetic expressions, e.g.

A2 or B*2/C

The object of using control characters is to be able to control the spacing of the line to be printed. We've already come across the use of ; in PRINT statements.

The program

```
10     LET X = 4
```



Would print:

THE ANSWER IS4UNITS

The semi-colon (;) makes the computer print out the expressions without any spaces between them. If you want spaces you have to include then in the literal strings to be printed.

The comma (,) is used as a tab. Each display line of 32 characters on the screen is divided into 4 fields each of which is 8 characters in length. Each time the computer comes across a comma in a PRINT statement, it starts printing the next expression at the beginning of the next available field.

The effect of this is to provide a display on the screen in 4 columns. If a string variable or literal string is more than 7 characters long and is followed by a comma the computer uses two or more fields in which to print the string and starts the next expression at the beginning of the third field.

ClibPDF - www.fastio.com

You can add extra commas to skip fields:

PRINT "FIELD 1", "FIELD 2", "FIELD 3"

would produce the output

FIELD 1 FIELD 2 FIELD 3

but

PRINT "FIELD 1",,"FIELD 2", "FIELD 3"
extra comma

would produce

FIELD 1 ↑ FIELD 2 FIELD 3
this field skipped

The statement

PRINT "THIS IS A FIELD",,"FIELD1", "FIELD 2"

would produce

THIS IS A FIELD FIELD 1
FIELD 2

The computer used 2 fields for the first string, skipped a field and then continued. It runs out of fields on the first line and has to continue in the first field of the second line.

There is no limit to the number of fields you may skip using commas, (Unless you run out of display area, of course.)

If you use a control character right at the end of a PRINT statement the next PRINT statement will obey the control character. For instance

```
1Ø    LET X = 4
2Ø    PRINT "THE ANSWER IS ";
3Ø    PRINT X
```

will produce the output

THE ANSWER IS 4

If no control character occurs at the end of a line the next PRINT statement will start printing on a new line.

The best way of finding out about PRINT is to experiment with it.

ClibPDF - www.fastio.com

CHAPTER 12

COPING WITH CHARACTERS

ClibPDF - www.fastio.com

COPING WITH CHARACTERS

You will have noticed that, in addition to the normal typewriter characters there are several other characters not usually found on a typewriter. In addition there are some usable characters not written on the keyboard.

Before we go on to investigate what characters can be printed, meet an interesting function:

CHR$(X) where X is a number, integer variable or expression.

It is used mainly in conjunction with the PRINT statement.

CHR$(X) means "the character whose code is x"

Try this:

```
1Ø    INPUT X
2Ø    PRINT CHR$(X)
3Ø    GO TO 1Ø
```

What this program does is to ask for a number and then to print out the character whose code is the number entered. The codes lie between Ø and 255.

A useful program for listing the symbols is given below.

```
1Ø    PRINT "ENTER CODE VALUE"
2Ø    INPUT X
3Ø    PRINT X; "...";  CHR$(X)
4Ø    PRINT
5Ø    LET X = X +1
6Ø    GO TO 3Ø
```

This lists 11 symbols together with their codes, starting with the code entered at the start of the program.

Table of characters corresponding to different codes:

Code	Character	Code	Character
Ø	space	14	:
1	null string	15	?
2	graphics (codes 2–11)	16	(
3		17	)
4		18	-
5		19	+
6		20	*
7		21	/
8		22	=
9		23	>
10		24	<
11		25	;
12	£	26	
13	$	27	

ClibPDF - www.fastio.com

Code	Character	Code	Character
28	Ø	55	R
29	1	56	S
30	2	57	T
31	3	58	U
32	4	59	V
33	5	60	W
34	6	61	X
35	7	62	Y
36	8	63	Z
37	9	64 to 127	?
38	A	128	Inverse space
39	B	129	Inverse quote
40	C	130	
41	D	131	
42	E	132	
43	F	133	
44	G	134	Inverse graphics (130–139)
45	H	135	
46	I	136	
47	J	137	
48	K	138	
49	L	139	
50	M	140	INVERSE £
51	N	141	" $
52	O	142	" :
53	P	143	" ?
54	Q	144	" (

Code	Character	Code	Character
145	Inverse)	172	Inverse G
146	" -	173	" H
147	" +	174	" I
148	" *	175	" J
149	" /	176	" K
150	" =	177	" L
151	" >	178	" M
152	" <	179	" N
153	" ;	180	" O
154	" ,	181	" P
155	" .	182	" Q
156	" Ø	183	" R
157	" 1	184	" S
158	" 2	185	" T
159	" 3	186	" U
160	" 4	187	" V
161	" 5	188	" W
162	" 6	189	" X
163	" 7	190	" Y
164	" 8	191	" Z
165	" 9	192 to 211	?
166	" A	212	"
167	" B	213	THEN
168	" C	214	TO
169	" D	215	;
170	" E	216	,
171	" F	217	)

ClibPDF - www.fastio.com

Code	Character	Code	Character
218	(	237	POKE
219	NOT	238	INPUT
220	-	239	RANDOMISE
221	+	240	LET
222	*	241	?
223	/	242	?
224	AND	243	NEXT
225	OR	244	PRINT
226	**	245	?
227	=	246	NEW
228	>	247	RUN
229	<	248	STOP
230	LIST	249	CONTINUE
231	RETURN	250	IF
232	CLS	251	GO SUB
233	DIM	252	LOAD
234	SAVE	253	CLEAR
235	FOR	254	REM
236	GO TO	255	?

Inverse means that the character appears white on a black background.

Inverse space is a black square, for instance.

The graphics characters are shown on the next page:

CHR$ (X) allows us to print any character we want.

There is a further group of facilities which enables us to handle characters;

These are

(a) TL$ (string)

—this gives the string minus its first character.

The string can be a literal string inside quotes or a string variable.

1Ø PRINT TL$ ("ABC") would give

BC

1Ø PRINT TL$(G$) will cut off the first character of string G$

(b) CODE (string)

—this gives the code corresponding to the first character in a string (either a string variable or a literal string)

1Ø PRINT CODE("ABC")

would print 38 which is the code for A.

An example of the way in which these can be used is the following program which accepts a string and prints it out in inverse video.

```
1Ø PRINT "ENTER YOUR STRING"
2Ø INPUT G$
3Ø PRINT G$
```

ClibPDF - www.fastio.com

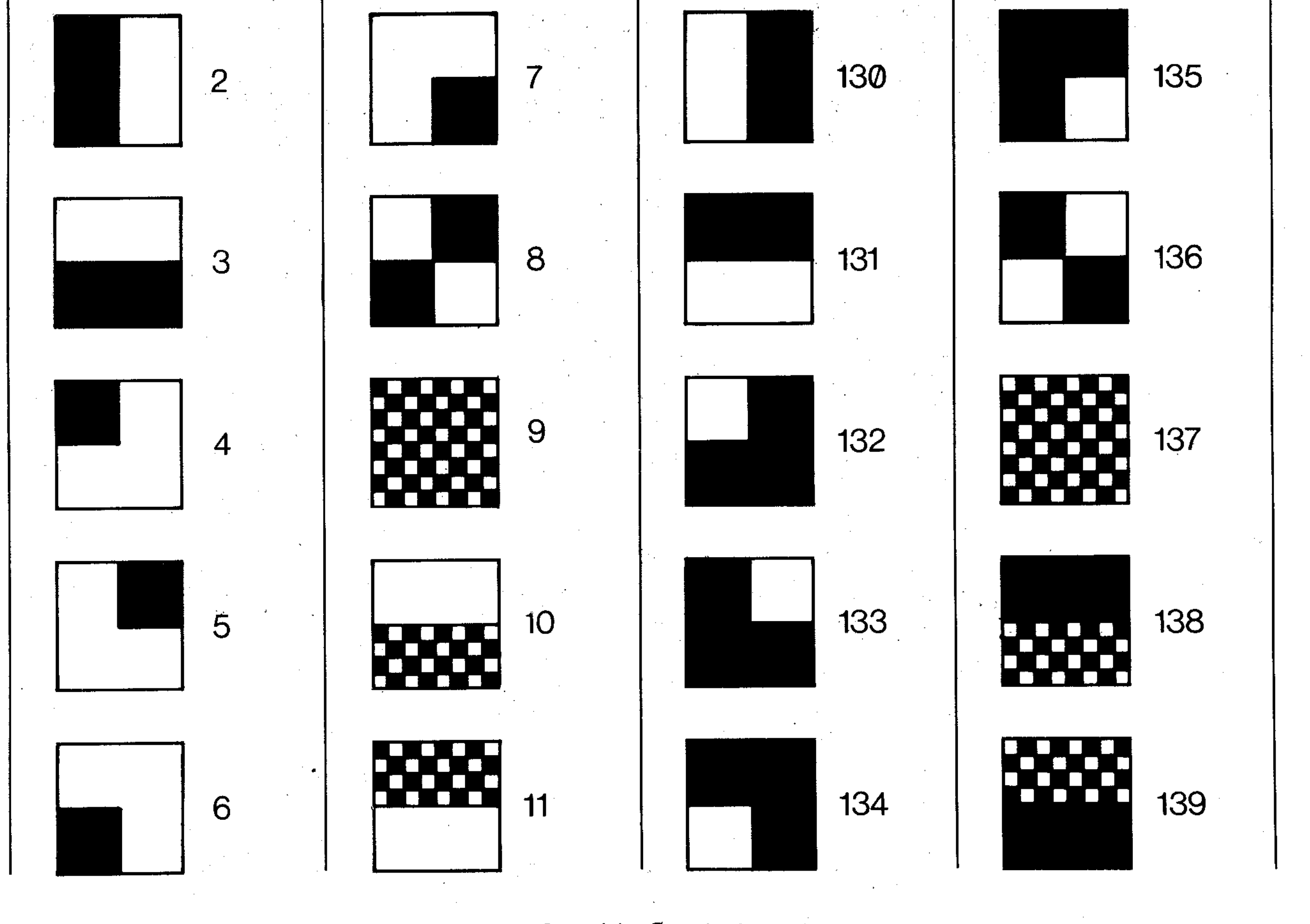


Table of Graphic Symbols

ClibPDF - www.fastio.com

```
 40  LET X = CODE (G$)
 50  LET X = X + 128
 60  IF G$ = CHR$ (1) THEN GO TO 100
 70  PRINT CHR$ (X);
 80  LET G$ = TL$ (G$)
 90  GO TO 40
100  STOP
```

Statement:

40 sets X = code of the first character of G$.

50 adds 128 to the code (this gives inverse of the letters, digits and graphics).

60 test the string to see if it is a null string, i.e. has no characters in it. If it is a null string either the input string was a null string to begin with or all the characters have been converted to inverse video and printed.

70 prints the inverse video character.

80 chops off the character which has just been printed and the program then jumps back to 40 and the code for the next character is extracted. The program goes on until the string has been shortened to the null string.

There is a further function which can be useful. This is:

STR$ (variable or number).

This allows an integer number or variable to be treated as a string variable

```
10 LET G$ = STR$(1234)
```

sets G$ = "1234"

```
10 LET G$ = STR$(X)
```

does the same thing for an integer variable — if X = 4852 then G$ = "4852".

So far we've not said very much about the graphics symbols. These have been designed so as to double the effective resolution of the display, which gives 23 lines of 32 characters each.

Here is a program which plots TWO bar charts on the same display.

ClibPDF - www.fastio.com

BAR CHART PLOTTER

```
1Ø  LET X = Ø
2Ø  PRINT "YZ-AXIS"
3Ø  PRINT "X ="
4Ø  FOR I = 1 TO 21
5Ø  LET Y = X
6Ø  LET Z = 24–X
7Ø  PRINT X,
```

Defines Variables and prints titles; generates Y and Z

```
8Ø  FOR J = 1 TO 2Ø
85  IF J>Y AND J=Z THEN PRINT CHR$ (3);
9Ø  IF J>Y AND J>Z THEN GO TO 135
95  IF J=Y AND J>Z THEN PRINT CHR$
(11);
1ØØ IF J<Y AND J<Z THEN PRINT CHR$
(139);
1Ø5 IF J<Y AND J=Z THEN PRINT CHR$
(139);
11Ø IF J<Y AND J>Z THEN PRINT CHR$
(11);
115 IF J=Y AND J<Z THEN PRINT CHR$
(139);
12Ø IF J>Y AND J<Z THEN PRINT CHR$
(3);
125 IF J=Y AND J=Z THEN PRINT CHR$
(139);
13Ø NEXT J

135 PRINT
14Ø LET X = X+1
15Ø NEXT I
```

The program prints Z as black bars, Y as grey bars. Statements 8Ø to 13Ø determine what graphic symbol is to be used: , , or nothing at all.

This program calculates the graphs of Y = X and Z = 24 – X and plots them in bar chart form. As you will see if you run the program it produces a very clear, unambiguous display. You can use the same sort of trick to achieve greater resolution along the line as well as from line to line.

Lines 85 to 125 decide what character will be printed, depending on the relative size of J, Z and Y.

These examples are only scratching the surface when it comes to character manipulation and graphics. The possibilities are almost literally endless.

ClibPDF - www.fastio.com

CHAPTER 13

HELP! OR, WHAT TO DO WHEN DESPERATE

ClibPDF - www.fastio.com

HELP! or "What to do when desperate"

You may occasionally get into difficulties.

These can be divided into two sorts:

(a) Problems with the system

(b) Problems with your programs.

Most problems with the system are likely to occur when you're entering string variables into your programs. One favourite is deleting the quotation marks round the string by accident. When the computer is waiting for you to enter a string variable it prints "[L]".

If you enter the wrong string and then use RUBOUT to delete it you may delete one or other of the quotation marks. This will give rise to a [S] syntax error symbol. If you see this check to see that both quotes are there.

If you have written a program which calls for a null string to be entered, and the program is recursive, you may find it very difficult to get out of the program back into command mode. Whatever you enter seems to have no effect! When (or if) this happens delete the quotes using RUBOUT ⇨ RUBOUT and enter (say) CHR$ (1Ø**6) which causes arithmetic overflow, giving a 6/N error message (when N is the line number of the INPUT statement reading the string).

Generally speaking the BREAK key (SHIFT SPACE) is the first thing to resort to. If the program is caught in an endless loop, or if it is LOADing unsuccessfully, the screen will go grey or black for an indefinite period. BREAK will get you back under these conditions.

If BREAK does not work in this situation there is nothing left to do but switch the ZX-80 off for a few seconds and then switch on. You do lose whatever program was in the ZX-80 if you do this.

When it comes to faults in the program it is difficult to offer such specific advice.

Some problems arise if you type an O instead of a Ø.

LET J = O would be accepted as a valid program line but would give a 2/N error code (variable not found) when the program was run. This sort of thing can be very difficult to spot. Similarly S and $ do get confused.

When a program stops unexpectedly or does something peculiar it may be difficult to work out exactly what went wrong. It is possible to carry out a post-mortem by using the immediate

ClibPDF - www.fastio.com

PRINT statement to find out what the value of variables (especially loop control variables) was at the time the program stopped.

If you type

PRINT J (or whatever variable you want)

the ZX-80 will print it even after the program has stopped running.

Sometimes there is not room to print the whole of the line number in an error message, particularly with error numbers 4 and 5. In this case usually only the first digit of the line number is printed. For example, error 4 on line 25Ø may cause the message

4/2

Another useful technique is to put STOP statements into programs at key points. When the program reaches the STOP statement it will stop (of course) and you will be able to see how it has performed up to that point. You can then get back into command mode, type CONTINUE (key T) NEWLINE and the program will continue from the STOP statement.

Luckily the ZX-80 is quite choosy about the program lines it will accept and this eliminates many of the problems which can happen with other BASIC's.

ClibPDF - www.fastio.com

CHAPTER 14

A RAGBAG OF FUNCTIONS

ClibPDF - www.fastio.com

A RAGBAG OF FUNCTIONS

This chapter covers all those statements and functions that haven't already been dealt with elsewhere.

RND(X) — provides a random number in the range 1 to X

Typical statement using RND:

```
10  LET J = RND(X)
```

sets J equal to a random number

Every time the ZX-80 executes the function RND it uses a random number generator to generate a pseudo-random number. It is called pseudo-random because the numbers occur in a fixed sequence. However, the sequence is very long and hence appears random.

RANDOMISE — sets the starting point of the sequence to a number equal to the number of frames supplied to the TV since the machine was turned on (unless POKE is used to alter the count, see below).

RANDOMISE n — sets the starting point of the sequence to n unless n is Ø, when it behaves as RANDOMISE.

The RANDOMISE statement allows the random sequence to be initialised at any time in the program. Typical forms:

```
1Ø    RANDOMISE 6
1Ø    RANDOMISE
```

RANDOMISE n ($n \neq o$) generates a sequence depending only on the value of n (which will be the same from run to run if n is unchanged).

RANDOMISE generates a different sequence each time.

One highly useful statement is POKE.

This is of the form

POKE A,B — where A is the ADDRESS of a location in store and B is an expression (the value of which should be less than 256 for sensible results).

Typically A might be the address of one of the two bytes which form the variable which acts as the frame counter. This is demonstrated in the next example program.

ClibPDF - www.fastio.com

Thus:

LET X = PEEK (A) sets X equal to the contents of address A

PEEK (A) is always less than 256 (in range Ø to 255)

The reason that the variables associated with PEEK and POKE should always be less than 256 (255 is maximum) is that everything in the ZX-80 is stored in 8 bit bytes. Bit is short for Binary digit. The maximum which can be stored in 8 bits is 255, and so all the integer variables use up 2 bytes, which allows numbers up to 32,767 to be stored.

Each half of a variable is stored in one byte, and every byte has an address of its own.

Here is a program which uses PEEK and POKE to gain access to the TV frame counter.

PEEK/POKE REACTION TIMER

```
1Ø  FOR I = 1 to 2Ø * RND(1ØØ)
2Ø  NEXT I
3Ø  POKE 16414,Ø
4Ø  POKE 16415,Ø
5Ø  PRINT "HIT RETURN"
6Ø  INPUT C$
7Ø  LET A = PEEK (16414)
8Ø  LET B = PEEK (16415)
9Ø  PRINT "YOUR REACTION TIME WAS";
    (B*256 + A – 4) *2Ø; "MILLISECS"
```

16414 and 16415 are the addresses of the two halves of a 16 bit number which counts the frames on the TV – increasing by 1 every 1/50th of a second. Lines 3Ø to 4Ø set the count to zero, and the count is stopped when a null string is input to C$. There is a delay on all operations (mainly between pressing the return key and this signal getting to the CPU) of around 8Ø mS, hence the 4 subtracted from the expression in line 9Ø.

This could have a repeat mechanism tacked on the end, for example:

```
1ØØ  PRINT "DO YOU WANT ANOTHER GO?"
11Ø  PRINT "TYPE Y OR N"
12Ø  INPUT C$
13Ø  IF C$ = "Y" THEN GO TO 1Ø
14Ø  STOP
```

Another function which allows the user to communicate directly with the ZX-80 is

ClibPDF - www.fastio.com

USR(A) — this calls a machine-code subroutine at the address A. The value is whatever the routine leaves in HL (a storage register within the central processor of the ZX-80) or, if the subroutine has not altered HL, the result is A

Typical form: LET J = USR(A)

PEEK, POKE and USR(A) are really facilities provided for very experienced users who understand the detailed working of the ZX-80.

CLEAR is a statement which resets all the variables in the program.

Typical form:

```
1Ø    CLEAR
```

(RUN automatically clears the variables everytime a program is run.)

And now, last but not least, DIM. This is of the form

1Ø DIM A(B) — sets up an array A which contains B+1 variables.

Each variable is called an element of array A.

Arrays can have any single-letter name and can have any number of elements (providing that there is enough room to store all the elements).

It is possible, though not a good idea, to have a variable and an array with the same name example:

```
1Ø    LET A = 4
2Ø    DIM A(A)
```

Each element is referred to by its subscript. For instance

B(Ø) is the first element of array B

B(2) is the third element of array B

B(N) is the (N+1)th element of array B.

Because you can use any integer expression as a subscript it is possible to process array elements easily and quickly.

Here is an example of the use of arrays for character manipulation.

CHEESE NIBBLER

```
1Ø    DIM A(1Ø)
2Ø    DIM B(1Ø)
3Ø    DIM C(1Ø)
```

ClibPDF - www.fastio.com

```
100  FOR J = 1 TO 10
110  LET A(J) = 1
120  LET B(J) = 1
130  LET C(J) = 1
140  NEXT J

200  FOR J = 1 TO 10
205  IF NOT A(J) = 1 THEN GO TO 220
210  PRINT "■";
215  GO TO 230
220  PRINT " ";
230  NEXT J
240  PRINT

300  FOR J = 1 TO 10
305  IF NOT B(J) = 1 THEN GO TO 320
310  PRINT "■";
315  GO TO 330
320  PRINT " ";
330  NEXT J
340  PRINT

400  FOR J = 1 TO 10
405  IF NOT C(J)=1 THEN GO TO 420
410  PRINT "■";
415  GO TO 430
420  PRINT " ";
430  NEXT J
440  PRINT

445  PRINT "HIT NEWLINE TO NIBBLE THE
     CHEESE"
450  INPUT Y$
460  IF NOT Y$ = " " THEN GO TO 1000
470  CLS

500  LET I = RND(10)
510  LET K = RND(3)
520  IF K = 1 THEN LET A(I) = 0
530  IF K = 2 THEN LET B(I) = 0
540  IF K = 3  THEN LET C(I) = 0
550  GO TO 200

1000 STOP
```

When typing in a program like this where there are several similar lines the EDIT facility is very useful, because you can edit line numbers.

ClibPDF - www.fastio.com

For example; after entering line 21Ø, type:

EDIT

⇦

⇦

RUBOUT

3

NEWLINE

Statements 1Ø to 3Ø set up 3 arrays, A, B, and C of 1Ø elements each.

Statements 1ØØ to 14Ø set all the elements of all arrays to 1.

2ØØ to 24Ø examine each element of the array A in turn. If an element is equal to 1 a ■ is printed. If an element is not 1 a space is printed.

Similarly for 3ØØ to 34Ø and 4ØØ to 43Ø.

On the first pass all the elements are set to 1 and ■ is printed all the way through.

445 to 47Ø print the instruction and if the user then hits NEWLINE the screen is cleared.

5ØØ to 54Ø select a random element of a random array to be set to Ø.

At 55Ø the program jumps back to print out all the arrays. The element which has been set to Ø is printed as a space – or nibble.

As more nibbles are taken there is a greater chance that the element selected will already be Ø and thus the effective nibble rate slows down as the game progresses.

ABS(n) this function gives n if n≥Ø, and –n if n<Ø. n is any integer expression.

e.g. ABS(5) = 5

ABS(–5) = 5

ClibPDF - www.fastio.com

CHAPTER 15

OVER TO YOU

ClibPDF - www.fastio.com

OVER TO YOU

If you have read all the way through this book, running the programs in it and writing your own, you should be well on the way to becoming a fluent BASIC programmer. Remember, though, that there are many things you can do with ZX-80 BASIC which you can't do using other BASIC's. Equally, some BASIC's have features which are not present in ZX-80 BASIC.

The appendices present a concise summary of the error codes and the ZX-80 BASIC. Section 2 of Appendix 2 is written with experienced users in mind, but most of it does not require a detailed knowlege of the working of the central processor.

From now on it's up to you!

ClibPDF - www.fastio.com

ERROR CODES

ClibPDF - www.fastio.com

ERROR CODES

When results are displayed, a code

n/m

is displayed also: n is the error number, m is a line number in the program

stmt type	code	meaning
	Ø	(m = next line that would have been executed) BREAK pressed. (m = –1 or –2) Command successfully completed. (m < Ø or > largest line number in program) GO TO m was executed. (m = largest line number in program) end of program.
NEXT	1	* NEXT <id> where <id> is not the control variable of an active FOR loop.
any	2	* variable name not found (any variable being used, or array name of an array element being assigned to).
any	3	* subscript out of range, or error of any kind while evaluating a subscript.
LET INPUT DIM PRINT	4	* no room to add new variable or to assign a longer string to a string variable or to screen.
PRINT	5	* no more room on screen.
any	6	arithmetic overflow (result > 32767 or < –32768, also result = –32768 in some circumstances.
RETURN	7	* RETURN with no corresponding GOSUB.
INPUT	8	INPUT can only be used in a program, not directly.
STOP	9	STOP statement executed.

* m = line number of offending statement.

CONTINUE is the same as GO TO m, except after code 9 when it is GO TO m + 1.

ClibPDF - www.fastio.com

APPENDIX

4K BASIC FOR ZX-80

ClibPDF - www.fastio.com

4K BASIC FOR ZX–80

1. ZX-80 user's view.

(a) Program Listing

The user inputs via the keyboard, lines of BASIC for insertion into the program and commands for immediate execution. While he is doing this he sees a display which is divided into two parts: the upper part is a "window" on the program listing, while the lower part displays the line or command he is currently outputting. Normally, the lower part is large enough to hold the whole line (which will take more than one screen line if it contains more than 32 characters), there is a blank line between the two parts, and the upper part occupies the remainder of the screen. If, however, there is insufficient RAM to hold a display of this size (each character on screen occupies 1 byte of RAM*) the upper part of the display will be shrunk line by line until the display file is small enough. When the upper part has disappeared altogether the lower part shrinks character by character.

The computer maintains a "current line number" for editing the program and the display is always organised so that the line with that number, or the preceding line if no line with that number exists, is on the screen if at all possible. If there is a line with the current number, it is displayed with a symbol consisting of a reverse video [>] between its line number and the text of the line; if there is none then the reverse video [>] will not appear.

Three keys are provided for changing the current line number: "⇧" changes it to the line number of the preceding line, "⇩" changes it to the line number of the following line, and "HOME" resets it to zero. If there is no preceding line "⇧" sets it to the line number of the first line in the program; similarly "⇩" will set it to the last line.

There are two other ways which the current line number can change: inserting a line into the program sets it to the line number of that line, and the command "LIST n" will set it to n.

When the current line is off the top of the screen, the window moves up so that it becomes the first line. When it is at or just off the bottom the window moves down a line. If it is well beyond the bottom of the window, the window moves down so that it becomes the second line on the screen.

* RAM stands for random access memory, or store.

ClibPDF - www.fastio.com

(b) Input area

The lower part of the screen contains the line the user is currently typing in. This line may be a command or a line of program; in the latter case it will begin with a line number (in the range 1 to 9999) and in the former case there should be no number, although zero in practice counts as "no number" here.

Somewhere in the line a "cursor" is displayed. This indicates two things: the position in the line where symbols will be inserted, and whether an unshifted alphabetic key will be treated as a keyword (eg "LIST") or a letter (eg "A"). The cursor is in the form of an inverse video [K] for keywords or [L] for letters.

Note that this cursor, although displayed in the line and occupying a character position on the screen, does not form part of the line and is ignored by anything interpreting the line.

A second symbol, similar in principle to the cursor, may also be displayed: this is in the form of an inverse video [S] and indicates that the line is not a syntactically correct BASIC statement. It is positioned such that the part to the left of it could be the beginning (or the whole) of a syntactically correct BASIC statement eg if "20 LET A = B + 5" is input from left to right then. [S] symbol will be displayed at the end of

20 LET	and be absent from	2
20 LET A		20
20 LET A=		20 LET A=B
20 LET A=B+		20 LET A=B+5

In most cases the symbol is displayed as far to the right as is consistent with the above description; however there are a few circumstances where this is not quite so, for instance in

LET A=ASC("X")

although "LET A=ASC" is a syntactically correct statement (ASC here being an integer variable) the [S] is not displayed after "ASC" but rather before it. This is because "ASC(. . .)" has already been identified as a function call, but as no built-in function with the name "ASC" is available it is faulted. Having identified it as a function call, however, the computer does not then consider other possible parses.

The following keys are available to alter the input line:

(i) single-character symbols: letters, digits, punctuation, etc the symbol is inserted to the left of the cursor.

ClibPDF - www.fastio.com

(ii) multi-character "tokens": "**", "AND", "OR", "NOT", "TO", "THEN" keywords.

Each of these is stored in the computer as a single byte, which, as in (i), is inserted to the left of the cursor. However, they appear on the screen as more than one character. Those that are alphanumeric (ie all except "**") are preceded and followed by a space, the preceding space being omitted (a) at the beginning of the line (b) where it follows another alphanumeric token. (This rule means that programs appear well-laid-out on the screen without using up scarce RAM space for explicit space characters. Inserting an explicit space character before or after an alphanumeric token always inserts one extra space in the displayed form.)

(iii) "RUBOUT" deletes the symbol or token to the left of the cursor.

(iv) cursor control keys '⇨' and '⇦' skip the cursor past the next symbol or token to the right and left respectively.

(v) "EDIT" replaces the input line with a copy of the current line from the program. If no line has the current line number, the first line after it is used. If the current line is after the last line in the program, the last line is used. If there are no lines of program at all, then an empty line is used. Note that any existing input line is lost; "EDIT" followed by "NEWLINE" is in fact the quickest way to get rid of an unwanted line, but beware typing "EDIT" in mistake for "NEWLINE"!

(vi) "NEWLINE" is ignored if the inverse video [S] symbol is present. Otherwise it marks the end of input and the line or command is submitted to the system (see next section). Note that the whole line is submitted, not just the part to the left of the cursor.

(c) After input

When a line with a nonzero line number is submitted to the system, it is inserted into the program, any existing line with the same number being first deleted. The input area is then cleared.

A special case is where the new line consists only of a line number, possibly preceded by spaces: the existing line (if any) is deleted but nothing replaces it and it therefore simply disappears from the listing. The "current line number" is still set to its number, however, so the inverse video [>] disappears also (see (a) above). If the line being inserted has one or more spaces after the line number but no other symbols or tokens,

ClibPDF - www.fastio.com

it is still inserted in the program and appears in the listing as a line which is blank except for its number; when the program is run such lines are ignored.

A line which has no line number, or which has line number zero, is a "command" and is obeyed immediately. For as long as it takes to obey the command (which for most commands is very brief) the screen is blank, then on completion the upper part of the display contains any output generated and the lower part contains a display of the form

m/n

where m is a single digit and n is "–2" for most commands.

If m = Ø, execution was successful; if m = 9 a STOP command was executed; otherwise m is an error code (see Appendix I).

Where a command (RUN, GO TO, GOSUB, CONTINUE) has caused the program to be entered, n is the line number of the offending instruction if m is an error code (exception: if the error is in a GO TO or GOSUB then n may be the target of the jump), the line number of the STOP if m = 9, and the line number of the last line in the program if m = Ø. Except in the case of m = Ø or m = 9, CONTINUE is a jump to line number n (but see 3(c)). If m = 9, CONTINUE is a jump to line number n +1.

Sometimes only the first digit of n is displayed because there is no room in the RAM for any more display file. For example beware confusing line number 24Ø, of which only the first digit is displayed, with line number 2.

A jump to a line number which is beyond the end of the program, or greater than 9999, or negative, gives m=Ø, n= the line number jumped to.

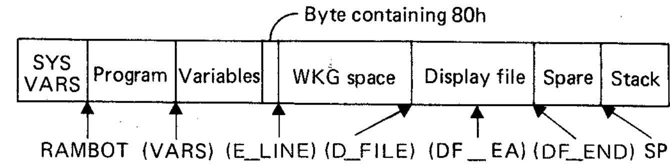
The commands are described individually in section 3.

2. Computer's view

N.B. 35h means Hexadecimal 35 (35h=3*16+5 =53. ØAh to ØFh are the decimal numbers 10 to 15.

(a) RAM

The contents of the RAM are:

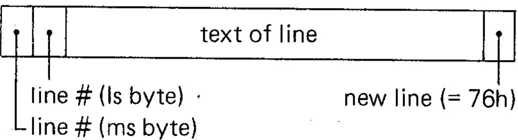


ClibPDF - www.fastio.com

The first area is fixed in size and contains various "system variables" which store various items of information such as the current line number, the line number to which CONTINUE jumps, the seed for the random number generator, etc etc. Those that could possibly be useful with PEEK etc have been documented elsewhere (Appendix 3).

An important subset of the system variables are the five contiguous words labelled VARS to DF_END which hold pointers into the RAM and define the extent of the remaining areas (apart from the stack).

The program consists of zero or more lines, each of the form



ie beginning with the line number (stored ms byte first contary to the usual practice on Z80's) and ending with a newline. The line number is in the range 1 to 9999 so that the ms 2 bits of the first byte are zeroes. The ms 2 bits of the byte pointed to by (VARS) will not both be zeroes; this gives a simple test for end-of-program.

The program lines are stored in ascending order of line number. The text consists of ordinary characters (codes Ø to 3Fh) and tokens (codes CØh to FFh), although reverse video characters (codes 8Øh to BFh) have also been allowed for.

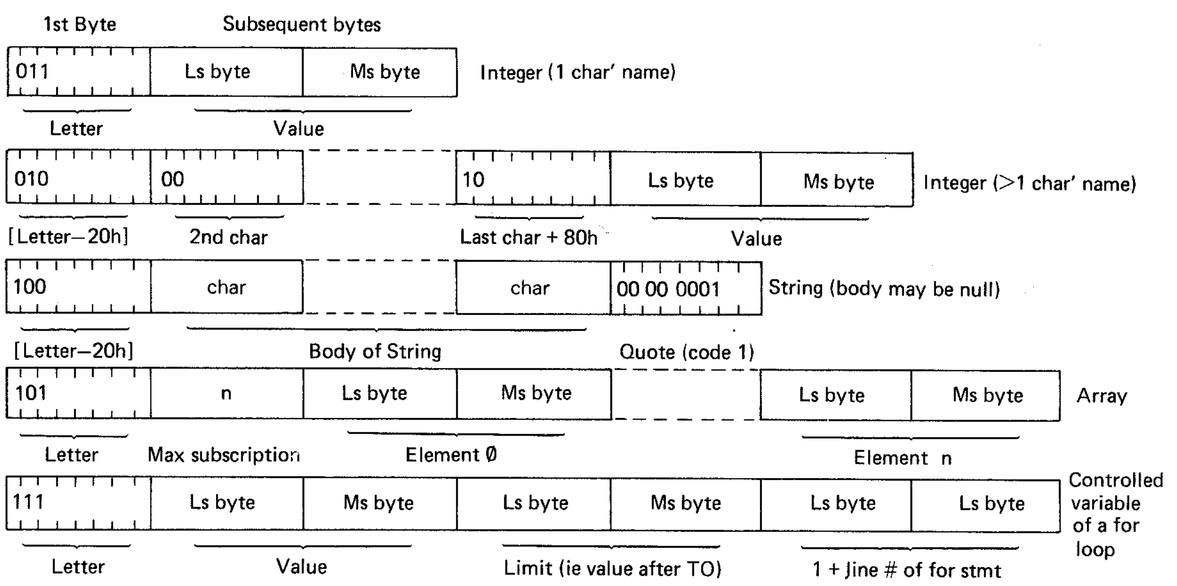
The variables take the forms shown on the next page.

They are not stored in any particular order; in practice each new variable is added onto the end. When a string variable is assigned to, the old copy is deleted and a new one created at the end. (Created first – "LET A$ = A$" does work!) Note that apart from the ms bit of the first byte a single-character integer is the same as the controlled variable of a FOR loop. The characters in a name, being all alphanumeric, have 6-bit codes as in the character code table. The first character in a name, being perforce alphabetic (ie in the range 26h to 3Fh) effectively has a 5-bit code.

The "variables" area is terminated by a single byte holding 80h (which can't be the name of a string!).

The working space holds the line being input (or edited, hence "E_LINE") except when statements are being obeyed when it is used for

ClibPDF - www.fastio.com



temporary strings (e.g. the results of CHR$ and STR$) and any other similar requirements. The subroutine X_TEMP is called after each statement to clear it out, so there is no need to explicitly release space used for these purposes.

The display file always contains 25 newline characters (hex 76); the first and last bytes are always 76h and in between are 24 lines each of from 0 to 32 (inclusive) characters. (DF_EA) points to the start of the lower part of the screen.

The stack (pointed to by register SP) has at the bottom (high-address end) a stack of 2-byte records. GOSUB adds a record to this stack consisting of 1+ its own line number; RETURN removes a record and jumps to the line number

ClibPDF - www.fastio.com

stored therein. The last 2 bytes of RAM contain a value which RETURN recognises as not being a line number.

The expression evaluator (which is also used to check the syntax of expressions) pushes 4 bytes onto the top of the stack for each intermediate result and pops them again when the appropriate operator is found, eg.

A+B*C**D*E+F*G $\quad$ X=B*C^{D}
Y=A+X*E

stack	A+	B*	C**	X*	Y+	F*
		A+	B*	A+		Y+
			A+			

Thus the above expression uses a maximum of 12 bytes of stack.

Parentheses use an additional 6 bytes each, eg.

A + (B*C**D)*E + F*G

would use 12 + 2 x 6 = 24 bytes

Apart from these two cases, the stack is only used for subroutine calls and for saving registers.

(b) Actions

The actions taken by the computer in response to the user's keystrokes are as follows.

Each time a symbol or token is inserted into or deleted from the input line, also each time the cursor is moved, this change is put into effect in the input line held in working-space after deleting the lower part of the display file (viz that part from (DF_EA to DF_END) – note that during this period the display file may be incomplete in that less than 25 newline characters are present, although the display file is never allowed to become large enough that there will not be room to add the remaining newline characters). Then the input line is checked to see if it is syntactically correct. The input line contains an inverse video [K] at the point where the cursor is; the syntax checker notes the address of the cursor in the variable (P_PTR) and sets variable (X_PTR) to point to the first wrong symbol or to zero if there is none. It also notes whether the cursor should be displayed as [K] or an [L]. Finally the lower part of the display file is rebuilt, inserting the [S] symbol and changing the cursor from [K] to [L] if required, as well as converting tokens into characters.

If there is now insufficient room for the display file the display file area is cleared and the upper part is remade with fewer lines by re-copying from the program stored in the area RAMBOT to (VARS), again converting tokens into characters

ClibPDF - www.fastio.com

as part of this process, and the lower part is then output afresh.

When a line is to be inserted into the program, its line number is converted into binary, space is made at the appropriate place by copying everything else up, and the text of the line (from which the cursor has already been deleted) is copied in. The working-space and display file are then re-made, the former now containing just the cursor and a newline.

When a command is executed, it is interpreted in situ in the working-space area. Program lines are of course interpreted in their place in the program.

3. Statements

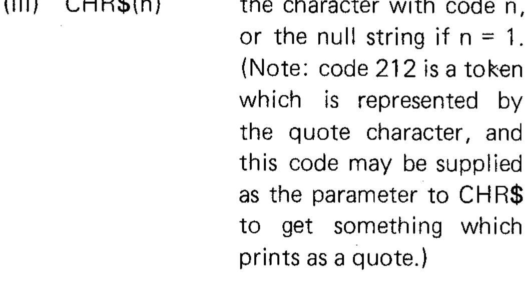
(a) expressions

Throughout section 3, "n" will be used to represent any space integer expression and "s" to represent any string expression.

String expressions are:

(i) string variable: $\alpha$$ where α is any letter (no intervening space allowed). Example: A$. May be used as "dest" for INPUT (see 3(b)).

(ii) literal string: delimited by quotes, any symbols or tokens other than quote permitted inside the string. Examples: "ABC" "++**".



(iii) CHR$(n) the character with code n, or the null string if n = 1. (Note: code 212 is a token which is represented by the quote character, and this code may be supplied as the parameter to CHR$ to get something which prints as a quote.)

(iv) TL$(s) the string s minus its first character, or the null string if s is null or contains only one character.

(v) STR$(n) the decimal representation of n: from 1 to 5 digits, leading minus sign if negative, no spaces — hence from 1 to 6 characters in all.

(vi) (s) parenthesised string expression.

ClibPDF - www.fastio.com

Integer expressions are:

(i) integer variable: first character alphabetic, subsequent characters alphanumeric, no embedded spaces, no limit on length of name. May be used as "dest" for INPUT (see 3(b)).
Examples: J AB37Q.

(ii) literal number: decimal, unsigned, value must be <32768 (syntax error otherwise), as many leading zeroes as you like.
Examples: 23 32767 ØØØØ32767.

(iii) prefix and infix operations:

form	priority	notes
n**n	10	"to the power"
—n	9	
n*n	8	
n/n	7	
n+n	6	
n—n	6	n—n behaves identically to n+—n
n=n	5	value is —1 if "true", 0 if "false".
n>n	5	
n<n	5	
s=s	5	
s>s	5	
s<s	5	
NOT n	4	bitwise Boolean operations
n AND n	3	
n OR n	2	

Ambiguities in parsing operations are resolved by considering the priority of the operators in question: higher priorities bind tighter, equal priorities associate from the left. Example:

—A**B+C*D/E*F—G—H

is the same as

(((—(A**B))+((C*D)/(E*F)))—G)—H

(iv) PEEK(n) the value at address n, being a single byte in the range Ø to 255.

(v) CODE(s) the code for the first character in strings.

ClibPDF - www.fastio.com

(vi) RND(n) — a pseudo-random number in the range 1 to n if n>Ø, in the range 1 to 32767 or –32768 to n–1 if n<Ø. If n=Ø returns 1 always.

(vii) USR(n) — call machine-code sub-routine at address n. Value is whatever the subroutine leaves in HL, or n if it doesn't alter HL.

(viii) ABS(n) — <u>if</u> n<Ø <u>then</u> –n <u>else</u> n.

(ix) array element: α(n) Example: A(I+1). May be used as "dest" for INPUT (see 3(b)).

(x) (n) — parenthesized integer expression.

Values of string expressions can be of any length and can contain any codes except 1 (the closing quote). Values of integer expressions must be in the range –32768 to +32767; any value outside this range causes a run-time error (number 6).

Note that relations yield –1 for "true" and Ø for "false" and that

–1 AND n	is the same as n.
Ø AND n	is the same as Ø
–1 OR n	is the same as –1.
Ø OR n	is the same as n.

also

NOT n	is the same as –n–1.

so that for instance

I AND I>Ø OR –I AND I<Ø.

is the same as ABS (I).

However constructions such as A>B>C do not have the obvious effect, being parsed as

(A>B)>C ie as

(A>B) AND C<–1 OR (NOT A>B) AND C<0.

<u>(b) Statements</u>

The statements available are:

NEW — re-initialise the computer to the state it has at initial switch-on. Loses all program, variables, etc.

LOAD — read system variables, program, and variables, from tape, and re-initialise input line and remake display file. Does not affect any GOSUB blocks that may be on the stack (although the effect of doing a RETURN is unlikely to be sensible). BREAK after data has been found on the

ClibPDF - www.fastio.com

	tape does the same as NEW: BREAK during the lead-in preserves the current program and variables.
SAVE	Write system variables, program, and variables to tape for subsequent reading by LOAD. Precedes data with 5 secs of silence; starting the tape before executing SAVE (which is recommended practice) writes frame sync pulses to the tape.

The above can all be used in programs, but are intended to be used as commands, and use in programs is not particularly sensible.

RUN n	is the same as CLEAR followed by GO TO n.
RUN	is the same as RUN 1.
CONTINUE	is the same as GO TO n, where n is the last number in an end-of-program message m/n with m>Ø (see 1(c) and 3(c). However, after message 9/n, CONTINUE is the same as GO TO n+1.
REM any text	no effect (ie is comment).
IF n THEN statement	executes the statement unless n is zero. (ie unless n is "false").
INPUT dest	where "dest" can take any of the forms marked above (see 3(a)). Returns to the input mode with any output so far produced by the PRINT statement in the upper part of the screen, and the input line initialised to contain the cursor alone if "dest" is an integer and the cursor inside a pair of quotes if "dest" is a string. These quotes can, however, be deleted as the input can be any expression (not necessarily a literal). The expression is checked for correct syntax and "NEWLINE" is ignored if the [S] marker is present. The upper part of the screen consists of as many lines as have been written by PRINT and the lower part follows on rather than being placed at the bottom of the screen. It is possible for the upper part to contain 24 lines, in which case the lower part is not

ClibPDF - www.fastio.com

visible although the keys all function normally, and, provided the user is able to work blind, the input can still be submitted.
The keys "⇧" and "⇩" have no effect; the effect of "EDIT" is rather comical and not very helpful, although it doesn't actually crash the system.

The INPUT statement cannot be used as a command because of the conflict in use of the working-space; however, in this situation LET can be used instead. Using INPUT as a command causes error code 8/–2.

If "dest" is an array element and there is an error in evaluating the subscript or the subscript is out of bounds the error is not reported until after the input value has been submitted.

PRINT <list>

in which <list> may be:

	<empty>
or	<expr>
or	<list>, <list>
or	<list>; <list>
and	<expr> may be s or n.

writes the value of each <expr> to the upper part of the screen, ie in the case of "s" writes the body of the string, converting tokens into characters, and in the case of "n" writes STR$(n). Each comma causes the output to tab to the 9th, 17th, or 25th column on the line of to the 1st column on the next line. A new line is output unless the <list> ends with a comma or semicolon.

LIST n — sets the "current line number" to n and enters "program input" mode without waiting to display any printed output error code (even if there is an error in evaluating n!) It is therefore only really suitable for use as a command.

LIST — is equivalent to LIST Ø.

STOP — causes "error" code 9, so that CONTINUE will carry on from the following statement. Useful for displaying results when no input is required.

GlibPDF - www.fastio.com

DIM α(n) — creates an array with name α and subscript range 0 to n inclusive. If one already exists, the new space will be reserved but the old array will continue to be used for all accesses. If a variable α already exists, it can still be used.

FOR α = n TO n — assigns to α (ie an integer with a single-character name) a FOR block as described in section 2(a). The effect is that a subsequent "NEXT α" will increment α and if it is not now greater than the TO value jump to the statement following the FOR. Note that we always enter the body of the loop at least once, and that the association between FOR and NEXT is entirely dynamic, ie not lexical.

GO TO n — jumps to line number n, or to the next line with a line number greater than n, or if neither exists stops showing code 0/n.

POKE n,n — the first n is an address to which the value of the second (modulo 256) is written as a single byte.

RANDOMISE n — set the seed of the pseudo-random number generator to n.

RANDOMISE — as above with n = number of frames displayed on the TV since initialisation (modulo 256).

CLEAR — delete all variables.

CLS — clear the upper part of the screen, eg to get rid of a message appropriate to an INPUT as in

```
2Ø PRINT "TYPE THE VALUE
FOR A".
3Ø INPUT A.
4Ø CLS.
```

GOSUB n — as GO TO n, but also push a GOSUB block on the stack so that RETURN will jump to the line following this one.

RETURN — pop a GOSUB block off the stack and jump to the line number contained in it.

NEXT α — equivalent to LET $\alpha = \alpha + 1$.
IF NOT $\alpha >$ (value supplied with TO) THEN
GO TO (line following FOR α).

ClibPDF - www.fastio.com

The combined effect of

m FOR $\alpha = n_1$ TO n_2 (m is the line number).

and

NEXT α

is of

m { LET $\alpha = n_1$
LET LIMIT $\alpha = n_2$ }

(assuming LIMITα isn't used elsewhere in the program).

and

LET $\alpha = \alpha + 1$.
IF NOT $\alpha >$LIMITα THEN GO TO m+1.

Note that this does not preclude (i) assignment to α (ii) several NEXTs matching one FOR, or one NEXT matching several FORs.

Note also that FOR – GOSUB – NEXT – RETURN, and FOR I – FOR J – NEXT I – NEXT J are possible, though not very useful.

(c) BREAK

If the BREAK key is found to be pressed at the end of execution of a line, execution does not follow on to the next line but stops showing 0/n where n is the line number of the next line that would have been executed but for the break-in.

4 Character set

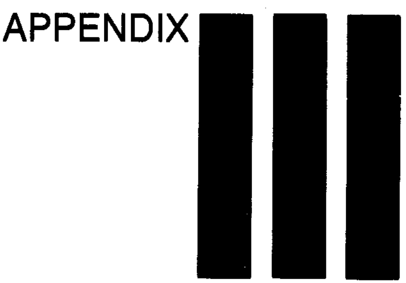
Code	Character	Code	Character
0	space	†23	>
1	"	†24	<
2	GRAPHICS	†25	;
3	GRAPHICS	†26	,
4	GRAPHICS	27	.
5	GRAPHICS	28	0
6	GRAPHICS	29	1
7	GRAPHICS	30	2
8	GRAPHICS	31	3
9	GRAPHICS	32	4
10	GRAPHICS	33	5
11	GRAPHICS	34	6
12	£	35	7
13	$	36	8
14	:	37	9
15	?	38	A
†16	(	39	B
†17	)	40	C
†18	–	41	D
†19	+	42	E
†20	*	43	F
†21	/	44	G
†22	=	45	H

ClibPDF - www.fastio.com

46 I
47 J
48 K
49 L
50 M
51 N
52 O
53 P
54 Q
55 R
56 S
57 T
58 U
59 V
60 W
61 X
62 Y
63 Z
† 64 –127 print as ?
† 128 –191 inverse video of 0 to 63
† 192 – 211 print as ?
† 212 "
213 THEN
214 TO
215 ;
216 ,
217)
218 (
219 NOT
220 –
221 +
222 *
223 /
224 AND
225 OR
226 **
227 =
228 <
229 >
230 LIST
231 RETURN
232 CLS
233 DIM
234 SAVE
235 FOR
236 GO TO
237 POKE
238 INPUT
239 RANDOMISE
248 LET
241 ?
242 ?
243 NEXT
244 PRINT
245 ?
246 NEW
247 RUN
288 STOP
249 CONTINUE
250 IF
251 GOSUB
252 LOAD
253 CLEAR
254 REM
255 ?

† not available from the keyboard. Codes 38–63 only available when cursor is [L], codes 230–255 only when cursor is [K].

ClibPDF - www.fastio.com

APPENDIX 

SYSTEM VARIABLES

ClibPDF - www.fastio.com

SYSTEM VARIABLES

The contents of the first 40 bytes of RAM are as follows. Some of the variables are 1 byte, and can be POKEd and PEEKed directly. The others are each 2 bytes, and have the low order byte at the given address (n, say) and the high order byte at the next address. Thus to poke value v at address n do

```
POKE n, v
POKE n+1, v/256
```

and to PEEK at the value in address n use the expression

```
PEEK(n) + PEEK(n+1)*256
```

if the value is known to be in the range 0 to 32767, and something like

```
LET MSB = PEEK(n+1)
IF MSB>127 THEN LET MSB = MSB–256.
LET VALUE = PEEK(n) + MSB*256
```

if it may be negative, ie. if n is 16412 or 16414. The notes at the lefthand side of the table have the following meanings:

X — The variable should not be altered by POKE as this could cause the BASIC to hang up.

N — POKE will have no lasting effect, as the variable will be rewritten either at the end of the POKE statement or the next time "edit" mode is entered (either at the end of the run or to get some input).

1 or 2 — number of bytes in variable.

U — "unsigned" number in the range 0 to 65535; the BASIC will treat values in the range 32768 to 65535 as –32768 to –1 respectively. These are the only variables likely to yield negative values.

notes	address	contents
1	16384	1 less than the "run-time error number": PEEK yields 255 normally, 1 less than the error code if an error has already occurred in this statement, eg PRINT (10**6 AND Ø) OR PEEK (16384) prints 5 because 10**6 overflows. Do not POKE any value other than 255 or 0 to 8. POKE 16384,255 does nothing; POKE 16384, n (n in the range 0

ClibPDF - www.fastio.com

		to 8) causes error n+1. Thus POKE 16384, 8 is the same as STOP.
X1	16385	sundry flags which control the BASIC system.
2	16386	statement number of current statement: POKE has no effect unless this is the last line in the program.
N2	16388	position in RAM of [K] or [L] cursor last line any input or editing was done.
2	16390	statement number of [>] cursor.
X2	16392	VARS } see section 2a of Appendix 2.
X2	16394	E–LINE } see section 2a of Appendix 2.
X2	16396	D–FILE } see section 2a of Appendix 2.
X2	16398	DF–EA } see section 2a of Appendix 2.
X2	16400	DF–END } see section 2a of Appendix 2.
X1	16402	number of lines in lower part of screen, including blank line separating the two parts.
2	16403	statement number of first line on screen. The LIST statement sets this, and scrolling alters it.
2	16405	address of character or token preceding the [S] marker.
2	16407	statement number to which CONTINUE jumps.
N1	16409	sundry flags which control the syntax analysis.
N2	16410	address of next item in syntax table (very unlikely to be useful).
U2	16412	"seed" for the random number generator. This is set by the RANDOMISE statement, q.v., and updated each time RND is called: see note below.
U2	16414	number of frames displayed since the ZX-80 was switched on (more exactly, the remainder when this is divided by 65536). While a picture is on the screen, this number is incremented 50 times

ClibPDF - www.fastio.com

per second in the UK version, 60 times per second in the US version.

N2	16416	address of 1st character of 1st variable name in last LET, INPUT, FOR, NEXT, or DIM statement. Thus in LET ABC=PEEK(16416) ABC is assigned the remainder when the address of the A is divided by 256. (Not likely to be very useful).
N2	16418	value of the last expression or variable: in practice this is the parameter of PEEK so that PEEK(16418) yields 34 and PEEK(16419) yields 64 always.
X1	16420	position on line of next line character to be written to screen: 33 = lefthand column, 32 = second from left, etc, up to 2 = righthand column. Also, 1 = 1st column in next line because current line is full, 0 = 1st column on next line because end-of-line has been signalled. Thus after a PRINT not ending in a comma or a semi-colon PEEK(16420) always yields Ø. It only yields 33 if the screen is empty (eg after CLS).
X1	16421	position of current line on screen: 23 = top line, 22 = second line down, etc.
X2	16422	address of the character after the closing bracket of the call of PEEK, or of the newline at the end of the POKE statement.

The first line of program begins at address 16424.

The function RND generates a pseudo-random number from the current "seed" as follows:

let n be the seed.

if n is zero, take n = 65536 instead.

let m be the remainder when n*77 is divided by 65537.

if m is 65536, take m = o instead.

the result of RND(x) is now x*m/65536.

the new seed is m.

ClibPDF - www.fastio.com

The TAB function in printing can be implemented as follows:

```
1ØØØ  REM   GO SUB  1Ø5Ø TABS TO
1ØØ1  REM   COLUMN I
1Ø1Ø  REM   GO SUB  1Ø4Ø TABS TO
1Ø11  REM   COLUMN I OUTPUTTING
1Ø12  REM   AT LEAST ONE SPACE
1Ø3Ø
1Ø4Ø  PRINT " ";
1Ø5Ø  IF I+PEEK(1642Ø) = 33 THEN RETURN
1Ø6Ø  IF I=Ø THEN IF PEEK(1642Ø)<2
      THEN RETURN
1Ø7Ø  GO TO 1Ø4Ø
```

ClibPDF - www.fastio.com

INDEX

ClibPDF - www.fastio.com

ABS 91, 112
Addition 35
AND. 50
Arithmetic overflow. 36
Arrays. 89
BASIC. 15, 103
Bit 87
Brackets, use of. 33
Branches 43
BREAK. 83, 116
Byte 87
Character set 75, 116
CHR$(n) 75
CLEAR 89, 115
CLS 53, 115
CODE 77, 111
Codes (for different characters) 75
Commands. 16
Command mode 23
Connections to tape recorder. 11
Connection to television 9
CONTINUE 84, 113
Control character 71
Control variable 57
Current line pointer 26
Cursor. 22, 104
– keyword cursor 22, 104
– letter cursor 23, 104
Cursor control keys 26, 103
DIM 89, 115
Display 9, 106
Display file. 108
Division. 36
EDIT 26
Element – see Array
Error codes (or messages) 25, 99
Expressions 35, 110
Expression evaluator 109
Field. 71
Flowchart 32, 43
FOR . . . TO. 61, 115
GO SUB 64, 115
GO TO 47, 115
Graphics characters 78
HOME – see Cursor control keys
IF . . . THEN 48, 113
INPUT 24, 113
Integer variables – see Variables
Iterative programs 57
Jumps 47
Keyboard. 21, 22
Keyword 22, 105
LET 35
Line numbers 17
LIST. 26, 114
LOAD. 11, 112
Literal number 111
Literal string. 71
Loops 61
Multiplication. 35
NEW. 22, 112
NEWLINE 23
NEXT 61, 115
NOT 50
Operators. 35
– arithmetic. 35
– relational 50
– logical 50

ClibPDF - www.fastio.com

OR 50
PEEK 88, 111
POKE 87, 115
Power connections. 9
PRINT 71, 114
Priority (of operators) 37, 111
Program listing 26
RAM. 103
RANDOMISE. 87, 115
Recording programs on tape 11
REM. 63
RETURN. 64, 115
RND. 87, 112
RUBOUT. 24, 105
RUN. 23, 113
SAVE 11, 113
SHIFT. 22
SPACE 105, 110
Stack 107
Statement numbers – see line numbers
STOP 85, 114
String variables – see Variables
STR$ 79
Subroutines 64
Subscripts 89
Subtraction 36
Syntax error marker. 23, 104
System variables 107, 121
TL$ 77
Token 22, 105
Truncation. 36
USR 89, 112
Variables 17
– integer variables 17, 111
– string variables. 17, 110
Zero – confusion between letter O and
number 0. 22

Printed by Heffers Printers Ltd Cambridge England

ClibPDF - www.fastio.com