THE AMSTRAD NOTEPAD *ADVANCED USER GUIDE*

SIGMA

Robin Nixon

THE AMSTRAD NOTEPAD ADVANCED USER GUIDE

ROBIN NIXON
Programs written and documented by Chris Nixon

SIGMA PRESS – Wilmslow, United Kingdom

First published in 1993 by

Sigma Press, 1 South Oak Lane, Wilmslow, Cheshire SK9 6AR, England.

British Library Cataloguing in Publication Data

A CIP catalogue record for this book is available from the British Library.

ISBN: 1-85058-515-6

Typesetting and design by

Sigma Press, Wilmslow

Printed in Malta by
Interprint Ltd.

Distributed by

John Wiley & Sons Ltd., Baffins Lane, Chichester, West Sussex, England.

Acknowledgement of copyright names

Within this book, various proprietary trade names and names protected by copyright are mentioned for descriptive purposes. Full acknowledgment is hereby made of all such protection.

PREFACE

At first sight it might seem a step backwards and a rather surprising move for Amstrad to release a 64K Z80-based 'laptop'-type computer when the market definitely looks like PC-compatible is the way to go, and laptops being the fastest growth area.

But maybe that's the point. There is a large group of people who know that all they need is a cheap, easy-to-use and (as the Notepads proudly proclaim) *user-friendly* interface - without having to learn about using DOS or Windows.

Amstrad are renowned for using tried and tested formulae, which the Z80 certainly is - just look at how well they did with the PCW family. What's more, Z80s are cheap, as are the additional chip sets that go with them and, because of their low power consumption, you get up to 40 hours use out of an NC100 - about 10 times more than with most PC-compatible laptops.

And Amstrad made a very sensible decision in their choice of software. By porting Protext across to it they have a top-selling word-processor also available on a number of platforms, including the Amstrad CPC and PCW, Archimedes, Atari ST, Amiga and PC compatibles. In one go, Amstrad have a product that can be file compatible with just about every other popular computer.

By removing Protext's command mode (well, not entirely, as you'll see later), they came up with a very simple system of colour-coded key combinations so that, no matter where you are in the Notepad, you can move to any other in-built application at a key press. But, in my opinion, one of the best things to be incorporated was the BBC Basic ROM which allows you to adapt the computer fully to your own requirements.

And that is what this book concentrates on. In it you will discover how you can use BBC Basic to write applications to complement seamlessly the in-built applications, with the user never even knowing they are in Basic.

Even if you're not a programmer, the explanations of how the Notepad works should interest you, but if not a large proportion of this book is taken up with ready-made

programs ready for you to type in and run. So, without needing to know a thing about programming, you can add a writing style checker to your Notepad, or there's a full scientific calculator, a food additive database, a graphical world time zone viewer, a mortgage calculator and a whole lot more.

For the more technically minded, full details on the Notepad's firmware calls, input/output ports and system variables are provided, including how you can make use of them yourself. You'll even learn how to create entire system applications to run from a RAM card. Everything you need to know in order to do this is in this book, even down to a fully working Z80 disassembler you can type in and use immediately.

In fact this book is packed with undocumented information about the Notepad (and even the Z80 microprocessor itself) that you are unlikely to find anywhere else. Along with the comprehensive index, you will find it to be a complete, one-stop reference to using and writing Notepad programs, as well as a valuable source of additional software for your Notepad.

Thanks are due to Mark Tilley and Gavin Every at Arnor Ltd (the programming team) as well as to Cliff Lawson the Notepad Project Manager at Amstrad Plc for much of the technical information that appears here. Because these details came directly from the programmers you can be sure that they are as accurate as possible.

Thanks also to John Blackburn for his invaluable assistance in the preparation of this book, and to Richard Russell, the author of the Notepad BBC Basic interpreter, for his assistance with the final manuscript.

Robin Nixon

To Julie and Naomi

Contents

SECTION 1

THE PROGRAMS

Whether or not you are a programmer, this collection of programs has been designed to accompany the applications provided with your Notepad. They have been written in such a way that no knowledge of programming is required to use them and you can call them up by simply pressing [Function][B], so you don't even need to know any Basic commands. Just type in the programs, check them and save them.

On the other hand, detailed descriptions accompany every program in this section, including line by line running commentaries, descriptions of the functions and procedures used, and an explanation of all the main arrays and variables.

Using all this information in conjunction with the listings you will be able to pick up on the various methods used and then incorporate any ideas you like into your own programs. To this end the variety of programs has been kept as wide as possible and a broad range of programming styles and techniques have been used, covering areas such as using the built-in assembler, handling strings, variables and arrays, processing and storing data, directly accessing the display RAM, the non-standard Notepad VDU codes and much more.

All the programs are written in BBC Basic and some contain sections of assembly language to achieve effects that, if written in Basic, wouldn't be possible, would take too much space to write, or would run unacceptably slowly.

The programs are self-contained, and range from a useful scientific calculator to a world clock featuring a map of the earth. There's also a version of the classic game Towers of Hanoi to while away the odd hour, and if you are fond of writing you

might like to pass your efforts through the compact style checker and see how they measure up.

Programmers may be particularly interested in the assembly language routine that performs an instant scan of the keyboard, in a similar fashion to BBC Basic's negative INKEY(-n), (as opposed to Basic's INKEY or INKEY$, which can only return keys at the speed of the current keyboard repeat rate). There's a full chart of the Notepad's character set – useful for designing screen layouts or games, and a complete Z80 disassembler so you can delve into the inner workings of other programs.

THREE GOLDEN RULES

There are three Golden Rules you must bear in mind when typing in program listings, and which you should always follow in order to prevent typing mistakes – or even crashing or erasing programs.

Golden Rule number one:
Make sure you save your work before you try it out. It's very tempting to type RUN every so often to see the effect so far, but even if you save the program first you are strongly advised against it – especially where there is machine code involved, as you could lock-up your Notepad.

Golden Rule number two:
Read the listing carefully. A common error is to confuse any of the following for each other: Lower case "l" (lower L), "i" (lower I) and "1" (one). Another common mistake is to confuse the capital letter "O" with the number "0".

Golden Rule number three:
Don't delete any REM lines or lines containing just a colon ":". As a matter of style most of the programs in this book avoid GOTOs in the main code, but all of them contain at least one ON ERROR GOTO line (and listings from other sources may make more liberal use of GOTOs). So, if any GOTOs happen to point at a line that you've deleted as being *unnecessary* you'll get into all sorts of bother.

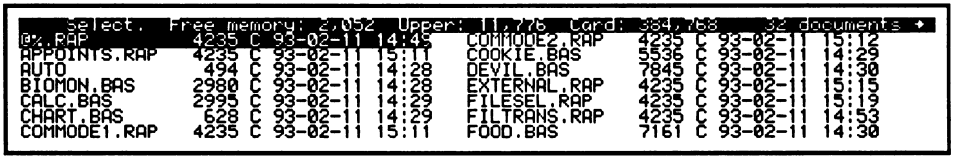
GET IT RIGHT

If all else fails (or even if you simply prefer not to type in the programs), you can order a disk containing all the listings fully tested and ready to run, along with a lead and software to transfer them to the Notepad, using the form in Appendix 6.

Please also remember that a Notepad without a RAM card may only be able to hold two to three or so small to medium Basic programs at any one time. If you really want to make use of your computer and not run into memory storage problems you should buy a RAM card. One such source is also given in Appendix 6.

AUTO

Menu system



AUTO, the menu system for Basic programs

This is the first and most important of the programs because it provides an easy-to-use interface for running the other programs, without needing to enter BBC Basic's command mode.

It works by taking advantage of a feature built into Notepad Basic which checks for a file called AUTO whenever you enter Basic (normally by pressing [Function][B]). If such a file exists Basic proceeds to load and run it, rather than just dropping into command mode.

USING THE PROGRAM

Type in the listing and save it as AUTO before trying it out. Note that you must NOT call it AUTO.BAS (although all the other programs should use the .BAS extension), or the file will not be recognised by Basic's initialisation routines. It is also essential that you save the program before running it in case you have made any typing mistakes and something goes wrong or, perhaps, you did type it in correctly but accidentally loaded in another program while testing it.

Once saved type:

```
RUN
```

and press [Return]. You should then see the Notepad's standard file selector which you can now use to call up a program in the same way you might select a file for editing in the word processor.

Because this program prevents access to Basic's command mode you might wonder how you are now going to be able to type in more programs. The answer's simple: you can exit from AUTO at any time by pressing [Stop]. You are then dropped into command mode and, if you want to enter a new program, type:

```
NEW
```

and off you go. Or, if you accidentally exited from AUTO you can get back in by pressing [Function][B].

HOW IT WORKS

The program is described as follows, with line numbers on the left and explanations on the right.

30-40	Clear memory and dimension A% so that it's just big enough to hold the machine code which will be assembled.
50-60	Assemble the code and call it.
70	If the first character of the file name is 0 then there is no file name so print a message and exit.
80-110	Copy the file name returned by the machine code routine into the variable R$.
120	Check whether the file name has an extension of .BAS. If not, it is not a Basic program (at least, not as far as the program is concerned, because the .BAS extension is the recommended method of declaring whether a file is a Basic program). So, if not, refuse to attempt to run it and call the file selector again.
130	Load the selected program into memory (replacing AUTO) and run it.
160-190	Prepare for a two-pass assembly using a FOR...NEXT loop and set the program point (P%) to the machine code destination address at the start of each pass.
200	Call the Notepad's built-in File selector.
210	Set the register DE to point to the start of where the File selector will have stored a file name if one was selected.
220-250	If the Carry flag is set then no file was selected because the user pressed [Stop], so set the first byte of the file name which is to be passed back to Basic (pointed to by DE) to a zero to indicate this, and then return.
270-340	A file name was selected so copy all the characters in the name to a known location in memory starting at 'buffer' and then return.

Functions and procedures

PROCselect	Assembles the machine code required to call the File selector and then return the name of any selected file to a known area of memory that can be accessed from Basic.

Main variables and arrays

A%	22 bytes of memory used to hold the assembled machine code.

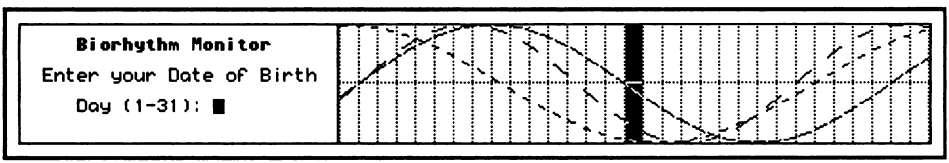
buffer	The start of 13 bytes of memory within A% which are used to hold any file names.
R$	Holds a copy of a selected file name ready to CHAIN it in.
J%	Temporary loop counter used to control the copying of a file name from memory into the string R$.
P%	The pointer to where machine code is to be assembled by the assembler.
found	Start of machine routine where a file name has been found.
loop	Label marking the start of the machine code loop to copy a file name to a known location, useable from Basic.

The program

```
   10 REM BBC Basic menu system
   20 :
   30 CLEAR
   40 DIM A% 22
   50 PROCselect
   60 CALL A%
   70 IF buffer?0 = 0 THEN CLS:PRINT "Press [Function][B] for
menu.":PRINT:END
   80 R$=""
   90 FOR J%=0 TO 11
  100   IF buffer?J% THEN R$=R$+CHR$(buffer?J%) ELSE J%=12
  110 NEXT
  120 IF RIGHT$(R$,4) <> ".BAS" THEN GOTO 60
  130 CHAIN R$
  140 :
  150 DEF PROCselect
  160 FOR PASS=0 TO 2 STEP 2
  170 P%=A%
  180 [
  190 OPT PASS
  200 CALL &B8C3
  210 LD DE,buffer
  220 JR C,found
  230 LD A,0
  240 LD (DE),A
  250 RET
  260 .found
  270 LD B,12
  280 .loop
  290 LD A,(HL)
  300 LD (DE),A
  310 INC HL
  320 INC DE
  330 DJNZ loop
  340 RET
  350 .buffer
  360 ]
  370 NEXT
  380 ENDPROC
```

BIOMON.BAS

Biorhythm Monitor



BIOMON.BAS, showing physical, intellectual and emotional strength

The study of biorhythms is based on the ancient belief that our physical, intellectual and emotional states run in fixed, regular cycles from the day that we are born. Whether you believe this or not, it means that we can calculate these states for any day of a person's life, given just their date of birth and today's date. And as the cycles are regular, they lend themselves to rather attractive looking sine wave charts, which used to be hand-drawn by astrologists in the days before computers.

However, this is extremely time consuming, and as the formulae for calculating the number of days that lie between a person's birthday and any other date are complex, they make an ideal subject for a computer program to handle. In fact, there probably isn't a computer in existence that hasn't had a biorhythm calculator written for it (as a demonstration of the machine's graphics capabilities as much as for any other reason).

The program BIOMON.BAS uses the standard biorhythm cycles to plot a personal chart for a 35-day period with today's date in the middle. It differs slightly from other programs of this type by telling the user in plain English what each line on the chart represents, and whether today's level is good or bad for that particular chart line.

USING THE PROGRAM

Type in the listing and save it as BIOMON.BAS before trying it out. This is essential in case you have made any typing mistakes and something goes wrong.

Now type:

```
RUN
```

and press [Return] and you will be prompted to enter your date of birth. Type in the day of the month on which you were born, and press [Return]. Then type in the number of the month, press [Return], and then enter the year – you can enter this either in full, as in 1964, or in shorthand, as in 64. Don't forget to press [Return] after

entering the year. If you make a mistake at any one of these three stages, Biomon will repeat that stage until it's happy with the result.

Biomon now makes a final, more involved check to see if the date you have just entered actually existed. It does this by checking that the month you have entered has at least the number of days you have given as the day of the month on which you were born, and leap years are taken into account at this stage. If Biomon finds an error it will report *Bad date – press SPACE*, and you will have to re-enter the whole date.

If all is well with the date, the screen will clear and the plotting will begin. Each line is drawn with a different dot pattern, making it easier to tell them apart.

When the plot is finished (it takes about half a minute), the box on the chart representing today's date will be highlighted in inverse, and the window on the left will show a key for each of the three chart lines together with a one-word summary of how good or bad each one is today.

If you can't wait for the full plot you can cut it short by pressing [Q], which jumps straight to the summary screen – useful if you're not interested in seeing the general pattern of cycles.

HOW IT WORKS

40	Points the Basic error handler to Biomon's own error handling routine at line 940.
50	Calls PROCsetup to initialise everything.
60	Calls PROCinput to get a birth date, followed by PROCdays to count the number of days that have elapsed. Then calls PROCgraph to plot the chart, and finally PROCreport to summarise the current state of each chart line.
90-130	Initialise the main graphics constants. Altering these values will have a major effect on the resulting chart.
140-180	Dimension all arrays and read in all the data.
190	Draws a box enclosing the entire screen area.
200	Draws a vertical line to separate the graph area from the information window.
210	Prints the program title.
220	Calls the date prompt window into operation and returns from the procedure.
250	Flushes the keyboard buffer by calling INKEY$(0) until no keys are returned.
260	Prompts for the date of birth to be entered.

270	Repeatedly prompts for the day of the month until the input is within legal limits.
280	Repeatedly prompts for the month until the input is within legal limits.
290	Repeatedly prompts for the year until the input is within legal limits.
300-320	Check to see if the date specified exists.
330	Informs the user if the date doesn't exist and waits for the message to be acknowledged.
340	Repeats the entire input process if the date entered doesn't exist.
410	Extracts the current day of the month and the current year from the system clock.
420	Extracts the name of the month from the system clock and converts it into a number between 1 and 12, by comparing it against each entry in the array m$().
430	Multiplies the elapsed years by 365 to get the rough number of days involved.
440-450	Adjust the days according to the birth month and current month.
460	Adjusts the days further according to the day of the month of birth and the current day of the month.
470-520	Adjust the days further according to the number of leap days involved.
560	Calculates whether y% is a leap year or not.
590-600	Clear the information box and print the current date, together with the birthdate being plotted, in preparation for the plot.
610	Sets up a graphics window and origin, and clears the new graphics window.
620-630	Draw the chart axis, and dotted boxes to delimit each of the 35 days to be charted.
640	Sets the start day to be 17 days ago and starts the main FOR...NEXT loop for the X coordinate, checking for the [Q] key at the start of each loop.
650	If [Q] wasn't pressed, calls PROCbio() inside a further FOR...NEXT loop to plot the current Y position of each line.
660	Ends the main plot loop, inverts today's box on the chart and exits the procedure.
690	Fetches the cycle length for the current line and whether it is in its dot or dash phase.
700	Plots a new point for the current line if it is inside its dot phase.

710	Checks to see if current line's dot or dash phase has reversed. If so, flips the line's dot phase flag.
720	Stores the new dot phase flag setting for the current line.
760-830	Draw a short sample of each line's dot pattern to be used as a key, during which time a score for each is calculated in line 810, representing an entry in the array of comments well$(). Display the report header when the loop is complete.
840-860	Print the name of each cycle, together with a single-word comment from the list held in well$(), pointed to by the relevant entry in the score table well%() (which was calculated back in line 810).
870	Waits until [Space] is pressed before returning from the procedure.
890	Holds the data for the number of days in each month.
900	Holds the names of each month, as used in line 420 to calculate the number of the current month.
910	Holds the cycle length in days for each of the three chart lines, followed by the length of the dot and dash for that line, followed by the initial dot-dash phase to start with.
920	Holds the single-word comments which are used at the end of each plot to summarise the state of each chart line.
940	Points the Basic error handler to a full error report in the event of a further error occurring while attempting to run the menu program AUTO. This is in case AUTO isn't present on your Notepad.
950	Attempts to run the menu program AUTO if the error was generated by pressing the [STOP] key.
960	If the error was caused by something else, or if AUTO isn't on your Notepad, a full error report is displayed.
970	After the error report the Notepad will be left in BBC Basic, so this message is displayed to remind users of how to return to the Notepad main menu.

Functions and procedures

PROCsetup	Dimensions arrays and reads in all data, initialises main variables and draws a box.
PROCinput	Prompts for the day, month and year of birth.
PROCwin1	Sets up a text window for the birthdate prompt.
PROCwin2	Sets up a date window for the birthdate input.
PROCdays	Calculates number of days elapsed since birth.

PROCgraph	Plots a biorhythm chart for the birthdate just entered.
PROCbio	Plots a dot at the current X coordinate for any of the three lines. Called by PROCgraph.
PROCreport	Displays a key for the graph and a single-word summary for each line.
FNleap	Returns TRUE or FALSE according to whether the passed variable is a leap year.

Main variables and arrays

m%()	Number of days in each month.
m$()	Abbreviation of month names.
period%()	Length of each cycle in days.
gap%()	Length of each line's dot and dash in pixels.
flag%()	Keeps count of each line's current dot or dash phase.
well%()	Numbers representing how good or bad each cycle is today.
well$()	Store of single-word summaries.
yc%	Y centre of graphic window.
xc%	X centre of graphic window.
px%	Number of pixels per day horizontally.
xm%	Width of graphic window.
ym%	Height of graphic window.
d%	Number of days since birth, as returned by PROCdays.
sd%	Number of days since birth up to 15 days ago (the start of the plot).
d1%	Day of birth.
m1%	Month of birth.
y1%	Year of birth.
d2%	Today's day of the month.
m2%	Today's month.
m3%	Today's year.
t%	Number of current line being processed (1, 2 or 3).
quit%	Whether Q was pressed during plot.

The program

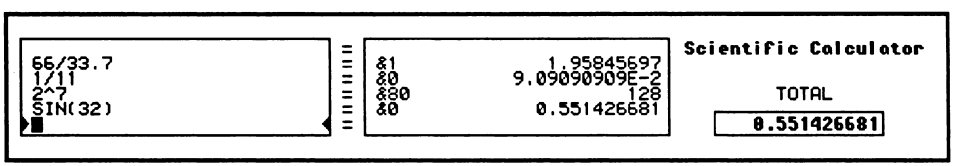
```
10 REM Biorhythms
20 :
30 CLS
40 ON ERROR GOTO 940
50 PROCsetup
```

```
  60 REPEAT:PROCinput:PROCdays:PROCgraph:PROCreport:UNTIL FALSE
  70 :
  80 DEF PROCsetup
  90 yc%=31:REM Y Centre of graphic window
 100 xc%=155:REM X Centre of graphic window
 110 px%=10:REM No. of pixels per day horizontally
 120 xm%=310:REM Width of graphic window
 130 ym%=61:REM Height of graphic window
 140 DIM m%(12):FOR d%=1 TO 12:READ m%(d%):NEXT
 150 DIM m$(12):FOR d%=1 TO 12:READ m$(d%):NEXT
 160 DIM period%(3),gap%(3,2),flag%(3),well%(3),well$(7)
 170 FOR t%=1 TO 3:READ period%(t%),gap%(t%,0),gap%(t%,1),flag%(t%):NEXT
 180 FOR w%=1 TO 7:READ well$(w%):NEXT
 190 MOVE 0,0:DRAW 479,0:DRAW 479,63:DRAW 0,63:DRAW 0,0
 200 MOVE 167,0:DRAW 167,63:PRINT TAB(5,1);CHR$(17);
 210 PRINT"Biorhythm Monitor";CHR$(18)
 220 PROCwin1:CLS:ENDPROC
 230 :
 240 DEF PROCinput
 250 REPEAT:UNTIL INKEY$(0)="":REM Flush keyboard buffer
 260 PROCwin1:CLS:PRINT TAB(1,0);"Enter your Date of Birth";:PROCwin2:
REPEAT
 270 REPEAT:CLS:INPUT"    Day (1-31): "d1%:UNTIL d1%>0 AND d1%<32
 280 REPEAT:CLS:INPUT"    Month (1-12): "m1%:UNTIL m1%>0 AND m1%<13
 290 REPEAT:CLS:INPUT"    Year (1900-): "y1%:UNTIL y1%<100 OR y1%>1900
 300 IF y1%<100 y1%=y1%+1900
 310 leg%=TRUE:IF y1%<1900 OR y1%>2020 leg%=FALSE
 320 IF d1%>m%(m1%)+FNleap(y1%)*(m1%=2) leg%=FALSE
 330 IF leg%=0 CLS:PRINT CHR$(17)"  Bad date - press SPACE";CHR$(18);:
g%=GET
 340 UNTIL leg%:ENDPROC
 350 :
 360 DEF PROCwin1:VDU 28,1,6,26,3:ENDPROC
 370 :
 380 DEF PROCwin2:VDU 28,1,5,26,5:ENDPROC
 390 :
 400 DEF PROCdays
 410 d2%=VAL(MID$(TIME$,5,2)):y2%=VAL(MID$(TIME$,12,4))
 420 m2%=0:REPEAT:m2%=m2%+1:m$=m$(m2%):UNTIL m$=MID$(TIME$,8,3)
 430 d%=365*(y2%-y1%)
 440 IF m2%>1 FOR m%=1 TO m2%-1:d%=d%+m%(m%):NEXT
 450 IF m1%>1 FOR m%=1 TO m1%-1:d%=d%-m%(m%):NEXT
 460 d%=d%+d2%-d1%
 470 y%=y1%-y1% MOD 4:REPEAT:y%=y%+4
 480 IF y%<y2% IF FNleap(y%) d%=d%+1
 490 UNTIL y%>y2%
 500 IF y1%=y2% IF FNleap(y1%) AND m1%<3 AND m2%>2 d%=d%+1:ENDPROC
 510 IF FNleap(y1%) AND m1%<3 d%=d%+1
 520 IF FNleap(y2%) AND m2%>2 d%=d%+1
 530 ENDPROC
 540 :
 550 DEF FNleap(y%)
 560 IF y%MOD4=0 AND (y%MOD100<>0 OR y%MOD400=0) THEN =TRUE ELSE =FALSE
 570 :
 580 DEF PROCgraph
 590 PROCwin1:CLS:PRINT TAB(3,0);"Plot on ";MID$(TIME$,5,11)
 600 PRINT TAB(4,2);"For DoB ";CHR$(17);d1%;"-";m1%;"-";y1%;CHR$(18)
 610 VDU24,168;1;478;62;29,168;1;:CLG
 620 MOVE 0,yc%:PLOT 21,xm%,yc%:MOVE xc%-px%/2,yc%:PLOT 1,px%,0
 630 FOR x%=0 TO xm% STEP px%:MOVE x%,0:PLOT 21,x%,ym%:NEXT
```

```
  640 sd%=d%-17:quit%=FALSE:FOR x%=0 TO xm%:IF INKEY(0)=81
quit%=TRUE:x%=xm%
  650 IF NOT quit% FOR t%=1 TO 3:PROCbio(t%):NEXT
  660 NEXT:MOVE xc%-4,0:PLOT 102,xc%+4,ym%:VDU26:ENDPROC
  670 :
  680 DEF PROCbio(t%)
  690 period%=period%(t%):flag%=flag%(t%):gap%=gap%(t%,flag%)
  700 IF flag% PLOT 69,x%,yc%+(yc%*SIN(2*PI/period%*(sd%+x%/px%)))
  710 IF x% MOD gap%=0 flag%=flag%+1:IF flag%=2 flag%=0
  720 flag%(t%)=flag%:ENDPROC
  730 :
  740 DEF PROCreport
  750 PROCwin1:CLS
  760 FOR t%=1 TO 3:period%=period%(t%):flag%=0
  770 FOR x%=8 TO 28:gap%=gap%(t%,flag%)
  780 IF flag% PLOT 69,x%,36-t%*8
  790 IF x% MOD gap%=0 flag%=flag%+1:IF flag%=2 flag%=0
  800 NEXT
  810 well%(t%)=(yc%+(yc%*SIN(2*PI/period%*(sd%+(xc%+px%)/px%))))/
(ym%/6)+2
  820 IF well%(t%)>7 well%(t%)=7
  830 NEXT:PRINT TAB(0,0);CHR$(19);"Your Constitution Today Is";CHR$(20)
  840 PRINT TAB(5,1);"Physically";SPC(5);well$(well%(1));
  850 PRINT TAB(5,2);"Emotionally";SPC(4);well$(well%(2));
  860 PRINT TAB(5,3);"Intellectually ";well$(well%(3));
  870 REPEAT:UNTIL INKEY(0)=32:ENDPROC
  880 :
  890 DATA 31,28,31,30,31,30,31,31,30,31,30,31
  900 DATA Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec
  910 DATA 23,8,8,0,28,1,8,0,33,4,4,0
  920 DATA Awful,Poor,Fair,Normal,Good,Great,Superb
  930 :
  940 ON ERROR GOTO 960
  950 VDU 26:CLS:IF ERR=17 THEN CHAIN "AUTO"
  960 REPORT:PRINT" at line ";ERL
  970 PRINT:PRINT"Press [Function][X] for Notepad Main Menu"
```

CALC.BAS

Scientific Calculator



CALC.BAS, a powerful scientific calculator

Most computer users complain at some time or another about the lack of a *real* calculator program for their system, which on the whole is a justifiable complaint –

especially in the PC compatible world, where hardware costing thousands of pounds often comes with no software installed.

By contrast, Amstrad Notepad users are lucky enough to have a built-in calculator featuring a large, friendly display. But sometimes it just isn't up to the job, especially where you need to use scientific functions, or recall the results of previous calculations.

The program CALC.BAS aims to solve some of these frustrations by providing a large *scratch pad* on to which you can jot calculations of a highly complex nature. You are allowed to use all the functions normally available from BBC Basic inside your calculations, and you enter these in a large window on the left of the screen, while a matching window on the right displays the results of each calculation.

The windows scroll in both directions and are synchronised, allowing you to recall previous entries and their results. You can even modify earlier calculations without having to type in the whole lot again.

Calc remembers the result of the current calculation and displays it in a separate, stationary window so that you can scroll freely through several screens of work and not lose your position. It clears this value when you next enter a line.

A special feature of CALC is its ability to treat the value in this window as a *running total* accumulator. So, putting a +, –, * or / symbol at the start of your calculation turns it into an expression that takes the value in the Total window as its input. (See *USING THE PROGRAM* for a more detailed explanation of how this works).

Although it doesn't support the use of variables or *memories*, you will be surprised at how useful this program is (it even gives the result of each calculation in hexadecimal, for any programmers who wouldn't otherwise have found Calc suited to their particular needs).

USING THE PROGRAM

Type in the listing and save it as CALC.BAS before trying it out. Type:

```
RUN
```

and the cursor will now be sitting in the bottom left of the Input window, between the two arrows that indicate where your input will go. Now type in any number, or legal BBC Basic expression such as:

```
30+(SIN(45))
```

Notice that your input is shown in bold text as you type. In fact, the contents of the bottom line of the Input window is always shown in bold, because when you are

scrolling through previous calculations it serves to highlight the one currently under the cursor.

Press [Return] and Calc will scroll both the Input and Result windows up one line, and the line in the Result window opposite the expression you have just entered shows the result in both hexadecimal (on the left) and decimal (on the right). The Total window just gives the result in decimal.

To try out the scrolling facility enter a few simple expressions, until the first one has completely scrolled off the top of the display. Now press [Up] a few times, watching as your previous entries (and their results) scroll back into view. Note the expressions turning bold one by one as they pass through the bottom line of the Input window.

Now stop at any time and edit an expression (one of the features of Calc is that it is permanently in edit mode, so you can change whatever is under the cursor at any time). Remember that you MUST press [Return] to register the change – if you move off the line with [Up] or [Down], Calc will restore the old contents of the line.

You might think that Calc is limited by the seeming inability to *pass on* any results to the next calculation you enter. For example, if you were to enter:

```
10
```

then both the Result and Total windows would show the answer 10. But what if you wanted to add 10 to the result from the last calculation? Even the most basic pocket calculators allow you to do this by default. If you enter a sum like 4 + 30 + 15 on any calculator, it displays the interim total each time another operator key is pressed.

Calc allows you to emulate this quite well, simply by adding one of the four basic arithmetic operators to the start of an expression. For example, if you were to enter this line instead:

```
+10
```

Calc assumes that you meant *add 10 to the current running total* – which is exactly what is wanted. The same goes for more complex expressions such as:

```
*(COS(100)+PI)/9.073
```

which means *multiply the current total by the result of this expression.* In Basic the process might look something like this:

```
total=total*((COS(100)+PI)/9.073)
```

Notice the added brackets around everything after the *. This is because syntactically, Calc evaluates the whole expression (minus the operator, of course) BEFORE applying it to the total.

The next time you enter an expression without a preceding operator Calc clears the running total. If you would prefer to clear it to zero, just enter 0. Or to clear the entire scratch pad, type:

```
CLEAR
```

in upper case (because BBC Basic requires upper case for all keywords) before pressing [Return], and then confirm your decision with the [Y] key.

Full line editing is provided by Calc, and while it may not be quite as good as the Notepad's default line editor, it does include all the standard editing key functions you would expect. Here's a complete list of the movement and editing keys used in Calc:

[Right]	Cursor right – Moves the cursor one character to the right.
[Left]	Cursor left – Moves the cursor one character to the left.
[Up]	Previous line – Scrolls the Input and Result windows down, and places the previous expression on the editing line.
[Down]	Next line – Scrolls the Input and Result windows up, and places the next expression on the editing line.
[Del->]	Delete character under cursor – The rest of the line is shunted to the left, while the cursor remains stationary.
[<-Del]	Delete character to left of cursor – The rest of the line is shunted to the left, and the cursor also moves one position to the left.
[Control][E]	Delete to end of line – All characters to the right of the cursor are deleted, as well as the character under the cursor (ideal for clearing an old line ready for a new expression).

HOW IT WORKS

30	Calls the setup procedure, and points the Basic error handler to Calc's own error handling routine.
40	Endlessly calls PROCinput until [Stop] is pressed.
70	Draws the editing line arrows.
80-100	Draw all three window borders.
110	Prints a column of equal signs between the main windows and prints the Total window's title.
120	Prints the program title.
130	Dimensions the arrays, calls PROCclear to print a 0 in the Total window, and tells Basic to display all numbers to 10 significant figures (the maximum).
160	Runs through the arrays A$() and B$(), setting all elements to "" (empty).

170	Resets both array pointers, clears the total and displays it in the Total window.
190-250	Set up four text windows. In order of appearance they are the editing line, the Input window, the Results window and the Total window.
280	Sets up the edit window, pulls the current calculation from A$() into e$, gets its length, sets the editing cursor to the left edge of the window, prints the expression in bold, starts the main input loop and reads a key press into key%.
290-350	Check the key in key%, and carry out the appropriate editing or movement function.
360	If the key press was a normal character, inserts it into e$ at the current position by calling PROCinsert.
370	When [Return] is pressed, checks if CLEAR was typed. If so, calls PROCwipe – but if e$ is empty, it's forced to contain 0 for the sake of appearance.
380	Puts the new expression into A$() at the current position, calls PROCcalc to evaluate it and update B$(), and advances the array pointer ptr% (and max% if ptr% was already at the highest element used so far).
390	Checks that max% hasn't exceeded the limits of the arrays A$() and B$() – otherwise adjusts max%.
400	Draws the new Input and Result window contents and returns.
430-440	If x% isn't already at the left-hand side, move it left and redraw the editing line to show the new cursor position.
470-480	If x% isn't already at the end of the line, move it right and redraw the editing line to show the new cursor position.
510-520	If the pointer isn't already at the start of the array, move it to the previous expression, display the new window contents and fetch the new expression for editing.
550-560	If the pointer isn't already at the last entry in the array, move it to the next expression, display the new window contents and fetch the new expression for editing.
590	Calls PROClist to update the main windows, pulls the current calculation from A$() into e$, gets its length, sets the editing cursor to the left edge of the window, sets up the edit window and prints the expression in bold.
630-640	Insert the character key% into e$, if it isn't already at maximum length.
670-690	Remove character to left of current character from e$, unless at start of e$.

730-740	Remove current character from e$, unless at end of e$.
770-790	Truncate e$ at the current position, unless at end of e$.
820-830	Print e$ in bold, followed by the current character in inverse to act as the cursor.
910-920	Clear Input window and fill it from A$(), starting from either five lines before the current line, or the start of the array if less than five entries exist.
950-960	Clear Result window and fill it from B$(), starting from either six lines before the current line, or the start of the array if less than six entries exist.
990	Fetches the current expression from A$() into e$, and exits if it's a null string.
1000	Splits the first character of e$ and puts its ASCII code into o%, to check for an operator on the next line.
1010	If o% is one of the four main maths symbols, sets the flag carry% to TRUE.
1020	If carry% is TRUE, passes the operator and the rest of e$ to FNcarry() to do an accumulative operation on the expression – otherwise, just evaluates e$ as normal. Either way, puts the result in the accumulator "tot".
1030	Makes separate strings holding the decimal and hex versions of the new total.
1040	Joins both strings together, padding so that the hex number is on the left and the decimal is on the right, puts the resulting string into B$() at the current position, then displays the new total.
1080-1090	Convert *tot* into a string, padded out to fill the Total window exactly, and print it bold in that window.
1130-1160	Perform addition, subtraction, multiplication or division with the accumulator *tot* and the result of the expression in e$.
1190-1210	In answer to the user typing CLEAR, display a safety message on the editing line in bold. If the user presses [Y] in response, clear the entire calculator with PROCclear.
1220	Calls PROCnewline to redraw the main windows and put the current calculation back in the editing line.
1270	Resets Basic's numeric accuracy to normal and attempts to run the menu program AUTO if the error was generated by pressing the [Stop] key.
1280	If the error was *No such file*, AUTO isn't on your Notepad so jump to the full error report.
1290-1300	If the program gets to here an illegal calculation was made. The user is informed and asked to acknowledge by pressing [Space].

	PROCnewline is called to redraw the main windows and redisplay the current calculation on the editing line, and a direct jump is made back to main loop at line 40. **Important:** This can only be allowed to happen a certain number of times before the Basic stack overflows with PROC calls that the error handler has jumped out of before reaching the ENDPROC.
1310	Displays a full error report.
1320	After the error report the Notepad will be left in BBC Basic, so this message is displayed to remind users of how to return to the Notepad main menu.

Functions and procedures

PROCsetup	Draws the screen and sets up arrays and main variables.
PROCclear	Clears the Input and Result windows, resets the Total window.
PROCinput	Takes input from the keyboard, and calls relevant routines for inserting and deleting characters, or moving around.
PROCleft	Moves the cursor one character to the left.
PROCright	Moves the cursor one character to the right.
PROCup	Scrolls both windows down, and places the previous expression on the editing line.
PROCdown	Scrolls both windows up, and places the next expression on the editing line.
PROCnewline	Redraws both windows at the current position and fetches the current expression for editing.
PROCinsert	Inserts a character into the input line.
PROCdel1	Performs [<- DEL].
PROCdel2	Performs [DEL ->].
PROCdel3	Performs [Control][E].
PROChilite	Prints the current expression in bold, and inverses the current character to act as a screen cursor.
PROClist	Main routine to update both Input and Result windows by calling PROClistlt and PROClistrt.
PROClistlt	Redraws the left-hand (Input) window.
PROClistrt	Redraws the right-hand (Result) window.
PROCcalc	Evaluates the expression just entered, and decides whether to make this the new total, or take the current total as the expression's input, on the basis of the first character. If the first character is + – * or /, FNcarry() is called to evaluate the new total.
PROCshowacc	Displays the current running total in the Total window.

PROCwipe	Displays a safety prompt before calling PROCclear to clear the entire "scratch pad".
FNcarry()	Takes the current total and either adds to it, subtracts from it, multiplies it by or divides it by the result of the expression just entered.

Main variables and arrays

A$()	The input array, which holds all the calculations.
B$()	The results array, as displayed in the Results window.
max%	Pointer to the highest element of A$() and B$() currently used.
ptr%	Pointer to the current element of A$() being edited, and also the current element of B$() into which the result will be placed.
tot	The running total *accumulator*.
key%	The current key press being examined.
e$	The expression currently being edited.
l%	The current length of the expression being edited.
x%	The current cursor position on the editing line.
o%	The mathematical operator at the start of the expression, if any.
t$	Result of current expression in decimal
h$	Result of current expression in hexadecimal.

The program

```
   10 REM Scientific Calculator
   20 :
   30 PROCsetup:ON ERROR GOTO 1270
   40 REPEAT:PROCinput:UNTIL FALSE
   50 :
   60 DEF PROCsetup
   70 VDU 26:CLS:PRINT TAB(0,6);CHR$(27);CHR$(16);TAB(26,6);CHR$(27);
CHR$(17);
   80 MOVE 0,6:DRAW 162,6:DRAW 162,57:DRAW 0,57:DRAW 0,6
   90 MOVE 180,6:DRAW 340,6:DRAW 340,57:DRAW 180,57:DRAW 180,6
  100 MOVE 364,6:DRAW 452,6:DRAW 452,18:DRAW 364,18:DRAW 364,6
  110 FOR y%=1 TO 6:PRINT TAB(28,y%);"=":NEXT:PRINT TAB(66,4);"TOTAL";
  120 PRINT TAB(58,1);CHR$(17);"Scientific Calculator";CHR$(18)
  130 DIM A$(100),B$(100):PROCclear:@%=&A0C:ENDPROC
  140 :
  150 DEF PROCclear
  160 FOR p%=0 TO 100:A$(p%)="":B$(p%)="":NEXT
  170 max%=0:ptr%=0:tot=0:PROCshowacc:ENDPROC
  180 :
  190 DEF PROCwinin:VDU 28,1,6,25,6:ENDPROC
  200 :
  210 DEF PROCwinlist:VDU 28,1,5,25,1:ENDPROC
  220 :
  230 DEF PROCwintot:VDU 28,31,6,55,1:ENDPROC
  240 :
```

```
250 DEF PROCwinacc:VDU 28,61,6,74,6:ENDPROC
260 :
270 DEF PROCinput
280 PROCwinin:e$=A$(ptr%):x%=1:l%=LEN(e$):PROChilite:REPEAT:key%=GET
290 IF key%=242 PROCleft
300 IF key%=243 PROCright
310 IF key%=240 PROCup
320 IF key%=241 PROCdown
330 IF key%=127 PROCdel1
340 IF key%=33 PROCdel2
350 IF key%=5 PROCdel3
360 IF key%<>33 AND key%<>5 AND key%>31 AND key%<127 PROCinsert
370 UNTIL key%=13:IF e$="CLEAR" PROCwipe:ENDPROC ELSEIF e$="" e$="0"
380 A$(ptr%)=e$:PROCcalc:ptr%=ptr%+1:IF ptr%>max% max%=max%+1
390 IF max%>100 max%=max%-1:ptr%=ptr%-1
400 PROClist:ENDPROC
410 :
420 DEF PROCleft
430 IF x%=1 ENDPROC
440 x%=x%-1:PROChilite:ENDPROC
450 :
460 DEF PROCright
470 IF x%=l%+1 ENDPROC
480 x%=x%+1:PROChilite:ENDPROC
490 :
500 DEF PROCup
510 IF ptr%=0 ENDPROC
520 CLS:ptr%=ptr%-1:PROCnewline:ENDPROC
530 :
540 DEF PROCdown
550 IF ptr%=max% ENDPROC
560 CLS:ptr%=ptr%+1:PROCnewline:ENDPROC
570 :
580 DEF PROCnewline
590 PROClist:e$=A$(ptr%):x%=1:l%=LEN(e$):PROCwinin:PROChilite:ENDPROC
600 :
610 DEF PROCinsert
620 IF l%=24 ENDPROC
630 e$=LEFT$(e$,x%-1)+CHR$(key%)+RIGHT$(e$,l%+1-x%)
640 l%=l%+1:x%=x%+1:PROChilite:ENDPROC
650 :
660 DEF PROCdel1
670 IF x%=1 ENDPROC
680 e$=LEFT$(e$,x%-2)+RIGHT$(e$,l%+1-x%)
690 x%=x%-1:l%=l%-1:PROChilite:ENDPROC
700 :
710 DEF PROCdel2
720 IF x%=l%+1 ENDPROC
730 e$=LEFT$(e$,x%-1)+RIGHT$(e$,l%-x%)
740 l%=l%-1:PROChilite:ENDPROC
750 :
760 DEF PROCdel3
770 IF x%=l%+1 ENDPROC
780 e$=LEFT$(e$,x%-1)
790 l%=x%-1:PROChilite:ENDPROC
800 :
810 DEF PROChilite
820 c$=MID$(e$,x%,1):IF c$="" c$=" "
830 CLS:PRINT CHR$(17);e$;TAB(x%-1,0);CHR$(14);c$;CHR$(15);CHR$(18);
840 ENDPROC
```

```
  850 :
  860 DEF PROClist
  870 PROCwinlist:CLS:IF ptr%<>0 PROClistlt
  880 PROClistrt:ENDPROC
  890 :
  900 DEF PROClistlt
  910 IF ptr%<5 top%=5-ptr%:p%=0 ELSE top%=0:p%=ptr%-5
  920 FOR y%=top% TO 4:PRINT TAB(0,y%);A$(p%);:p%=p%+1:NEXT:ENDPROC
  930 :
  940 DEF PROClistrt
  950 PROCwintot:CLS:IF ptr%<6 top%=5-ptr%:p%=0 ELSE top%=0:p%=ptr%-5
  960 FOR y%=top% TO 5:PRINT TAB(0,y%);B$(p%);:p%=p%+1:NEXT:ENDPROC
  970 :
  980 DEF PROCcalc
  990 e$=A$(ptr%):IF e$="" ENDPROC
 1000 o%=ASC(LEFT$(e$,1))
 1010 IF o%=42 OR o%=43 OR o%=45 OR o%=47 carry%=TRUE ELSE carry%=FALSE
 1020 IF carry% tot=FNcarry(o%,RIGHT$(e$,LEN(e$)-1)) ELSE tot=EVAL(e$)
 1030 t$=STR$(tot):h$="&"+STR$ (tot)
 1040 B$(ptr%)=h$+STRING$((25-LEN(h$))-LEN(t$),CHR$(32))+t$:PROCshowacc
 1050 ENDPROC
 1060 :
 1070 DEF PROCshowacc
 1080 t$=STR$(tot):tot$=STRING$(14-LEN(t$),CHR$(32))+t$
 1090 PROCwinacc:CLS:PRINT CHR$(17);tot$;CHR$(18);
 1100 ENDPROC
 1110 :
 1120 DEF FNcarry(o%,e$)
 1130 IF o%=42 THEN =tot*EVAL(e$)
 1140 IF o%=43 THEN =tot+EVAL(e$)
 1150 IF o%=45 THEN =tot-EVAL(e$)
 1160 IF o%=47 THEN =tot/EVAL(e$)
 1170 :
 1180 DEF PROCwipe
 1190 PROCwinin:CLS
 1200 PRINT TAB(6,0);CHR$(17);"Clear (Y/N)?";CHR$(18);
 1210 REPEAT:g%=GET AND 223:UNTIL g%=89 OR g%=78:CLS:IF g%=89 PROCclear
 1220 PROCnewline:ENDPROC
 1230 :
 1240 REM This last section handles lines rejected by EVAL.
 1250 REM Note: Repeated errors will eventually overflow the stack.
 1260 :
 1270 IF ERR=17 @%=&90A:VDU 26:CLS:CHAIN"AUTO"
 1280 IF ERR=214 GOTO 1310
 1290 PROCwinin:CLS:PRINT TAB(2,0);CHR$(17);"Error - press SPACE";
CHR$(18);
 1300 REPEAT:UNTIL GET=32:CLS:PROCnewline:GOTO 40
 1310 VDU 26:CLS:REPORT:PRINT" at line ";ERL
 1320 PRINT:PRINT"Press [Function][X] for Notepad Main Menu"
```

CHART.BAS

Programmer's ASCII Chart

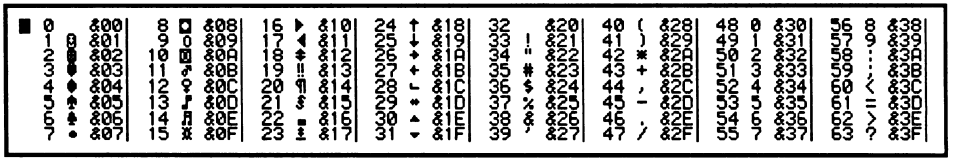


CHART.BAS, showing the first 64 Notepad characters

A program that displays the entire character set for a particular computer is always a welcome addition to any programmer's library of utilities. CHART.BAS is such a program for the Amstrad Notepad, listing each of the 256 available characters together with their ASCII codes in both decimal and hexadecimal. It's worth mentioning that you can't normally display characters with an ASCII code of less than 32 as these are control characters, used for controlling virtually every aspect of your Notepad's screen display, and any attempt to print them with a command such as:

```
PRINT CHR$(12)
```

will usually have a quite unexpected effect (unless, of course, that was your intention!) – in this case, the screen would be cleared exactly as if you had typed a CLS command. It might seem odd, then, that each of the 32 control characters (from 0 to 31) has been given a real, six-by-six pixel shape, just like all the others in the character set (a quick peek at the User Guide shows this to be true). So how can they be displayed?

The solution lies in the *unprintable* control code, 27. To be precise: The author of BBC Basic thoughtfully included this control code to allow any ASCII character to be printed as a character, without being interpreted by the VDU drivers as a control code.

Here is the corrected version of the example above which, using ASCII code 27, manages to print the actual symbol represented by ASCII code 12:

```
PRINT CHR$(27);CHR$(12)
```

Remember that whatever character you want to display MUST be the very next thing printed. If you split the command across two separate lines, you can't miss off the semicolon, otherwise the carriage return that would normally happen will be interpreted as a symbol by ASCII code 27, and printed as such. The next line will no longer be under the control of code 27, so it will behave like any other control code itself and (you guessed it) the screen will clear again.

So if you want to use any of the characters in the Notepad's character set, and you aren't sure if they will be interpreted literally or as control codes, your best bet is always to precede them with a CHR$(27);. Alternatively you can use the equivalent VDU command, like this:

```
VDU 27,12
```

and you won't have to worry about appending a semi-colon in order to prevent an unwanted carriage return.

USING THE PROGRAM

Chart is one of the simplest of all the programs in this book, but you still need to know how to use it. So type in the listing and save it as CHART.BAS before trying it out. Type:

```
RUN
```

(remembering to press [Return]), and the first 64 characters of the Notepad's character set will be displayed in eight lines, each containing eight columns of codes separated by vertical bars.

In any column, the information for a code is shown in the order: Decimal, ASCII code, the Character itself, hexadecimal ASCII code. To see another 64 codes, press [Down] and the screen will be redrawn with the next page of characters. Press [Up] to go back to the previous page, and [Stop] to exit the program altogether.

When you see a character you would like to include in your programs, jot down its decimal (or hexadecimal) code for later on. BBC Basic will happily understand either format in a CHR$() or VDU statement.

HOW IT WORKS

30	Points the Basic error handler to Chart's own error handling routine at line 290.
40	Calls PROCsetup to initialise the program, then sits in an infinite REPEAT...UNTIL loop calling PROCkeys.
80	Sets the page counter (page%) to zero and calls PROCpage to show the first screen of 64 characters.
120	Sits in a REPEAT...UNTIL loop reading all key presses into g%, until one has the ASCII value of either the [Up] or [Down] key.
130	If the ASCII value of the key just pressed (g%) was that of the [Up] key (240), and page% isn't already zero, decrements page% and calls PROCpage to show the new page before exiting the procedure.
140	If the ASCII value of the key just pressed (g%) was that of the

	[Down] key (241), and page% isn't already zero, decrements page% and calls PROCpage to show the new page before exiting the procedure.
180	Begins a FOR...NEXT loop of the column counter (x%).
190	Begins a nested FOR...NEXT loop of the row counter (y%).
200	Calculates the ASCII code of the current character by first multiplying the page counter (page%) by 64 to obtain the code of the first character in the current page. Next the correct offset within the page is obtained by multiplying the column counter (x%) by 8 and adding the row counter (y%). Finally this offset is added to the code of the first character in the page to obtain the current character's ASCII code.
210	Converts the current character's ASCII code into the string dec$, and pads it out with enough spaces to ensure that it will be right-justified when printed.
220	Converts the current character's ASCII code into hexadecimal and stores it in the string hex$, and if necessary pads it with a zero character to ensure that it will be the conventional width for hex characters, which is two characters for eight bit numbers (three once the ampersand & is added to the front).
230	Prints dec$ at the correct column position, which is x%*10 as each field is 10 characters wide. Adds a trailing space and a semicolon – to ensure the next data printed follows immediately after the space.
240	Prints the character itself, using CHR$(27) to ensure it is not interpreted by the VDU drivers as a control code, and adds a space, followed lastly by hex$.
250-270	End the nested row and column loops, and return from the procedure.
290	Points the Basic error handler to a full error report in the event of a further error occurring while attempting to run AUTO. This is in case AUTO isn't present on your Notepad.
300	Attempts to run the menu program AUTO if the error was generated by pressing the [Stop] key.
310	If the error was caused by something else, or if AUTO isn't on your Notepad, a full error report is displayed.
320	After the error report the Notepad will be left in BBC Basic, so this message is displayed to remind users of how to return to the Notepad main menu.

Functions and procedures

PROCsetup	Resets the page number and displays the first page of codes.

PROCkeys	Reads the keyboard for the [Up] and [Down] keys, changing and displaying pages if appropriate.
PROCpage	Displays the current page of 64 ASCII characters.

Main variables and arrays

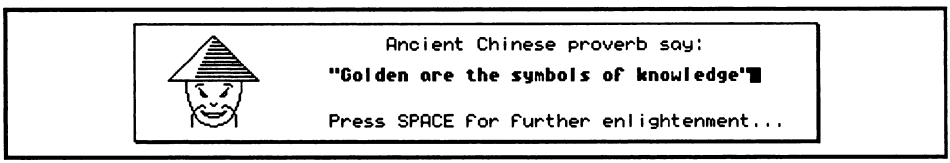
page%	The current page number, from 0 to 3.
char%	The current ASCII character being displayed.
dec$	A string containing the decimal ASCII code of the current character. It is padded with one or two spaces as necessary to right-align the number.
hex$	A string containing the hexadecimal ASCII code of the current character. It is padded with a single zero as necessary to right-align the number.
g%	The ASCII value of the current key press.
x%	The current column (0 – 7).
y%	The current row (0 – 7).

The program

```
 10 REM Programmers' ASCII Chart
 20 :
 30 ON ERROR GOTO 290
 40 PROCsetup:REPEAT:PROCkeys:UNTIL FALSE
 50 :
 60 CLS
 70 DEF PROCsetup
 80 page%=0:PROCpage
 90 ENDPROC
100 :
110 DEF PROCkeys
120 REPEAT:g%=INKEY(0):UNTIL g%=240 OR g%+241
130 IF g%=240 AND page%>0 page%=page%-1:PROCpage:ENDPROC
140 IF g%=241 AND page%<3 page%=page%+1:PROCpage:ENDPROC
150 ENDPROC
160 :
170 DEF PROCpage
180 FOR x%=0 TO 7
190 FOR y%=0 TO 7
200 char%=page%*64+(x%*8+y%)
210 dec$=STR$(char%):dec$=STRING$(3-LEN(dec$),CHR$(32))+dec$
220 hex$=STR$ (char%):hex$="&"+STRING$(2-LEN(hex$),CHR$(48))+hex$
230 PRINT TAB(x%*10,y%);dec$;SPC(1);
240 PRINT CHR$(27);CHR$(char%);CHR$(18);SPC(1);hex$;CHR$(179);
250 NEXT
260 NEXT
270 ENDPROC
280 :
290 ON ERROR GOTO 310
300 VDU 26:CLS:IF ERR=17 THEN CHAIN "AUTO"
310 REPORT:PRINT" at line ";ERL
320 PRINT:PRINT"Press [Function][X] for Notepad Main Menu"
```

COOKIE.BAS

Random Proverb Generator



COOKIE.BAS, with one of its pearls of wisdom

Playing with words is one of the first thing a Basic programmer learns to do. The program Eliza, which managed successfully to imitate the couch-side manner of a friendly psychiatrist is probably the most famous example of this sort of idea.

COOKIE.BAS is based on the same principle of stringing a selection of carefully chosen words together to create a meaningful sentence – in this case, a plausible sounding proverb of the type that you might find inside a Chinese fortune cookie.

You can waste many a mirthful hour playing with Cookie, but it's worth pointing out that the principles used do (believe it or not) have a serious application from the programmer's point of view. Not only are Basic's powerful string-slicing functions demonstrated – albeit to a very small degree – but some of the basic rules of simulating natural language on a computer are presented in an approachable fashion.

Feel free to extend the scope of this program, even if it's only by expanding the vocabulary in order to reduce the chances of repetition from proverb to proverb – a chance that is at present fairly high, despite the provision of 30 different subjects, objects and adjectives which together make the chance of the same proverb appearing twice in a row some 27,000 to 1.

USING THE PROGRAM

Type in the listing and save it as COOKIE.BAS before trying it out. Then type:

```
RUN
```

and the screen will clear to show a shadowed card in the centre, and the face of an oriental gentleman being drawn on the left of the card.

When the drawing is complete, a proverb will immediately be printed in the centre of the card in bold type enclosed between quotation marks. Cookie will then invite you to press [Space] for some further enlightenment, which you may do until you have had enough insights to last you a lifetime. At this point press [Stop], and you will

(probably mercifully) leave the Chinese gentleman and his profound sayings for another day...

HOW IT WORKS

30	Points the Basic error handler to Cookie's own error handling routine at line 980.
40	Calls PROCsetup to create and fill the word arrays, and then sits in an endless REPEAT...UNTIL loop calling PROCproverb.
70	Resets all windows, clears the screen, reads in the absolute number of words in each array (the arrays must all contain the same number of words) and assembles the screen loader.
80	Dimensions the three word arrays to max%-1, as the zeroeth element of each will be used.
90-110	Read in max% number of words into the three arrays.
120	Calls PROCcard to display the card, picture and static text, and sets up a text window large enough to hold the largest proverb possible.
160	Loads previously saved screen file from disk, if it exists.
170	Draws the outline of the card.
180	Draws the card shadow.
190	Begins the picture drawing FOR...NEXT loops. y% is controlled by the outer loop, within which a single string (s$) is read from the picture DATA. x% is controlled by the inner loop, within which each character of s$ is extracted with MID$(). If the character is a 1, PROCdot is called to plot a single point within the picture at the current coordinates of x% and y% (plus the correct offsets).
200	Ends the x% and y% loops and prints the first static message.
210	Prints the second static message.
250	Plots a point within the growing picture at the current coordinates of x% and y%. x% has a constant added to ensure that the picture is plotted within the card, and y% is subtracted from a different constant to ensure that the picture is both the right way up and the correct distance from the card edge.
280	Clears the proverb window and picks a random adjective (adj$), object (obj$) and subject (sub$) by generating three random numbers, each of which is used to index into the relevant word array.
290	Constructs the finished proverb (p$) by joining adj$ to obj$ with one static string, and obj$ to sub$ with another. Quotes are joined to p$ at both ends.

300	Prints the proverb p$ bold and centred within the current window, by starting printing at half the window width minus half the length of p$.
310	Repeatedly fetches key presses until [Space] has been pressed, before exiting the procedure.
330	Stores the maximum size of the three arrays.
350-380	Store 30 adjectives.
400-430	Store 30 objects.
450-480	Store 30 subjects.
500-970	Store 48 picture lines, each of 48 pixels.
990	Points the Basic error handler to a full error report in the event of a further error occurring while attempting to run AUTO. This is in case AUTO isn't present on your Notepad.
1000	Attempts to run the menu program AUTO if the error was generated by pressing the [Stop] key.
1010	If the error was caused by something else, or if AUTO isn't on your Notepad, a full error report is displayed.
1020	After the error report the Notepad will be left in BBC Basic, so this message is displayed to remind users of how to return to the Notepad main menu.
1050	Start of procedure that assembles the screen saver/loader, which saves a copy of the screen after everything is drawn the first time, and loads it in each time thereafter.
1060-1100	Define the five NC100 jump block routines to be used.
1120-1130	Begin the two-pass assembly and set P% (the assembly destination pointer) to the start of the previously dimensioned Z%.
1190	Pages the 16K of RAM with the video memory in at address &C000.
1200-1230	Copy the contents of video RAM down to &8000.
1240	Puts back the video RAM.
1250-1260	Open a file for saving the screen data.
1270	Returns if unable to open the file.
1280-1300	Save &1000 bytes from &8000 to the file.
1310	Closes the file and exit.
1350-1360	Open a file for reading.
1370-1400	If unable to open the file set the contents of flag to zero and return.
1440-1470	Read the &1000 bytes to location &8000 then close the file.

1480-1530	Map the video RAM 16K block into &C000, copy the &1000 bytes from location &8000 up to &F000 and then put back the screen RAM.
1540-1560	To indicate successful loading, set the contents of flag to 1 then return.
1600-1610	Save the current status of the bank switcher for block 4 (&C000-&FFFF).
1620-1650	Map the video RAM into main RAM then return.
1690-1700	Restore the state of the bank 4 bank switcher and its copy at &B003.
1760	The file name COOKIE.SCN.
1800	The flag to indicate successful file loading.
1840	Temporary storage of the state of the bank switcher.
1890-1920	A function to allocate memory for a string and store the string in that memory.
1940-1970	A function to allocate space for a byte of data and store the data in that location.

Functions and procedures

PROCsetup	Dimensions the three word arrays, reads in all the words, calls PROCcard to draw the card, the Chinese gentleman and the static text messages, and sets up a text window in which to print the proverbs.
PROCcard	Prints the outline of the card, draws the picture and prints the static text messages.
PROCdot	Prints a dot from the picture at the current coordinates – called by PROCcard.
PROCproverb	Constructs and prints a new proverb.

Main variables and arrays

adj$()	Holds all the adjectives.
obj$()	Holds all the subjects.
sub$()	Holds all the objects.
adj$	A randomly selected adjective picked from adj$().
obj$	A randomly selected object picked from obj$().
sub$	A randomly selected subject picked from sub$.
max%	The maximum number of adjectives, subjects and objects which are to be read from DATA and manipulated.
s$	The current line of the picture while it is being printed.

x%	The current column of the picture being printed.
y%	The current row of the picture being printed.
p$	The proverb under construction and eventually printed.

The program

```
   10 REM Fortune Cookie
   20 :
   30 ON ERROR GOTO 990
   40 PROCsetup:REPEAT:PROCproverb:UNTIL 0
   50 :
   60 DEF PROCsetup
   70 VDU 26:CLS:RESTORE:READ max%:DIM Z% &80:PROCassemble
   80 DIM adj$(max%-1),obj$(max%-1),sub$(max%-1)
   90 FOR w%=0 TO max%-1:READ adj$(w%):NEXT
  100 FOR w%=0 TO max%-1:READ obj$(w%):NEXT
  110 FOR w%=0 TO max%-1:READ sub$(w%):NEXT
  120 PROCcard:VDU 28,24,3,68,3
  130 ENDPROC
  140 :
  150 DEF PROCcard
  160 CALL scrn_from_disk:IF ?flag=0 THEN CLS ELSE ENDPROC
  170 MOVE 60,1:DRAW 419,1:DRAW 419,63:DRAW 60,63:DRAW 60,1
  180 MOVE 61,0:DRAW 420,0:DRAW 420,62
  190 FOR y%=0 TO 47:READ s$:FOR x%=0 TO 47:IF MID$(s$,x%+1,1)="1"
PROCdot
  200 NEXT:NEXT:PRINT TAB(32,1);"Ancient Chinese proverb say:"
  210 PRINT TAB(27,6);"Press SPACE for further enlightenment..."
  220 CALL scrn_to_disk:ENDPROC
  230 :
  240 DEF PROCdot
  250 PLOT 69,x%+78,56-y%:ENDPROC
  260 :
  270 DEF PROCproverb
  280 CLS:adj$=adj$(RND(max%-1)):obj$=obj$(RND(max%-
1)):sub$=sub$(RND(max%-1))
  290 p$=CHR$(34)+adj$+" are the "+obj$+" of "+sub$+CHR$(34)
  300 PRINT TAB(22-LEN(p$)/2,0);CHR$(17);p$;CHR$(18);
  310 REPEAT:UNTIL GET=32:ENDPROC
  320 :
  330 DATA 30
  340 :
  350 DATA Subtle,Bold,Many,Rewarding,Brutal,Few,Bland,Blessed,Blind
  360 DATA Cursed,Sinister,Wondrous,Vague,Deadly,Strange,Black,Golden
  370 DATA Precious,Sweet,Bitter,Varied,Monstrous,Terrible,Simple,Cheap
  380 DATA Tainted,Futile,Promising,Painful,Empty
  390 :
  400 DATA fires,pathways,penalties,temples,benefits,pleasures,sins,
symbols
  410 DATA revelations,seeds,treasures,ways,workings,perils,qualities,
joys
  420 DATA origins,follies,enigmas,dividends,rewards,deeds,evils,politics
  430 DATA fruits,mysteries,methods,motives,crimes,desires
  440 :
  450 DATA the flesh,passion,hate,seduction,the soul,charity,knowledge
  460 DATA the spirit,wisdom,heaven,hell,mercy,freedom,life,the
heart,destiny
```

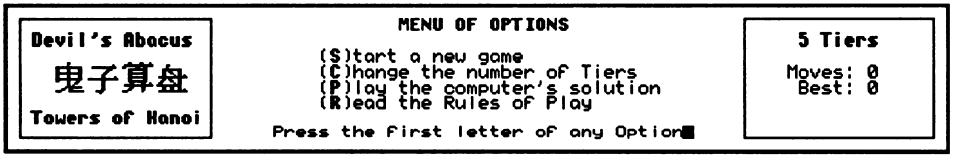
```
 470 DATA love,death,truth,lust,greed,hope,delight,despair,disease,peace
 480 DATA envy,religion,deceit,guilt
 490 :
 500 DATA "00000000000000000000000011000000000000000000000000"
 510 DATA "00000000000000000000000101100000000000000000000000"
 520 DATA "00000000000000000000001000010000000000000000000000"
 530 DATA "00000000000000000000010011111100000000000000000000"
 540 DATA "00000000000000000001000000001000000000000000000000"
 550 DATA "00000000000000000100011111111000000000000000000000"
 560 DATA "00000000000000001000000000001000000000000000000000"
 570 DATA "00000000000000010000111111111100000000000000000000"
 580 DATA "00000000000000100000000000000010000000000000000000"
 590 DATA "00000000000001000001111111111111100000000000000000"
 600 DATA "00000000000010000000000000000000010000000000000000"
 610 DATA "00000000000100000011111111111111110000000000000000"
 620 DATA "00000000001000000000000000000000100000000000000000"
 630 DATA "00000000010000000111111111111111111110000000000000"
 640 DATA "00000000100000000000000000000000001000000000000000"
 650 DATA "00000001000000001111111111111111111111100000000000"
 660 DATA "00000010000000000000000000000000000010000000000000"
 670 DATA "00000100000000011111111111111111111111111000000000"
 680 DATA "00001000000000000000000000000000000000100000000000"
 690 DATA "00010000000000111111111111111111111111111110000000"
 700 DATA "00100000000000000000000000000000000000001000000000"
 710 DATA "01000000000011111111111111111111111111111100000000"
 720 DATA "10000000000000000000000000000000000000000010000000"
 730 DATA "11111111111111111111111111111111111111111111111111"
 740 DATA "00000000010000000000000000000000000001000000000000"
 750 DATA "00000000010000011110000000000111100001000000000000"
 760 DATA "00000000010000100000000000000000100001000000000000"
 770 DATA "00000000010000010000000000000010000001000000000000"
 780 DATA "00000000010000001000000000001000000010000000000000"
 790 DATA "00000000010000001100000000011000000010000000000000"
 800 DATA "00000000010000001010000000101000000010000000000000"
 810 DATA "00000000010000000111000001110000000010000000000000"
 820 DATA "00000000010000000000000000000000000010000000000000"
 830 DATA "00000000010000000000000000000000000010000000000000"
 840 DATA "00000000010000000000000000000000000100000000000000"
 850 DATA "00000000000100000000001001000000000100000000000000"
 860 DATA "00000000000100000110000110000110000100000000000000"
 870 DATA "00000000000010011001000000010011001000000000000000"
 880 DATA "00000000000010100000000000000000101000000000000000"
 890 DATA "00000000000001000110000000001100010000000000000000"
 900 DATA "00000000000001000011111111111100001000000000000000"
 910 DATA "00000000000010100001100000011000010100000000000000"
 920 DATA "00000000000010100000011111100000010100000000000000"
 930 DATA "00000000000010010000000000000000100100000000000000"
 940 DATA "00000000000010001000000000000001000100000000000000"
 950 DATA "00000000000010000110000000001100001000000000000000"
 960 DATA "00000000000010000001110000111000000100000000000000"
 970 DATA "00000000000010000000001111000000000100000000000000"
 980 :
 990 ON ERROR GOTO 1010
1000 VDU 26:CLS:IF ERR=17 THEN CHAIN "AUTO"
1010 REPORT:PRINT" at line ";ERL
1020 PRINT:PRINT"Press [Function][X] for Notepad Main Menu"
1030 END
1040 :
1050 DEF PROCassemble
1060 fopenout=&B8A5
```

```
1070 fopenin=&B8A2
1080 foutblock=&B8AB
1090 finblock=&B896
1100 fclose=&B890
1110 :
1120 FOR PASS = 0 TO 2 STEP 2
1130 P%=Z%
1140 [
1150 OPT PASS
1160 :
1170 .scrn_to_disk
1180 :
1190 CALL map_scrn_in
1200 LD HL,&F000
1210 LD DE,&8000
1220 LD BC,&1000
1230 LDIR
1240 CALL map_scrn_out
1250 LD HL,filename
1260 CALL fopenout
1270 RET NC
1280 LD HL,&8000
1290 LD BC,&1000
1300 CALL foutblock
1310 JP fclose
1320 :
1330 .scrn_from_disk
1340 :
1350 LD HL,filename
1360 CALL fopenin
1370 JR C,froml
1380 LD HL,flag
1390 LD (HL),0
1400 RET
1410 :
1420 .froml
1430 :
1440 LD HL,&8000
1450 LD BC,&1000
1460 CALL finblock
1470 CALL fclose
1480 CALL map_scrn_in
1490 LD HL,&8000
1500 LD DE,&F000
1510 LD BC,&1000
1520 LDIR
1530 CALL map_scrn_out
1540 LD HL,flag
1550 LD (HL),1
1560 RET
1570 :
1580 .map_scrn_in
1590 :
1600 LD A,(&B003)
1610 LD (state),A
1620 LD A,67
1630 LD (&B003),A
1640 OUT (&13),A
1650 RET
1660 :
```

```
1670 .map_scrn_out
1680 :
1690 LD A,(state)
1700 LD (&B003),A
1710 OUT (&13),A
1720 RET
1730 :
1740 .filename
1750 :
1760 DEFM "COOKIE.SCN":DEFB 0
1770 :
1780 .flag
1790 :
1800 DEFB 0
1810 :
1820 .state
1830 :
1840 DEFB 0
1850 ]
1860 NEXT
1870 ENDPROC
```

DEVIL.BAS

Towers of Hanoi



DEVIL.BAS, getting ready to start a game

Towers of Hanoi is a popular and very old game of the patience variety, where a lone player must move a stack of rings from one of three vertical poles to another, one at a time – but the stack is built from rings of different sizes, and at no time during the game may any ring be placed on top of one that is smaller.

It's not hard to work out the solution, but the trick lies in completing the puzzle in the fewest number of moves – which if you have never played the game before is a lot harder than it sounds. In fact the minimum number of moves doubles with every ring added to the stack (rather like the binary number system, in fact).

DEVIL.BAS adheres strictly to the standard rules. You can choose to play with any number of rings from two to seven levels (often called *tiers* in this version) and if you get stuck you can ask the computer to play the full solution for that number of tiers. It will solve the game either at full speed (quite entertaining to watch at seven-tier level)

or run through it one step at a time, waiting for a key press before continuing (ideal for studying the solution).

USING THE PROGRAM

Type in the listing and save it as DEVIL.BAS before trying it out. Then type:

```
RUN
```

and the main screen will appear. The number of rings or tiers is initially set to five, as shown in the box to the right, but you can change this from the Main Menu at any time by pressing [C], followed by the number you would like to play with, and finally [Return].

To play the game with the current number of tiers, press [S] from the main menu. The centre area of the display will clear to show a stack of rings on the left, beneath the number 1. At the top of the screen in the centre is the number 2, and at the top-right of the are is the number 3. These numbers represent the three poles, which are not themselves drawn for reasons of clarity.

The game is played by first pressing the number of the pole from which you would like to move a ring – the *start* pole, followed by the number of the pole you want to move it to – the *end* pole. For example, if you wanted to move a ring from pole 1 to pole 2, just push [1] followed by [2].

There is no need to press [Return] to enter your choice – the keys are read instantly as you push them. Devil will not allow illegal moves to take place. An illegal move is considered to be any one of the following:

- Attempting to select a start pole that is empty
- Attempting to select an end pole whose top-most ring is smaller than the ring to be moved
- Attempting to select a ring that cannot possibly be moved because the other two poles top-most rings are both smaller in diameter
- Attempting to move a ring to the pole it is on.

The game continues until you either complete the puzzle, in which case you will be heartily congratulated (well, sort of), or you press [Stop], which will immediately exit the program, or you press [Q], which will prompt the computer to ask you if you are sure you want to abandon the game. If you are, press [Y] to return to the main menu – otherwise, press [N] and the game will resume.

During play the box on the right hand side of the screen always shows the number of the current move and also the highest score for this level. Changing the number of tiers resets the high score and, of course, starting a new game resets the move

counter. All prompts during a game (or an automatic solution) are displayed on the bottom line of this box which normally shows either *From?* or *To?*, depending on whether you next need to select a start or end pole.

If you can't solve a particular level you might like to watch the computer doing it for you in the shortest possible number of moves. To do this press [P] from the main menu to see the automatic solution menu. You are given the choice of either automatic playback (where the entire solution whips past at the speed of light), or manual playback (where the computer politely waits for you to press a key after each move).

Press [A] for automatic or [M] for manual, and the solution will unfold before your very eyes. In both automatic and manual modes you can abort the solution by pressing [Q], exactly as if it were a normal game being played, and you will be dropped back at the Main Menu.

If you want to review the rules at any time, press [R] from the main menu. When you've finished reading, press [Space] to return to the menu.

HOW IT WORKS

30	Points the Basic error handler to Devil's own error handling routine at line 2510.
40	Resets all windows, clears the screen and calls PROCsetup to initialise the program.
50	Calls the Main Menu inside an infinite REPEAT...UNTIL loop until [Stop] is pressed.
90	Dimensions the arrays and assembles the screen saver/loader.
100	Sets the default number of tiers to five.
110-120	Create seven 32-character width strings containing different size rings made from CHR$(223), and place them in tier$, smallest first.
130	Loads previously saved screen file from disk, if it exists.
140-170	Draw screen boxes and print the left-hand box static strings.
180-190	Draw the four Chinese characters held in DATA lines.
200-220	Print the right-hand box static strings.
260-320	Set up the main window and print the Main Menu.
330-340	Read keyboard until one of the highlighted menu option letters is pressed.
350-380	Call the appropriate procedure depending on the key just pressed.
420-450	Ask user for the new number of tiers, and wait until a valid number between 2 to 7 inclusive is entered.

460	Prints the new number of tiers in the right-hand status box.
470	Resets the move counter and the high score, and prints these in the status box.
510	Starts the main game REPEAT...UNTIL loop and repeatedly puts the result of FNmove into result%.
520	If result% equals FALSE then [Q] was pressed at some point during the move, and so calls FNquit to make sure the user wants to quit. If FNquit returns TRUE then forces an exit from the REPEAT...UNTIL loop and returns from the procedure.
530	[Q] wasn't pressed, so repeats the loop until won% equals TRUE. Then checks to see if the high score (hi%) is now higher than the number of moves just made (sc%), and if so sets hi% to sc%.
540	Prints high score and congratulations message.
550	Waits for [Space] key before returning from procedure.
580-600	Check poles 2 and 3 to see if a complete stack has been built by comparing stack%(pole) with tiers%. If equal, return TRUE. If not, return FALSE.
630-650	Print a *Quit (Y/N)* prompt and return TRUE if [Y] is pressed or FALSE if [N] is pressed.
680	Executes a FOR...NEXT loop to draw the initial stack on pole 1, to the height of tiers%.
690	Prints the pole numbers.
730-790	Display a the rules and wait for [Space] before returning.
820	Defines a text window for the main playing area.
850	Defines a text window to hold the score, high score and current number of tiers.
880	Defines a text window for the prompts issued when a game is under way.
910	Clears the prompt window and prints *From?*, signalling that a start pole is to be selected.
920	Calls FNkey inside a REPEAT...UNTIL loop to obtain the start pole, returning FALSE if FNkey returns FALSE ([Q] was pressed).
930	Repeats the loop unless FNlegal_start returns TRUE, in which case the start pole number is printed next to the *From?* message, and the word *To?* is added on the end to signal that an end pole is to be selected.
940	Calls FNkey inside a REPEAT...UNTIL loop to obtain the end

	pole, returning FALSE if FNkey returns FALSE ([Q] was pressed).
950	Repeats the loop unless FNlegal_end returns TRUE.
960	Prints the end pole number next to the *To?* message, removes the start ring with PROCget(start%), and puts it on the new pole with PROCput(end%).
970	Increments and prints the move counter, and returns TRUE.
1000-1010	Repeatedly scan the keyboard until any of [1], [2], [3] or [Q] is pressed (g% AND 223 forces the key into upper case to save two separate checks on its ASCII code).
1020	If [Q] was pressed, returns FALSE.
1030	Returns the ASCII value of the key minus 48, to bring it into the range 1 to 3.
1060	Returns FALSE if start pole is empty.
1070-1090	Set legal% to FALSE and check the other two poles to see if the start ring can legally be moved to either. If so, set legal% to TRUE.
1100	Returns the value of legal%.
1130	Returns FALSE if the end pole selected was in fact the start pole.
1140	Returns FALSE if the start ring is larger than the top-most ring on the end pole.
1150	Returns TRUE, because no problem was found with the choice of end pole.
1180	Returns 99 if the pole passed in p% is empty.
1190	Otherwise returns the number of the top-most ring on pole p%, as held in pole%(p%,height), where height is given by level%(p%).
1220	Removes a ring from the pole passed in p% by overprinting it with 32 spaces.
1230	Calls FNtop with p% to get the number of the top-most ring, and stores it in store% before decrementing the height of the pole held in level%(p%).
1270	Increments the height of the pole held in level%(p%) and places the ring number held in store% on the top of the pole.
1280	Places the new ring (still held in store%) on the pole passed in p% by fetching its image from tier$().
1320	Prints the current move counter.
1350	Prints the current high score.
1380	Sets up the left-hand pole to hold the number of each ring up to

	tiers%. The largest must be at the bottom – pole%(1,tiers%) – and the smallest at the top – pole%(1,1) – and so the rings are placed in reverse order. Sets the height of the first stack as held in level%(1) to tiers%.
1390	Clears the ring numbers on the other two poles to zero and sets their heights to zero.
1400	Draws the new stack and resets the move counter.
1430	Sets x% and y% to point to the top left graphics location of the Chinese character about to be printed, by multiplying the passed text coordinates. Also ensures the character is not inverted by subtracting the resulting Y coordinate from the top-most point on the screen.
1440	Starts the outer row loop, reading in each line of the character from DATA.
1450	Starts the inner column loop, extracting each character from the line just read.
1460	Plots a point at the current X and Y graphics coordinates if the character just extracted is a 1.
1470-1480	End the inner and outer loop before returning.
1510-1540	Display the automatic solution menu and prompt for a key press.
1550	Reads the keyboard until [A], [M] or [Q] is pressed.
1560	Returns if [Q] was pressed, otherwise sets ss% (single-step mode) to TRUE if [M] was pressed or FALSE if [A] was pressed.
1570	Sets up and draws the starting position, clears the status window and turns on bold type.
1580	Displays the selected playback mode according to the value of ss%.
1590	Turns bold off, activates the main window, resets the DATA pointer to the start of the solution, clears quit% and begins the playback REPEAT...UNTIL loop.
1600	Calls PROCwait if in single-step mode to wait for a key press before continuing, otherwise the keyboard is checked *on the fly* with INKEY$(0), to see if [Q] has been pressed. If so, sets quit% to TRUE.
1610	Reads each move of the solution into start% and end%, which allows simulation of a normal game by calling PROCget(start%) and PROCput(end%) immediately.
1620	Increments and prints the move counter and loops back to the REPEAT, until either quit% equals TRUE or the solution has been completed.

1630	Activates the status window and returns from the procedure if quit% equals TRUE.
1640-1650	Print a *Press SPACE* prompt and wait until [Space] is pressed before returning from the procedure.
1680	Waits for [Space] or [Q] to be pressed, setting quit% to TRUE if [Q] was pressed, or FALSE if not.
1720-2380	Hold the data for four Chinese characters which spell (roughly translated) *Devil Abacus*.
2400-2500	Hold the complete solution for up to seven rings, as pairs of start and end pole movements.
2520	Points the Basic error handler to a full error report in the event of a further error occurring while attempting to run AUTO. This is in case AUTO isn't present on your Notepad.
2530	Attempts to run the menu program AUTO if the error was generated by pressing the [Stop] key.
2540	If the error was caused by something else, or if AUTO isn't on your Notepad, a full error report is displayed.
2550	After the error report the Notepad will be left in BBC Basic, so this message is displayed to remind users of how to return to the Notepad main menu.
2580	Start of procedure that assembles the screen saver/loader, which saves a copy of the screen after everything is drawn the first time, and loads it in each time thereafter.
2590-2630	Define the five NC100 jumpblock routines to be used.
2650-2660	Begin the two-pass assembly and set P% (the assembly destination pointer) to the start of the previously dimensioned Z%.
2720	Pages the 16K of RAM with the video memory in at address &C000.
2730-2760	Copy the contents of video RAM down to &8000.
2770	Puts back the video RAM.
2780-2790	Open a file for saving the screen data.
2800	Returns if unable to open the file.
2810-2830	Save &1000 bytes from &8000 to the file.
2840	Closes the file and exit.
2880-2890	Open a file for reading.
2900-2930	If unable to open the file, set the contents of flag to zero and return.
2970-3000	Read the &1000 bytes to location &8000 then close the file.

3010-3060	Map the video RAM 16K block into &C000, copy the &1000 bytes from location &8000 up to &F000 and then put back the screen RAM.
3070-3090	To indicate successful loading, set the contents of flag to 1 then return.
3130-3140	Save the current status of the bank switcher for block 4 (&C000-&FFFF).
3150-3180	Map the video RAM into main RAM then return.
3220-3230	Restore the state of the bank 4 bank switcher and its copy at &B003.
3290	The file name DEVIL.SCN.
3330	The flag to indicate successful file loading.
3370	Temporary storage of the state of the bank switcher.
3420-3450	A function to allocate memory for a string and store the string in that memory.
3470-3500	A function to allocate space for a byte of data and store the data in that location.

Functions and procedures

PROCsetup	Sets up the main variables and draws the playing screen. Calls PROCchinese four times to display the Chinese characters. Fills tier$ with seven rings of decreasing diameter built from CHR$(223).
PROCmenu	Displays the Main Menu and calls the appropriate procedure according to the key pressed.
PROCtiers	Allows user to change the current number of tiers.
PROCplay	Plays the game.
PROCdrawstack	Draws the playing area ready for start of play or auto-solve, with a stack of rings (of the currently selected number) on the left, and the pole numbers on the top line.
PROCrules	Displays the rules.
PROCwin1	Defines a text window for the main playing area.
PROCwin2	Defines a text window for the status box (where the scores and current number of tiers are displayed).
PROCwin3	Defines a text window for the prompt box (where prompts are displayed during a game).
PROCget()	Takes a ring of the named stack and stores its number.
PROCput()	Places the stored ring on the named stack.
PROCscore	Updates the current move counter.

PROChiscore	Updates the fewest moves counter.
PROCprepare	Prepares program for a new game or automatic solution by resetting the ring stacks and the moves counter.
PROCchinese()	Prints a Chinese character at the passed text coordinates.
PROCsolve	Plays the complete solution for the current number of tiers, in automatic or single-step mode.
PROCwait	Waits for [Space] or [Q] to be pressed, setting quit% to TRUE if [Q] was pressed, or FALSE if not.
FNwin	Sets won% to TRUE if any stack contains all the rings.
FNquit	Prints a *Quit (Y/N)* prompt and returns TRUE if [Y] is pressed or FALSE if [N] is pressed.
FNmove	Plays a full move, returning FALSE if [Q] was pressed instead of a start or end pole number.
FNkey	Waits for [1], [2], [3], or [Q] to be pressed, returning FALSE if the latter, or TRUE if any of the former.
FNlegal_start	Returns FALSE if the selected start pole is illegal for any reason.
FNlegal_end	Returns FALSE if the selected end pole is illegal for any reason.
FNtop()	Returns the number of the ring at the top of the named stack (1 to 7), or 99 if the stack is empty.

Main variables and arrays

tier$()	Holds strings representing pictures of all seven rings. tier$(1) is the smallest, tier$(7) the largest.
pole%()	Holds the current position of rings on each pole.
level%()	Holds the current number of rings held on each pole (1 to 7)
start%	The start pole number (1 to 3).
end%	The end pole number (1 to 3).
sc%	The move counter.
hi%	The fewest moves managed so far for the current level.
ss%	Single-step flag, holds TRUE if user selected manual playback of the solution, otherwise holds FALSE.
quit%	Holds TRUE if [Q] was pressed during a game or automatic solution.
g%	Used throughout the program to hold key presses.

The program

```
10 REM Devil's Abacus
20 :
30 ON ERROR GOTO 2520
40 VDU 26:CLS:PROCsetup
```

```
   50 REPEAT:PROCmenu:UNTIL FALSE
   60 END
   70 :
   80 DEF PROCsetup
   90 DIM tier$(7),pole%(3,7),level%(3),Z% &80:PROCassemble
  100 tiers%=5:sc%=0:hi%=0
  110 FOR t%=1 TO 7:pad$=STRING$(7-t%,CHR$(32))
  120 tier$(t%)=pad$+STRING$(t%*2,CHR$(223))+pad$:NEXT
  130 CALL scrn_from_disk:IF ?flag=0 THEN CLS ELSE GOTO 200
  140 MOVE 0,0:DRAW 100,0:DRAW 100,63:DRAW 0,63:DRAW 0,0
  150 MOVE 380,0:DRAW 479,0:DRAW 479,63:DRAW 380,63:DRAW 380,0
  160 PRINT TAB(1,1);CHR$(17);"Devil's Abacus";CHR$(18);
  170 PRINT TAB(1,6);CHR$(17);"Towers of Hanoi";CHR$(18)
  180 RESTORE 1720:PROCchinese(3,3):RESTORE 1890:PROCchinese(6,3)
  190 RESTORE 2060:PROCchinese(9,3):RESTORE 2230:PROCchinese(12,3)
  200 PROCwin2:CLS:PRINT TAB(4,0);CHR$(17);tiers%;CHR$(32);"Tiers";
CHR$(18)
  210 PRINT TAB(3,2);"Moves: ";sc%
  220 PRINT TAB(4,3);"Best: ";hi%
  230 CALL scrn_to_disk:ENDPROC
  240 :
  250 DEF PROCmenu
  260 PROCwin1:CLS
  270 PRINT TAB(15,0);CHR$(17);"MENU OF OPTIONS";CHR$(18)
  280 PRINT TAB(8,2);:VDU 40,17,83,18,41:PRINT"tart a new game"
  290 PRINT TAB(8,3);:VDU 40,17,67,18,41:PRINT"hange the number of Tiers"
  300 PRINT TAB(8,4);:VDU 40,17,80,18,41:PRINT"lay the computer's
solution"
  310 PRINT TAB(8,5);:VDU 40,17,82,18,41:PRINT"ead the Rules of Play"
  320 PRINT TAB(4,7);"Press the first letter of any Option";
  330 REPEAT:g%=GET AND 223
  340 UNTIL g%=83 OR g%=67 OR g%=80 OR g%=82
  350 IF g%=83 PROCplay:ENDPROC
  360 IF g%=67 PROCtiers:ENDPROC
  370 IF g%=80 PROCsolve:ENDPROC
  380 IF g%=82 PROCrules:ENDPROC
  390 ENDPROC
  400 :
  410 DEF PROCtiers
  420 CLS:PRINT TAB(13,2);CHR$(17);"SET NUMBER OF TIERS";CHR$(18)
  430 PRINT TAB(2,6);"Minumum Moves: 2 Tiers = 3, 7 Tiers = 127";
  440 REPEAT:PRINT TAB(11,4);"How many Tiers (2-7) ";
  450 INPUT tiers%:UNTIL tiers%>=2 AND tiers%<=7
  460 PROCwin2:PRINT TAB(4,0);CHR$(17);tiers%;CHR$(18)
  470 sc%=0:hi%=0:PROCscore:PROChiscore:ENDPROC
  480 :
  490 DEF PROCplay
  500 PROCprepare:PROCscore
  510 REPEAT:result%=FNmove
  520 IF result%=FALSE IF FNquit UNTIL TRUE:ENDPROC
  530 UNTIL FNwin:IF sc%<hi% OR hi%=0 hi%=sc%
  540 PROChiscore:PROCwin3:CLS:PRINT SPC(3);CHR$(17);"Well
Done!";CHR$(18);
  550 REPEAT:UNTIL GET=32:CLS:ENDPROC
  560 :
  570 DEF FNwin
  580 won%=FALSE:FOR p%=2 TO 3
  590 IF level%(p%)=tiers% won%=TRUE
  600 NEXT:=won%
  610 :
```

```
 620 DEF FNquit
 630 PROCwin3:CLS:PRINT SPC(2);CHR$(17);"Quit (Y/N)?";CHR$(18);
 640 REPEAT:g%=GET AND 223:UNTIL g%=89 OR g%=78
 650 CLS:IF g%=89 THEN = TRUE ELSE = FALSE
 660 :
 670 DEF PROCdrawstack:PROCwin1:CLS
 680 FOR t%=1 TO tiers%:PRINT TAB(0,8-t%);tier$(pole%(1,t%));:NEXT
 690 PRINT TAB(6,0);"1";TAB(21,0);"2";TAB(36,0);"3"
 700 ENDPROC
 710 :
 720 DEF PROCrules
 730 CLS:PRINT TAB(20,0);CHR$(17);"RULES";CHR$(18)'
 740 PRINT"The object of Devil's Abacus is to move the"
 750 PRINT"entire tower to any empty position, tier by"
 760 PRINT"tier, in as few moves as possible. However,"
 770 PRINT"you can't put larger tiers on smaller ones."
 780 PRINT TAB(10,7);"Press SPACE for the Menu";
 790 REPEAT:UNTIL GET=32:ENDPROC
 800 :
 810 DEF PROCwin1
 820 VDU 28,18,7,62,0:ENDPROC
 830 :
 840 DEF PROCwin2
 850 VDU 28,64,5,78,1:ENDPROC
 860 :
 870 DEF PROCwin3
 880 VDU 28,64,6,78,6:ENDPROC
 890 :
 900 DEF FNmove
 910 PROCwin3:CLS:VDU 17:PRINT TAB(1,0);"From ";
 920 REPEAT:start%=FNkey:IF start%=FALSE UNTIL TRUE:VDU 18:=FALSE
 930 UNTIL FNlegal_start:VDU start%+48:PRINT TAB(10,0);"To: ";
 940 REPEAT:end%=FNkey:IF end%=FALSE UNTIL TRUE:VDU 18:=FALSE
 950 UNTIL FNlegal_end:VDU 18
 960 VDU end%+48:PROCwin1:PROCget(start%):PROCput(end%)
 970 sc%=sc%+1:PROCscore:=TRUE
 980 :
 990 DEF FNkey
1000 REPEAT:g%=GET
1010 UNTIL (g%>=49 AND g%<=51) OR (g% AND 223)=81
1020 IF (g% AND 223)=81 THEN = FALSE
1030 =g%-48
1040 :
1050 DEF FNlegal_start
1060 IF level%(start%)=0 THEN =FALSE
1070 legal%=FALSE:FOR p%=1 TO 3
1080   IF p%<>start% IF FNtop(start%)<FNtop(p%) legal%=TRUE
1090 NEXT
1100 =legal%
1110 :
1120 DEF FNlegal_end
1130 IF start%=end% THEN =FALSE
1140 IF FNtop(start%)>FNtop(end%) THEN =FALSE
1150 =TRUE
1160 :
1170 DEF FNtop(p%)
1180 IF level%(p%)=0 THEN = 99
1190 =pole%(p%,level%(p%))
1200 :
1210 DEF PROCget(p%)
```

```
 1220 PRINT TAB(15*p%-15,8-level%(p%));STRING$(14,CHR$(32));
 1230 store%=FNtop(p%):level%(p%)=level%(p%)-1
 1240 ENDPROC
 1250 :
 1260 DEF PROCput(p%)
 1270 level%(p%)=level%(p%)+1:pole%(p%,level%(p%))=store%
 1280 PRINT TAB(15*p%-15,8-level%(p%));tier$(store%);
 1290 ENDPROC
 1300 :
 1310 DEF PROCscore
 1320 PROCwin2:PRINT TAB(10,2);sc%;SPC(2):ENDPROC
 1330 :
 1340 DEF PROChiscore
 1350 PROCwin2:PRINT TAB(10,3);hi%;SPC(2):ENDPROC
 1360 :
 1370 DEF PROCprepare
 1380 FOR t%=1 TO tiers%:pole%(1,t%)=tiers%+1-t%:NEXT:level%(1)=tiers%
 1390 FOR p%=2 TO 3:FOR t%=1 TO 7:pole%(p%,t%)=0:NEXT:level%(p%)=0:NEXT
 1400 PROCdrawstack:sc%=0:ENDPROC
 1410 :
 1420 DEF PROCchinese(col%,row%)
 1430 x%=col%*6:y%=64-row%*8
 1440 FOR r%=1 TO 16:READ r$
 1450 FOR c%=1 TO 16:p%=VAL(MID$(r$,c%,1))
 1460 IF p%=1 PLOT 69,x%+(c%-1),y%-(r%-1)
 1470 NEXT:NEXT
 1480 ENDPROC
 1490 :
 1500 DEF PROCsolve
 1510 CLS:PRINT TAB(10,2);CHR$(17);"PLAY COMPUTER'S SOLUTION";CHR$(18)
 1520 PRINT TAB(5,4);:VDU 40,17,65,18,41:PRINT"utomatic or";
 1530 VDU 32,40,17,77,18,41:PRINT"anual playback?";
 1540 PRINT TAB(0,6);"Q Quits playback - SPACE moves in Manual Mode";
 1550 REPEAT:g%=GET AND 223:UNTIL g%=65 OR g%=77 OR g%=81
 1560 IF g%=81 ENDPROC ELSE IF g%=77 ss%=TRUE ELSE ss%=FALSE
 1570 PROCprepare:PROCwin3:CLS:VDU 17
 1580 IF ss% PRINT SPC(2);"Single Step"; ELSE PRINT SPC(1); "Auto
Playback";
 1590 VDU 18:PROCwin1:RESTORE 2400:TIME=0:quit%=FALSE:REPEAT
 1600 IF ss% PROCwait ELSE i%=INKEY(0) AND 223:IF i%=81 quit%=TRUE
 1610 READ start%,end%:PROCget(start%):PROCput(end%)
 1620 sc%=sc%+1:PROCscore:PROCwin1:UNTIL sc%=2^tiers%-1 OR quit%
 1630 PROCwin3:CLS:IF quit% ENDPROC
 1640 PRINT SPC(2);CHR$(17);"Press SPACE";CHR$(18);
 1650 REPEAT:UNTIL GET=32:CLS:ENDPROC
 1660 :
 1670 DEF PROCwait
 1680 REPEAT:i%=INKEY(0):quit%=(i%=81):UNTIL i%=32 OR quit%
 1690 ENDPROC
 1700 :
 1710 REM Chinese Character "Devil" #1
 1720 DATA "0000000110000000"
 1730 DATA "0000000100000000"
 1740 DATA "0111111111111110"
 1750 DATA "0110000110000110"
 1760 DATA "0110000110000110"
 1770 DATA "0111111111111110"
 1780 DATA "0110000110000110"
 1790 DATA "0110000110000110"
 1800 DATA "0111111111111110"
```

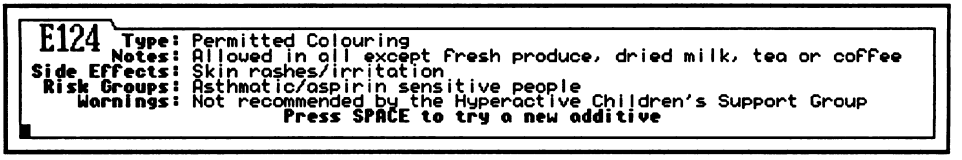
```
1810 DATA "0110000110001100"
1820 DATA "0000000110011000"
1830 DATA "0000000110100000"
1840 DATA "0000001011111110"
1850 DATA "0000110011000000"
1860 DATA "0111000011111111"
1870 DATA "0000000000000000"
1880 REM Chinese Character "Devil" #2
1890 DATA "0000000000000000"
1900 DATA "0000000000000100"
1910 DATA "0011111111111110"
1920 DATA "0000000000111000"
1930 DATA "0000000001100000"
1940 DATA "0000000011000000"
1950 DATA "0000000110000000"
1960 DATA "0000000110000010"
1970 DATA "1111111111111111"
1980 DATA "0000000110000000"
1990 DATA "0000000110000000"
2000 DATA "0000000110000000"
2010 DATA "0000000110000000"
2020 DATA "0000000110000000"
2030 DATA "0000111100000000"
2040 DATA "0000001000000000"
2050 REM Chinese Character "Abacus" #1
2060 DATA "0001100001100010"
2070 DATA "0111111111111111"
2080 DATA "1100000110011000"
2090 DATA "0001111111111000"
2100 DATA "0001100000011000"
2110 DATA "0001111111111000"
2120 DATA "0001100000011000"
2130 DATA "0001111111111000"
2140 DATA "0001100000011000"
2150 DATA "0001111111111000"
2160 DATA "0000110000110010"
2170 DATA "0111111111111111"
2180 DATA "0000110000110000"
2190 DATA "0001100000110000"
2200 DATA "1110000000100000"
2210 DATA "0000000000000000"
2220 REM Chinese Character "Abacus" #2
2230 DATA "0000000000000000"
2240 DATA "0000011000000000"
2250 DATA "0000010000000000"
2260 DATA "0001111111111000"
2270 DATA "0001101100011000"
2280 DATA "0001100110011000"
2290 DATA "0001100001011010"
2300 DATA "0111111111111111"
2310 DATA "0001101100011000"
2320 DATA "0011000110011000"
2330 DATA "0110000001011000"
2340 DATA "1111111111111000"
2350 DATA "0001100110011000"
2360 DATA "0001100110011010"
2370 DATA "0111111111111111"
2380 DATA "0000000000000000"
2390 REM Solution for all Tiers up to Seven
2400 DATA 1,2,1,3,2,3,1,2,3,1,3,2,1,2,1,3,2,3,2,1,3,1,2,3
```

```
2410 DATA 1,2,1,3,2,3,1,2,3,1,3,2,1,2,3,1,2,3,2,1,3,1,3,2
2420 DATA 1,2,1,3,2,3,1,2,3,1,3,2,1,2,1,3,2,3,2,1,3,1,2,3
2430 DATA 1,2,1,3,2,3,2,1,3,1,3,2,1,2,3,1,2,3,2,1,3,1,2,3
2440 DATA 1,2,1,3,2,3,1,2,3,1,3,2,1,2,1,3,2,3,2,1,3,1,2,3
2450 DATA 1,2,1,3,2,3,1,2,3,1,3,2,1,2,3,1,2,3,2,1,3,1,3,2
2460 DATA 1,2,1,3,2,3,1,2,3,1,3,2,1,2,3,1,2,3,2,1,3,1,2,3
2470 DATA 1,2,1,3,2,3,2,1,3,1,3,2,1,2,3,1,2,3,2,1,3,1,3,2
2480 DATA 1,2,1,3,2,3,1,2,3,1,3,2,1,2,1,3,2,3,2,1,3,1,2,3
2490 DATA 1,2,1,3,2,3,1,2,3,1,3,2,1,2,3,1,2,3,2,1,3,1,3,2
2500 DATA 1,2,1,3,2,3,1,2,3,1,3,2,1,2
2510 :
2520 ON ERROR GOTO 2540
2530 VDU 26:CLS:IF ERR=17 THEN CHAIN "AUTO"
2540 REPORT:PRINT" at line ";ERL
2550 PRINT:PRINT"Press [Function][X] for Notepad Main Menu"
2560 END
2570 :
2580 DEF PROCassemble
2590 fopenout=&B8A5
2600 fopenin=&B8A2
2610 foutblock=&B8AB
2620 finblock=&B896
2630 fclose=&B890
2640 :
2650 FOR PASS = 0 TO 2 STEP 2
2660 P%=Z%
2670 [
2680 OPT PASS
2690 :
2700 .scrn_to_disk
2710 :
2720 CALL map_scrn_in
2730 LD HL,&F000
2740 LD DE,&8000
2750 LD BC,&1000
2760 LDIR
2770 CALL map_scrn_out
2780 LD HL,filename
2790 CALL fopenout
2800 RET NC
2810 LD HL,&8000
2820 LD BC,&1000
2830 CALL foutblock
2840 JP fclose
2850 :
2860 .scrn_from_disk
2870 :
2880 LD HL,filename
2890 CALL fopenin
2900 JR C,froml
2910 LD HL,flag
2920 LD (HL),0
2930 RET
2940 :
2950 .froml
2960 :
2970 LD HL,&8000
2980 LD BC,&1000
2990 CALL finblock
3000 CALL fclose
```

```
3010 CALL map_scrn_in
3020 LD HL,&8000
3030 LD DE,&F000
3040 LD BC,&1000
3050 LDIR
3060 CALL map_scrn_out
3070 LD HL,flag
3080 LD (HL),1
3090 RET
3100 :
3110 .map_scrn_in
3120 :
3130 LD A,(&B003)
3140 LD (state),A
3150 LD A,67
3160 LD (&B003),A
3170 OUT (&13),A
3180 RET
3190 :
3200 .map_scrn_out
3210 :
3220 LD A,(state)
3230 LD (&B003),A
3240 OUT (&13),A
3250 RET
3260 :
3270 .filename
3280 :
3290 DEFM "DEVIL.SCN":DEFB 0
3300 :
3310 .flag
3320 :
3330 DEFB 0
3340 :
3350 .state
3360 :
3370 DEFB 0
3380 ]
3390 NEXT
3400 ENDPROC
```

FOOD.BAS

Food Additive Guide



FOOD.BAS, better watch out for this additive

In these days of increasing consumer awareness, it's surprising how little we still know about exactly what goes into our food. The consumer laws that forced manufacturers to put detailed ingredients on food labels still have a long way to go, as labels often list a great number of the additives under their notorious E numbers, rather than by name. And although you can use a specialised dictionary to look up the plain English names of most additives, when they're listed by number only, the job is that much harder.

FOOD.BAS is a program that tells you a great deal more about your food than the label on the side does, and it identifies nearly every additive that has a significant side-effect – as a worryingly large number of them do.

It manages this by recognising that every "E" number lies within a band that have similar functions, and therefore it doesn't need to know about every additive in existence to be able to tell you at least something about each.

However, Food maintains an extra database (currently 73 additives) of some of the most doubtful "E" numbers presently in use within the EEC to give you more specific information, such as the possible side-effects and the groups considered to be most sensitive to (or even at risk from) these effects.

But when using Food please bear in mind that the information and advice given is gleaned from the various consumer guides available, and is not guaranteed by the program author to be 100 percent accurate, reliable or up to date. If in doubt, please consult your doctor before panicking and removing a particular food from your diet purely on the strength of information contained in this program.

The program is called Food for Thought as, hopefully, it will at least give you that.

USING THE PROGRAM

Type in the listing and save it as FOOD.BAS before trying it out. Type:

```
RUN
```

and immediately the program will prompt you to enter an "E" number.

Food labels sometimes show additive numbers with an "E" before them, and some don't. It makes no difference to the program whether or not you include the letter E in your input, but you may be interested to know that additives have not been fully approved by the EEC unless they start with this letter (the most notorious example is 621 – good old Monosodium Glutamate).

So enter a number about which you would like some information and press [Return]. If the number doesn't lie within a recognised additive band you will be told so, and asked to press [Space] before returning to the input prompt (note that the bands, while at present stable, may have been added to or widened since this program was written

– in which case, you won't be able to get information on some otherwise recognised additives).

If Food recognises the number then almost straightaway it will draw an information card, and list the details for the additive under these headings:

Type:	This is the broad, generic name of the band within which the additive lies; for example Permitted Colours.
Notes:	Here a brief description of the additive's general purpose is given, or perhaps a special note.
Side Effects:	Empty, unless the additive is one of the few for which recognised side effects exist.
Risk Groups:	Empty, unless side effects exist – in which case the specific groups of people affected are listed, if any.
Warnings:	Empty, unless side effects exist – in which case any particular warning that applies is given.

At the top left of the card, Food will print the additive number in large type on the card tab, and if it is an additive it has specific information about it will also know whether the additive is officially recognised by the EEC, and if so will print an "E" before it.

When you have read the card press [Space] to return to the input screen, where you can enter a new additive number.

HOW IT WORKS

30	Points the Basic error handler to Food's own error handling routine at line 1950.
40	Calls PROCsetup to initialise the program, then sits in a REPEAT...UNTIL loop taking additive numbers with PROCinput, looking them up with PROCsearch and displaying the information with PROCcard.
80-140	Set the number of special-case additives, dimension the arrays and read in all data.
180-200	Clear the screen, print the program title and prompt for an additive number.
210	Draws a small box large enough for the user's input and clears legal% before entry and validation of a number.
220	Sets up an outer REPEAT...UNTIL loop within which a small text window is created inside the input box. Then sets up an inner REPEAT...UNTIL loop within which the user's input is read into e$, exiting only when e$ is not empty. Then prints the number in the window again in case it scrolled during input.

230	Sets i% to point at the first actual digit of e$, by examining the first character to see if it is an upper case E.
240	Sets e% to the VALue of e$, using i% to make sure the E (if present) is skipped. Cancels the window and validates the number with a call to FNlegal().
250-270	Print an error message if the number is invalid, wait for [Space], and overprint the message with spaces. This is split over three lines for clarity.
280	Repeats the outer loop until the number is validated (legal%=TRUE).
310-400	Validate the number passed in n% by checking if it falls within the currently active additive bands. If so, set cat% to the value of that band and return TRUE. If not, return FALSE.
440	Fetches the category (type$) and the notes (note$) for the current additive, as any legal number has at least this much information available.
450	Sets the default value of effects$, group$ and warn$ before searching the database. Stores the additive number as a string in num$.
460	Sets a *found* flag, f%, to FALSE, and starts a FOR...NEXT loop (with a% as the loop counter) to search the database for a match, using e%=enum%(a%,1). If one exists, sets the temporary variable m% as a place-marker into the database at the position where the match was found, sets f% to TRUE, and forces an early (but legal) exit from the FOR...NEXT loop by setting the loop counter a% equal to the loop limit max%.
470	Ends the search loop, and returns from the procedure if f% is false. Otherwise, pulls the side effect of the additive from se$() into effect$.
480	Pulls the risk group from rg$ into group$ and the warning from sc$ into warn$.
490	If the current database entry contains a 1 in its first subscript then this is an official "E" number and num$ is set to E, otherwise num$ is set to empty.
500	Builds the "E" number to be displayed on the card tab by converting e% into a string and adding it to whatever num$ currently holds.
530-540	Cancel any windows and print the outline of the card.
550-590	Print the five headings and the messages obtained for the additive.

600-610	Print a prompt and wait for [Space] to be pressed before returning.
640-660	Print the additive number, num$, on the card tab in customised characters, by first setting the X and Y origin far enough from the tab's right edge to accommodate the whole number, and then by calling PROCbig() for each character in the string.
690	Sets c% to point into the custom character array at the right place for the character passed in c$ by subtracting 48 from its ASCII code, or if c$ is an E, by forcing c% to equal 10.
700	Starts the outer row loop.
710	Starts the inner column loop
720	Extracts each character from the line in the array chr$() pointed to by y%, and plots a point at the current X and Y graphics coordinates if the character just extracted is a 1.
730-740	End the inner and outer loop before returning.
780-970	List the name and notes for each additive category recognised by the program, to be read into cat$().
1010-1150	List the special case database of additives to be read into enum%(). Each entry is five numbers long, in the following order: ***Official "E" number:*** 1 if Yes, 0 if No ***Additive number:*** The number proper. ***Side effects:*** Index into se$(). ***Risk Groups:*** Index into rg$(). ***Warning:*** Index into sc$().
1190-1300	List the side effects to be read into se$().
1340-1380	List the risk groups to be read into rg$().
1420-1470	List the warnings to be read into sc$().
1510-1530	List the bit map of the custom character 0, as read into chr$().
1510-1530	List the bit map of the custom character 0, as read into chr$().
1550-1930	List the remaining custom character bit maps, in the order 1, 2, 3, 4, 5, 6, 7, 8, 9, E.
1950	Points the Basic error handler to a full error report in the event of a further error occurring whilst attempting to run AUTO. This is in case AUTO isn't present on your Notepad.
1960	Attempts to run the menu program AUTO if the error was generated by pressing the [Stop] key.
1970	If the error was caused by something else, or if AUTO isn't on your Notepad, a full error report is displayed.

1980	After the error report the Notepad will be left in BBC Basic, so this message is displayed to remind users of how to return to the Notepad main menu.

Functions and procedures

PROCsetup	Sets up the arrays, reads in the special case additive database and the customised numeric typeface.
PROCinput	Displays program title and prompts for an additive number to be entered.
PROCsearch	Performs a search on the database to see if the additive is one with known side effects.
PROCcard	Displays the information card and the additive information.
PROCenumber	Prints the full "E" number on the card tab in the customised typeface.
PROCbig	Prints a single digit or the letter E in the customised typeface.
FNlegal()	Checks if the additive lies within a currently established band.

Main variables and arrays

max%	The number of additives in the special case database.
cat$()	Holds the category name and notes for each currently established band of additives.
enum%()	Holds the database of additives with side effects.
se$()	Lists the side effects used by the database.
rg$()	Lists the risk groups used by the database.
sc$()	Lists the warnings used by the database.
chr$()	Holds bit maps of the 11 customised characters (digits 0 to 9 and the letter E).
e$	The additive number entered by the user.
num$	The additive number stripped of any E and held as a string, for later use in PROCenumber.
legal%	Whether the current additive exists.
cat%	The category or *band* number within which the user's additive lies.
type$	The current additive's category.
note$	The current additive's category notes.
effect$	The current additive's side effects (if any).
group$	The current additive's risk groups (if any).
warn$	The current additive's warning (if any).

The program

```
   10 REM Food for Thought
   20 :
   30 ON ERROR GOTO 1950
   40 PROCsetup:REPEAT:PROCinput:PROCsearch:PROCcard:UNTIL 0
   50 :
   60 DEF PROCsetup
   70 CLS:PRINT "Please wait..."
   80 max%=73:DIM cat$(9,1),enum%(max%,4),se$(11),rg$(4),sc$(5),
chr$(10,13)
   90 FOR n%=0 TO 9:READ cat$(n%,0),cat$(n%,1):NEXT
  100 FOR y%=0 TO max%:FOR x%=0 TO 4:READ enum%(y%,x%):NEXT:NEXT
  110 FOR n%=0 TO 11:READ se$(n%):NEXT
  120 FOR n%=0 TO 4:READ rg$(n%):NEXT
  130 FOR n%=0 TO 5:READ sc$(n%):NEXT
  140 FOR y%=0 TO 10:FOR x%=0 TO 13:READ chr$(y%,x%):NEXT:NEXT
  150 ENDPROC
  160 :
  170 DEF PROCinput
  180 CLS:PRINT TAB(22,1);CHR$(17);"Food for Thought";
  190 PRINT " - The Additive Guide";CHR$(18):PRINT TAB(19,3);CHR$(17);
  200 PRINT "Type in the ""E"" Number you wish to identify";CHR$(18)
  210 MOVE 218,14:DRAW 266,14:DRAW 266,26:DRAW 218,26:DRAW
218,14:legal%=0
  220 REPEAT:VDU 28,37,5,42,5:REPEAT:CLS:INPUT "" e$:UNTIL e$<>"":PRINT
e$;
  230 IF (ASC(LEFT$(e$,1))AND 223)=69 i%=1 ELSE i%=0
  240 e%=VAL(RIGHT$(e$,LEN(e$)-i%)):VDU 26:legal%=FNlegal(e%)
  250 IF NOT legal% PRINT TAB(19,7);"Sorry, not a valid ""E"" Number";
  260 IF NOT legal% PRINT " - press SPACE";: REPEAT:UNTIL GET=32
  270 IF NOT legal% PRINT TAB(19,7);SPC(45);
  280 UNTIL legal%:ENDPROC
  290 :
  300 DEF FNlegal(n%)
  310 IF n%>=100 AND n%<=180 THEN cat%=0:=TRUE
  320 IF n%>=200 AND n%<=290 THEN cat%=1:=TRUE
  330 IF n%>=300 AND n%<=321 THEN cat%=2:=TRUE
  340 IF n%>=322 AND n%<=495 THEN cat%=3:=TRUE
  350 IF n%>=500 AND n%<=620 THEN cat%=4:=TRUE
  360 IF n%>=621 AND n%<=637 THEN cat%=5:=TRUE
  370 IF n%>=900 AND n%<=904 THEN cat%=6:=TRUE
  380 IF n%>=905 AND n%<=907 THEN cat%=7:=TRUE
  390 IF n%>=920 AND n%<=927 THEN cat%=8:=TRUE
  400 IF n%>=1400 AND n%<=1442 THEN cat%=9:=TRUE
  410 =FALSE
  420 :
  430 DEF PROCsearch
  440 type$=cat$(cat%,0):note$=cat$(cat%,1)
  450 effect$="Not known":group$="":warn$="":num$=STR$(e%)
  460 f%=FALSE:FOR a%=0 TO max%:IF e%=enum%(a%,1) m%=a%:f%=TRUE:a%=max%
  470 NEXT:IF NOT f% THEN ENDPROC ELSE effect$=se$(enum%(m%,2))
  480 group$=rg$(enum%(m%,3)):warn$=sc$(enum%(m%,4))
  490 IF enum%(m%,0)=1 num$="E" ELSE num$=""
  500 num$=num$+STR$(e%):ENDPROC
  510 :
  520 DEF PROCcard
  530 VDU 26:CLS:MOVE 0,0:DRAW 479,0:DRAW 479,58
  540 DRAW 50,58:DRAW 48,63:DRAW 0,63:DRAW 0,0:
```

```
  550 PRINT TAB(9,1);CHR$(17);"Type: ";CHR$(18);type$
  560 PRINT TAB(8,2);CHR$(17);"Notes: ";CHR$(18);note$
  570 PRINT TAB(1,3);CHR$(17);"Side Effects: ";CHR$(18);effect$
  580 PRINT TAB(2,4);CHR$(17);"Risk Groups: ";CHR$(18);group$
  590 PRINT TAB(5,5);CHR$(17);"Warnings: ";CHR$(18);warn$;:PROCenumber
  600 PRINT TAB(23,6);CHR$(17);"Press SPACE to try a new additive";
CHR$(18)
  610 REPEAT:UNTIL GET=32:ENDPROC
  620 :
  630 DEF PROCenumber
  640 len%=LEN(num$):xo%=43-8*len%:yo%=60:FOR n%=1 TO len%
  650 PROCbig(MID$(num$,n%,1)):xo%=xo%+8
  660 NEXT:ENDPROC
  670 :
  680 DEFPROCbig(c$)
  690 IF c$="E" c%=10 ELSE c%=ASC(c$)-48
  700 FOR y%=0 TO 13
  710 FOR x%=0 TO 7
  720 IF MID$(chr$(c%,y%),x%+1,1)="1" PLOT 69,x%+xo%,yo%-y%
  730 NEXT
  740 NEXT:ENDPROC
  750 :
  760 REM Categories and Notes
  770 :
  780 DATA "Permitted Colouring"
  790 DATA "Allowed in all except fresh produce, dried milk, tea or
coffee"
  800 DATA "Preservative"
  810 DATA "Helps ensure food safety. Avoid only when you know food is
fresh"
  820 DATA "Permitted Antioxidant"
  830 DATA "Prevents ready-packed foods reacting adversely with the air"
  840 DATA "Emulsifier / Stabiliser"
  850 DATA "Alters the handling properties of foods, especially packet
mixes"
  860 DATA "Miscellaneous Additive"
  870 DATA "Part of the firming, gelling and anti-caking agent group"
  880 DATA "Flavour Enhancer"
  890 DATA "Works by increasing saliva flow or by stimulating taste buds"
  900 DATA "Glazing Agent"
  910 DATA "Provides a polish to sugar confectionaries such as chewing
gum"
  920 DATA "Mineral Hydrocarbon"
  930 DATA "Prevents some dried foods drying out "
  940 DATA "Bleaching Agent"
  950 DATA "Used to bleach, mature and improve various types of flour"
  960 DATA "Modified Starch"
  970 DATA "Where did you buy this food? This additive is illegal in the
UK"
  980 :
  990 REM Additives with noticeable side effects
 1000 :
 1010 DATA 1,102,3,2,2,1,104,3,2,2,0,107,3,2,2,1,110,3,2,2,1,120,3,2,2
 1020 DATA 1,122,3,2,2,1,123,3,2,2,1,124,3,2,2,1,127,1,2,2,0,128,3,2,2
 1030 DATA 1,131,2,3,2,1,132,5,3,2,1,133,3,2,2,1,150,1,1,2,1,151,3,2,2
 1040 DATA 0,153,0,0,5,0,154,1,1,2,0,155,3,2,2,1,200,3,0,0,1,210,7,2,0
 1050 DATA 1,211,2,2,0,1,212,2,2,0,1,213,2,2,0,1,214,2,2,0,1,215,2,2,0
 1060 DATA 1,216,2,2,0,1,217,2,2,0,1,218,2,0,0,1,219,2,0,0,1,220,6,0,0
 1070 DATA 1,221,11,2,3,1,222,11,2,0,1,223,7,2,3,1,224,11,2,3,1,226,7,2,3
 1080 DATA 1,227,7,2,3,1,230,5,0,0,1,231,5,0,0,1,236,3,0,0,1,239,7,0,5
```

```
 1090 DATA 1,249,8,1,1,1,250,5,1,1,1,251,5,1,1,1,252,7,1,1,1,261,0,4,4
 1100 DATA 1,270,8,1,0,1,310,7,2,1,1,311,7,2,1,1,312,7,2,1,1,320,8,1,1
 1110 DATA 1,321,8,1,1,1,325,10,1,0,1,385,8,0,0,1,407,6,0,5,1,412,7,0,0
 1120 DATA 1,413,3,0,0,1,421,5,0,0,1,430,2,3,0,1,431,3,3,0,0,503,6,0,0
 1130 DATA 0,508,6,0,0,0,510,8,4,4,0,514,8,4,4,0,518,10,4,4,0,525,9,0,0
 1140 DATA 0,621,4,1,1,0,622,5,1,1,0,623,0,0,1,0,627,0,0,1,0,631,0,0,1
 1150 DATA 0,635,0,0,1,0,905,6,0,0,0,924,9,0,0,0,925,3,0,0
 1160 :
 1170 REM Specific Side Effects
 1180 :
 1190 DATA "None known"
 1200 DATA "Hyperactivity in children"
 1210 DATA "Allergic reactions"
 1220 DATA "Skin rashes/irritation"
 1230 DATA "Headaches and/or dizziness"
 1240 DATA "Nausea and/or vomiting"
 1250 DATA "Internal irritation"
 1260 DATA "Gastric/digestive upsets"
 1270 DATA "Metabolic disturbances"
 1280 DATA "Severe internal upsets"
 1290 DATA "Toxicity"
 1300 DATA "Respiratory difficulties"
 1310 :
 1320 REM Specific groups at risk
 1330 :
 1340 DATA "None"
 1350 DATA "Very young children"
 1360 DATA "Asthmatic/aspirin sensitive people"
 1370 DATA "People with allergies"
 1380 DATA "Kidney and/or heart patients"
 1390 :
 1400 REM Special Conditions
 1410 :
 1420 DATA "None"
 1430 DATA "Not permitted in foods intended for babies and/or young
children"
 1440 DATA "Not recommended by the Hyperactive Children's Support Group"
 1450 DATA "Should be avoided by asthmatics as potentially dangerous"
 1460 DATA "Should be avoided by people with impaired kidneys"
 1470 DATA "Suspected carcinogen"
 1480 :
 1490 REM Sprite Data
 1500 :
 1510 DATA "00111000","01000100","11000110","11000110","11000110"
 1520 DATA "11000110","11000110","11000110","11000110","11000110"
 1530 DATA "11000110","11000110","01000100","00111000"
 1540 :
 1550 DATA "00011000","00111000","01111000","00011000","00011000"
 1560 DATA "00011000","00011000","00011000","00011000","00011000"
 1570 DATA "00011000","00011000","00011000","01111110"
 1580 :
 1590 DATA "00111000","01000100","11000110","11000110","00000110"
 1600 DATA "00000110","00001100","00011000","00110000","00100000"
 1610 DATA "01100000","01000000","11000110","11111110"
 1620 :
 1630 DATA "00111000","01000100","11000110","11000110","00000110"
 1640 DATA "00000100","00011000","00000100","00000110","00000110"
 1650 DATA "11000110","11000110","01000100","00111000"
 1660 :
 1670 DATA "00001100","00001100","00011100","00011100","00101100"
```

```
1680 DATA "00101100","01001100","01001100","10001100","10001100"
1690 DATA "11111110","00001100","00001100","00011110"
1700 :
1710 DATA "11111110","11000110","11000000","11000000","11000000"
1720 DATA "11111000","11000100","10000110","00000110","00000110"
1730 DATA "11000110","11000110","01000100","00111000"
1740 :
1750 DATA "00111000","01000100","11000110","11000110","11000000"
1760 DATA "11111000","11000100","11000110","11000110","11000110"
1770 DATA "11000110","11000110","01000100","00111000"
1780 :
1790 DATA "11111110","11000110","11000110","11000110","00000110"
1800 DATA "00000100","00001100","00001000","00011000","00011000"
1810 DATA "00011000","00011000","00011000","00011000"
1820 :
1830 DATA "00111000","01000100","11000110","11000110","11000110"
1840 DATA "01000100","00111000","01000100","11000110","11000110"
1850 DATA "11000110","11000110","01000100","00111000"
1860 :
1870 DATA "00111000","01000100","11000110","11000110","11000110"
1880 DATA "11000110","01000110","00111110","00000110","00000110"
1890 DATA "11000110","11000110","01000100","00111000"
1900 :
1910 DATA "11111110","01100110","01100010","01100000","01100000"
1920 DATA "01100100","01111100","01100100","01100000","01100000"
1930 DATA "01100000","01100010","01100110","11111110"
1940 :
1950 ON ERROR GOTO 1970
1960 VDU 26:CLS:IF ERR=17 THEN CHAIN "AUTO"
1970 REPORT:PRINT" at line ";ERL
1980 PRINT:PRINT"Press [Function][X] for Notepad Main Menu"
```

INKEY.BAS

Negative INKEY emulator

```
INKEY(-N) emulator
Press any key combinations, or [Stop] to end...

The value is: &F3
```

INKEY.BAS, simple, but very useful

One of the most useful features of Basic on the BBC Micros is the INKEY() statement in conjunction with a minus value whereby, (for example), the function INKEY(-99) would test whether [Space] is currently held down.

Unfortunately, only the simpler INKEY(time) and INKEY$ are supported by the Notepad. However, there is a routine built into the Notepad's firmware which will test the keys (with the exception of [Shift], [Control] and [Symbol] pressed on their

own), and the program(ette) INKEY.BAS illustrates how you can incorporate this into your own programs.

USING THE PROGRAM

Type in the listing and save it as INKEY.BAS before trying it out. Then type:

```
RUN
```

and the screen will clear and wait for you to press keys. When you do, the value associated with that key will be shown in hexadecimal. If a particular key combination is inactive, the previous value will remain displayed. Certain (unlikely to be needed) combinations will return a value, but it's always the same: &29F. You are therefore recommended not to use any combinations that return this value because so many other combinations also return it.

You will find a full list of values for every possible legal key combination in Appendix 3.

HOW IT WORKS

10-30	These set up the display and prompt the user.
40	Dimensions A% ready to store the machine code.
50	Assembles the machine code into A%.
60	Calls the machine code.
70	Assigns F% the value returned by extracting the two-byte value that has been placed in buffer. If it is zero then no key was pressed so GOTO 60 and keep looking.
80	The value returned (F%) is converted to a single hexadecimal number and displayed.
100-130	The assembly procedure. PASS is used in a FOR...NEXT loop for a two-pass assembly and each pass P% is set to the start of the destination area for the machine code (A%).
150	The label inkey is assigned so that Basic can call the more recognisable inkey, rather than CALL A%, which could mean anything.
160	The firmware routine KMREADKBD is called.
170	The register pair HL is set to point to a buffer that was created during assembly in order to hold the value returned by KMREADKBD.
170-210	The buffer is now loaded with the values of the registers B and C which, together, make up the two-byte result, and the code returns to Basic.

Functions and procedures

PROCassemble	Assembles the machine code.

Main variables and arrays

A%	An array to hold the assembled machine code.
inkey	The start of the machine code.
F%	The value returned.
buffer	Area following the machine code in which the two-byte value returned by KMREADKBD is placed.
P%	Pointer to the area of memory to assemble to.

The program

```
 10 CLS
 20 PRINT "INKEY(-N) emulator"
 30 PRINT:PRINT "Press any key combinations, or [Stop] to end..."
 40 DIM A% 100
 50 PROCassemble
 60 CALL inkey
 70 F%=buffer?0+buffer?1:IF F%=0 THEN GOTO 60
 80 VDU 31,0,6:PRINT "The value is: &"; F%;"   ";
 90 GOTO 60
100 DEF PROCassemble
110 FOR PASS=0 TO 2 STEP 2
120 P%=A%
130 [
140 OPT PASS
150 .inkey
160 CALL &B806
170 LD HL,buffer
180 LD (HL),B
190 INC HL
200 LD (HL),C
210 RET
220 .buffer
230 ]
240 NEXT
250 ENDPROC
```

MORTGAGE.BAS

Loan calculator

```
Mortgage calculator

Enter amount of mortgage: £40000
Enter interest rate     : %9
Enter monthly payment   : £200
  Repayment must be at least £300
Enter monthly payment   : £350
```

MORTGAGE.BAS, how big a mortgage can you afford?

This is a very simple program to check how many years a mortgage will take to clear. With it you can enter the total amount of loan, current interest rate and monthly payment and the program will calculate how many years it will be before you have paid off the loan, and what the total repayments amount to.

Having done that you will then be able to enter varying amounts for your monthly repayments to see what difference the effect of paying more or less each month would have. However, the program will not allow for a repayment less than the minimum monthly payment required to pay off a mortgage because below a certain amount a mortgage would never get paid back and would actually increase each year.

A further point to note is that calculations assume all interest due in each year is paid in 12 equal monthly instalments and that an amount extra is also paid towards reducing the balance. So you cannot use this program to check an endowment (or with profits) mortgage. Also, it is assumed that interest rates remain static throughout the entire period of the loan.

If interest rates are currently fairly high (15%) or fairly low (5%), it might be an idea to adjust the rate to take an educated guess for future changes. For example, a 15% rate might level out over 25 years to an average of 11% or 12%, while a rate of 5% might more realistically average out at 9% or 10%. In any event, when you do this, you should not be optimistic.

USING THE PROGRAM

Type in the listing and save it as MORTGAGE.BAS before trying it out. Then type:

```
RUN
```

You will then be prompted to enter the amount of the loan, the prevailing interest rate and your current monthly repayment. Having done this the program will work out how many years the mortgage will take to pay off and the total amount repaid.

HOW IT WORKS

30	Clears the screen.
40	Points the Basic error handler to a new routine at line 220.
50-80	Prompt the user for the three items of data.
90-110	If the monthly payment is not sufficient, tell the user what the minimum is, and ask for the input again.
120	Sets the year counter to year one.
140	Repeats until finished.
150	Adds the accrued interest for the current year to the amount of mortgage outstanding. Then deducts the total repayments made this year.

160	Prints the current year and how much money is still to be repaid.
170	Increments the year.
180	Looks back to line 140 until the balance is 0 or less.
190-200	Print the total amount repaid and re-run the program.
210	If there was a typing error, or the menu program AUTO was not found then this line tells Basic to GOTO line 230.
220	The user pressed [Stop] so the program has finished. Now call up the menu program, AUTO.
230-240	Either there was a typing error in the listing or the file AUTO was not found. In any event, print the error message and the line at which it occurred and remind the user how to get back into the Notepad.

Main variables

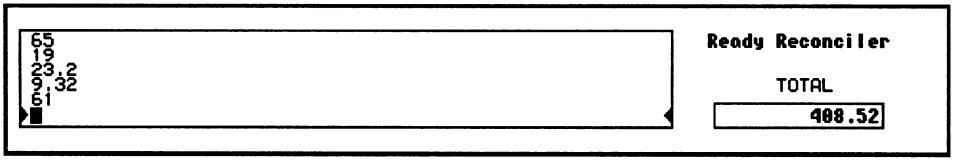
amount%	The amount of the loan.
rate%	The interest rate.
payment%	The monthly repayment.
year%	The current year.

The program

```
 10 REM Mortgage & Loan Calculator
 20 :
 30 CLS
 40 ON ERROR GOTO 220
 50 PRINT "Mortgage calculator":PRINT
 60 INPUT "Enter amount of mortgage: _"amount%
 70 INPUT "Enter interest rate     : %"rate%
 80 INPUT "Enter monthly payment   : _"payment%
 90 IF payment% > (amount%*(rate%/100))/12 THEN GOTO 120
100 PRINT "  Repayment must be at least _";A$ (amount%*(rate%/100))/12
110 GOTO 80
120 year%=1
130 PRINT
140 REPEAT
150 amount%=amount%+amount%*(rate%/100)-payment%*12
160 PRINT "Year ";year%;" Outstanding: _";amount%
170 year%=year%+1
180 UNTIL amount% <= 0
190 PRINT:PRINT "Total repaid _";year%*payment%*12
200 PRINT "Press any key for another calculation...";:G=GET:RUN
210 ON ERROR GOTO 230
220 VDU26:CLS:IF ERR=17 THEN CHAIN "AUTO"
230 REPORT:PRINT " at line ";ERL
240 PRINT:PRINT "Press [Function][X] for Notepad main menu"
```

READYREC.BAS

Statement reconciler



READYREC.BAS, makes reconciling a doddle.

How often have you wished that your calculator could tot up a column of figures, but allow you to make corrections to the entries afterwards? It's a common need, both in business and the home. Whether you're trying to make sense of an order book, your cheque book or even a till receipt, the problem is identical, and is the main reason for the huge popularity of spreadsheet programs.

Spreadsheets allow you to quickly trace discrepancies in a list of numbers or calculations, and make amendments if need be. You can even try out *what if?* scenarios with your figures, by adding one or more hypothetical purchases or sales and then seeing if the new total benefits you in some way.

READYREC.BAS is a program that allows you to do just this. It's based on the more complex program CALC.BAS, and even borrows some of the same procedures, but it's a lot shorter and simpler to use and understand.

Like Calc, you enter a list of figures or complex calculations in a large Input window, scrolling back and forth to make changes where needed, while a separate Totals window displays the current total.

But unlike Calc, the total in Readyrec is calculated from the SUM of every figure or calculation in the list, and also unlike Calc you cannot enter *accumulative* expressions such as this:

```
+10
```

This is because by its very nature, Readyrec automatically adds the result of every new entry to the running total, making such expressions meaningless.

Depending on your profession you will find Readyrec either more or less useful than Calc, and it is because the two programs fill two such different needs that they have both been included in this book, despite the unavoidable repetition of some sections of code from each.

USING THE PROGRAM

Type in the listing and save it as READYREC.BAS before trying it out. Then type:

```
RUN
```

and the cursor will now be sitting in the bottom left of the Input window, between the two arrows that indicate where your typed input will go. Now type in any number, or legal BBC Basic expression such as:

```
10*37/100
```

Notice that your input is shown in bold text as you type. In fact, the contents of the bottom line of the Input window are always shown in bold, because when you are scrolling through previous calculations it serves to highlight the one currently under the cursor. Press [Return] and Readyrec will scroll the Input window up one line, and the Total window will show the result of the calculation.

Now try entering a few simple calculations until the first has completely scrolled off the top of the display, and see how the total in the Total window changes as the result of each entry is added on. Now press [Up] a few times, watching as your previous entries scroll back into view. Note the lines turning bold one by one as they pass through the bottom line of the Input window.

Stop at any time and edit an expression (one of the features of Readyrec is that it is permanently in edit mode, so you can change whatever is under the cursor at any time). Remember that you MUST press [Return] to register the change – if you move off the line with [Up] or [Down], Readyrec will restore the old contents of the line.

To clear all entries, instead of a calculation type:

```
CLEAR
```

(in upper case) and then confirm your decision with the [Y] key.

Line editing is provided by Readyrec, including all the standard editing key functions you would expect. Here's a complete list of the movement and editing keys used in Readyrec:

[Right] Cursor right – Moves the cursor one character to the right.

[Left] Cursor left – Moves the cursor one character to the left.

[Up] Previous line – Scrolls the Input window down, and places the previous entry on the editing line.

[Down] Next line – Scrolls the Input window up, and places the next entry on the editing line.

[Del->] Delete character under cursor – The rest of the line is shunted to the left, while the cursor remains stationary.

[<-Del] Delete character to left of cursor – The rest of the line is shunted to the left, and the cursor also moves one position to the left.

[Control][E] Delete to end of line – All characters to the right of the cursor are deleted, as well as the character under the cursor (ideal for clearing an old line ready for a new entry).

HOW IT WORKS

30	Calls the setup procedure, and points the Basic error handler to Readyrec's own error handling routine.
40	Endlessly calls PROCinput and PROCcalc until [Stop] is pressed.
70	Draws the editing line arrows.
80-90	Draws both window borders.
100-110	Print the program title and the Total window title.
120-170	Print a summary of the instructions in the Input window, which will disappear once the first line is entered.
180	Dimensions the calculation storage array, calls PROCclear to print a 0 in the Total window, and tells Basic to display all numbers to 10 significant figures (the maximum).
210	Runs through A$(), setting all elements to "" (empty).
210	Resets both array pointers, clears the total and displays it in the Total window.
240-280	Set up three text windows. In order of appearance they are the editing line, the Input window and the Total window.
310	Sets up the edit window, pulls the current calculation from A$() into e$, gets its length, sets the editing cursor to the left edge of the window, prints the expression in bold, starts the main input loop and reads a keypress into key%.
320-380	Check the key in key%, and carry out the appropriate editing or movement function.
390	If the keypress was a normal character, inserts it into e$ at the current position by calling PROCinsert.
400	When [Return] is pressed, checks if CLEAR was typed. If so, calls PROCwipe – but if e$ is empty, it's forced to contain 0 for the sake of appearance.
410	Puts the new expression into A$() at the current position and advances the array pointer ptr% (and max% if ptr% was already at the highest element used so far).

420	Checks that max% hasn't exceeded the limits of the array A$() – otherwise adjusts max%.
430	Draws the new Input window contents and returns.
460-470	If x% isn't already at the left-hand side, move it left and redraw the editing line to show the new cursor position.
500-510	If x% isn't already at the end of the line, move it right and redraw the editing line to show the new cursor position.
540-550	If the pointer isn't already at the start of the array, move it to the previous line, display the new window contents and fetch the new line for editing.
580-590	If the pointer isn't already at the last entry in the array, move it to the next line, display the new window contents and fetch the new line for editing.
620	Calls PROClist to update the Input window, pulls the current line from A$() into e$, gets its length, sets the editing cursor to the left edge of the window, sets the edit window up and prints the expression in bold.
650-670	Insert the character key% into e$, if it isn't already at maximum length.
700-720	Remove character to left of current character from e$, unless at start of e$.
750-770	Remove current character from e$, unless at end of e$.
800-820	Truncate e$ at the current position, unless at end of e$.
850-860	Print e$ in bold, followed by the current character in inverse to act as the cursor.
900-920	Clear Input window and fill it from A$(), starting from either five lines before the current line, or the start of the array if less than five entries exist.
950	Clears the total and starts running through each entry in the array A$(), evaluating and adding its total to tot if it isn't a blank line.
960	Finishes adding the totals, converts the new total to a string so that it can be padded with spaces and appear right-justified.
970	Prints the new total bin the Total window, in bold text before returning.
1000-1020	In answer to the user typing CLEAR, display a safety message on the editing line in bold. If user presses [Y] in response, clear all entries with PROCclear.
1030	Calls PROCnewline to redraw the Input window and put the current calculation back in the editing line before returning.
1080	Resets Basic's numeric accuracy to normal and attempts to run

	the menu program AUTO if the error was generated by pressing the [Stop] key.
1090	If the error was *No such file*, AUTO isn't on your Notepad so jump to the full error report.
1100-1120	If the program gets to here an illegal calculation was made. The user is informed and asked to acknowledge by pressing [Space]. PROCnewline is called to redraw the Input window and redisplay the current calculation on the editing line, and a direct jump is made back to main loop at line 40. **Important:** This can only be allowed to happen a certain number of times before the Basic stack overflows with PROC calls that the error handler has jumped out of before reaching the ENDPROC.
1130	Displays a full error report.
1140	After the error report the Notepad will be left in BBC Basic, so this message is displayed to remind users of how to return to the Notepad main menu.

Functions and procedures

PROCsetup	Draws the screen and sets up arrays and main variables.
PROCclear	Clears the Input window, resets the Total window.
PROCinput	Takes input from the keyboard, and calls relevant routines for inserting and deleting characters or moving around.
PROCleft	Moves the cursor one character to the left.
PROCright	Moves the cursor one character to the right.
PROCup	Scrolls the window down and places the previous entry on the editing line.
PROCdown	Scrolls the window up and places the next entry on the editing line.
PROCnewline	Redraws the Input windows at the current position and fetches the current line for editing.
PROCinsert	Inserts a character into the input line.
PROCdel1	Performs [<-Del].
PROCdel2	Performs [Del->].
PROCdel3	Performs [Control][E].
PROChilite	Prints the current line in bold, and inverses the current character to act as a screen cursor.
PROClist	Updates the Input window.
PROCcalc	Clears the total, evaluates all the calculations entered so far, adds the result of each to the total and displays the new total.

PROCwipe	Displays a safety prompt before calling PROCclear to clear all entries.

Main variables and arrays

A$()	The input array which holds all the calculations.
max%	Pointer to the highest element of A$() currently used.
ptr%	Pointer to the current element of A$() being edited.
tot	The total of all calculations entered.
key%	The current keypress being examined.
e$	The expression currently being edited.
l%	The current length of the expression being edited.
x%	The current cursor position on the editing line.

The program

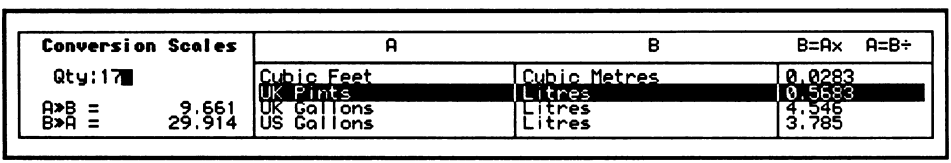
```
   10 REM Ready Reconciler
   20 :
   30 PROCsetup:ON ERROR GOTO 1080
   40 REPEAT:PROCinput:PROCcalc:UNTIL FALSE
   50 :
   60 DEF PROCsetup
   70 VDU 26:CLS:PRINT TAB(0,6);CHR$(27);CHR$(16);TAB(56,6);CHR$(27);
CHR$(17);
   80 MOVE 0,6:DRAW 342,6:DRAW 342,57:DRAW 0,57:DRAW 0,6
   90 MOVE 364,6:DRAW 452,6:DRAW 452,18:DRAW 364,18:DRAW 364,6
  100 PRINT TAB(60,1);CHR$(17);"Ready Reconciler";CHR$(18)
  110 PRINT TAB(66,4);"TOTAL";:PROCwinlist
  120 PRINT TAB(21,0);CHR$(17);CHR$(19);"Instructions";CHR$(20);CHR$(18);
  130 PRINT TAB(1,2);"Enter a list of formulae to be summed, using ";
  140 PRINT CHR$(17);CHR$(27);CHR$(30);CHR$(18);", ";
  150 PRINT CHR$(17);CHR$(27);CHR$(31);CHR$(18);" and "
  160 PRINT TAB(1,3);CHR$(17);"Return";CHR$(18);" to edit any line. ";
  170 PRINT"Type ";CHR$(17);"CLEAR";CHR$(18);" to clear the list."
  180 DIM A$(255):PROCclear:@%=&A0C:ENDPROC
  190 :
  200 DEF PROCclear
  210 FOR p%=0 TO 255:A$(p%)="":NEXT
  220 max%=0:ptr%=0:tot=0:PROCcalc:ENDPROC
  230 :
  240 DEF PROCwinin:VDU 28,1,6,55,6:ENDPROC
  250 :
  260 DEF PROCwinlist:VDU 28,1,5,55,1:ENDPROC
  270 :
  280 DEF PROCwintot:VDU 28,61,6,74,6:ENDPROC
  290 :
  300 DEF PROCinput
  310 PROCwinin:e$=A$(ptr%):x%=1:l%=LEN(e$):PROChilite:REPEAT:key%=GET
  320 IF key%=242 PROCleft
  330 IF key%=243 PROCright
  340 IF key%=240 PROCup
  350 IF key%=241 PROCdown
  360 IF key%=127 PROCdell
```

```
370 IF key%=33 PROCdel2
380 IF key%=5 PROCdel3
390 IF key%<>33 AND key%<>5 AND key%>31 AND key%<127 PROCinsert
400 UNTIL key%=13:IF e$="CLEAR" PROCwipe:ENDPROC ELSEIF e$="" e$="0"
410 A$(ptr%)=e$: ptr%=ptr%+1:IF ptr%>max% max%=max%+1
420 IF max%>255 max%=max%-1:ptr%=ptr%-1
430 PROClist:ENDPROC
440 :
450 DEF PROCleft
460 IF x%=1 ENDPROC
470 x%=x%-1:PROChilite:ENDPROC
480 :
490 DEF PROCright
500 IF x%=l%+1 ENDPROC
510 x%=x%+1:PROChilite:ENDPROC
520 :
530 DEF PROCup
540 IF ptr%=0 ENDPROC
550 CLS:ptr%=ptr%-1:PROCnewline:ENDPROC
560 :
570 DEF PROCdown
580 IF ptr%=max% ENDPROC
590 CLS:ptr%=ptr%+1:PROCnewline:ENDPROC
600 :
610 DEF PROCnewline
620 PROClist:e$=A$(ptr%):x%=1:l%=LEN(e$):PROCwinin:PROChilite:ENDPROC
630 :
640 DEF PROCinsert
650 IF l%=54 ENDPROC
660 e$=LEFT$(e$,x%-1)+CHR$(key%)+RIGHT$(e$,l%+1-x%)
670 l%=l%+1:x%=x%+1:PROChilite:ENDPROC
680 :
690 DEF PROCdel1
700 IF x%=1 ENDPROC
710 e$=LEFT$(e$,x%-2)+RIGHT$(e$,l%+1-x%)
720 x%=x%-1:l%=l%-1:PROChilite:ENDPROC
730 :
740 DEF PROCdel2
750 IF x%=l%+1 ENDPROC
760 e$=LEFT$(e$,x%-1)+RIGHT$(e$,l%-x%)
770 l%=l%-1:PROChilite:ENDPROC
780 :
790 DEF PROCdel3
800 IF x%=l%+1 ENDPROCA
810 e$=LEFT$(e$,x%-1)
820 l%=x%-1:PROChilite:ENDPROC
830 :
840 DEF PROChilite
850 c$=MID$(e$,x%,1):IF c$="" c$=" "
860 CLS:PRINT CHR$(17);e$;TAB(x%-1,0);CHR$(14);c$;CHR$(15);CHR$(18);
870 ENDPROC
880 :
890 DEF PROClist
900 PROCwinlist:CLS:IF ptr%=0 ENDPROC
910 IF ptr%<5 top%=5-ptr%:p%=0 ELSE top%=0:p%=ptr%-5
920 FOR y%=top% TO 4:PRINT TAB(0,y%);A$(p%);:p%=p%+1:NEXT:ENDPROC
930 :
940 DEF PROCcalc
950 tot=0:FOR p%=0 TO max%:IF A$(p%)<>"" tot=tot+EVAL(A$(p%))
960 NEXT:PROCwintot:t$=STR$(tot):t$=STRING$(14-LEN(t$),CHR$(32))+t$
```

```
  970 CLS:PRINT CHR$(17);t$;CHR$(18);:ENDPROC
  980 :
  990 DEF PROCwipe
 1000 PROCwinin:CLS
 1010 PRINT TAB(11,0);CHR$(17);"Clear list - Are you sure (Y/N)?";
CHR$(18);
 1020 REPEAT:g%=GET AND 223:UNTIL g%=89 OR g%=78:CLS:IF g%=89 PROCclear
 1030 PROCnewline:ENDPROC
 1040 :
 1050 REM This last section handles lines rejected by EVAL.
 1060 REM Note: Repeated errors will eventually overflow the stack.
 1070 :
 1080 IF ERR=17 @%=&90A:VDU 26:CLS:CHAIN"AUTO"
 1090 IF ERR=214 GOTO 1130
 1100 err$="Error in "+A$(p%)+" - press SPACE":PROCwinin:CLS
 1110 PRINT TAB(27-LEN(err$)/2,0);CHR$(17);err$;CHR$(18);
 1120 REPEAT:UNTIL GET=32:CLS:ptr%=p%:PROCnewline:GOTO 40
 1130 VDU 26:CLS:REPORT:PRINT" at line ";ERL
 1140 PRINT:PRINT"Press [Function][X] for Notepad Main Menu"
```

SCALES.BAS

Conversion Scales



SCALES.BAS, stay conversant with conversions.

It's often useful to convert figures from one unit of measurement to another, and by now most of us know the old trick of multiplying inches by 2.54 to get centimetres. But it would be nice to have a decent set of conversion tables handy for all occasions, because you never know when you might need to change ounces to grams or kilograms, or perhaps litres into gallons.

SCALES.BAS was designed to fill this gap, and it's much easier to use than those little bits of paper you can buy to fit inside your Personal Organiser because it does the conversions for you. Not only that, it will store any number you enter and show it converted between any one of the many units of measurement it supports (currently 34) – all you have to do is scroll up and down the list, and the program does the rest.

And it's expandable, should you find its current repertoire too limited. The method is detailed in the line-by-line program explanation, in the discussions of lines 790 onward (especially lines 930 – 1260).

USING THE PROGRAM

Type in the listing and save it as SCALES.BAS before trying it out. Type:

```
RUN
```

and after a short while (during which it reads in and expands the data) the screen will change to show a large, scrollable window containing the first few units of measurement, with the highest pair highlighted in inverse colours.

On the left is the status box into which you can enter a number to be converted. This is shown next to the word *Qty*. Below this are two totals, labelled *A>B=* and *B>A=*. The purpose of these will become clear when you look at the larger window which is divided into three parts: the two units to convert between and the conversion factor (included purely for show, and as found in standard paper conversion tables).

Above the left and right-hand units of measurement are the letters *A* and *B* respectively, and these are used in the status box to show the direction of conversion between the two units.

Whatever value is currently entered next to *Qty*, the number shown immediately below is the result of converting this value from unit A to unit B, while below this is the result going the other way – that is, from unit B to unit A.

To take the example of converting from inches to centimetres (which is at the top of the list when you first run the program), a Qty of 1 would give results of 2.54 (representing inches to centimetres) and 0.393 (centimetres to inches) respectively. Initially the Qty value is set to zero. To enter a number for conversion, press [Return] at any time. Type in the number and press [Return] again, and the results in both directions are immediately shown below.

You can now scroll through the various conversions available using the [Up] and [Down] keys; new results for your original input will be calculated for the currently highlighted unit pairs as you move through the table, and your input will remain unchanged until you press [Return] to enter a new value.

HOW IT WORKS

30	Points the Basic error handler to Scale's own error handling routine at line 1280, and restores Basic's original format of numeric output in the event of an error.
40	Cancels any windows and asks the user to wait while the DATA is read in and expanded.
50	Calls PROCsetup to read in the data, PROChighlight to invert the top line of the units of measurement window and then sits in an infinite REPEAT...UNTIL loop reading keys and fetching the user's input.

90	Stores Basic's original numeric output format and then sets it always to display numbers in floating point with a maximum of three decimal places in a field width of 12. Then clears the input variable *input*, sets the array pointer scale% to 1 and sets the window row pointer row% to 0.
100	Reads the size of the lexicon, dimensions the lexicon store array and reads in the words.
110-130	Read the size of the table, dimension the array and read in the data, expanding the numbers into the words that are stored in the lexicon.
140-200	Clear and draw the screen, making good use of VDU statements to cut down on the number of PRINT CHR$() commands that would otherwise be necessary.
210	Prints the starting value, 0, and calls PROCoutput to print both conversions (quicker than printing 0.000 on the result lines).
220	Activates the main window and prints the first screenful of available conversions.
260-280	Read the keyboard until [Return] is pressed, checking for the [Up] and [Down] keys and calling the relevant scroll routine if either is pressed and the pointer is able to move in that direction.
310	Un-inverses the current line and decrements both the pointer and the current screen row.
320	If the screen pointer is off screen, adjusts it and redraws the window to simulate a downward scroll.
330	Inverses the new current line, calculates and displays the new conversions and returns.
360	Un-inverses the current line and increments both the pointer and the current screen row.
370	If the screen pointer is off screen, adjusts it and executes a line feed from the bottom line to force an upward scroll, after which the new bottom line is pulled from the array and printed.
380	Inverses the new current line, calculates and displays the new conversions and returns.
410	Fills the main window from the units of measurement array, with the current line at the top.
440	Inverses the entire width of the main window at the current row position by using PLOT 102 (inverse rectangle).
480	Flushes the keyboard buffer by reading INKEY(0) until no key press is returned, then clears the input window and prompts for a number.

490	Clears the input window again, and prints the user's input so that it is formatted the same way as the results.
520	Selects and clears the A>B result window and prints the conversion going from left to right.
530	Selects and clears the B>A result window and prints the conversion going from right to left.
570-720	Define the windows used, which in order are program title, table header, the table itself, input, A>B and B>A.
750	Returns the passed string with the passed number of spaces tacked on the end.
790	Starts reading a complete unit of measurement from data consisting of four numbers. Each number is an index into the lexicon, and has been ORed with 64 if the last letter isn't wanted (to turn a plural into a singular) or with 128 if the last two letters aren't wanted (special case – turns *inches* into *inch*). This system saves on space, effectively doubling the number of words in the lexicon.
800-810	The numbers are ANDed in turn with 64 and 128 and chopped by one or two characters respectively.
820	If the number pointed to a real lexicon entry and was not zero (denoting no word) it is joined to p$ and a space is added – unless it is the last of the four words.
830	Closes the loop and returns the newly-concatenated string.
860	Holds the number of words in the lexicon.
870-900	List the words in the lexicon – all the complex strings in the table are made from these building blocks.
920	Holds the number of entries in the conversion table.
930-1260	Hold the entire conversion table as pairs of four numbers followed by a conversion factor for that pair. The four numbers each represent a word from the lexicon, but some are further coded by being ORed with 64 or 128, denoting that the last one or two letters respectively of that word are not wanted when the table is constructed. This allows re-using plural words as singular, so long as all that is required to turn them singular is to remove no more than two letters from the end.

Functions and procedures

PROCsetup	Dimensions the arrays, reads in the data and draws the screen.
PROCscan	Reads the keyboard until [Return] is pressed, checking for the [Up] and [Down] keys and calling the relevant scroll routine if either is pressed.

PROCscroll_up	Scrolls the units of measurement window down to highlight the previous pair of units.
PROCscroll_down	Scrolls the units of measurement window up to highlight the next pair of units.
PROCscreen	Redraws the units of measurement window with the current pair of units at the top.
PROChighlight	Inverses the entire width of the units of measurement window at the current row. Called twice in a row to cancel the effect.
PROCinput	Takes a new input from the user and displays it.
PROCoutput	Converts the current user input both ways using the current pair of units and displays the two results.
FNpad()	Returns the passed string with the passed number of spaces tacked on the end.
FNconcat	Builds the left and right-hand units of measurement strings from data.

Main variables and arrays

lex$()	Holds the lexicon of words from which the units of measurement names are built.
table$()	Holds the text of the conversion table.
scale()	Holds the conversion factors for all units of measurement pairs.
lex%	The number of words in the lexicon.
max%	The number of entries in the conversion table.
scale%	Pointer to the current units of measurement pair in table$() and its conversion factor in scale().
row%	The current screen row in the units of measurement window.
input	The current user's input.

The program

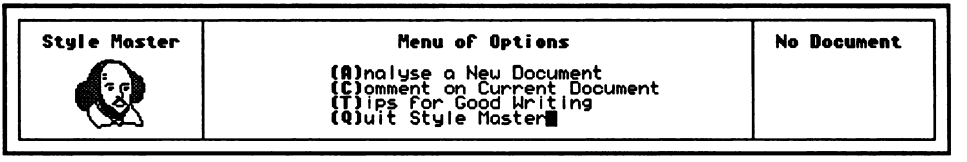
```
  10 REM Sliding Conversion Scales
  20 :
  30 ON ERROR @%=fmt%:GOTO 1280
  40 VDU 26:CLS:PRINT "Please wait..."
  50 PROCsetup:PROChighlight:REPEAT:PROCscan:PROCinput:UNTIL FALSE
  60 END
  70 :
  80 DEF PROCsetup
  90 fmt%=@%:@%=&2030C:input=0:scale%=1:row%=0
 100 READ lex%:DIM lex$(lex%):FOR n%=1 TO lex%:READ lex$(n%):NEXT
 110 READ max%:DIM table$(max%):DIM scale(max%):FOR n%=1 TO max%
 120 t1$=FNpad(FNconcat,22)+CHR$(179):t2$=FNpad(FNconcat,22)+CHR$(179)
 130 READ f$:scale(n%)=VAL(f$):t3$=FNpad(f$,11):table$(n%)=t1$+t2$+t3$:
NEXT
 140 CLS:MOVE 0,4:DRAW 479,4:DRAW 479,59:DRAW 0,59:DRAW 0,4
```

```
  150 MOVE 123,4:DRAW 123,59:MOVE 123,43:DRAW 479,43
  160 MOVE 260,4:DRAW 260,43:MOVE 398,4:DRAW 398,43
  170 PROCwin1:PRINT TAB(1,0);CHR$(17);"Conversion Scales";CHR$(18)
  180 PRINT TAB(2,2);"Qty:";:VDU 31,1,4,65,175,66,32,61,31,1,5,66,175,65,
32,61
  190 PROCwin2:VDU 31,11,0,65,31,34,0,66
  200 VDU 31,47,0,66,61,65,120,32,32,65,61,66,246
  210 PROCwin_in:CLS:PRINT input;:PROCoutput
  220 PROCwin3:PROCscreen
  230 ENDPROC
  240 :
  250 DEF PROCscan
  260 REPEAT:i%=INKEY(0):IF i%=240 AND scale%>1 PROCscroll_up
  270 IF i%=241 AND scale%<max% PROCscroll_down
  280 UNTIL i%=13:ENDPROC
  290 :
  300 DEF PROCscroll_up
  310 PROChighlight:scale%=scale%-1:row%=row%-1
  320 IF row%<0 row%=0:PROCwin3:PROCscreen
  330 PROChighlight:PROCoutput:ENDPROC
  340 :
  350 DEF PROCscroll_down
  360 PROChighlight:scale%=scale%+1:row%=row%+1
  370 IF row%=4 row%=3:PROCwin3:VDU 31,0,3,10:PRINT table$(scale%);
  380 PROChighlight:PROCoutput:ENDPROC
  390 :
  400 DEF PROCscreen
  410 FOR g%=0 TO 3:PRINT TAB(0,g%);table$(g%+scale%);:NEXT:ENDPROC
  420 :
  430 DEF PROChighlight
  440 MOVE 126,64-(row%+3)*8:PLOT 102,474,64-(row%+4)*8
  450 ENDPROC
  460 :
  470 DEF PROCinput
  480 REPEAT:UNTIL INKEY(0):PROCwin_in:CLS:INPUT""input;
  490 CLS:PRINT input;:PROCoutput:ENDPROC
  500 :
  510 DEF PROCoutput
  520 PROCwin_to:CLS:PRINT input*scale(scale%);
  530 PROCwin_from:CLS:PRINT input/scale(scale%);
  540 ENDPROC
  550 :
  560 DEF PROCwin1
  570 VDU 28,1,6,19,1:ENDPROC
  580 :
  590 DEF PROCwin2
  600 VDU 28,21,1,78,1:ENDPROC
  610 :
  620 DEF PROCwin3
  630 VDU 28,21,6,78,3:ENDPROC
  640 :
  650 DEF PROCwin_in
  660 VDU 28,7,3,19,3:ENDPROC
  670 :
  680 DEF PROCwin_to
  690 VDU 28,7,5,19,5:ENDPROC
  700 :
  710 DEF PROCwin_from
  720 VDU 28,7,6,19,6:ENDPROC
  730 :
```

```
 740 DEF FNpad(p$,p%)
 750 p$=p$+STRING$(p%-LEN(p$),CHR$(32))
 760 =p$
 770 :
 780 DEF FNconcat
 790 p$="":FOR w%=1 TO 4:READ v%:w$=lex$(v% AND 31)
 800 IF v% AND 64 w$=LEFT$(w$,LEN(w$)-1)
 810 IF v% AND 128 w$=LEFT$(w$,LEN(w$)-2)
 820 IF w$<>"" p$=p$+w$:IF w%<4 p$=p$+CHR$(32)
 830 NEXT:=p$
 840 :
 850 REM Lexicon
 860 DATA 29
 870 DATA Inches,Feet,Yards,Miles,Hectares,Acres,Gallons,Ounces
 880 DATA Pounds,Hundredweights,Long tons,Centimetres,Metres
 890 DATA Kilometres,Litres,Grammes,Kilogrammes,Tonnes,Square
 900 DATA Cubic,Nautical,UK,US,per,Minute,Hour,Pints,Kilos,cm
 910 REM Conversion Table
 920 DATA 34
 930 DATA 1,0,0,0,12,0,0,0,2.54
 940 DATA 2,0,0,0,12,0,0,0,30.48
 950 DATA 2,0,0,0,13,0,0,0,0.3048
 960 DATA 3,0,0,0,13,0,0,0,0.9144
 970 DATA 4,0,0,0,13,0,0,0,1609.3
 980 DATA 4,0,0,0,14,0,0,0,1.609
 990 DATA 21,4,0,0,14,0,0,0,1853.27
1000 DATA 19,1,0,0,19,12,0,0,6.452
1010 DATA 19,2,0,0,19,12,0,0,929.0304
1020 DATA 19,2,0,0,19,13,0,0,0.092903
1030 DATA 19,3,0,0,19,13,0,0,0.836
1040 DATA 19,4,0,0,19,14,0,0,2.58999
1050 DATA 6,0,0,0,19,13,0,0,4046.86
1060 DATA 6,0,0,0,5,0,0,0,0.404686
1070 DATA 6,0,0,0,19,14,0,0,0.004047
1080 DATA 20,1,0,0,20,12,0,0,16.387
1090 DATA 20,2,0,0,15,0,0,0,28.317
1100 DATA 20,3,0,0,20,13,0,0,0.76
1110 DATA 20,2,0,0,20,13,0,0,0.0283
1120 DATA 22,27,0,0,15,0,0,0,0.5683
1130 DATA 22,7,0,0,15,0,0,0,4.546
1140 DATA 23,7,0,0,15,0,0,0,3.785
1150 DATA 22,7,0,0,23,7,0,0,1.20095
1160 DATA 8,0,0,0,16,0,0,0,28.3495
1170 DATA 9,0,0,0,16,0,0,0,453.59237
1180 DATA 9,0,0,0,17,0,0,0,0.45359
1190 DATA 11,0,0,0,18,0,0,0,1.01605
1200 DATA 11,0,0,0,17,0,0,0,1016.05
1210 DATA 10,0,0,0,17,0,0,0,50.8
1220 DATA 4,24,26,0,2,24,25,0,88
1230 DATA 4,24,26,0,14,24,26,0,1.609344
1240 DATA 4,24,22,71,14,24,79,0,0.35401
1250 DATA 4,24,23,71,14,24,79,0,0.42514
1260 DATA 9,24,19,129,28,24,19,29,0.0703
1270 :
1280 ON ERROR GOTO 1300
1290 VDU 26:CLS:IF ERR=17 THEN CHAIN "AUTO"
1300 REPORT:PRINT" at line ";ERL
1310 PRINT:PRINT"Press [Function][X] for Notepad Main Menu"
```

STYLE.BAS

Style Checker



STYLE.BAS, correct your writing wrongs.

Style checkers are programs that attempt to analyse your writing for errors in grammar and style, and the better versions manage this seemingly incomputable task with a fairly high degree of accuracy. It has been argued that you shouldn't take the advice offered by these programs too seriously as it tends to be rather pedantic and robotic, but it's surprising just what can be learnt from more modest examples of the genre, given their limited scope.

STYLE.BAS is just such a program, and while it doesn't exactly jangle with bells and whistles it can certainly give some of the big guns a run for their money.

Firstly, though, you should note that Style can seem slow at times, especially when digesting large documents – but after all, it is written in Basic as opposed to machine code, and it gets there in the end.

As a fully working, seriously written program Style serves as a good example of just what you can do with the much maligned Basic language – just don't let it loose on your first novel, unless you've got some time on your hands.

And now a brief word about style checking (very brief – a full discussion could easily fill the pages of this book) and in particular the points checked for by Style.

Four distinct areas of writing style are looked at by the program: use of passive verbs, hidden verbs, abstract nouns and complex sentences. On top of this a full readability score is generated at the end of each analysis, giving both the standard Fog and Flesch-Kincaid Indices for the piece.

Passive verbs are by far the worst offenders in writing (readers interested in exploring the subject further should consult a copy of *Fowler's Modern English Usage*, published by Oxford University Press as part of their Oxford Reference series).

And although passive verbs are easy to avoid, they account for most bad writing habits. Here is an example:

It has been decided that all coffee breaks are now banned.

Who decided it? You can't tell from the wording, and this is a trick people often use when they wish to remain anonymous and unconnected with a particular memo or announcement. The result, apart from being rather impersonal, is a very stuffy and lifeless writing style.

The trouble is that most people think that this is the correct way to write formally, when it is actually the worst way to write anything. Here is the corrected version, this time with the verb *to be* made active:

I have decided that all coffee breaks are now banned.

This is much better. Not only is it now clear who was responsible for the decision, but the whole sentence has come alive, and is more approachable. Well, sort of.

If you can catch passive verbs and, where possible, remove them from your writing, you'll probably improve it more than by any other single method.

USING THE PROGRAM

Type in the listing and save it as STYLE.BAS before trying it out. Type:

```
RUN
```

and after a short while (during which it reads in and displays a rather inspirational bust of Shakespeare) the Main Menu will appear.

Press [A] to analyse a new document, and the Notepad File Selector window will appear. Choose the document you want to analyse and press [Return], and it will be read in word by word. There is no restriction on the size of documents that Style will handle, but you'll have to be pretty patient if you want to see how War and Peace stacks up against the collected works of Stephen King.

As soon as the document is read, you will get an instant judgement on the right-hand side of the screen. This is based purely on the reading age that Style judges you would need to understand the document without difficulty.

Also displayed on the right is the famous Fog Index, as well as the not quite as famous Flesch-Kincaid Index, should you be interested. Briefly, the higher the Fog Index the harder the writing is to understand, and a value of 12 is about average.

If you want a more subjective assessment of the piece, press [C] from the Main Menu for a brief, plain English commentary on the overall style, structure, meaning and impact of the document.

Pressing [T] displays a short list of tips for good writing, all of them widely recognised and well-proven. Finally, pressing [Q] will quit Style. So does pressing

[Stop] for that matter, but exiting *properly* ensures that a friendly safety warning is always displayed, just in case you pressed the wrong key.

HOW IT WORKS

30	Points Basic's error handler to line 60, where the program ends up when it has finished.
90	Dimensions two memory arrays to hold the two machine code routines used by Style.
100	Loads previously saved screen file from disk, if it exists.
110-120	Draw a border around the screen, and two vertical lines to delineate the central window.
130	Sets up the left-hand window and prints the program title.
140	Calls PROCbust to draw a picture of Shakespeare – this is jumped over if the program has been used at least once, in which case the ready-drawn screen will be loaded into display ram.
150	Calls PROCnew to reset the readability scores and document name, and sets up various thresholds used during analysis.
160-180	Dimension the comment arrays and read in all the style comments.
190-200	Dimension the passive and hidden verb arrays and read in the passive verb partners and hidden verb endings.
210	Saves the screen, if no screen file for this program exists.
240-250	Clear the document flag and current document name, and reset the readability scores.
260	Displays the current document status.
280-320	Define the three text windows, which in order are: the left hand box, the central window and the right hand status box.
350-360	Display the current document name, and if none is loaded return from the procedure.
370-440	Display the overall readability rating.
450	Sets up Basic's numeric output format to display numbers in floating point with always one decimal place, in a field width of five.
460-480	Print the document readability statistics: average sentence length, Fog Index and Flesch-Kincaid Index.
490	Restores Basic's numeric output format.
520	Sets the origin for the top left of the bust.
530-590	Draw Shakespeare's bust, by decoding the bit map data into a 36x36 pixel sprite and plotting the pixels individually.
620-640	Plot the point passed by PROCbust at the position of the current

	X and Y loop variables plus the original top-left offset in xo%, yo%.
670-710	Print the menu title and options.
720-750	Read the keyboard until a valid key is pressed, and call the appropriate procedure.
760	If the interpreter got this far, [Q] was pressed so Style confirms that the user wants to quit, setting quit% TRUE if [Y] is pressed in response.
800	If a document is currently loaded, Style ensures user really wants to analyse another.
810	Calls the machine code routine for displaying the Notepad file selector, after which the screen is reloaded from disk.
820	If the user pressed [Stop] to exit the file selector, FNselect will have returned "", so return.
830-900	Read in each word from the selected document, counting the end-of-sentence flags, calculate the overall readability scores, set the *document loaded* flag and print the readability statistics.
920-960	Prompt the user to press [Y] or [N] to confirm the action passed in m$. Return TRUE if [Y] pressed, otherwise FALSE.
1020-1040	Calculate the Fog Index and the Flesch-Kincaid Index for the document just analysed.
1080-1110	Calculate the percentage of sentences in which each of the four style errors occurred.
1150-1220	Read the next word from the open document, discarding non-alphabetic characters and setting the end-of-sentence flag eos% TRUE if any of the three main stop characters is found.
1250-1260	Return TRUE if the passed character is a letter.
1280-1300	Return TRUE if the passed character is an end-of-sentence marker.
1320-1340	Set up various counters ready for the start of the next sentence.
1360-1400	Check the passed word for any one of these three style errors: passive verb, hidden verb, abstract noun.
1420-1470	Return the number of syllables in the passed word.
1490-1510	Return TRUE if the passed character is a vowel.
1530-1580	Check to see if the passed word is the final part of a passive verb – these are spread across two words, so a flag is already set if the previous word was the start of a possible passive verb.
1600-1630	Check if the passed word is a hidden verb.
1650-1670	Check if the passed word is an abstract noun.
1690-1710	Check if the sentence just read is complex.

1730-1800	Display a style commentary based on the percentage of the four errors checked for that were found.
1820-1870	Return an index into one of the four commentary arrays, taking a style fault percentage as input.
1890-1960	Display four tips for good writing.
1980-2100	The complete bit map of Shakespeare's bust.
2130	The words which invariable start passive verb word pairs.
2160	The letters with which hidden verbs invariably end.
2190-2230	List five possible comments pertinent to the overall style of the piece.
2260-2300	List five possible comments pertinent to the overall structure of the piece.
2330-2370	List five possible comments pertinent to the overall clarity of the piece.
2400-2440	List five possible comments pertinent to the overall impact of the piece.
2460-2530	The function responsible for calling the Notepad file selector code. The function returns either a file name, or a null string if [Stop] was pressed.
2550-2780	Assemble machine code that will call the Notepad file selector.
2800	Points the Basic error handler to a full error report in the event of a further error occurring while attempting to run the menu program AUTO. This is in case AUTO isn't present on your Notepad.
2810	Attempts to run the menu program AUTO if the error was generated by pressing the [STOP] key.
2820	If the error was caused by something else, or if AUTO isn't on your Notepad, a full error report is displayed.
2830	After the error report the Notepad will be left in BBC Basic, so this message is displayed to remind users of how to return to the Notepad main menu.
2860	Start of procedure that assembles the screen saver/loader, which saves a copy of the screen after everything is drawn the first time, and loads it in each time thereafter.
2870-2910	Define the five NC100 jump block routines to be used.
2930-2940	Begin the two-pass assembly and set P% (the assembly destination pointer) to the start of the previously dimensioned Z%.
3000	Pages the 16K of RAM with the video memory in at address &C000.

3010-3040	Copy the contents of video RAM down to &8000.
3050	Puts back the video RAM.
3060-3070	Open a file for saving the screen data.
3080	Returns if unable to open the file.
3090-3110	Save &1000 bytes from &8000 to the file.
3120	Closes the file and exits.
3160-3170	Open a file for reading.
3190-3210	If unable to open the file set the contents of flag to zero and return.
3250-3280	Read the &1000 bytes to location &8000 then close the file.
3290-3340	Map the video RAM 16K block into &C000, copy the &1000 bytes from location &8000 up to &F000 and then put back the screen RAM.
3350-3370	To indicate successful loading, set the contents of flag to 1 then return.
3410-3420	Save the current status of the bank switcher for block 4 (&C000-&FFFF).
3430-3460	Map the video RAM into main RAM then return.
3500-3510	Restore the state of the bank 4 bank switcher and its copy at &B003.
3570-1760	The file name STYLE.SCN.
3610	The flag to indicate successful file loading.
3630	Temporary storage of the state of the bank switcher.

Functions and procedures

PROCsetup	Dimensions all arrays, reads in the data and draws the bust (unless a screen file is found, in which case this is loaded instantly).
PROCnew	Resets all the document pointers ready for a new file to be analysed.
PROCstatus	Displays the readability scores in the status box.
PROCbust	Decodes and draws Shakespeare's bust.
PROCplot	Plots a single point from the bust image.
PROCmenu	Displays the main menu and calls one of the three main procedures, according to the key pressed.
PROCanalyse	Analyses the document just selected for style faults.
PROCfog	Generates a Fog Index and Flesch-Kincaid Index.
PROCscore	Converts the number of style faults of each of the four types into

	numbers representing the percentage of sentences in which they occur.
PROCnewsen	Clears certain variables in preparation for reading a new sentence.
PROCcheck_word()	Checks the passed word for style faults.
PROCpv()	Checks the passed word for a passive verb event.
PROChv()	Checks the passed word for hidden verbs.
PROCan()	Checks the passed word for abstract nouns.
PROCcomplex	Checks the sentence just read for complexity.
PROCcomment	Gives subjective commentary on the overall style.
PROCadvice	Displays four tips for good writing.
FNsure()	Asks for confirmation of a named operation, returning TRUE if [Y] is pressed.
FNread_word()	Returns the next word from the file.
FNisalpha()	Returns TRUE if the passed character is a letter.
FNisend()	Returns TRUE if the passed character is an end-of-sentence marker.
FNsyllables()	Returns the number of syllables in the passed word.
FNv()	Returns TRUE if the passed character is a vowel.
FNindex()	Returns the passed percentage as an index suitable for pointing into one of the four comment arrays.

Main variables and arrays

style$()	Contains comments on style.
structure$()	Contains comments on structure.
clarity$()	Contains comments on clarity.
impact$()	Contains comments on impact.
pv$()	Contains the words which usually begin a passive verb pair.
hv$()	Contains the letter which usually end a hidden verb.
doc%	Contains TRUE if a document has just been analysed.
doc$	Contains the name of the current document, or *No Document.*
words%	The number of words in the document.
sen%	The number of sentences in the document.
age%	The recommended reading age for the document.
sy%	The total number of syllables in the document.
hard%	The number of hard words in the document.
hv%	The number of hidden verbs in the document.

an%	The number of abstract nouns in the document.
cs%	The number of complex sentences in the document.
pv%	The number of passive verbs in the document.
avlen	The average sentence length.
fog	The Fog Index of the current document.
fk	The Flesch-Kincaid index.
senlimit%	The current sentence length threshold.

The program

```
   10 REM Style Master
   20 :
   30 ON ERROR GOTO 60
   40 CLS:PROCsetup
   50 quit%=FALSE:REPEAT:PROCmenu:UNTIL quit%
   60 GOTO 2800
   70 :
   80 DEF PROCsetup
   90 CLEAR:DIM A% 40,Z% &80:PROCassemble:PROCassemble2
  100 CALL scrn_from_disk:IF ?flag=0 THEN CLS ELSE GOTO 150
  110 MOVE 0,0:DRAW 479,0:DRAW 479,63:DRAW 0,63:DRAW 0,0
  120 MOVE 96,0:DRAW 96,63:MOVE 384,0:DRAW 384,63
  130 PROCwin1:CLS:PRINT TAB(1,0);CHR$(17);"Style Master";CHR$(18)
  140 PROCbust
  150 PROCnew:ex%=5:gd%=15:av%=30:bd%=50:senlimit%=40:@%=&90A
  160 DIM style$(4),structure$(4),clarity$(4),impact$(4):RESTORE 2190
  170 FOR s%=0 TO 4:READ style$(s%):NEXT:FOR s%=0 TO 4:READ structure$
(s%):NEXT
  180 FOR s%=0 TO 4:READ clarity$(s%):NEXT:FOR s%=0 TO 4:READ impact$
(s%):NEXT
  190 DIM pv$(6):RESTORE 2130:FOR w%=0 TO 6:READ pv$(w%):NEXT
  200 DIM hv$(3):RESTORE 2160:FOR w%=0 TO 3:READ hv$(w%):NEXT
  210 CALL scrn_to_disk:ENDPROC
  220 :
  230 DEF PROCnew
  240 doc%=FALSE:doc$="No Document":words%=0:sen%=0:fog%=0:fk%=0:age%=0
  250 sy%=0:hard%=0:hv%=0:an%=0:cs%=0:pv%=0
  260 PROCstatus:ENDPROC
  270 :
  280 DEF PROCwin1:VDU 28,1,6,14,1:ENDPROC
  290 :
  300 DEF PROCwin2:VDU 28,17,6,62,1:ENDPROC
  310 :
  320 DEF PROCwin3:VDU 28,65,6,78,1:ENDPROC
  330 :
  340 DEF PROCstatus
  350 PROCwin3:CLS:PRINT TAB(7-LEN(doc$)/2,0);CHR$(17);doc$;CHR$(18)
  360 IF doc%=FALSE THEN ENDPROC
  370 PRINT TAB(0,2);"Rating: ";
  380 IF age%<=5 PRINT"Great!"
  390 IF age%>5 AND age%<=10 PRINT"Good"
  400 IF age%>10 AND age%<= 14 PRINT"Average"
  410 IF age%>14 AND age%<=16 PRINT"Poor"
  420 IF age%>16 AND age%<=18 PRINT"Bad"
  430 IF age%>18 AND age%<=20 PRINT"Awful"
```

```
  440 IF age%>20 PRINT"Abysmal"
  450 @%=&20105
  460 PRINT TAB(0,3);"SenLen: ";avlen;
  470 PRINT TAB(3,4);"Fog: ";fog;
  480 PRINT TAB(3,5);"F/K: ";fk;
  490 @%=&90A:ENDPROC
  500 :
  510 DEF PROCbust
  520 xo%=30:yo%=43
  530 RESTORE 1990
  540 FOR y%=1 TO 36:x%=0
  550 FOR p%=1 TO 3:READ c%:V%=0
  560 FOR b%=15 TO 0 STEP-1: IF (c% AND 2^b%)>0 PROCplot(x%,y%)
  570 x%=x%+1:NEXT
  580 NEXT
  590 NEXT
  600 ENDPROC
  610 :
  620 DEF PROCplot(x%,y%)
  630 PLOT 69,x%+xo%,yo%-y%
  640 ENDPROC
  650 :
  660 DEF PROCmenu:PROCwin2:CLS
  670 PRINT TAB(16,0);CHR$(17);"Menu of Options";CHR$(18)
  680 VDU 31,10,2,17,40,65,41,18:PRINT"nalyse a New Document"
  690 VDU 31,10,3,17,40,67,41,18:PRINT"omment on Current Document"
  700 VDU 31,10,4,17,40,84,41,18:PRINT"ips for Good Writing"
  710 VDU 31,10,5,17,40,81,41,18:PRINT"uit Style Master";
  720 REPEAT:g%=GET AND 223:UNTIL g%=65 OR g%=67 OR g%=84 OR g%=81
  730 IF g%=65 PROCanalyse:ENDPROC
  740 IF g%=67 PROCcomment:ENDPROC
  750 IF g%=84 PROCadvice:ENDPROC
  760 IF FNsure("Quit Style Master") quit%=TRUE
  770 ENDPROC
  780 :
  790 DEF PROCanalyse
  800 PROCwin2:IF doc% THEN IF NOT FNsure("Analyse New Document") THEN
ENDPROC
  810 filename$=FNselect:CALL scrn_from_disk:IF ?flag=0 THEN CLS
  820 IF filename$="" THEN ENDPROC
  830 PROCnew:doc$=filename$:PROCwin2:CLS
  840 l$="Analysing "+doc$:PRINT TAB(23-LEN(l$)/2,1);CHR$(17);l$;CHR$(18)
  850 PROCnewsen:in%=OPENIN doc$:eos%=FALSE:PRINT TAB(15,3)"Words read:"
  860 REPEAT:word$=FNread_word(in%):PRINT TAB(28,3);words%
  870 PROCcheck_word(word$):senlen%=senlen%+1
  880 IF eos% sen%=sen%+1:PROCcomplex:PROCnewsen
  890 UNTIL EOF#in%:CLOSE #in%:PROCfog:PROCscore
  900 doc%=TRUE:PROCstatus:ENDPROC
  910 :
  920 DEF FNsure(m$)
  930 CLS:PRINT TAB(23-LEN(m$)/2,1);CHR$(17);m$;CHR$(18)
  940 PRINT TAB(13,3)"Are you sure (Y/N)?";
  950 REPEAT:g%=GET AND 223:UNTIL g%=78 OR g%=89:IF g%=89 THEN =TRUE
  960 =FALSE
  970 :
  980 DEF PROCopen(f$)
  990 =OPENIN f$
 1000 :
 1010 DEF PROCfog
 1020 avlen=words%/sen%:perhard=(hard%/words%)*100
```

```
1030 fog=(avlen+perhard)*0.4:spw=sy%/words%
1040 fk=0.39*avlen+11.8*spw-15.59:age%=fog+3.5
1050 ENDPROC
1060 :
1070 DEF PROCscore
1080 pv=pv%/sen%*100
1090 an=an%/sen%*100
1100 hv=hv%/sen%*100
1110 cs=cs%/sen%*100
1120 ENDPROC
1130 :
1140 DEF FNread_word(ch%)
1150 w$="":REPEAT:c%=BGET#ch%:UNTIL EOF#ch% OR FNisalpha(c%)
1160 IF EOF#ch% THEN = "" ELSE w$=w$+CHR$(c%)
1170 REPEAT:c%=BGET#ch%:IF FNisalpha(c%) w$=w$+CHR$(c%)
1180 UNTIL EOF#ch% OR NOT FNisalpha(c%)
1190 IF c%=44 commas%=commas%+1 ELSEIF FNisend(c%) eos%=TRUE
1200 IF w$<>"" words%=words%+1
1210 l$="":FOR l%=1 TO LEN(w$):l$=l$+CHR$(ASC(MID$(w$,l%,1))OR 32):NEXT
1220 =l$
1230 :
1240 DEF FNisalpha(c%)
1250 c%=c% AND 223:IF c%>=65 AND c%<=90 THEN =TRUE
1260 =FALSE
1270 :
1280 DEF FNisend(c%)
1290 IF c%=33 OR c%=46 OR c%=63 THEN = TRUE
1300 =FALSE
1310 :
1320 DEF PROCnewsen
1330 eos%=FALSE:senlen%=0:hardper%=0:commas%=0:pvf%=FALSE
1340 ENDPROC
1350 :
1360 DEF PROCcheck_word(w$)
1370 IF w$="" ENDPROC
1380 y%=FNsyllables(w$):sy%=sy%+y%:IF y%>=3 hard%=hard%+1
1390 PROCpv(w$):PROChv(w$):PROCan(w$)
1400 ENDPROC
1410 :
1420 DEF FNsyllables(w$)
1430 IF LEN(w$)<=6 THEN =1
1440 s%=0:f%=FNv(LEFT$(w$,1))
1450 FOR i%=2 TO LEN(w$):v%=FNv(MID$(w$,i%,1))
1460 IF v%<>f% f%=v%:s%=s%+1
1470 NEXT:=s%/2
1480 :
1490 DEF FNv(c$):v$=CHR$(ASC(c$) AND 223)
1500 IF v$="A" OR v$="E" OR v$="I" OR v$="O" OR v$="U" OR v$="Y" THEN
=TRUE
1510 =FALSE
1520 :
1530 DEF PROCpv(w$)
1540 IF pvf%=TRUE THENIF RIGHT$(w$,2)="ed" pvf%=FALSE:pv%=pv%+1:ENDPROC
1550 IF pvf%=TRUE pvf%=FALSE:ENDPROC
1560 f%=FALSE:FOR w%=0 TO 6:IF w$=pv$(w%) f%=TRUE
1570 NEXT:IF f% pvf%=TRUE
1580 ENDPROC
1590 :
1600 DEF PROChv(w$)
1610 f%=FALSE:FOR w%=0 TO 3:IF RIGHT$(w$,4)=hv$(w%) f%=TRUE
```

```
 1620 NEXT:IF f% hv%=hv%+1
 1630 ENDPROC
 1640 :
 1650 DEF PROCan(w$)
 1660 IF RIGHT$(w$,5)="ation" an%=an%+1
 1670 ENDPROC
 1680 :
 1690 DEF PROCcomplex
 1700 IF (senlen%<(hardper%*3))OR(senlen%>senlimit%)OR(commas%=0 AND
senlen%>25)OR(commas%>0 AND senlen%>((commas%+1)*20))OR(commas%>2 AND
(senlen%<(commas%*4)))) cs%=cs%+1
 1710 ENDPROC
 1720 :
 1730 DEF PROCcomment
 1740 IF NOT doc% THEN ENDPROC
 1750 PROCwin2:CLS:PRINT TAB(16,0);CHR$(17);"Style Criticism";CHR$(18)
 1760 PRINT TAB(0,2);CHR$(17);"Style:     ";CHR$(18);style$(FNindex(pv))
 1770 PRINT TAB(0,3);CHR$(17);"Structure: ";CHR$(18);structure$(FNindex
(cs))
 1780 PRINT TAB(0,4);CHR$(17);"Meaning:   ";CHR$(18);clarity$(FNindex
(an))
 1790 PRINT TAB(0,5);CHR$(17);"Impact:    ";CHR$(18);impact$(FNindex
(hv));
 1800 G=GET:ENDPROC
 1810 :
 1820 DEF FNindex(i)
 1830 IF i<=ex% THEN =0
 1840 IF i>ex% AND i<=gd% THEN =1
 1850 IF i>gd% AND i<=av% THEN =2
 1860 IF i>av% AND i<=bd% THEN =3
 1870 =4
 1880 :
 1890 DEF PROCadvice
 1900 PROCwin2:CLS:PRINT TAB(8,0);CHR$(17);
 1910 PRINT"Golden Rules for Good Writing";CHR$(18)'
 1920 PRINT"* Use simple words, even in 'serious' writing"
 1930 PRINT"* Avoid abstract words and clever jargon"
 1940 PRINT"* Keep your style alive - avoid passive verbs"
 1950 PRINT"* Watch out for, and uncover, all hidden verbs";
 1960 G=GET:ENDPROC
 1970 :
 1980 REM Shakespeare Bit map
 1990 DATA &1,&F000,&0,&7,&FF00,&0,&1B,&1C0,&0
 2000 DATA &2A,&60,&0,&54,&30,&0,&AC,&18,&0
 2010 DATA &D8,&1E,&0,&1AB,&D,&0,&158,&A,&C000
 2020 DATA &2AB,&5,&6000,&358,&6,&E000,&EA8,&5,&6000
 2030 DATA &3558,&F0F6,&B000,&6AA9,&105,&5000,&5558,&7176,&B000
 2040 DATA &AAA8,&6165,&5000,&D558,&106,&B000,&AAA8,&85,&6000
 2050 DATA &D55C,&8A,&A000,&AAAC,&68D,&6000,&D55C,&10A,&A000
 2060 DATA &AABE,&78D,&4000,&D56E,&CCF,&8000,&6AAF,&1868,&0
 2070 DATA &5517,&C794,&0,&2A17,&F034,&0,&1E0B,&F872,&0
 2080 DATA &205,&FCE1,&0,&202,&FFC0,&8000,&201,&BFC0,&8000
 2090 DATA &100,&5F80,&4000,&100,&2780,&4000,&80,&1880,&2000
 2100 DATA &40,&3DFE,&2000,&27,&C301,&E000,&18,&0,&0
 2110 :
 2120 REM Passive verb partners
 2130 DATA are,be,been,being,is,was,were
 2140 :
 2150 REM Hidden verb endings
 2160 DATA sion,tion,ment,ance
```

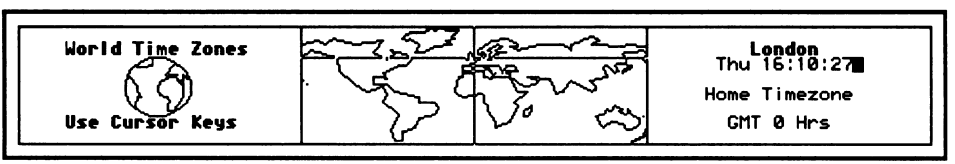
```
2170 :
2180 REM Style
2190 DATA "Direct and friendly"
2200 DATA "Slightly impersonal"
2210 DATA "Rather stuffy"
2220 DATA "Too impersonal"
2230 DATA "Pompous and bureaucratic"
2240 :
2250 REM Structure
2260 DATA "Well balanced"
2270 DATA "Complex but balanced"
2280 DATA "Complex and unbalanced"
2290 DATA "Overly complex"
2300 DATA "Unacceptably complex"
2310 :
2320 REM Clarity
2330 DATA "Straightforward"
2340 DATA "Fairly clear"
2350 DATA "Hard to follow"
2360 DATA "Quite obscure"
2370 DATA "Incomprehensible"
2380 :
2390 REM Impact
2400 DATA "Very punchy"
2410 DATA "Fairly high"
2420 DATA "Quite low key"
2430 DATA "Barely measurable"
2440 DATA "Nonexistant"
2450 :
2460 DEF FNselect
2470 CALL A%
2480 IF buffer?0 = 0 THEN CLS:=""
2490 R$=""
2500 FOR J%=0 TO 11
2510 IF buffer?J% THEN R$=R$+CHR$(buffer?J%) ELSE J%=12
2520 NEXT
2530 =R$
2540 :
2550 DEF PROCassemble
2560 FOR PASS=0 TO 2 STEP 2
2570 P%=A%
2580 [
2590 OPT PASS
2600 CALL &B8C3
2610 LD DE,buffer
2620 JR C,found
2630 LD A,0
2640 LD (DE),A
2650 RET
2660 .found
2670 LD B,12
2680 .loop
2690 LD A,(HL)
2700 LD (DE),A
2710 INC HL
2720 INC DE
2730 DJNZ loop
2740 RET
2750 .buffer
2760 ]
```

```
2770 NEXT
2780 ENDPROC
2790 :
2800 ON ERROR GOTO 2820
2810 VDU 26:CLS:IF ERR=17 THEN CHAIN "AUTO"
2820 REPORT:PRINT" at line ";ERL
2830 PRINT:PRINT"Press [Function][X] for Notepad Main Menu"
2840 END
2850 :
2860 DEF PROCassemble2
2870 fopenout=&B8A5
2880 fopenin=&B8A2
2890 foutblock=&B8AB
2900 finblock=&B896
2910 fclose=&B890
2920 :
2930 FOR PASS = 0 TO 2 STEP 2
2940 P%=Z%
2950 [
2960 OPT PASS
2970 :
2980 .scrn_to_disk
2990 :
3000 CALL map_scrn_in
3010 LD HL,&F000
3020 LD DE,&8000
3030 LD BC,&1000
3040 LDIR
3050 CALL map_scrn_out
3060 LD HL,filename
3070 CALL fopenout
3080 RET NC
3090 LD HL,&8000
3100 LD BC,&1000
3110 CALL foutblock
3120 JP fclose
3130 :
3140 .scrn_from_disk
3150 :
3160 LD HL,filename
3170 CALL fopenin
3180 JR C,froml
3190 LD HL,flag
3200 LD (HL),0
3210 RET
3220 :
3230 .froml
3240 :
3250 LD HL,&8000
3260 LD BC,&1000
3270 CALL finblock
3280 CALL fclose
3290 CALL map_scrn_in
3300 LD HL,&8000
3310 LD DE,&F000
3320 LD BC,&1000
3330 LDIR
3340 CALL map_scrn_out
3350 LD HL,flag
3360 LD (HL),1
```

```
3370 RET
3380 :
3390 .map_scrn_in
3400 :
3410 LD A,(&B003)
3420 LD (state),A
3430 LD A,67
3440 LD (&B003),A
3450 OUT (&13),A
3460 RET
3470 :
3480 .map_scrn_out
3490 :
3500 LD A,(state)
3510 LD (&B003),A
3520 OUT (&13),A
3530 RET
3540 :
3550 .filename
3560 :
3570 DEFM "STYLE.SCN":DEFB 0
3580 :
3590 .flag
3600 :
3610 DEFB 0
3620 :
3630 .state
3640 :
3650 DEFB 0
3660 ]
3670 NEXT
3680 ENDPROC
```

TIMEZONE.BAS

World Clock



TIMEZONE.BAS, goodness, is that the time?

World clocks of one kind or another are included with most electronic organisers and even portables these days. They usually look very pretty, with a nice little picture of a globe spinning around, but their main purpose in life seems to be letting you check at a pinch whether your contact in Honolulu is likely to be dragged out of warm bed to give you a frosty reception if you make that ever-so-vital call right now.

Although the Notepad has a Time Manager it only supports six time zones and is text-only. TIMEZONE.BAS aims to improve on this by providing a graphically attractive world clock, of the sort that wouldn't disgrace even the yuppiest pocket computer. Complete with 24 time zones placed at strategic spots of worldwide power and influence (like Noumea), you need never again fret over the current time of day in Anchorage, Alaska.

As well as showing you the current time in all 24 time zones worldwide, the program displays the time relative to your current home time, and also relative to GMT. This implies that you can change the home time zone, and indeed you can, although you'll have to alter the program to fix it permanently. It's easy to do, and is fully covered in the line-by-line explanation of the program.

USING THE PROGRAM

Type in the listing and save it as TIMEZONE.BAS before trying it out. Type:

```
RUN
```

and the program will take a few seconds to draw a world map, and there's even a little globe that sits under the program title.

On the right of the screen is the status window, showing the current time zone. This is always *London* when you first run the program, though it's easily changed. Below this is the time in that zone, and initially this will be the same time as held in the Notepad's system clock.

Below this is a message telling you the relative difference in hours between the home time and the current time zone, and when the two are the same (as they will be to start with) it simply says *Home Timezone.*

Finally, at the foot of the window is another message telling you the relative difference in hours between the current time zone and GMT (although it doesn't take into account British Summer Time or any other daylight saving time).

On the map a cross hair is centred over the current time zone, which initially is London. Pressing [Left] and [Right] moves the cross hair west and east respectively, through each time zone in turn, updating the information in the status window.

If you want to make another time zone the *home* time zone, press [Return] at any time – but see the line-by-line explanation for how to change it permanently from being London each time you start the program.

HOW IT WORKS

30 Points the Basic error handler to Timezone's own error handling routine at line 2150.

40	Sets up the variables and the display.
50	Loads previously saved screen file from disk, if it exists.
80-120	Draw the world map by continent.
180	Sets both the start time zone and the home time zone, and the timer to zero. Should you want to make either start zone different, set them to any number between 1 and 24 (the zones themselves are listed on lines 1450 – 1680). You will usually want to keep these both the same as each other, but you don't have to.
190	Dimensions the arrays.
200-220	Draw a box around the screen and display static strings.
230	Sets xm and ym to display the globe small, and then fixes the graphics origin at the bottom-left of the world map area.
240-260	Read in all the data.
270	Draws two vertical lines delineating the world map window.
310	Displays data for the home time zone and calls PROCscankeys in an infinite loop.
350-370	Read the keyboard, checking for [Left], [Right] and [Return]. Call PROCprevzone, PROCnextzone or PROChomezone respectively.
380	If at least one second has elapsed since the last time this line was visited, resets the timer and forces an immediate display of the clock for the current zone.
420	Removes the cross-hair and selects the previous zone in the list (wrapping to the last zone if already at the start of the list).
430	Places the cross-hair at the new location, and displays the new zone's time.
470	Removes the cross-hair and selects the next zone in the list (wrapping to the first zone if already at the end of the list).
480	Places the cross-hair at the new location, and displays the new zone's time.
520	Removes the cross-hair, sets the home time zone equal to the current time zone, replaces the cross-hair and updates the status window.
560	Places the cross-hair on the world map.
570-580	Extract the GMT offset for the current zone and convert it to a string, adding a leading + if positive, and the letters *Hrs* on the end.
590-610	Calculate the time zone's offset from the home time zone by subtracting the home time zone's GMT offset from the current

	time zone's GMT offset, and then add a leading + if the result is positive. If already on the home time zone, just display *Home Timezone*.
620-640	Print the time zone's name, its offset from the home time zone and its offset from GMT.
650	Calculates and prints the current time in that time zone.
690-720	Read the day, hour, minutes and seconds from the system clock.
730-740	Calculate the day number by counting day% up from one, until the current day matches an entry in week$(). Whatever day% equals at that point is the day number.
750	Calculates the hour in the current time zone by first adding the current zone's GMT offset, and then by subtracting the home time zone's GMT offset.
760	If the hour is less than zero, wraps around to the other end of the day and decrements the day of the week, wrapping to the last day of the week if necessary.
770	If the hour is greater than 24, wraps around to the other end of the day and increments the day of the week, wrapping to the first day of the week if necessary.
780	Converts the new day number to a name.
790	Constructs the string that holds the time for the current zone from the day of the week and the time for that zone.
800	Prints the completed string.
840-890	Construct a time string in hh:mm:ss format from the current values of hour%, mins%, secs% and returns it.
920-950	Exclusive-OR a cross-hair at the current city coordinates using PLOT 96. One call displays the cross-hair, the next (if at the same coordinates) removes it.
990-1030	Display a discrete land mass by first reading in the number of lines to plot (max%) and plotting the start coordinates, followed by drawing a line to each new coordinate pair read until the count in p% reaches max%. The multipliers xm and ym are used, allowing the maps to be scaled as required.
1050-1310	The main continent procedures. Some of these contain more than one call to PROCisland, where there are several land masses to be drawn. Occasionally single points are plotted to represent important islands (Honolulu is plotted as part of PROCamerica).
1350-1370	Draw the three land masses visible on the globe.
1380-1410	Draw the outline of the globe as a polygon with 32 corners.
1450-1680	The time zone data held listed as follows: name, offset in hours

	from GMT, X coordinate of the named location, Y coordinate of the named location.
1700	Lists the days of the week.
1720-2170	List the land mass data for both the world map and the globe.
2190	Points the Basic error handler to a full error report in the event of a further error occurring while attempting to run AUTO. This is in case AUTO isn't present on your Notepad.
2200	Attempts to run the menu program AUTO if the error was generated by pressing the [Stop] key.
2210	If the error was caused by something else, or if AUTO isn't on your Notepad, a full error report is displayed.
2220	After the error report the Notepad will be left in BBC Basic, so this message is displayed to remind users of how to return to the Notepad main menu.
2250-3170	Assemble the screen saver/loader, which saves a copy of the screen after everything is drawn the first time, and loads it in each time thereafter.
2250	Start of procedure that assembles the screen saver/loader, which saves a copy of the screen after everything is drawn the first time, and loads it in each time thereafter.
2260-2300	Define the five NC100 jump block routines to be used.
2320-2330	Begin the two-pass assembly and set P% (the assembly destination pointer) to the start of the previously dimensioned Z%.
2390	Pages the 16K of RAM with the video memory in at address &C000.
2400-2430	Copy the contents of video RAM down to &8000.
2440	Puts back the video RAM.
2450-2460	Open a file for saving the screen data.
2470	Returns if unable to open the file.
2480-2500	Save &1000 bytes from &8000 to the file.
2510	Closes the file and exits.
2550-2560	Open a file for reading.
2570-2600	If unable to open the file, set the contents of flag to zero and returns.
2640-2670	Read the &1000 bytes to location &8000 then close the file.
2680-2730	Map the video RAM 16K block into &C000, copy the &1000 bytes from location &8000 up to &F000 and then put back the screen RAM.

2740-2760	To indicate successful loading, set the contents of flag to 1 then return.
2800-2810	Save the current status of the bank switcher for block 4 (&C000-&FFFF).
2820-2850	Map the video RAM into main RAM then returns.
2890-2900	Restore the state of the bank 4 bank switcher and its copy at &B003.
2960	The file name TIMEZONE.SCN.
3000	The flag to indicate successful file loading.
3040	Temporary storage of the state of the bank switcher.
3090-3120	A function to allocate memory for a string and store the string in that memory.
3140-3170	A function to allocate space for a byte of data and store the data in that location.

Functions and procedures

PROCsetup	Dimensions the arrays, draws the screen boxes, the globe and prints static text.
PROCselect	Displays data for the home time zone and calls PROCscankeys in an infinite loop.
PROCscankeys	Reads the keyboard, checking for [Left], [Right] and [Return].
PROCprevzone	Selects the next time zone west, wrapping round to the east of the map in necessary.
PROCnextzone	Selects the next time zone east, wrapping round to the west of the map if necessary.
PROChomezone	Makes the current time zone the home timezone.
PROCshowzone	Displays the cross-hair and updates the status window for the current zone.
PROCzonetime	Polls the system clock and displays it at the top right adjusted for the current zone.
PROCcrosshair	Displays or removes a cross-hair at the current zone coordinates.
PROCisland	Draws an isolated land mass on the main world map.
PROCamerica	Draws the land mass that makes up North and South America.
PROCafrica	Draws the land masses that make up Africa.
PROCeurope	Draws the land masses that make up Europe.
PROCgreenland	Draws the land mass that makes up Greenland.
PROCaustralia	Draws the land masses that make up the Antipodes.
PROCglobe	Draws a circle and the land masses that make up the visible face of the globe.

FNnewtime	Returns the adjusted time formatted to hh:mm:ss.

Main variables and arrays

city$()	Holds the name of each time zone's main city or island.
gmt%()	Holds the relative offset of each time zone from GMT.
xpos%()	Holds the cross-hair X coordinate of each time zone.
ypos%()	Holds the cross-hair Y coordinate of each time zone.
week$()	Holds the names of each day of the week.
xm	X coordinate multiplier for controlling width of graphics objects.
ym	Y coordinate multiplier for controlling height of graphics objects.
zone%	Current time zone.
home%	Home time zone.
tzo$	Time zone offset from home time-zone.
gmt$	Time zone offset from Greenwich Mean Time.
day$	The current day of the week in the home time zone.
hour%	The current hour of the day in the home time zone.
mins%	The current minute.
secs%	The current second.
h$	The hour in the current time zone padded to two digits with leading zero.
m$	The current minute padded to two digits with leading zero.
s$	The current seconds padded to two digits with leading zero.
time$	The time in the current zone formatted to hh:mm:ss.

The program

```
 10 REM World Time Zones
 20 :
 30 ON ERROR GOTO 2190
 40 DIM Z% &80:VDU 26:CLS
 50 PROCassemble:CALL scrn_from_disk:IF ?flag=0 THEN CLS
 60 PROCsetup
 70 IF ?flag=1 THEN GOTO 140
 80 PROCamerica
 90 PROCgreenland
100 PROCafrica
110 PROCeurope
120 PROCaustralia
130 CALL scrn_to_disk
140 PROCselect
150 END
160 :
170 DEF PROCsetup
180 zone%=12:home%=12:TIME=0
```

```
  190 DIM city$(24),gmt%(24),xpos%(24),ypos%(24),week$(7)
  200 MOVE 0,0:DRAW 479,0:DRAW 479,63:DRAW 0,63:DRAW 0,0
  210 PRINT TAB(4,1);CHR$(17);"World Time Zones";CHR$(18)
  220 PRINT TAB(4,6);CHR$(17);"Use Cursor Keys";CHR$(18):VDU 28,56,6,76,1
  230 xm=0.8:ym=0.8:VDU 29,58;14;:PROCglobe:xm=1.9:ym=1.4:VDU 29,148;0;
  240 RESTORE 1450:FOR z%=1 TO 24
  250 READ city$(z%),gmt%(z%),xpos%(z%),ypos%(z%)
  260 NEXT:FOR d%=1 TO 7:READ week$(d%):NEXT
  270 MOVE 0,0:DRAW 182,0:DRAW 182,64:DRAW 0,64:DRAW 0,0
  280 ENDPROC
  290 :
  300 DEF PROCselect
  310 PROCshowzone:REPEAT:PROCscankeys:UNTIL FALSE
  320 ENDPROC
  330 :
  340 DEF PROCscankeys
  350 i%=INKEY(0):IF i%=242 PROCprevzone:ENDPROC
  360 IF i%=243 PROCnextzone:ENDPROC
  370 IF i%=13 SOUND 1,1,100,1:PROChomezone:ENDPROC
  380 IF TIME>100 THEN TIME=0:PROCzonetime
  390 ENDPROC
  400 :
  410 DEF PROCprevzone
  420 PROCcrosshair:zone%=zone%-1:IF zone%=0 zone%=24
  430 PROCshowzone
  440 ENDPROC
  450 :
  460 DEF PROCnextzone
  470 PROCcrosshair:zone%=zone%+1:IF zone%=25 zone%=1
  480 PROCshowzone
  490 ENDPROC
  500 :
  510 DEF PROChomezone
  520 PROCcrosshair:home%=zone%:PROCshowzone
  530 ENDPROC
  540 :
  550 DEF PROCshowzone
  560 CLS:PROCcrosshair
  570 IF gmt%(zone%)>0 gmt$="+"+STR$(gmt%(zone%)) ELSE
gmt$=STR$(gmt%(zone%))
  580 gmt$="GMT "+gmt$+" Hrs"
  590 tzo%=gmt%(zone%)-gmt%(home%)
  600 IF tzo%>0 tzo$="+"+STR$(tzo%) ELSE tzo$=STR$(tzo%)
  610 IF tzo%<>0 tzo$="Home Time "+tzo$+" Hrs" ELSE tzo$="Home Timezone"
  620 PRINT TAB(11-LEN(city$(zone%))/2,0);CHR$(17);city$(zone%);CHR$(18)
  630 PRINT TAB(11-LEN(tzo$)/2,3);tzo$
  640 PRINT TAB(11-LEN(gmt$)/2,5);gmt$;
  650 PROCzonetime
  660 ENDPROC
  670 :
  680 DEF PROCzonetime
  690 day$=LEFT$(TIME$,3)
  700 hour%=VAL(MID$(TIME$,17,2))
  710 mins%=VAL(MID$(TIME$,20,2))
  720 secs%=VAL(MID$(TIME$,23,2))
  730 day%=0:REPEAT:day%=day%+1
  740 UNTIL week$(day%)=day$
  750 hour%=hour%+gmt%(zone%)-gmt%(home%)
  760 IF hour%<0 hour%=24+hour%:day%=day%-1:IF day%=0 day%=7
  770 IF hour%>23 hour%=hour%-24:day%=day%+1:IF day%=8 day%=1
```

```
 780 day$=week$(day%)
 790 time$=day$+CHR$(32)+FNnewtime
 800 PRINT TAB(11-LEN(time$)/2,1);time$;
 810 ENDPROC
 820 :
 830 DEF FNnewtime
 840 h$=STR$(hour%):m$=STR$(mins%):s$=STR$(secs%)
 850 h$=STRING$(2-LEN(h$),"0")+h$
 860 m$=STRING$(2-LEN(m$),"0")+m$
 870 s$=STRING$(2-LEN(s$),"0")+s$
 880 time$=h$+":"+m$+":"+s$
 890 =time$
 900 :
 910 DEF PROCcrosshair
 920 MOVE xpos%(zone%)*xm,0
 930 PLOT 6,xpos%(zone%)*xm,48*ym
 940 MOVE 0,ypos%(zone%)*ym
 950 PLOT 6,96*xm,ypos%(zone%)*ym
 960 ENDPROC
 970 :
 980 DEF PROCisland
 990 READ max%
1000 READ x%,y%:MOVE x%*xm,y%*ym
1010 FOR p%=1 TO max%
1020 READ x%,y%:DRAW x%*xm,y%*ym
1030 NEXT:ENDPROC
1040 :
1050 DEF PROCamerica
1060 RESTORE 1720:PROCisland
1070 PLOT 69,1*xm,25*ym
1080 PLOT 69,2*xm,24*ym
1090 ENDPROC
1100 :
1110 DEF PROCafrica
1120 RESTORE 1780:PROCisland
1130 RESTORE 1830:PROCisland
1140 PLOT 69,40*xm,29*ym
1150 ENDPROC
1160 :
1170 DEF PROCeurope
1180 RESTORE 1860:PROCisland
1190 RESTORE 2000:PROCisland
1200 RESTORE 2100:PROCisland
1210 ENDPROC
1220 :
1230 DEF PROCgreenland
1240 RESTORE 1960:PROCisland
1250 ENDPROC
1260 :
1270 DEF PROCaustralia
1280 RESTORE 2030:PROCisland
1290 RESTORE 2070:PROCisland
1300 PLOT 69,95*xm,11*ym
1310 ENDPROC
1320 :
1330 DEF PROCglobe
1340 IF ?flag=1 THEN ENDPROC
1350 RESTORE 2130:PROCisland
1360 RESTORE 2140:PROCisland
1370 RESTORE 2150:PROCisland
```

```
 1380 MOVE ((SIN(0)*22)+19)*xm,((COS(0)*20)+22)*ym
 1390 FOR a=0 TO 2*PI STEP 2*PI/32
 1400 DRAW ((SIN(a)*22)+19)*xm,((COS(a)*20)+22)*ym
 1410 NEXT
 1420 ENDPROC
 1430 :
 1440 REM Time Zone Data
 1450 DATA Midway,-11,1,25
 1460 DATA Honolulu,-10,2,24
 1470 DATA Anchorage,-9,3,37
 1480 DATA Los Angeles,-8,13,27
 1490 DATA Denver,-7,16,31
 1500 DATA Chicago,-6,20,32
 1510 DATA New York,-5,25,30
 1520 DATA Caracas,-4,28,20
 1530 DATA Rio de Janeiro,-3,35,12
 1540 DATA Recife,-2,37,16
 1550 DATA Azores,-1,40,29
 1560 DATA London,0,48,34
 1570 DATA Paris,1,49,32
 1580 DATA Cairo,2,58,27
 1590 DATA Jeddah,3,60,25
 1600 DATA Dubai,4,64,26
 1610 DATA Karachi,5,69,26
 1620 DATA Dhaka,6,75,26
 1630 DATA Bangkok,7,78,23
 1640 DATA Hong Kong,8,83,26
 1650 DATA Tokyo,9,91,30
 1660 DATA Sydney,10,93,7
 1670 DATA Noumea,11,95,11
 1680 DATA Wellington,12,95,4
 1690 REM Week Days
 1700 DATA Mon,Tue,Wed,Thu,Fri,Sat,Sun
 1710 REM North and South America
 1720 DATA 43
 1730 DATA 0,39,5,40,7,39,9,41,15,39,23,40,20,36,24,33,26,35,25,37,29,
35,31,33
 1740 DATA 29,31,27,31,23,28,24,26,22,27,20,27,20,24,24,24,26,21,33,19,
33,17
 1750 DATA 37,17,38,14,37,12,33,10,32,7,30,6,28,5,28,2,26,2,25,6,27,12,
23,17
 1760 DATA 24,20,23,23,20,22,18,21,15,24,11,29,10,34,3,37,0,34
 1770 REM Africa
 1780 DATA 23
 1790 DATA 51,29,45,28,45,27,44,27,43,21,45,19,50,20,51,19,51,17,52,15,
52,13
 1800 DATA 54,9,54,7,58,9,60,15,59,17,63,21,60,21,58,27,53,27,54,28,53,
26,51,27
 1810 DATA 51,29
 1820 REM Madagascar
 1830 DATA 5
 1840 DATA 62,11,63,11,64,14,64,15,62,13,62,11
 1850 REM Europe
 1860 DATA 85
 1870 DATA 45,31,45,29,47,29,48,31,50,31,53,28,53,29,51,31,55,31,56,28,
56,29
 1880 DATA 57,30,57,31,59,31,60,29,57,29,57,28,58,28,58,27,62,23,65,24,
66,25
 1890 DATA 63,27,69,26,71,20,73,21,73,22,75,25,81,21,82,22,82,23,80,25,
83,26
```

```
 1900 DATA 85,27,84,30,85,31,87,28,88,29,87,31,88,32,90,32,90,34,89,35,
90,36
 1910 DATA 92,36,95,37,96,35,95,33,96,32,96,38,91,39,89,38,81,39,83,40,
81,42
 1920 DATA 75,39,75,37,73,39,66,39,60,37,58,38,60,38,60,39,57,41,53,40,
51,38
 1930 DATA 49,37,49,35,51,36,52,34,53,36,52,37,55,39,56,38,54,37,55,36,
57,36
 1940 DATA 54,35,53,33,52,33,51,35,50,35,50,34,47,32,47,31,45,31
 1950 REM Greenland
 1960 DATA 18
 1970 DATA 32,36,33,39,30,41,29,40,28,41,30,43,34,43,38,45,37,44,41,45,
40,44
 1980 DATA 44,44,42,42,43,41,40,40,42,39,37,38,34,36,32,36
 1990 REM Britain
 2000 DATA 3
 2010 DATA 46,33,48,34,47,36,46,33
 2020 REM Australia
 2030 DATA 23
 2040 DATA 84,7,83,8,83,9,82,11,84,12,85,14,86,13,87,15,89,14,88,13,90,
12,91,13
 2050 DATA 91,14,92,12,94,10,94,8,92,6,90,6,89,8,88,8,87,9,86,9,85,8,84,7
 2060 REM New Zealand
 2070 DATA 4
 2080 DATA 96,7,95,4,92,2,92,1,96,4
 2090 REM Japan
 2100 DATA 6
 2110 DATA 91,31,91,30,89,29,89,28,92,30,92,31,91,31
 2120 REM Globe
 2130 DATA 5,4,8,6,15,8,16,5,21,5,23,0,29
 2140 DATA 3,6,36,11,36,15,37,16,40
 2150 DATA 28,29,37,24,31,25,29,28,31,30,31,32,30,30,29,28,29,24,28,20,
25,19
 2160 DATA 20,21,17,25,16,26,15,27,12,26,9,26,5,29,6,32,9,35,16,35,20,36,
24,35
 2170 DATA 23,33,25,32,28,35,25,36,28,35,30,38,28
 2180 :
 2190 ON ERROR GOTO 2210
 2200 VDU 26:CLS:IF ERR=17 THEN CHAIN "AUTO"
 2210 REPORT:PRINT" at line ";ERL
 2220 PRINT:PRINT"Press [Function][X] for Notepad Main Menu"
 2230 END
 2240 :
 2250 DEF PROCassemble
 2260 fopenout=&B8A5
 2270 fopenin=&B8A2
 2280 foutblock=&B8AB
 2290 finblock=&B896
 2300 fclose=&B890
 2310 :
 2320 FOR PASS = 0 TO 2 STEP 2
 2330 P%=Z%
 2340 [
 2350 OPT PASS
 2360 :
 2370 .scrn_to_disk
 2380 :
 2390 CALL map_scrn_in
 2400 LD HL,&F000
 2410 LD DE,&8000
```

```
2420 LD BC,&1000
2430 LDIR
2440 CALL map_scrn_out
2450 LD HL,filename
2460 CALL fopenout
2470 RET NC
2480 LD HL,&8000
2490 LD BC,&1000
2500 CALL foutblock
2510 JP fclose
2520 :
2530 .scrn_from_disk
2540 :
2550 LD HL,filename
2560 CALL fopenin
2570 JR C,from1
2580 LD HL,flag
2590 LD (HL),0
2600 RET
2610 :
2620 .from1
2630 :
2640 LD HL,&8000
2650 LD BC,&1000
2660 CALL finblock
2670 CALL fclose
2680 CALL map_scrn_in
2690 LD HL,&8000
2700 LD DE,&F000
2710 LD BC,&1000
2720 LDIR
2730 CALL map_scrn_out
2740 LD HL,flag
2750 LD (HL),1
2760 RET
2770 :
2780 .map_scrn_in
2790 :
2800 LD A,(&B003)
2810 LD (state),A
2820 LD A,67
2830 LD (&B003),A
2840 OUT (&13),A
2850 RET
2860 :
2870 .map_scrn_out
2880 :
2890 LD A,(state)
2900 LD (&B003),A
2910 OUT (&13),A
2920 RET
2930 :
2940 .filename
2950 :
2960 DEFM "TIMEZONE.SCN":DEFB 0
2970 :
2980 .flag
2990 :
3000 DEFB 0
3010 :
```

```
3020 .state
3030 :
3040 DEFB 0
3050 ]
3060 NEXT
3070 ENDPROC
```

ZAP.BAS

Z80 disassembler

```
0004  30 30        JR NC,&0034          00
0006  52           LD D,D               R
0007  1B           DEC DE               ←
0008  2A 62 36     LD HL,(&3662)        *b6
000B  30 57        JR NC,&0052          0W
000D  21 00 00     LD HL,&0000          !
Escape at line 520
>
```

ZAP.BAS, peek under your Notepad's bonnet.

An essential tool for every assembly language programmer is a disassembler. With one you can check that programs you have written have assembled correctly or get a feel for how other programs work. In particular, if you wish to examine code in the Notepad's ROMs in order to, perhaps, call undocumented routines directly yourself (only recommended if you make careful checks to ensure that a new version of the firmware has not changed or moved such a routine), you can adapt the screen saving functions of programs such as COOKIE.BAS or TIMEZONE.BAS to page a selected ROM in (rather than the video RAM) and save a copy to disk ready for disassembly.

Note that if you do disassemble parts of the ROMs you may only use the information you obtain to call the routines in the ROMs. You MAY NOT copy and/or modify any of the routines and incorporate them in your own programs as they are protected by copyright.

ZAP.BAS was written to be as easy to understand as possible and therefore a brute force method was used for its data storage. Rather than storing the main assembler mnemonics such as ADC or LD, and then doing some complicated calculations to determine the various combinations such as ADC A or ADC B and LD HL,A or LD (DE),&1234, every single possible combination is stored on its own data line.

This means that the program only has to check whether the current opcode sequence is one, two, three or four bytes, noting the opcodes against all the sequences stored next to the accompanying mnemonics.

USING THE PROGRAM

Type in the listing and save it as ZAP.BAS before trying it out, then type:

```
RUN
```

The screen will then clear and you will see *Please wait* while the program reads in all its data. This takes about eight seconds, after which the familiar File Selector appears, and you should now choose a file you wish to disassemble. If you wish you can choose non-machine code files too, just to test the program, but the disassembly will be completely nonsensical.

Once a file has been selected you are then asked for the *ORG address*. This should be the exact address to which the first byte of the code you are disassembling was originally assembled. For example, if you were to disassemble ROM 5 which is the BBC Basic Rom, you should give an ORG address of &C000, which is the area of memory to which BBC Basic is paged in when you call it up.

Incidentally, the reason Zap reads its input from a file is because the program is rather large and, being in BBC Basic, it would not be easy to allow it to disassemble from ANY RAM address in ANY RAM or ROM. Therefore, it is left up to you to save any areas of memory you wish to assemble out to disk first.

Of course, the program could have been written completely in assembler itself and it would then have been blindingly fast. However, the Basic version is more than sufficient for examining code segments.

HOW IT WORKS

30-70	Declare the main variables.
80-150	Dimension the arrays.
170	Assembles the machine code for the File selector.
180	Reads the mnemonic and opcode data.
190	Calls the File selector.
200	Asks for the ORG address
210	Opens the selected file for reading.
250	Gets the first opcode.
260	Is it a one-byte opcode? If yes, GOTO done.
270	Gets the second opcode
280-300	Is it one of the two-byte family of opcodes? If yes, GOTO done.
310	Gets the third opcode
320-330	Is it one of the three-byte family of opcodes? If yes, GOTO done.
340	Gets the fourth opcode

350-370	Is it one of the four-byte family of opcodes? If yes, GOTO done.
380	If we got here then we found an unrecognised opcode. So it is either data or an undocumented opcode.
390	Continues until the end of the file.
400-410	Finished so close the file and exit.
430-470	Is it a one-byte opcode? If yes, print the mnemonic.
490-530	Is it a type-one two-byte opcode? If yes, print the mnemonic.
550-590	Is it a type-two two-byte opcode? If yes, print the mnemonic.
610-670	Is it a type-three two-byte opcode? If yes, print the mnemonic.
690-730	Is it a type-one three-byte opcode? If yes, print the mnemonic.
750-790	Is it a type-two three-byte opcode? If yes, print the mnemonic.
810-850	Is it a type-one four-byte opcode? If yes, print the mnemonic.
870-910	Is it a type-two four-byte opcode? If yes, print the mnemonic.
930-970	Is it a type-three four-byte opcode? If yes, print the mnemonic.
990-1100	Print a given mnemonic, replacing any asterisks in the original with the passed values which are the actual opcodes.
1120-1320	Print the values of opcodes 1, 2, 3 and 4 in hexadecimal.
1340-1440	Read the opcodes and mnemonic data into the arrays.
1460-1490	Read the hexadecimal data, preface each with an & to make it useable by the EVAL statement then return its value.
1510-1580	Call the firmware File selector and return the name of any file selected.
1600-1830	Assemble the File selector calling code.
1890-3900	Single-byte opcode and mnemonic data.
3940-7130	Type-one two-byte opcode and mnemonic data.
7170-7340	Type-two two-byte opcode and mnemonic data.
7380-7430	Type-three two-byte opcode and mnemonic data.
7470-7720	Type-one three-byte opcode and mnemonic data.
7760-8230	Type-two three-byte opcode and mnemonic data.
8270-8280	Type-one four-byte opcode and mnemonic data.
8320-8430	Type-two four-byte opcode and mnemonic data.
8470-9080	Type-three four-byte opcode and mnemonic data.

Functions and procedures

PROCassemble	Assembles the File selector calling code.
PROCread_data	Reads the opcode and mnemonic data into the arrays.
PROCj	Prints the first opcode value in hexadecimal.

PROCk	Prints the second opcode value in hexadecimal.
PROCl	Prints the third opcode value in hexadecimal.
PROCm	Prints the fourth opcode value in hexadecimal.
PROCmnemon	Prints the mnemonic for the given opcode.
FNchk_one	Checks for a one-byte opcode.
FNchk_twoa	Checks for a type-one two-byte opcode.
FNchk_twob	Checks for a type-two two-byte opcode.
FNchk_twoc	Checks for a type-three two-byte opcode.
FNchk_threea	Checks for a type-one three-byte opcode.
FNchk_threeb	Checks for a type-two three-byte opcode.
FNchk_foura	Checks for a type-one four-byte opcode.
FNchk_fourb	Checks for a type-two four-byte opcode.
FNchk_fourc	Checks for a type-three four-byte opcode.
FNread	Reads the hexadecimal text data and converts to a number.
FNselect	Prompts the user for a file using the File selector.

Main variables and arrays

start%	The ORG address of the code being disassembled.
one%	Number of one-byte opcodes.
twoa%	Number of type-one two-byte opcodes.
twob%	Number of type-two two-byte opcodes.
twoc%	Number of type-three two-byte opcodes.
threea%	Number of type-one three-byte opcodes.
threeb%	Number of type-two three-byte opcodes.
foura%	Number of type-one four-byte opcodes.
fourb%	Number of type-two four-byte opcodes.
fourc%	Number of type-three four-byte opcodes.
A%	Space for the File selector machine code.
a1%	One-byte opcode array.
b1%	Type-one two-byte opcode array.
b2%	Type-two two-byte opcode array.
b3%	Type-three two-byte opcode array.
c1%	Type-one three-byte opcode array.
c2%	Type-two three-byte opcode array.
d1%	Type-one four-byte opcode array.
d2%	Type-two four-byte opcode array.
d3%	Type-three four-byte opcode array.

d4%	Type-four four-byte opcode array.
a1$	One-byte mnemonic array.
b1$	Type-one two-byte mnemonic array.
b2$	Type-two two-byte mnemonic array.
b3$	Type-three two-byte mnemonic array.
c1$	Type-one three-byte mnemonic array.
c2$	Type-two three-byte mnemonic array.
d1$	Type-one four-byte mnemonic array.
d2$	Type-two four-byte mnemonic array.
d3$	Type-three four-byte mnemonic array.
d4$	Type-four four-byte mnemonic array.
file$	The file name returned by the File selector.
handle	File handle of the opened file.
chk%	This is set and returned by the chk_... functions if one of them is successful in matching with the current opcode.
oldstart%	The previous value of start% before the current opcode was tested. This allows start% to be incremented as each new byte is read in, while oldstart% retains the correct address of the start of the current opcode.
J%	Opcode one. (Also used as a temporary loop counter during initialisation).
K%	Opcode two (if there is one).
L%	Opcode three (if there is one).
M%	Opcode four (if there is one).
C%,X%	Temporary loop counters.
flag%	If set, PROCmnemonic knows it need not search for any asterisks to replace with values, as there are none.
pos1%	The position of the first asterisk if there are two of them in a mnemonic.
pos2%	The position of the asterisk if there is just one in a mnemonic.
str1$,str2$	Left and right halves of a mnemonic having the asterisk stripped from it.
text$	The mnemonic, both before and after asterisk replacement.
B$	A string holding a four-character number representing the value of oldstart%, the current offset from the ORG address.
Z$	The two-byte hex string printed by PROCs j, k, l and m.
R$	Temporary string created to read the hexadecimal data into the arrays via an EVAL command.

buffer	Holds the file name obtained by the file selector in the machine code section of the program (part of the A% array).
found	Label in the machine code section from which point a file has been selected.
loop	Label in the machine codes section where a loop transfers the file name to a known location addressable from Basic.

The program

```
   10 CLS:PRINT "Z80 Disassembler"
   20 PRINT:PRINT "Please wait..."
   30 start%=0
   40 one%=202
   50 twoa%=320:twob%=18:twoc%=6
   60 threea%=26:threeb%=48
   70 foura%=2:fourb%=12:fourc%=62
   80 DIM A% 40
   90 DIM a1% one%,a1$(one%)
  100 DIM b1% twoa%,b2% twob%,b3% twoc%,b1$(twoa%),b2$(twob%),b3$(twoc%)
  110 DIM b1a% twoa%
  120 DIM c1% threea%,c2% threeb%,c1$(threea%),c2$(threeb%)
  130 DIM c2a% threeb%
  140 DIM d1% foura%,d2% fourb%,d3% fourc%,d1$(foura%),d2$(fourb%),
d3$(fourc%)
  150 DIM d1a% foura%,d2a% fourb%,d3a% fourc%,d3b% fourc%
  160 :
  170 PROCassemble
  180 PROCread_data
  190 file$=FNselect:IF file$="" THEN END
  200 I$="0":PRINT "Disassembling """;file$;""""":PRINT:INPUT "Please
enter ORG address: "I$:IF LEN(I$)>0 THEN start%=EVAL I$
  210 PRINT:handle=OPENIN(file$)
  220 :
  230 REPEAT
  240 chk%=0:oldstart%=start%
  250 J%=BGET #handle:start%=start%+1
  260 IF FNchk_one THEN GOTO 390
  270 K%=BGET #handle:start%=start%+1
  280 IF FNchk_twoa THEN GOTO 390
  290 IF FNchk_twob THEN GOTO 390
  300 IF FNchk_twoc THEN GOTO 390
  310 L%=BGET #handle:start%=start%+1
  320 IF FNchk_threea THEN GOTO 390
  330 IF FNchk_threeb THEN GOTO 390
  340 M%=BGET #handle:start%=start%+1
  350 IF FNchk_foura THEN GOTO 390
  360 IF FNchk_fourb THEN GOTO 390
  370 IF FNchk_fourc THEN GOTO 390
  380 PRINT "Unrecognised opcode"
  390 UNTIL EOF #handle
  400 CLOSE #handle
  410 PRINT:END
  420 :
  430 DEF FNchk_one
  440 FOR X%=0 TO one%-1
  450 IF J%=a1%?X% THEN PROCj:PRINT SPC(9);:
```

```
PROCmnemon(a1$(X%),0,0,0,2):VDU 27,J%:chk%=1
  460 NEXT
  470 =chk%
  480 :
  490 DEF FNchk_twoa
  500 FOR X%=0 TO twoa%-1
  510 IF J%=b1%?X% AND K%=b1a%?X% THEN PROCj:PROCk:PRINT SPC(6);:
PROCmnemon(b1$(X%),0,0,0,2):VDU 27,J%,27,K%:chk%=1
  520 NEXT
  530 =chk%;
  540 :
  550 DEF FNchk_twob
  560 FOR X%=0 TO twob%-1
  570 IF J%=b2%?X% THEN PROCj:PROCk:PRINT SPC(6);:
PROCmnemon(b2$(X%),1,K%,0,2):VDU 27,J%,27,K%:chk%=1
  580 NEXT
  590 =chk%
  600 :
  610 DEF FNchk_twoc
  620 offset=K%:IF K%>129 THEN offset=-(K%-129)
  630 IF (oldstart%+offset)<0 THEN offset=oldstart%
  640 FOR X%=0 TO twoc%-1
  650 IF J%=b3%?X% THEN PROCj:PROCk:PRINT SPC(6);:
PROCmnemon(b3$(X%),1,oldstart%+offset,0,4):VDU 27,J%,27,K%:chk%=1
  660 NEXT
  670 =chk%
  680 :
  690 DEF FNchk_threea
  700 FOR X%=0 TO threea%-1
  710 IF J%=c1%?X% THEN PROCj:PROCk:PROCl:PRINT SPC(3);:
PROCmnemon(c1$(X%),1,K%+L%*&100,0,4):VDU 27,J%,27,K%,27,L%:chk%=1
  720 NEXT
  730 =chk%
  740 :
  750 DEF FNchk_threeb
  760 FOR X%=0 TO threeb%-1
  770 IF J%=c2%?X% AND K%=c2a%?X% THEN PROCj:PROCk:PROCl:PRINT
SPC(3);:PROCmnemon(c2$(X%),1,L%,0,2):VDU 27,J%,27,K%,27,L%:chk%=1
  780 NEXT
  790 =chk%
  800 :
  810 DEF FNchk_foura
  820 FOR X%=0 TO foura%-1
  830 IF J%=d1%?X% AND K%=d1a%?X% THEN PROCj:PROCk:PROCl:PROCm:
PROCmnemon(d1$(X%),2,L%,M%,2):VDU 27,J%,27,K%,27,L%,27,M%:chk%=1
  840 NEXT
  850 =chk%
  860 :
  870 DEF FNchk_fourb
  880 FOR X%=0 TO fourb%-1
  890 IF J%=d2%?X% AND K%=d2a%?X% THEN PROCj:PROCk:PROCl:PROCm:
PROCmnemon(d2$(X%),1,L%+M%*&100,0,4):VDU 27,J%,27,K%,27,L%,27,M%:chk%=1
  900 NEXT
  910 =chk%
  920 :
  930 DEF FNchk_fourc
  940 FOR X%=0 TO fourc%-1
  950 IF J%=d3%?X% AND K%=d3a%?X% AND M%=d3b%?X% THEN PROCj:PROCk:PROCl:
PROCm:PROCmnemon(d3$(X%),1,L%,0,2):VDU 27,J%,27,K%,27,L%,27,M%:chk%=1
  960 NEXT
```

```
  970 =chk%
  980 :
  990 DEF PROCmnemon(text$,flag%,val1%,val2%,size%)
 1000 IF flag%=0 THEN GOTO 1090
 1010 pos1%=0:pos2%=0
 1020 FOR C%=1 TO LEN(text$)
 1030 IF MID$(text$,C%,1)="*" THEN pos1%=pos2%:pos2%=C%
 1040 NEXT
 1050 str1$=STR$ (val1%):str1$="&"+STRING$(size%-LEN(str1$),"0")+str1$
 1060 str2$=STR$ (val2%):str2$="&"+STRING$(size%-LEN(str2$),"0")+str2$
 1070 text$=LEFT$(text$,pos2%-1)+str1$+RIGHT$(text$,LEN(text$)-pos2%)
 1080 IF flag%=2 THENtext$=LEFT$(text$,pos1%-1)+str2$+RIGHT$
(text$,LEN(text$)-pos1%)
 1090 PRINT text$;:VDU 31,40,VPOS
 1100 ENDPROC
 1110 :
 1120 DEF PROCj
 1130 B$=STR$ (oldstart%):B$=STRING$(4-LEN(B$),"0")+B$
 1140 PRINT:PRINT ;B$;SPC(2);
 1150 Z$=STR$ (J%):IF J%<16 THEN Z$="0"+Z$
 1160 PRINT ;Z$;" ";
 1170 ENDPROC
 1180 :
 1190 DEF PROCk
 1200 Z$=STR$ (K%):IF K%<16 THEN Z$="0"+Z$
 1210 PRINT ;Z$;" ";
 1220 ENDPROC
 1230 :
 1240 DEF PROCl
 1250 Z$=STR$ (L%):IF L%<16 THEN Z$="0"+Z$
 1260 PRINT ;Z$;" ";
 1270 ENDPROC
 1280 :
 1290 DEF PROCm
 1300 Z$=STR$ (M%):IF M%<16 THEN Z$="0"+Z$
 1310 PRINT ;Z$;" ";
 1320 ENDPROC
 1330 :
 1340 DEF PROCread_data
 1350 FOR J%=0 TO one%-1:a1%?J%=FNread:READ a1$(J%):NEXT
 1360 FOR J%=0 TO two%-1:b1%?J%=FNread:b1a%?J%=FNread:READ b1$(J%):NEXT
 1370 FOR J%=0 TO twob%-1:b2%?J%=FNread:READ b2$(J%):NEXT
 1380 FOR J%=0 TO twoc%-1:b3%?J%=FNread:READ b3$(J%):NEXT
 1390 FOR J%=0 TO threea%-1:c1%?J%=FNread:READ c1$(J%):NEXT
 1400 FOR J%=0 TO threeb%-1:c2%?J%=FNread:c2a%?J%=FNread:READ
c2$(J%):NEXT
 1410 FOR J%=0 TO foura%-1:d1%?J%=FNread:d1a%?J%=FNread:READ d1$(J%):NEXT
 1420 FOR J%=0 TO fourb%-1:d2%?J%=FNread:d2a%?J%=FNread:READ d2$(J%):NEXT
 1430 FOR J%=0 TO fourc%-1:d3%?J%=FNread:d3a%?J%=FNread:d3b%?J%=FNread:
READ d3$(J%):NEXT
 1440 ENDPROC
 1450 :
 1460 DEF FNread
 1470 READ R$
 1480 R$="&"+R$
 1490 =EVAL R$
 1500 :
 1510 DEF FNselect
 1520 CALL A%
 1530 IF buffer?0 = 0 THEN CLS:=""
```

```
1540 R$=""
1550 FOR J%=0 TO 11
1560 IF buffer?J% THEN R$=R$+CHR$(buffer?J%) ELSE J%=12
1570 NEXT
1580 =R$
1590 :
1600 DEF PROCassemble
1610 FOR PASS=0 TO 2 STEP 2
1620 P%=A%
1630 [
1640 OPT PASS
1650 CALL &B8C3
1660 LD DE,buffer
1670 JR C,found
1680 LD A,0
1690 LD (DE),A
1700 RET
1710 .found
1720 LD B,12
1730 .loop
1740 LD A,(HL)
1750 LD (DE),A
1760 INC HL
1770 INC DE
1780 DJNZ loop
1790 RET
1800 .buffer
1810 ]
1820 NEXT
1830 ENDPROC
1840 :
1850 REM Instruction codes and mnemonics
1860 :
1870 REM Single byte
1880 :
1890 DATA 00,NOP
1900 DATA 02,"LD (BC),A"
1910 DATA 03,INC BC
1920 DATA 04,INC B
1930 DATA 05,DEC B
1940 DATA 07,RLCA
1950 DATA 08,"EX AF,AF"'
1960 DATA 09,"ADD HL,BC"
1970 DATA 0A,"LD A,(BC)"
1980 DATA 0B,DEC BC
1990 DATA 0C,INC C
2000 DATA 0D,DEC C
2010 DATA 0F,RRCA
2020 DATA 12,"LD (DE),A"
2030 DATA 13,INC DE
2040 DATA 14,INC D
2050 DATA 15,DEC D
2060 DATA 17,RLA
2070 DATA 19,"ADD HL,DE"
2080 DATA 1A,"LD A,(DE)"
2090 DATA 1B,DEC DE
2100 DATA 1C,INC E
2110 DATA 1D,DEC E
2120 DATA 1F,RRA
2130 DATA 23,INC HL
```

```
2140 DATA 24,INC H
2150 DATA 25,DEC H
2160 DATA 27,DAA
2170 DATA 29,"ADD HL,HL"
2180 DATA 2B,DEC HL
2190 DATA 2C,INC L
2200 DATA 2D,DEC L
2210 DATA 2F,CPL
2220 DATA 33,INC SP
2230 DATA 34,INC (HL)
2240 DATA 35,DEC (HL)
2250 DATA 37,SCF
2260 DATA 39,"ADD HL,SP"
2270 DATA 3B,DEC SP
2280 DATA 3C,INC A
2290 DATA 3D,DEC A
2300 DATA 3F,CCF
2310 DATA 40,"LD B,B"
2320 DATA 41,"LD B,C"
2330 DATA 42,"LD B,D"
2340 DATA 43,"LD B,E"
2350 DATA 44,"LD B,H"
2360 DATA 45,"LD B,L"
2370 DATA 46,"LD B,(HL)"
2380 DATA 47,"LD B,A"
2390 DATA 48,"LD C,B"
2400 DATA 49,"LD C,C"
2410 DATA 4A,"LD C,D"
2420 DATA 4B,"LD C,E"
2430 DATA 4C,"LD C,H"
2440 DATA 4D,"LD C,L"
2450 DATA 4E,"LD C,(HL)"
2460 DATA 4F,"LD C,A"
2470 DATA 50,"LD D,B"
2480 DATA 51,"LD D,C"
2490 DATA 52,"LD D,D"
2500 DATA 53,"LD D,E"
2510 DATA 54,"LD D,H"
2520 DATA 55,"LD D,L"
2530 DATA 56,"LD D,(HL)"
2540 DATA 57,"LD D,A"
2550 DATA 58,"LD E,B"
2560 DATA 59,"LD E,C"
2570 DATA 5A,"LD E,D"
2580 DATA 5B,"LD E,E"
2590 DATA 5C,"LD E,H"
2600 DATA 5D,"LD E,L"
2610 DATA 5E,"LD E,(HL)"
2620 DATA 5F,"LD E,A"
2630 DATA 60,"LD H,B"
2640 DATA 61,"LD H,C"
2650 DATA 62,"LD H,D"
2660 DATA 63,"LD H,E"
2670 DATA 64,"LD H,H"
2680 DATA 65,"LD H,L"
2690 DATA 66,"LD H,(HL)"
2700 DATA 67,"LD H,A"
2710 DATA 68,"LD L,B"
2720 DATA 69,"LD L,C"
2730 DATA 6A,"LD L,D"
```

```
2740 DATA 6B,"LD L,E"
2750 DATA 6C,"LD L,H"
2760 DATA 6D,"LD L,L"
2770 DATA 6E,"LD L,(HL)"
2780 DATA 6F,"LD L,A"
2790 DATA 70,"LD (HL),B"
2800 DATA 71,"LD (HL),C"
2810 DATA 72,"LD (HL),D"
2820 DATA 73,"LD (HL),E"
2830 DATA 74,"LD (HL),H"
2840 DATA 75,"LD (HL),L"
2850 DATA 76,HALT
2860 DATA 77,"LD (HL),A"
2870 DATA 78,"LD A,B"
2880 DATA 79,"LD A,C"
2890 DATA 7A,"LD A,D"
2900 DATA 7B,"LD A,E"
2910 DATA 7C,"LD A,H"
2920 DATA 7D,"LD A,L"
2930 DATA 7E,"LD A,(HL)"
2940 DATA 7F,"LD A,A"
2950 DATA 80,"ADD A,B"
2960 DATA 81,"ADD A,C"
2970 DATA 82,"ADD A,D"
2980 DATA 83,"ADD A,E"
2990 DATA 84,"ADD A,H"
3000 DATA 85,"ADD A,L"
3010 DATA 86,"ADD A,(HL)"
3020 DATA 87,"ADD A,A"
3030 DATA 88,"ADC A,B"
3040 DATA 89,"ADC A,C"
3050 DATA 8A,"ADC A,D"
3060 DATA 8B,"ADC A,E"
3070 DATA 8C,"ADC A,H"
3080 DATA 8D,"ADC A,L"
3090 DATA 8E,"ADC A,(HL)"
3100 DATA 8F,"ADC A,A"
3110 DATA 90,SUB B
3120 DATA 91,SUB C
3130 DATA 92,SUB D
3140 DATA 93,SUB E
3150 DATA 94,SUB H
3160 DATA 95,SUB L
3170 DATA 96,SUB (HL)
3180 DATA 97,SUB A
3190 DATA 98,"SBC A,B"
3200 DATA 99,"SBC A,C"
3210 DATA 9A,"SBC A,D"
3220 DATA 9B,"SBC A,E"
3230 DATA 9C,"SBC A,H"
3240 DATA 9D,"SBC A,L"
3250 DATA 9E,"SBC A,(HL)"
3260 DATA 9F,"SBC A,A"
3270 DATA A0,AND B
3280 DATA A1,AND C
3290 DATA A2,AND D
3300 DATA A3,AND E
3310 DATA A4,AND H
3320 DATA A5,AND L
3330 DATA A6,AND (HL)
```

```
3340 DATA A7,AND A
3350 DATA A8,XOR B
3360 DATA A9,XOR C
3370 DATA AA,XOR D
3380 DATA AB,XOR E
3390 DATA AC,XOR H
3400 DATA AD,XOR L
3410 DATA AE,XOR (HL)
3420 DATA AF,XOR A
3430 DATA B0,OR B
3440 DATA B1,OR C
3450 DATA B2,OR D
3460 DATA B3,OR E
3470 DATA B4,OR H
3480 DATA B5,OR L
3490 DATA B6,OR (HL)
3500 DATA B7,OR A
3510 DATA B8,CP B
3520 DATA B9,CP C
3530 DATA BA,CP D
3540 DATA BB,CP E
3550 DATA BC,CP H
3560 DATA BD,CP L
3570 DATA BE,CP (HL)
3580 DATA BF,CP A
3590 DATA C0,RET NZ
3600 DATA C1,POP BC
3610 DATA C5,PUSH BC
3620 DATA C7,RST &00
3630 DATA C8,RET Z
3640 DATA C9,RET
3650 DATA CF,RST &08
3660 DATA D0,RET NC
3670 DATA D1,POP DE
3680 DATA D5,PUSH DE
3690 DATA D7,RST &10
3700 DATA D8,RET C
3710 DATA D9,EXX
3720 DATA DF,RST &18
3730 DATA E0,RET PO
3740 DATA E1,POP HL
3750 DATA E3,"EX (SP),HL"
3760 DATA E5,PUSH HL
3770 DATA E7,RST &20
3780 DATA E8,RET PE
3790 DATA E9,JP (HL)
3800 DATA EB,"EX DE,HL"
3810 DATA EF,RST &28
3820 DATA F0,RET P
3830 DATA F1,POP AF
3840 DATA F3,DI
3850 DATA F5,PUSH AF
3860 DATA F7,RST &30
3870 DATA F8,RET M
3880 DATA F9,"LD SP,HL"
3890 DATA FB,EI
3900 DATA FF,RST &38
3910 :
3920 REM Two byte
3930 :
```

```
3940 DATA CB,00,RLC B
3950 DATA CB,01,RLC C
3960 DATA CB,02,RLC D
3970 DATA CB,03,RLC E
3980 DATA CB,04,RLC H
3990 DATA CB,05,RLC L
4000 DATA CB,06,RLC (HL)
4010 DATA CB,07,RLC A
4020 DATA CB,08,RRC B
4030 DATA CB,09,RRC C
4040 DATA CB,0A,RRC D
4050 DATA CB,0B,RRC E
4060 DATA CB,0C,RRC H
4070 DATA CB,0D,RRC L
4080 DATA CB,0E,RRC (HL)
4090 DATA CB,0F,RRC A
4100 DATA CB,10,RL B
4110 DATA CB,11,RL C
4120 DATA CB,12,RL D
4130 DATA CB,13,RL E
4140 DATA CB,14,RL H
4150 DATA CB,15,RL L
4160 DATA CB,16,RL (HL)
4170 DATA CB,17,RL A
4180 DATA CB,18,RR B
4190 DATA CB,19,RR C
4200 DATA CB,1A,RR D
4210 DATA CB,1B,RR E
4220 DATA CB,1C,RR H
4230 DATA CB,1D,RR L
4240 DATA CB,1E,RR (HL)
4250 DATA CB,1F,RR A
4260 DATA CB,20,SLA B
4270 DATA CB,21,SLA C
4280 DATA CB,22,SLA D
4290 DATA CB,23,SLA E
4300 DATA CB,24,SLA H
4310 DATA CB,25,SLA L
4320 DATA CB,26,SLA (HL)
4330 DATA CB,27,SLA A
4340 DATA CB,28,SRA B
4350 DATA CB,29,SRA C
4360 DATA CB,2A,SRA D
4370 DATA CB,2B,SRA E
4380 DATA CB,2C,SRA H
4390 DATA CB,2D,SRA L
4400 DATA CB,2E,SRA (HL)
4410 DATA CB,2F,SRA A
4420 DATA CB,38,SRL B
4430 DATA CB,39,SRL C
4440 DATA CB,3A,SRL D
4450 DATA CB,3B,SRL E
4460 DATA CB,3C,SRL H
4470 DATA CB,3D,SRL L
4480 DATA CB,3E,SRL (HL)
4490 DATA CB,3F,SRL A
4500 DATA CB,40,"BIT 0,B"
4510 DATA CB,41,"BIT 0,C"
4520 DATA CB,42,"BIT 0,D"
4530 DATA CB,43,"BIT 0,E"
```

```
4540 DATA CB,44,"BIT 0,H"
4550 DATA CB,45,"BIT 0,L"
4560 DATA CB,46,"BIT 0,(HL)"
4570 DATA CB,47,"BIT 0,A"
4580 DATA CB,48,"BIT 1,B"
4590 DATA CB,49,"BIT 1,C"
4600 DATA CB,4A,"BIT 1,D"
4610 DATA CB,4B,"BIT 1,E"
4620 DATA CB,4C,"BIT 1,H"
4630 DATA CB,4D,"BIT 1,L"
4640 DATA CB,4E,"BIT 1,(HL)"
4650 DATA CB,4F,"BIT 1,A"
4660 DATA CB,50,"BIT 2,B"
4670 DATA CB,51,"BIT 2,C"
4680 DATA CB,52,"BIT 2,D"
4690 DATA CB,53,"BIT 2,E"
4700 DATA CB,54,"BIT 2,H"
4710 DATA CB,55,"BIT 2,L"
4720 DATA CB,56,"BIT 2,(HL)"
4730 DATA CB,57,"BIT 2,A"
4740 DATA CB,58,"BIT 3,B"
4750 DATA CB,59,"BIT 3,C"
4760 DATA CB,5A,"BIT 3,D"
4770 DATA CB,5B,"BIT 3,E"
4780 DATA CB,5C,"BIT 3,H"
4790 DATA CB,5D,"BIT 3,L"
4800 DATA CB,5E,"BIT 3,(HL)"
4810 DATA CB,5F,"BIT 3,A"
4820 DATA CB,60,"BIT 4,B"
4830 DATA CB,61,"BIT 4,C"
4840 DATA CB,62,"BIT 4,D"
4850 DATA CB,63,"BIT 4,E"
4860 DATA CB,64,"BIT 4,H"
4870 DATA CB,65,"BIT 4,L"
4880 DATA CB,66,"BIT 4,(HL)"
4890 DATA CB,67,"BIT 4,A"
4900 DATA CB,68,"BIT 5,B"
4910 DATA CB,69,"BIT 5,C"
4920 DATA CB,6A,"BIT 5,D"
4930 DATA CB,6B,"BIT 5,E"
4940 DATA CB,6C,"BIT 5,H"
4950 DATA CB,6D,"BIT 5,L"
4960 DATA CB,6E,"BIT 5,(HL)"
4970 DATA CB,6F,"BIT 5,A"
4980 DATA CB,70,"BIT 6,B"
4990 DATA CB,71,"BIT 6,C"
5000 DATA CB,72,"BIT 6,D"
5010 DATA CB,73,"BIT 6,E"
5020 DATA CB,74,"BIT 6,H"
5030 DATA CB,75,"BIT 6,L"
5040 DATA CB,76,"BIT 6,(HL)"
5050 DATA CB,77,"BIT 6,A"
5060 DATA CB,78,"BIT 7,B"
5070 DATA CB,79,"BIT 7,C"
5080 DATA CB,7A,"BIT 7,D"
5090 DATA CB,7B,"BIT 7,E"
5100 DATA CB,7C,"BIT 7,H"
5110 DATA CB,7D,"BIT 7,L"
5120 DATA CB,7E,"BIT 7,(HL)"
5130 DATA CB,7F,"BIT 7,A"
```

```
5140 DATA CB,80,"RES 0,B"
5150 DATA CB,81,"RES 0,C"
5160 DATA CB,82,"RES 0,D"
5170 DATA CB,83,"RES 0,E"
5180 DATA CB,84,"RES 0,H"
5190 DATA CB,85,"RES 0,L"
5200 DATA CB,86,"RES 0,(HL)"
5210 DATA CB,87,"RES 0,A"
5220 DATA CB,88,"RES 1,B"
5230 DATA CB,89,"RES 1,C"
5240 DATA CB,8A,"RES 1,D"
5250 DATA CB,8B,"RES 1,E"
5260 DATA CB,8C,"RES 1,H"
5270 DATA CB,8D,"RES 1,L"
5280 DATA CB,8E,"RES 1,(HL)"
5290 DATA CB,8F,"RES 1,A"
5300 DATA CB,90,"RES 2,B"
5310 DATA CB,91,"RES 2,C"
5320 DATA CB,92,"RES 2,D"
5330 DATA CB,93,"RES 2,E"
5340 DATA CB,94,"RES 2,H"
5350 DATA CB,95,"RES 2,L"
5360 DATA CB,96,"RES 2,(HL)"
5370 DATA CB,97,"RES 2,A"
5380 DATA CB,98,"RES 3,B"
5390 DATA CB,99,"RES 3,C"
5400 DATA CB,9A,"RES 3,D"
5410 DATA CB,9B,"RES 3,E"
5420 DATA CB,9C,"RES 3,H"
5430 DATA CB,9D,"RES 3,L"
5440 DATA CB,9E,"RES 3,(HL)"
5450 DATA CB,9F,"RES 3,A"
5460 DATA CB,A0,"RES 4,B"
5470 DATA CB,A1,"RES 4,C"
5480 DATA CB,A2,"RES 4,D"
5490 DATA CB,A3,"RES 4,E"
5500 DATA CB,A4,"RES 4,H"
5510 DATA CB,A5,"RES 4,L"
5520 DATA CB,A6,"RES 4,(HL)"
5530 DATA CB,A7,"RES 4,A"
5540 DATA CB,A8,"RES 5,B"
5550 DATA CB,A9,"RES 5,C"
5560 DATA CB,AA,"RES 5,D"
5570 DATA CB,AB,"RES 5,E"
5580 DATA CB,AC,"RES 5,H"
5590 DATA CB,AD,"RES 5,L"
5600 DATA CB,AE,"RES 5,(HL)"
5610 DATA CB,AF,"RES 5,A"
5620 DATA CB,B0,"RES 6,B"
5630 DATA CB,B1,"RES 6,C"
5640 DATA CB,B2,"RES 6,D"
5650 DATA CB,B3,"RES 6,E"
5660 DATA CB,B4,"RES 6,H"
5670 DATA CB,B5,"RES 6,L"
5680 DATA CB,B6,"RES 6,(HL)"
5690 DATA CB,B7,"RES 6,A"
5700 DATA CB,B8,"RES 7,B"
5710 DATA CB,B9,"RES 7,C"
5720 DATA CB,BA,"RES 7,D"
5730 DATA CB,BB,"RES 7,E"
```

```
5740 DATA CB,BC,"RES 7,H"
5750 DATA CB,BD,"RES 7,L"
5760 DATA CB,BE,"RES 7,(HL)"
5770 DATA CB,BF,"RES 7,A"
5780 DATA CB,C0,"SET 0,B"
5790 DATA CB,C1,"SET 0,C"
5800 DATA CB,C2,"SET 0,D"
5810 DATA CB,C3,"SET 0,E"
5820 DATA CB,C4,"SET 0,H"
5830 DATA CB,C5,"SET 0,L"
5840 DATA CB,C6,"SET 0,(HL)"
5850 DATA CB,C7,"SET 0,A"
5860 DATA CB,C8,"SET 1,B"
5870 DATA CB,C9,"SET 1,C"
5880 DATA CB,CA,"SET 1,D"
5890 DATA CB,CB,"SET 1,E"
5900 DATA CB,CC,"SET 1,H"
5910 DATA CB,CD,"SET 1,L"
5920 DATA CB,CE,"SET 1,(HL)"
5930 DATA CB,CF,"SET 1,A"
5940 DATA CB,D0,"SET 2,B"
5950 DATA CB,D1,"SET 2,C"
5960 DATA CB,D2,"SET 2,D"
5970 DATA CB,D3,"SET 2,E"
5980 DATA CB,D4,"SET 2,H"
5990 DATA CB,D5,"SET 2,L"
6000 DATA CB,D6,"SET 2,(HL)"
6010 DATA CB,D7,"SET 2,A"
6020 DATA CB,D8,"SET 3,B"
6030 DATA CB,D9,"SET 3,C"
6040 DATA CB,DA,"SET 3,D"
6050 DATA CB,DB,"SET 3,E"
6060 DATA CB,DC,"SET 3,H"
6070 DATA CB,DD,"SET 3,L"
6080 DATA CB,DE,"SET 3,(HL)"
6090 DATA CB,DF,"SET 3,A"
6100 DATA CB,E0,"SET 4,B"
6110 DATA CB,E1,"SET 4,C"
6120 DATA CB,E2,"SET 4,D"
6130 DATA CB,E3,"SET 4,E"
6140 DATA CB,E4,"SET 4,H"
6150 DATA CB,E5,"SET 4,L"
6160 DATA CB,E6,"SET 4,(HL)"
6170 DATA CB,E7,"SET 4,A"
6180 DATA CB,E8,"SET 5,B"
6190 DATA CB,E9,"SET 5,C"
6200 DATA CB,EA,"SET 5,D"
6210 DATA CB,EB,"SET 5,E"
6220 DATA CB,EC,"SET 5,H"
6230 DATA CB,ED,"SET 5,L"
6240 DATA CB,EE,"SET 5,(HL)"
6250 DATA CB,EF,"SET 5,A"
6260 DATA CB,F0,"SET 6,B"
6270 DATA CB,F1,"SET 6,C"
6280 DATA CB,F2,"SET 6,D"
6290 DATA CB,F3,"SET 6,E"
6300 DATA CB,F4,"SET 6,H"
6310 DATA CB,F5,"SET 6,L"
6320 DATA CB,F6,"SET 6,(HL)"
6330 DATA CB,F7,"SET 6,A"
```

```
6340 DATA CB,F8,"SET 7,B"
6350 DATA CB,F9,"SET 7,C"
6360 DATA CB,FA,"SET 7,D"
6370 DATA CB,FB,"SET 7,E"
6380 DATA CB,FC,"SET 7,H"
6390 DATA CB,FD,"SET 7,L"
6400 DATA CB,FE,"SET 7,(HL)"
6410 DATA CB,FF,"SET 7,A"
6420 DATA DD,09,"ADD IX,BC"
6430 DATA DD,19,"ADD IX,DE"
6440 DATA DD,23,INC IX
6450 DATA DD,29,"ADD IX,IX"
6460 DATA DD,2B,DEC IX
6470 DATA DD,39,"ADD IX,SP"
6480 DATA DD,E1,POP IX
6490 DATA DD,E3,"EX (SP),IX"
6500 DATA DD,E5,PUSH IX
6510 DATA DD,E9,JP (IX)
6520 DATA DD,F9,"LD SP,IX"
6530 DATA ED,40,"IN B,(C)"
6540 DATA ED,41,"OUT (C),B"
6550 DATA ED,42,"SBC HL,BC"
6560 DATA ED,44,NEG
6570 DATA ED,45,RETN
6580 DATA ED,46,IM 0
6590 DATA ED,47,"LD I,A"
6600 DATA ED,48,"IN C,(C)"
6610 DATA ED,49,"OUT (C),C"
6620 DATA ED,4A,"ADC HL,BC"
6630 DATA ED,4D,RETI
6640 DATA ED,4F,"LD R,A"
6650 DATA ED,50,"IN D,(C)"
6660 DATA ED,51,"OUT (C),D"
6670 DATA ED,52,"SBC HL,DE"
6680 DATA ED,56,IM 1
6690 DATA ED,57,"LD A,I"
6700 DATA ED,58,"IN E,(C)"
6710 DATA ED,59,"OUT (C),E"
6720 DATA ED,5A,"ADC HL,DE"
6730 DATA ED,5E,IM 2
6740 DATA ED,5F,"LD A,R"
6750 DATA ED,60,"IN H,(C)"
6760 DATA ED,61,"OUT (C),H"
6770 DATA ED,62,"SBC HL,HL"
6780 DATA ED,67,RRD
6790 DATA ED,68,"IN L,(C)"
6800 DATA ED,69,"OUT (C),L"
6810 DATA ED,6A,"ADC HL,HL"
6820 DATA ED,6F,RLD
6830 DATA ED,72,"SBC HL,SP"
6840 DATA ED,78,"IN A,(C)"
6850 DATA ED,79,"OUT (C),A"
6860 DATA ED,7A,"ADC HL,SP"
6870 DATA ED,8B,OTDR
6880 DATA ED,A0,LDI
6890 DATA ED,A1,CPI
6900 DATA ED,A2,INI
6910 DATA ED,A3,OUTI
6920 DATA ED,A8,LDD
6930 DATA ED,A9,CPD
```

```
6940 DATA ED,AA,IND
6950 DATA ED,AB,OUTD
6960 DATA ED,B0,LDIR
6970 DATA ED,B1,CPIR
6980 DATA ED,B2,INIR
6990 DATA ED,B3,OTIR
7000 DATA ED,B8,LDDR
7010 DATA ED,B9,CPDR
7020 DATA ED,BA,INDR
7030 DATA FD,09,"ADD IY,BC"
7040 DATA FD,19,"ADD IY,DE"
7050 DATA FD,23,INC IY
7060 DATA FD,29,"ADD IY,IY"
7070 DATA FD,2B,DEC IY
7080 DATA FD,39,"ADD IY,SP"
7090 DATA FD,E1,POP IY
7100 DATA FD,E3,"EX (SP),IY"
7110 DATA FD,E5,PUSH IY
7120 DATA FD,E9,JP (IY)
7130 DATA FD,F9,"LD SP,IY"
7140 :
7150 REM Two byte: n
7160 :
7170 DATA 06,"LD B,*"
7180 DATA 0E,"LD C,*"
7190 DATA 16,"LD D,*"
7200 DATA 1E,"LD E,*"
7210 DATA 26,"LD H,*"
7220 DATA 2E,"LD L,*"
7230 DATA 36,"LD (HL),*"
7240 DATA 3E,"LD A,*"
7250 DATA C6,"ADD A,*"
7260 DATA CE,"ADC A,*"
7270 DATA D3,"OUT (*),A"
7280 DATA D6,SUB *
7290 DATA DB,"IN A,*"
7300 DATA DE,"SBC A,*"
7310 DATA E6,AND *
7320 DATA EE,XOR *
7330 DATA F6,OR *
7340 DATA FE,CP *
7350 :
7360 REM Two byte: e
7370 :
7380 DATA 10,DJNZ *
7390 DATA 18,JR *
7400 DATA 20,"JR NZ,*"
7410 DATA 28,"JR Z,*"
7420 DATA 30,"JR NC,*"
7430 DATA 38,"JR C,*"
7440 :
7450 REM Three byte: nn
7460 :
7470 DATA 01,"LD BC,*"
7480 DATA 11,"LD DE,*"
7490 DATA 21,"LD HL,*"
7500 DATA 22,"LD (*),HL"
7510 DATA 2A,"LD HL,(*)"
7520 DATA 31,"LD SP,*"
7530 DATA 32,"LD (*),A"
```

```
7540 DATA 3A,"LD A,(*)"
7550 DATA C2,"JP NZ,*"
7560 DATA C3,JP *
7570 DATA C4,"CALL NZ,*"
7580 DATA CA,"JP Z,*"
7590 DATA CC,"CALL Z,*"
7600 DATA CD,CALL *
7610 DATA D2,"JP NC,*"
7620 DATA D4,"CALL NC,*"
7630 DATA DA,"JP C,*"
7640 DATA DC,"CALL C,*"
7650 DATA E2,"JP PO,*"
7660 DATA E4,"CALL PO,*"
7670 DATA EA,"JP PE,*"
7680 DATA EC,"CALL PE,*"
7690 DATA F2,"JP P,*"
7700 DATA F4,"CALL P,*"
7710 DATA FA,"JP M,*"
7720 DATA FC,"CALL M,*"
7730 :
7740 REM Three byte: IX/IY n
7750 :
7760 DATA DD,34,INC (IX+*)
7770 DATA DD,35,DEC (IX+*)
7780 DATA DD,46,"LD B,(IX+*)"
7790 DATA DD,4E,"LD C,(IX+*)"
7800 DATA DD,56,"LD D,(IX+*)"
7810 DATA DD,5E,"LD E,(IX+*)"
7820 DATA DD,66,"LD H,(IX+*)"
7830 DATA DD,6E,"LD L,(IX+*)"
7840 DATA DD,70,"LD (IX+*),B"
7850 DATA DD,71,"LD (IX+*),C"
7860 DATA DD,72,"LD (IX+*),D"
7870 DATA DD,73,"LD (IX+*),E"
7880 DATA DD,74,"LD (IX+*),H"
7890 DATA DD,75,"LD (IX+*),L"
7900 DATA DD,77,"LD (IX+*),A"
7910 DATA DD,7E,"LD A,(IX+*)"
7920 DATA DD,86,"ADD A,(IX+*)"
7930 DATA DD,8E,"ADC A,(IX+*)"
7940 DATA DD,96,SUB (IX+*)
7950 DATA DD,9E,"SBC A,(IX+*)"
7960 DATA DD,A6,AND (IX+*)
7970 DATA DD,AE,XOR (IX+*)
7980 DATA DD,B6,OR (IX+*)
7990 DATA DD,BE,CP (IX+*)
8000 DATA FD,34,INC (IY+*)
8010 DATA FD,35,DEC (IY+*)
8020 DATA FD,46,"LD B,(IY+*)"
8030 DATA FD,4E,"LD C,(IY+*)"
8040 DATA FD,56,"LD D,(IY+*)"
8050 DATA FD,5E,"LD E,(IY+*)"
8060 DATA FD,66,"LD H,(IY+*)"
8070 DATA FD,6E,"LD L,(IY+*)"
8080 DATA FD,70,"LD (IY+*),B"
8090 DATA FD,71,"LD (IY+*),C"
8100 DATA FD,72,"LD (IY+*),D"
8110 DATA FD,73,"LD (IY+*),E"
8120 DATA FD,74,"LD (IY+*),H"
8130 DATA FD,75,"LD (IY+*),L"
```

```
8140 DATA FD,77,"LD (IY+*),A"
8150 DATA FD,7E,"LD A,(IY+*)"
8160 DATA FD,86,"ADD A,(IY+*)"
8170 DATA FD,8E,"ADC A,(IY+*)"
8180 DATA FD,96,SUB (IY+*)
8190 DATA FD,9E,"SBC A,(IY+*)"
8200 DATA FD,A6,AND (IY+*)
8210 DATA FD,AE,XOR (IY+*)
8220 DATA FD,B6,OR (IY+*)
8230 DATA FD,BE,CP (IY+*)
8240 :
8250 REM Four byte: n
8260 :
8270 DATA DD,36,"LD (IX+*),*"
8280 DATA FD,36,"LD (IY+*),*"
8290 :
8300 REM Four byte: nn
8310 :
8320 DATA DD,21,"LD IX,*"
8330 DATA DD,22,"LD (*),IX"
8340 DATA DD,2A,"LD IX,(*)"
8350 DATA ED,43,"LD (*),BC"
8360 DATA ED,4B,"LD BC,(*)"
8370 DATA ED,53,"LD (*),DE"
8380 DATA ED,5B,"LD DE,(*)"
8390 DATA ED,73,"LD (*),SP"
8400 DATA ED,7B,"LD SP,(*)"
8410 DATA FD,21,"LD IY,*"
8420 DATA FD,22,"LD (*),IY"
8430 DATA FD,2A,"LD IY,(*)"
8440 :
8450 REM Four byte: IX/IY n
8460 :
8470 DATA DD,CB,06,RLC (IX+*)
8480 DATA DD,CB,0E,RRC (IX+*)
8490 DATA DD,CB,16,RL (IX+*)
8500 DATA DD,CB,1E,RR (IX+*)
8510 DATA DD,CB,26,SLA (IX+*)
8520 DATA DD,CB,2E,SRA (IX+*)
8530 DATA DD,CB,3E,SRL (IX+*)
8540 DATA DD,CB,46,"BIT 0,(IX+*)"
8550 DATA DD,CB,4E,"BIT 1,(IX+*)"
8560 DATA DD,CB,56,"BIT 2,(IX+*)"
8570 DATA DD,CB,5E,"BIT 3,(IX+*)"
8580 DATA DD,CB,66,"BIT 4,(IX+*)"
8590 DATA DD,CB,6E,"BIT 5,(IX+*)"
8600 DATA DD,CB,76,"BIT 6,(IX+*)"
8610 DATA DD,CB,7E,"BIT 7,(IX+*)"
8620 DATA DD,CB,86,"RES 0,(IX+*)"
8630 DATA DD,CB,8E,"RES 1,(IX+*)"
8640 DATA DD,CB,96,"RES 2,(IX+*)"
8650 DATA DD,CB,9E,"RES 3,(IX+*)"
8660 DATA DD,CB,A6,"RES 4,(IX+*)"
8670 DATA DD,CB,AE,"RES 5,(IX+*)"
8680 DATA DD,CB,B6,"RES 6,(IX+*)"
8690 DATA DD,CB,BE,"RES 7,(IX+*)"
8700 DATA DD,CB,C6,"SET 0,(IX+*)"
8710 DATA DD,CB,CE,"SET 1,(IX+*)"
8720 DATA DD,CB,D6,"SET 2,(IX+*)"
8730 DATA DD,CB,DE,"SET 3,(IX+*)"
```

```
8740 DATA DD,CB,E6,"SET 4,(IX+*)"
8750 DATA DD,CB,EE,"SET 5,(IX+*)"
8760 DATA DD,CB,F6,"SET 6,(IX+*)"
8770 DATA DD,CB,FE,"SET 7,(IX+*)"
8780 DATA FD,CB,06,RLC (IY+*)
8790 DATA FD,CB,0E,RRC (IY+*)
8800 DATA FD,CB,16,RL (IY+*)
8810 DATA FD,CB,1E,RR (IY+*)
8820 DATA FD,CB,26,SLA (IY+*)
8830 DATA FD,CB,2E,SRA (IY+*)
8840 DATA FD,CB,3E,SRL (IY+*)
8850 DATA FD,CB,46,"BIT 0,(IY+*)"
8860 DATA FD,CB,4E,"BIT 1,(IY+*)"
8870 DATA FD,CB,56,"BIT 2,(IY+*)"
8880 DATA FD,CB,5E,"BIT 3,(IY+*)"
8890 DATA FD,CB,66,"BIT 4,(IY+*)"
8900 DATA FD,CB,6E,"BIT 5,(IY+*)"
8910 DATA FD,CB,76,"BIT 6,(IY+*)"
8920 DATA FD,CB,7E,"BIT 7,(IY+*)"
8930 DATA FD,CB,86,"RES 0,(IY+*)"
8940 DATA FD,CB,8E,"RES 1,(IY+*)"
8950 DATA FD,CB,96,"RES 2,(IY+*)"
8960 DATA FD,CB,9E,"RES 3,(IY+*)"
8970 DATA FD,CB,A6,"RES 4,(IY+*)"
8980 DATA FD,CB,AE,"RES 5,(IY+*)"
8990 DATA FD,CB,B6,"RES 6,(IY+*)"
9000 DATA FD,CB,BE,"RES 7,(IY+*)"
9010 DATA FD,CB,C6,"SET 0,(IY+*)"
9020 DATA FD,CB,CE,"SET 1,(IY+*)"
9030 DATA FD,CB,D6,"SET 2,(IY+*)"
9040 DATA FD,CB,DE,"SET 3,(IY+*)"
9050 DATA FD,CB,E6,"SET 4,(IY+*)"
9060 DATA FD,CB,EE,"SET 5,(IY+*)"
9070 DATA FD,CB,F6,"SET 6,(IY+*)"
9080 DATA FD,CB,FE,"SET 7,(IY+*)"
```

SECTION 2

REFERENCE

1

CONTINUED . . . FROM THE NOTEPAD MANUAL

There are a few features of BBC Basic which are essential for advanced programming on the Notepad, but which were not covered in the computer's manual. Therefore, they are fully documented here and you should read the following descriptions if you either want to use BBC Basic's in-built assembler, wish to have control over how Basic prints values to the screen, pass register values to machine code or integrate Basic files with Protext wordprocessor files.

*SPOOL & *EXEC

There are several very good reasons for wanting to use these two commands, the first being that you can actually type in your BBC Basic programs using the Notepad's built-in wordprocessor. So, for example, to enter a program called PROGRAM.BAS, press [Function][Word], followed by [Word] then enter the file name: PROGRAM.TXT, and start typing in the program.

When you have entered and checked the program press [Stop] to finish and then press [Function][B] to enter BBC Basic. If you have a Basic menu program installed – and you will know if you have – simply press [Stop] to exit it at this point and then type NEW. Otherwise you will see Basic's opening screen.

Now all you need to do is type:

```
*EXEC PROGRAM.TXT
```

and you will see each program line displayed on the screen in turn as if you had typed them all in very quickly yourself. When finished you can now save the program as a BBC Basic file by entering:

```
SAVE "PROGRAM.BAS"
```

The extension .BAS is recommended to:

❑ Distinguish it from a wordprocessor or other file

❑ Show visually that it is a Basic file.

What has happened is that you have typed all the program into the wordprocessor as an ASCII file (well, virtually). That is, a file containing all the keystrokes, just as you typed them. In the second part of the process you told the *EXEC (execute) command to execute all these keystrokes in order until the file is exhausted.

Now you are in a position to try out your program, so run it and see if it is working in the way it's supposed to. If not, you may want to make a couple of modifications here and now in BBC Basic and try again. If you're still not happy, it may need a more major change, so here's how to re-save the program as an ASCII file suitable for editing with a wordprocessor. Just type:

```
*SPOOL PROGRAM.TXT
LIST
```

Again you will see every line of the program scroll past as the ASCII file is created. Once done you need to type:

```
*SPOOL
```

on its own to turn the spooling off. You can now re-enter the wordprocessor by pressing [Function][Word], followed by [Calc] and then selecting the file PROGRAM.TXT for editing.

This technique may also be useful if you are using one of the larger programs from this book and don't have a RAM card. In this case Basic will try to store the entire program in what is called the Notepad's upper memory. This is a small area of RAM and you can't fit much in it.

However, the lower memory is about three times bigger but only wordprocessor documents are stored here. So a clever trick you can use is to keep your BBC Basic programs in ASCII form so that they stay in lower memory and then *EXEC them into BBC Basic as and when you need to use them.

Having said that, if you intend to use the Notepad to any extent, it is highly recommended that you buy a RAM card as one will make ALL the difference and improve your productivity no end. For details of one source of supply please refer to Appendix 6.

OPT n

This statement determines what output is produced on the screen when assembly language routines are processed by the interpreter. The OPT statement is followed by a number between 0 and 7, with the following results:

OPT 0	Assembler errors suppressed, no listing
OPT 1	Assembler errors suppressed, listing displayed
OPT 2	Assembler errors reported, no listing
OPT 3	Assembler errors reported, listing displayed
OPT 4	Assembler errors suppressed, no listing, assemble to O%
OPT 5	Assembler errors suppressed, listing displayed, assemble to O%
OPT 6	Assembler errors reported, no listing, assemble to O%
OPT 7	Assembler errors reported, listing displayed, assemble to O%

Usually you will only be concerned with OPTions 0 to 3 (4 to 7 are discussed later). So, taking the numbers 0 to 3, if they are shown in binary as follows, you will see that the right-hand bit defines whether a listing is to be displayed. If the bit is set then yes, otherwise no. The left-hand bit covers whether errors are reported or not. If the bit is set then yes, otherwise no:

```
0 = 00
1 = 01
2 = 10
3 = 11
```

The OPT statement can only occur inside the square brackets which signify the use of assembler directives. If labels are used in your assembly listings you will need to assemble them twice. This is known as a two-pass assembly, where the first pass assembles the instructions and the second pass, once the locations of all labels have been determined, adds all the label information (such as JP or JR addresses) to the correct instructions.

Therefore, an assembly procedure might look like the following, where the variable PASS is used in a FOR. . . NEXT loop to set the OPTion:

```
1000 DEF PROC assemble
1010 DIM A% 100
1020 FOR PASS=0 TO 3 STEP 3
1030 P%=A%
1040 [
1050 OPT PASS
1060 :
1070 \ Your code goes here. . .
1080 :
1090 ]
1100 NEXT
1110 ENDPROC
```

This will cause the assembler to list all the instructions as it assembles them. Once you are sure a program assembles correctly you may wish to replace line 1020 with

the following, which uses OPTions of 0 and 2, to suppress any assembly displays other than errors:

```
1020 FOR PASS=0 TO 2 STEP 2
```

If you become serious about writing assembler programs there will come a time when you'll need to assemble code to a memory address which is reserved or even occupied by the Basic program itself. Obviously this is a big problem, but with a simple solution. Luckily the Notepad allows you to perform Offset Assembly.

This is where a complete assembly goes ahead, as if it was assembled at the address pointed to by P% but, in fact, the assembled machine code is stored starting at the location pointed to by O%. In other words you can, for example, quite happily assemble code to run from &C000 onwards (as you might if you were writing a 16K system application to put in the first 16K of a RAM card), but actually only store the code in a *safe* area of ram at &6000-&9FFF.

A procedure to do just that might look like this:

```
1000 DEF PROC assemble
1010 FOR PASS=4 TO 7 STEP 3
1020 P%=&C000:O%=&6000
1030 [
1040 OPT PASS
1050 :
1060 \ Your code goes here. . .
1070 :
1080 ]
1090 NEXT
1100 ENDPROC
```

As well as assigning O% to point to an area of memory you are CERTAIN is free, note line 1010 where the variable PASS is assigned different values from before.

To explain: Going back to where OPT values 0 to 3 were examined in binary, if you look at the binary equivalents of the numbers 4 to 7 you will see that the right-hand bit pairs are still acting in the same way as before to control the display of listing and/or error reports, but there is a new third bit on the left. It is this one (if set) that tells the assembler to assemble at O% rather than P%:

```
4 = 100
5 = 101
6 = 110
7 = 111
```

Finally, once your assembly takes place without any errors, for a blank display during Offset assembly you could change line 1010 to:

```
1010 FOR PASS=4 TO 6 STEP 2
```

DEF

While in the assembler there are three undocumented commands you can use for allocating space. They are DEFB, DEFW and DEFM which, in turn, allocate a single byte, a two-byte word and a string of memory. These commands are used in place of the BBC Micro's EQUB, EQUW and EQUS keywords. Examples of acceptable command syntax include:

```
DEFB &FF
DEFB byte
DEFB ASC("A")
DEFW Z%
DEFW 0
DEFW &1234
DEFM "This is a test"
```

REGISTER VARIABLES

Just as you could directly pass values to the 6502's registers on the original BBC Micro by setting A%, F%, X% and Y%, so you can with Z80 BBC Basic.

The variables available are A%, B%, C%, D%, E%, F%, H% and L%, which directly correspond with the Z80's registers when treated as 8-bit single registers, rather than register pairs.

So, for example, you could print a single character to the screen (with the call TXTOUTPUT – &B833), using the following two commands:

```
A%=ASC("*")
CALL &B833
```

In addition, BBC Basic returns the contents of these registers to the variables when it returns from a CALL or USR command.

In fact, USR can be a handy replacement for CALL because it also returns the contents of the alternate registers H' and L'. It does this by returning a 32-bit value corresponding to H, L, H' and L', in that order. This is the way BBC Basic internally handles all 32-bit values.

@%

The NC100 manual mentions @% but gives you no details about using it. Using @% you can manipulate the way numbers are displayed by the PRINT and STR$ commands. With it you can control the field width, the total number of characters printed and the number of decimal places.

To use it you should consider @% as a four-byte number such as &01020304. The most significant byte (called B4) has a value of &01 in the above example, while B1 has a value of &04, and so on.

Byte B4

This is tested by the STR$ command to determine the format of strings created by it. If B4 = &01 then strings will be formatted using the settings in @%, otherwise @% will be ignored.

Byte B3

This selects the format type where &00 is General format (G), &01 is Exponent format (E) and &02 is Fixed format (F). In G format, numbers that are integers will be printed as integers. Numbers in the range 0.1 to 1 will be printed as 0.1 (and so on), while numbers less than 0.1 will be printed in exponential format.

Exponential format always prints numbers in scientific notation so, for example, 0.01 is printed as 1E-2, 100 is 1E2 and 1,234,567 is 1.234567E6.

Fixed format always prints numbers with a fixed number of decimal spaces. If a number cannot be fitted into a field it reverts to the G format. The decimal points are aligned vertically, making this format particularly useful for printing tables.

Byte B2

This controls the total number of digits printed by each format. By default B2 has a value of &09. In G format B2 states the maximum number of digits (between 1 and 9) that can be printed before reverting to E format

In E format B2 specifies the total number of digits (between 1 and 9) to be printed before and after the decimal point (not counting digits after the E).

In F format B2 specifies the number of digits (between 0 and 9) to follow the decimal point.

Byte B1

This sets the over all print field width and may have any value between &00 and &FF.

Here are some examples:

@%=	*&0000020A*	*&0000090A*	*&0002020A*	*&0001020A*
100=	*1E2*	*100*	*100.00*	*1.0E2*
10=	*10*	*10*	*10.00*	*1.0E1*
1=	*1*	*1*	*1.00*	*1.0E0*
0.1=	*0.1*	*0.1*	*0.10*	*1.0E-1*
0.01=	*1E-2*	*1E-2*	*0.01*	*1.0E-2*

You can omit any leading zeros if you prefer.

EDIT

There appears to be a slight problem with the Notepad's EDIT command when used incorrectly. For example if you, correctly, type:

```
EDIT 100
```

or:

```
EDIT100
```

you will get to edit line 100. But if you accidentally type any of:

```
EDIT #
EDIT#
EDIT [anything but a number]
```

(because, for example, your finger slipped and hit the # key which is next to [Return]), then the Notepad will take as many of the first lines of the program as it can and stick them all in Basic's editing buffer, ready for you to combine them. Actually you should receive an error message when this happens, but you can easily get out of this by pressing [Stop].

It appears to be offering you the equivalent of:

```
EDIT 10,50
```

which means edit lines 10 to 50 inclusive, placing them all in the edit buffer, but with a slight change of syntax making the command mean:

```
EDIT ,
```

shorthand for edit all the lines you can from the start of the program into the edit buffer.

Another problem you may encounter with EDIT is if you use Basic keyword shortcuts (where P. stands for PRINT, R. for RETURN, and so on), and enter the command E. (short for ENDPROC) at the start of a line while in AUTO mode. In this case Basic actually interprets this as the EDIT command and throws you out of AUTO mode and into the EDITor, again with as many lines crammed into the edit buffer as will fit.

The short and simple answer to this is to forget about using E. any more, use EN. instead and there will be no further confusion between EDIT and ENDPROC.

2

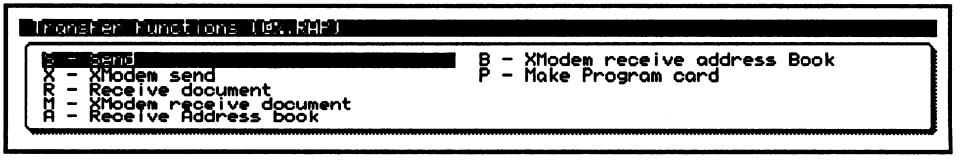
UNDOCUMENTED FEATURES

TRANSFERRING BBC BASIC PROGRAMS

There is what, at first sight, appears to be a problem with the NC100 which prevents you from backing up your BBC Basic programs to another computer, because by default they are not visible. That is because an unconfigured NC100 is set up NOT to display file dates and times, but in order for you to transfer them these MUST be visible. Apparently the default mode is the simple, *beginner's* mode, while this is the *advanced* mode.

Anyway, what you have to do to configure date displays is press [Function][X] to enter the front menu, then press [Menu] followed by the down cursor key twice and the right cursor key once, to set the display format of dd/mm/yy.

Now you're ready to transfer your files, so press [Function][L] to list the files, select the one to send using the cursor keys, press [Menu] and continue with your transfer as normal.

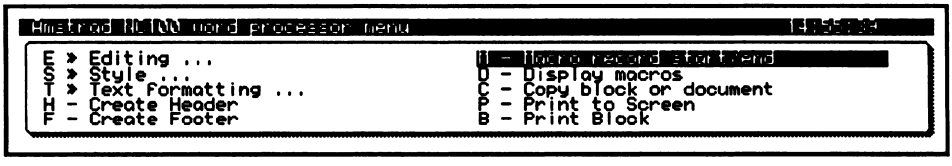


Ready to send a file

QUICK MACRO ASSIGNING

While editing documents you may know that you can assign a sequence of key strokes on to a single key press (known as a macro), by pressing [Menu], selecting [M] for Macros, pressing [Symbol][key] or [Symbol[Shift][key] (where key is any key between [A] and [Z]), then typing in the sequence of keys, and finally selecting [Menu] and [M] again to end the recording of the Macro.

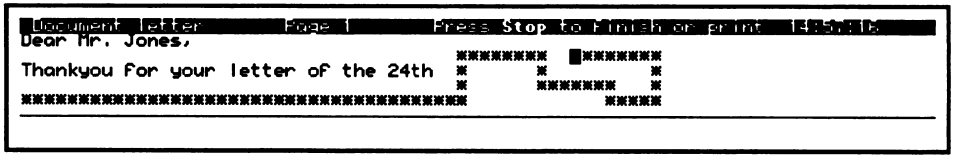
Thankfully there are two quick and easy short cuts available. The first is to press [Shift][Control][M] to initiate the recording, then press the key combination the macro is to be assigned to, the key macro sequence itself, and press [Shift][Control][M] a second time to end the recording. This feature is also available outside of the word processor but not in BBC Basic.



Defining a macro

LINE DRAWING CHARACTERS

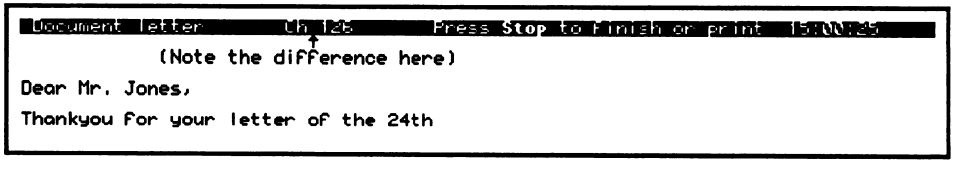
If you wish to change the character used for line drawing with the [Symbol] and cursor key (for example, to use an asterisk), press [Shift][Control[C] and then press a key such as * then, when you next draw lines, they will be made up of asterisks. When you have finished, as advised in the manual, to return to standard line drawing mode press [Shift][Control][L] and to switch between single and double line mode press [Shift][Control][D].



Line drawing using asterisks

PAGE DISPLAY MODE

To toggle Page Display mode on and off quickly , press [Shift][Control][P]. You will notice the display alternates between *Page n* and *Ch nnnnn*, where n is the page number and nnnnn is the offset of the current character from the document start. Also, all the triple-line page break markers between pages will not be displayed.



Toggling Page mode on and off

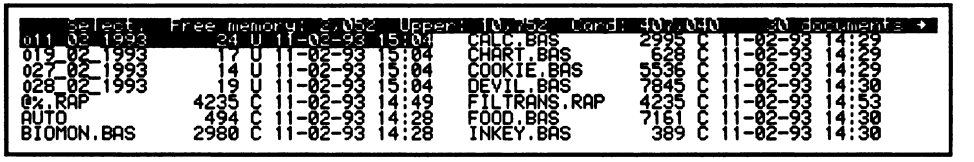
USING THE FILE SELECTOR

Wherever you need to load in or browse through files the File Selector function is called by the Notepad. This includes the *. (catalogue disk) command in BBC Basic and all listings in this book that use files.

However, the operation of the File Selector is not fully documented in the NC100 Notepad Manual so, in case you have not yet discovered the undocumented key presses, here's what else you can do with it:

Firstly, of course, you already know that the cursor keys move the highlight around, but if you press [Control] while doing so the [Up] and [Down] keys place the highlight on the first or last item in the current column, while the [Left] and [Right] keys respectively display the first and last set of (up to) 14 files (if there are more than 14).

In addition, you can display any hidden files (such as those created by the Diary program) by pressing [Shift][Control][H]. So, for example, if you have a diary entry set for the 1st of July, 1994, you would see the file name *o01_07_1994*. In fact a separate file is created for each and every day you enter in your diary. To make the files invisible again press [Shift][Control][H] a second time.



Four hidden appointment files

You will also be interested to hear that for some unknown reason, the undocumented macro assignment command [Shift][Control[M] also works from inside the File Selector (except when you are in BBC Basic).

Lastly, there's a quick and dirty way of deleting unwanted files. Simply move the highlight to your chosen file and press either [Del->] or [<-Del] (on the top-right of your keyboard). You will then be asked whether you want to delete the document. This is quicker than pressing [Menu] followed by [D], for delete, but do be careful.

PEEKING ABOUT

The last known undocumented key combination is [Shift][Control][Stop]. If you are editing a document and press it, the screen clears (except for the time) and you get a command line at the top-left. This appears to be the equivalent of Command Mode, found on all other implementations of Protext. So far three commands are known.

```
key a "This is a MACRO"
```

The command mode macro define command

There is KEY which is another means of defining macros. To use it type a command such as:

```
KEY A Testing
```

The other two commands are complementary. The first is DU which dumps any part of the Notepad's RAM or ROM (whatever is mapped in) to the screen. Simply type:

```
DU &4000
```

or whatever other address you are interested in, and all the data will be printed to the screen in hexadecimal and as Ascii characters. just press [Stop] to stop the screen from scrolling, any other key to resume printing, or [Stop] a second time to return to the command mode.

```
4010: 45 CC C3 CB C7 C3 EF C7 C3 DA CC C3 C1 F6 C3 66
4020: F3 C3 81 F3 C3 C1 F3 C3 33 F3 21 00 A2 36 0A 2C
4030: 36 09 2C AF 77 2C 20 FC 3E 37 32 FE A2 21 76 C0
4040: 22 EE A2 CD C1 F0 ED 53 E4 A2 22 DC A2 CD E7 C7
4050: C2 D3 CC CD 4E CC 42 42 43 20 42 41 53 49 43 20
4060: 28 4E 43 31 30 30 29 20 56 65 72 73 69 6F 6E 20
4070: 33 2E 31 30 0D 0A 28 43 29 20 43 6F 70 79 72 69
```

Dumping memory to the screen

Using the MM command you can map in other parts of the system memory to location &4000, which you can then view using the DU command. So, to page in the Basic ROM, you would type:

```
MM &5
```

The values you can use and the RAM or ROM that gets paged in are:

MM &00	ROM – Operating System
MM &01	ROM – Control code
MM &02	ROM – Calculator
MM &03	ROM – Address book
MM &04	ROM – Diary
MM &05	ROM – BBC Basic
MM &06	ROM – Protext
MM &07	ROM – Protext
MM &08	ROM – Spell Check Code
MM &09	ROM – Spell Check Code
MM &0A	ROM – Dictionary

MM &0B	ROM – Dictionary
MM &0C	ROM – Dictionary
MM &0D	ROM – Dictionary
MM &0E	ROM – Dictionary
MM &0F	ROM – Dictionary
MM &40	RAM – Internal RAM
MM &41	RAM – Internal RAM
MM &42	RAM – Internal RAM
MM &43	RAM – Internal RAM – Including Video RAM
MM &80-	RAM – Card RAM of up to 64 x 16K blocks (for 1Mb card)

See Chapter 3 for specific details on mapping the video RAM into main memory, and Chapter 4 for a detailed explanation of the NC100's memory map and how to map any parts of it into the core 64K area.

UNDOCUMENTED SELF-TEST

There is a POST (Power On Self-Test) built into the NC100 which performs a number of diagnostic tests. To call it up switch off your Notepad, hold down [Function] and [Symbol] and then switch it back on again, while still holding down these two keys.

You will then be able to go through the tests by pressing [Return]. The first test sets every pixel on the display, so that you can tell whether they are all functioning. Next all the characters in the character set are displayed. Next you see the value of the Memory/Battery/Status byte, the Real Time Clock and then the 12 internal ROMs are checked, followed by all the RAM.

Now you have an opportunity to test the keyboard to make sure all the keys are returning values. This may be useful if you are getting spurious key strokes. If you note a problem here it may be a good idea to try to suck out any material under the keys with a vacuum cleaner.

Next you get to test the parallel port by printing three lines of characters to a printer. If you have a laser printer you will not notice anything happening (even if this test is successful), until you send a Form Feed to it to eject the page (or press the Form Feed button on the printer).

Now it gets noisy because both the A and B sound channels are tested, so if you are on a train or something you'd be best advised to place your palm over the speaker grill first.

And finally, you come to the end of the tests where you get an opportunity to run through them again or return to the Notepad's front menu.

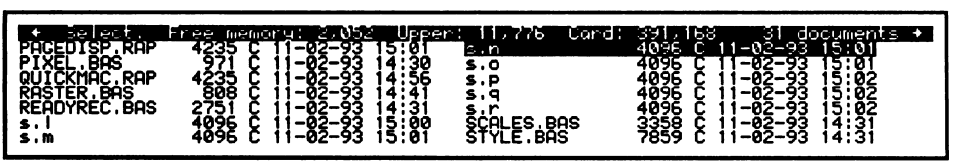
SAVING THE SCREEN

If you ever wondered how the screen dumps in this book and the NC100 manual were created, here's the answer. Each time you press [Shift][Control][S] the computer appears to lock up for a few seconds. In fact it's copying the entire contents of the screen to a file. The first file is saved with the file name *s.a*, while subsequent files are called *s.b*, *s.c* and so on, up to *s.z* and then through the ASCII set from *s.{* onwards.

To reset the file name to *s.a* again for your next screen grab you will need to enter Basic by pressing [Function][B] and then type:

```
?&B140=96
```

96 being the ASCII value one before the character a. The files created are 4,096 bytes long and consist of 64 rows of 64 bytes. Characters on the NC100 are six pixels wide and there are 80 of them on a line, making the screen 480 pixels wide. This equates to just 60 bytes, so the final four bytes at the end of each line are ignored. In all 256 bytes per screen are wasted but, for convenience and speed, all 4,096 bytes of screen memory are saved to the file.



The menu system showing several saved screens

Knowing this you can write your own programs to dump these screens to a printer or even convert them to industry standard PCX or TIFF files. But following are some example programs you may find useful for manipulating screen dumps both on the Notepad and on an IBM PC using Borland's Turbo C compiler.

GRAB2PCX.BAS

This BBC Basic program creates an exact PCX image of a screen dump. It uses the File Selector to choose files and only allows file names that begin S. to check that they are screen dump files. The PCX file created is totally uncompressed, takes up about 8K and should be readable by any program that can read the PCX format.

```
10 REM NC100 Screen grab to PCX converter
20 :
30 CLS:DIM A% 40,B% 128:PROCassemble
40 FOR J%=1 TO 128
50 READ B%?J%
```

```
   60 NEXT
   70 PRINT "GRAB2PCX.BAS: Press any key for the File Selector...";
:G$=GET$
   80 file$=FNselect:IF file$="" THEN CLS:END
   90 IF LEFT$(file$,2) <> "s." THEN PRINT "Not a screen grab..."':GOTO
70
  100 file2$=LEFT$(file$,1)+RIGHT$(file$,1)+".pcx"
  110 handlein=OPENIN(file$)
  120 handleout=OPENOUT(file2$)
  130 FOR J%=1 TO 128
  140 BPUT #handleout,B%?J%
  150 NEXT
  160 PRINT "GRAB2PCX.BAS: Creating file: ";file2$'
  170 PRINT "Processing line (of 64):"
  180 PRINT:PRINT "When finished this pogram will"
  190 PRINT "offer to convert another file."
  200 FOR K%=1 TO 64
  210 VDU 31,25,2:PRINT ;K%
  220 FOR J%=1 TO 60
  230 BPUT #handleout,&C1
  240 BPUT #handleout,BGET #handlein
  250 NEXT
  260 FOR N%=1 TO 4
  270 D%=BGET #handlein
  280 NEXT
  290 NEXT
  300 CLOSE #handlein
  310 CLOSE #handleout
  320 GOTO 80
  330 :
  340 REM PCX header block
  350 :
  360 DATA &A,5,1,1,0,0,0,0,&DF,1,&3F,0,0,0,0,0
  370 DATA 0,0,0,&FF,&FF,&FF,0,0,0,0,0,0,0,0,0,0
  380 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
  390 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
  400 DATA 0,1,&3C,0,1,0,0,0,0,0,0,0,0,0,0,0
  410 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
  420 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
  430 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
  440 :
  450 DEF FNselect
  460 CALL A%
  470 IF buffer?0 = 0 THEN CLS:=""
  480 R$=""
  490 FOR J%=0 TO 11
  500 IF buffer?J% THEN R$=R$+CHR$(buffer?J%) ELSE J%=12
  510 NEXT
  520 =R$
  530 :
  540 DEF PROCassemble
  550 FOR PASS=0 TO 2 STEP 2
  560 P%=A%
  570 [
  580 OPT PASS
  590 CALL &B8C3
  600 LD DE,buffer
  610 JR C,found
  620 LD A,0
  630 LD (DE),A
  640 RET
  650 .found
```

```
  660 LD B,12
  670 .loop
  680 LD A,(HL)
  690 LD (DE),A
  700 INC HL
  710 INC DE
  720 DJNZ loop
  730 RET
  740 .buffer
  750 ]
  760 NEXT
  770 ENDPROC
```

G2P-BORD.BAS

This program is identical to the one above except that it draws a border around the dump before saving it as a slightly larger PCX file with dimensions of 496x80. If you simply modify the first program rather than type all this one in, remember you will need to modify the data lines from line 730 to 800 too.

```
   10 REM NC100 Screen grab to PCX converter
   20 REM Version 2 - Creates a black border
   30 :
   40 CLS:DIM A% 40,B% 128:PROCassemble
   50 FOR J%=1 TO 128
   60 READ B%?J%
   70 NEXT
   80 PRINT "GRAB2PCX.BAS: Press any key for the File Selector...";
:G$=GET$
   90 file$=FNselect:IF file$="" THEN CLS:END
  100 IF LEFT$(file$,2) <> "s." THEN PRINT "Not a screen grab..."':GOTO
80
  110 file2$=LEFT$(file$,1)+RIGHT$(file$,1)+".pcx"
  120 handlein=OPENIN(file$)
  130 handleout=OPENOUT(file2$)
  140 FOR J%=1 TO 128
  150 BPUT #handleout,B%?J%
  160 NEXT
  170 PRINT "GRAB2PCX.BAS: Creating file: ";file2$'
  180 PRINT "Processing line (of 64):"
  190 PRINT:PRINT "When finished this pogram will"
  200 PRINT "offer to convert another file."
  210 FOR K%=1 TO 2
  220 FOR J=1 TO 62
  230 BPUT #handleout,&C1
  240 BPUT #handleout,&FF
  250 NEXT
  260 NEXT
  270 FOR K%=1 TO 6
  280 BPUT #handleout,&C1
  290 BPUT #handleout,&C0
  300 FOR J%=1 TO 60
  310 BPUT #handleout,&C1
  320 BPUT #handleout,&00
  330 NEXT
  340 BPUT #handleout,&C1
  350 BPUT #handleout,&03
```

```
360 NEXT
370 FOR K%=1 TO 64
380 VDU 31,25,2:PRINT ;K%
390 BPUT #handleout,&C1
400 BPUT #handleout,&C0
410 FOR J%=1 TO 60
420 BPUT #handleout,&C1
430 BPUT #handleout,BGET #handlein
440 NEXT
450 BPUT #handleout,&C1
460 BPUT #handleout,&03
470 FOR N%=1 TO 4
480 D%=BGET #handlein
490 NEXT
500 NEXT
510 FOR K%=1 TO 6
520 BPUT #handleout,&C1
530 BPUT #handleout,&C0
540 FOR J%=1 TO 60
550 BPUT #handleout,&C1
560 BPUT #handleout,&00
570 NEXT
580 BPUT #handleout,&C1
590 BPUT #handleout,&03
600 NEXT
610 FOR K%=1 TO 2
620 FOR J%=1 TO 62
630 BPUT #handleout,&C1
640 BPUT #handleout,&FF
650 NEXT
660 NEXT
670 CLOSE #handlein
680 CLOSE #handleout
690 GOTO 90
700 :
710 REM PCX header block
720 :
730 DATA &A,5,1,1,0,0,0,0,&EF,1,&4F,0,0,0,0,0
740 DATA 0,0,0,&FF,&FF,&FF,0,0,0,0,0,0,0,0,0,0
750 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
760 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
770 DATA 0,1,&3E,0,1,0,0,0,0,0,0,0,0,0,0
780 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
790 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
800 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
810 :
820 DEF FNselect
830 CALL A%
840 IF buffer?0 = 0 THEN CLS:=""
850 R$=""
860 FOR J%=0 TO 11
870 IF buffer?J% THEN R$=R$+CHR$(buffer?J%) ELSE J%=12
880 NEXT
890 =R$
900 :
910 DEF PROCassemble
920 FOR PASS=0 TO 2 STEP 2
930 P%=A%
940 [
950 OPT PASS
960 CALL &B8C3
970 LD DE,buffer
```

```
 980 JR C,found
 990 LD A,0
1000 LD (DE),A
1010 RET
1020 .found
1030 LD B,12
1040 .loop
1050 LD A,(HL)
1060 LD (DE),A
1070 INC HL
1080 INC DE
1090 DJNZ loop
1100 RET
1110 .buffer
1120 ]
1130 NEXT
1140 ENDPROC
```

GRABDISP.BAS

To complement the previous program this one will display an S. screen grab so you can determine whether it has saved correctly and is what you want before converting to PCX. Use the cursor keys to select a file and press [Return] to view it. Non-screen dumps will be displayed as garbage.

```
  10 ON ERROR GOTO 90
  20 VDU 26:CLS:DIM Z% &80:PROCassemble
  30 PRINT "GRABDISP"'
  40 PRINT "Press a key to select a screen grab to display...":G$=GET$
  50 CALL getfile:IF ?filename=0 THEN GOTO 90
  60 CALL scrn_from_disk:G$=GET$
  70 GOTO 50
  80 :
  90 ON ERROR GOTO 110
 100 VDU 26:CLS:IF ERR=17 THEN CHAIN "AUTO"
 110 REPORT:PRINT" at line ";ERL
 120 PRINT:PRINT"Press [Function][X] for Notepad Main Menu"
 130 END
 140 :
 150 DEF PROCassemble
 160 fopenin=&B8A2
 170 finblock=&B896
 180 fclose=&B890
 190 :
 200 FOR PASS = 0 TO 2 STEP 2
 210 P%=Z%
 220 [
 230 OPT PASS
 240 :
 250 .scrn_from_disk
 260 :
 270 LD HL,filename
 280 CALL fopenin
 290 JR C,fromI
 300 LD HL,flag
 310 LD (HL),0
 320 RET
```

```
330 :
340 .from1
350 :
360 LD HL,&8000
370 LD BC,&1000
380 CALL finblock
390 CALL fclose
400 CALL map_scrn_in
410 LD HL,&8000
420 LD DE,&F000
430 LD BC,&1000
440 LDIR
450 CALL map_scrn_out
460 LD HL,flag
470 LD (HL),1
480 RET
490 :
500 .map_scrn_in
510 :
520 LD A,(&B003)
530 LD (state),A
540 LD A,67
550 LD (&B003),A
560 OUT (&13),A
570 RET
580 :
590 .map_scrn_out
600 :
610 LD A,(state)
620 LD (&B003),A
630 OUT (&13),A
640 RET
650 :
660 .flag
670 :
680 DEFB 0
690 :
700 .state
710 :
720 DEFB 0
730 :
740 .getfile
750 :
760 CALL &B8C3
770 LD DE,filename
780 JR C,found
790 LD A,0
800 LD (DE),A
810 RET
820 .found
830 LD B,12
840 .loop
850 LD A,(HL)
860 LD (DE),A
870 INC HL
880 INC DE
890 DJNZ loop
900 RET
910 :
920 .filename
930 :
940 ]
```

```
  950 NEXT
  960 ENDPROC
```

NC2PCX.C

This IBM-compatible Borland Turbo C program is identical to the BBC Basic program GRAB2PCX.BAS except that, being compiled and running on a PC, it is extremely fast.

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>

char data[128]=
{
   0x0a,5,1,1,0,0,0,0,0xdf,1,0x3f,0,0,0,0,0,
   0,0,0,0xff,0xff,0xff,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,1,0x3c,0,1,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
};

main(argc,argv)
int argc;
char *argv[];
{
   FILE *fpin,*fpout;
   int j,k,byte;

   if (argc<3)
   {
      printf("Type: NCPCX filename filename");
      exit(0);
   }

   fpin=fopen(argv[1],"rb");

   if (fpin == NULL)
   {
      printf("\nFile %s not found.",argv[1]);
      exit(0);
   }

   fpout=fopen(argv[2],"wb");

   if (fpout == NULL)
   {
      printf("\nCannot create file %s.",argv[2]);
      exit(0);
   }

   for (j=0 ; j<128 ; ++j)
   {
```

```
      fputc(data[j],fpout);
   }

   for (k=0 ; k<64 ; ++k)
   {
      for (j=0 ; j<60 ; ++j)
      {
         byte=fgetc(fpin);
         fputc(0xc1,fpout);
         fputc(byte,fpout);
      }
      fgetc(fpin); fgetc(fpin); fgetc(fpin); fgetc(fpin);
   }
   fcloseall();
}
```

NC2PCXB.C

This IBM-compatible Borland Turbo C program is identical to the BBC Basic program G2P-BORD.BAS except that, being compiled and running on a PC, it is also extremely fast.

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>

char data[128]=
{
   0x0a,5,1,1,0,0,0,0,0xef,1,0x4f,0,0,0,0,0,
   0,0,0,0xff,0xff,0xff,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,1,0x3e,0,1,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
};

main(argc,argv)
int argc;
char *argv[];
{
   FILE *fpin,*fpout;
   int j,k,byte;

   if (argc<3)
   {
      printf("Type: NCPCX filename filename");
      exit(0);
   }

   fpin=fopen(argv[1],"rb");

   if (fpin == NULL)
   {
      printf("\nFile %s not found.",argv[1]);
      exit(0);
```

```
}

fpout=fopen(argv[2],"wb");

if (fpout == NULL)
{
   printf("\nCannot create file %s.",argv[2]);
   exit(0);
}

for (j=0 ; j<128 ; ++j)
{
   fputc(data[j],fpout);
}

for (k=0 ; k<2 ; ++k)
{
   for (j=0 ; j<62 ; ++j)
   {
      fputc(0xc1,fpout);
      fputc(0xff,fpout);
   }
}

for (k=0 ; k<6 ; ++k)
{
   fputc(0xc1,fpout);
   fputc(0xc0,fpout);

   for (j=0 ; j<60 ; ++j)
   {
      fputc(0xc1,fpout);
      fputc(0x00,fpout);
   }

   fputc(0xc1,fpout);
   fputc(0x03,fpout);
}

for (k=0 ; k<64 ; ++k)
{
   fputc(0xc1,fpout);
   fputc(0xc0,fpout);

   for (j=0 ; j<60 ; ++j)
   {
      byte=fgetc(fpin);
      fputc(0xc1,fpout);
      fputc(byte,fpout);
   }

   fputc(0xc1,fpout);
   fputc(0x03,fpout);

   fgetc(fpin); fgetc(fpin); fgetc(fpin); fgetc(fpin);
}

for (k=0 ; k<6 ; ++k)
{
   fputc(0xc1,fpout);
```

```
      fputc(0xc0,fpout);

      for (j=0 ; j<60 ; ++j)
      {
         fputc(0xc1,fpout);
         fputc(0x00,fpout);
      }

      fputc(0xc1,fpout);
      fputc(0x03,fpout);
   }

   for (k=0 ; k<2 ; ++k)
   {
      for (j=0 ; j<62 ; ++j)
      {
         fputc(0xc1,fpout);
         fputc(0xff,fpout);
      }
   }

   fcloseall();
}
```

PCX2NC.C

This final Turbo C program is for restoring non-bordered PCXs back to the original format, as saved by the Notepad. This is in case you may wish to then transfer one back to your Notepad or, perhaps, convert it to the bordered PCX format. It's also useful if you happen to have deleted your original S. files.

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>

char data[128]=
{
   0x0a,5,1,1,0,0,0,0,0xdf,1,0x3f,0,0,0,0,0,
   0,0,0,0xff,0xff,0xff,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,1,0x3c,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
};

main(argc,argv)
int argc;
char *argv[];
{
   FILE *fpin,*fpout;
   int n,byte,len,val,count,offset;

   if (argc<3)
   {
```

```
      printf("Type: NCPCX filename filename");
      exit(0);
   }

   fpin=fopen(argv[1],"rb");

   if (fpin == NULL)
   {
      printf("\nFile %s not found.",argv[1]);
      exit(0);
   }

   fpout=fopen(argv[2],"wb");

   if (fpout == NULL)
   {
      printf("\nCannot create file %s.",argv[2]);
      exit(0);
   }

   fseek(fpin,128L,0);

   offset=0;

   for (count=0 ; count<3840 ; )
   {
      byte=fgetc(fpin);

      if ((byte & 0xc0) == 0xc0)
      {
         len=byte & 0x3f;
         val=fgetc(fpin);

         for (n=0 ; n<len ; ++n)
         {
            fputc(val,fpout);
            ++offset;
            ++count;
         }
      }
      else
      {
         fputc(byte,fpout);
         ++offset;
         ++count;
      }

      if (offset == 60)
      {
         offset=0;
         fputc(0xff,fpout); fputc(0xff,fpout);
         fputc(0xff,fpout); fputc(0xff,fpout);
      }
   }
   fcloseall();
}
```

3

WRITING EXTERNAL PROGRAMS

You can create your own similar applications

The simplest and safest way to develop for the Notepad is to get a PCMCIA drive for your PC and write a binary image direct to the card using that. If this isn't possible then small programs (up to 16K) can be developed by transferring the binary card image into the Notepad using Xmodem from the PC. Then use the *Make program card* feature in the File transfer menu to write that file on to a newly formatted PCMCIA RAM card.

You can also use the BBC Basic assembler's Offset assembly facility which will allow you to write code that is assembled as if it were at &C000 but actually places the code elsewhere, so that you can save it to a RAM card and run it. Note that you will probably need two cards for this: the first for your source code and other files, the second for testing your application. See Chapter 1 for full details on using offset assembly.

However you create it, to run the resultant code, you just press [Function][X] and the first 16K page of the RAM card will be switched to the Z80 memory map at &C000-&FFFF. A Check is then made that location &C200 holds the ASCII text *NC100PRG* and also that locations &C210-&C212 contain a long jump to &C220.

All being well, the Z80 will start executing code at &C210 so that, once you have control, you can take over completely if you wish (driving all hardware functions

directly). Most people will probably want to cooperate with the in-built firmware as it provides most of the routines that you could want anyway.

But you MUST follow a few important rules in order for your application to be recognised by the system and to interact correctly with it. First of all the program's origin MUST be &C210, and the first instruction must be a JP &C220.

From &C213 to &C21F you need to store the name of your application, followed by a zero byte. The total length of the name including the zero terminator may not be longer than 13 characters. Here's an illustration:

```
ORG &C200 DB "NC100PRG"

ORG &C210 JP start DB "PROGRAM NAME",0

ORG &C220 [Your program goes here]
```

The available workspace is from &A000 to &A3FF, but it is shared with other programs so never assume certain data is left where you put it if another application has been executing in the meantime. You can also use &A800 to &AFFF, but beware that this well be overwritten if the File Selector is called.

For interaction with the rest of the system, add-on applications MUST handle Yellow events. For example: either exit when [Stop] is pressed or check for a yellow event with KMGETYELLOW, and return if the carry flag is set.

Serious developers may be interested in contacting Ranger Computers on 0604 589200 as they can produce a device that looks like RAM to a PC but ends in a PCMCIA header plug that connects directly to the Notepad's card slot and the PC RAM appears as card RAM to the Notepad.

Another alternative is the excellent shareware cross assembler, TASM, which can assemble code for 10 different microprocessors, including the Z80. You should be able to get hold of a copy from your favourite shareware library, or you can download it from the Assembler library in the IBMPRO forum on CompuServe, and it may be available on other bulletin boards.

In conjunction with the Lapcat lead and software available from Arnor (See Appendix 6), you will then be able to assemble object files and transfer them directly to a RAM card in your Notepad. But make sure the card is freshly formatted before doing so, to ensure that the code is stored in the first 16K of RAM.

USING THE NOTEPAD'S LCD DISPLAY

Because the Z80 is restricted to addressing an area of no more than 64K, if you want any more RAM or ROM you have to page it in to order. This way you can have 16K

blocks of memory containing code or data for different purposes and then use a bank switching device to map blocks in when they are needed.

And, of course, this is what the NC100 does. In fact it has 256K ROM and 64K RAM of memory built in so it uses very sophisticated memory management techniques to page everything into the small 64K area at exactly the right times. Although the screen is only 4K long, that is still too precious an amount of memory to give up permanently, so even the screen ram is only paged in when it has to be written to or read from (although the LCD display does have permanent access to it, in order to keep it visible all the time).

For technical reasons only 16K chunks of memory can be switched in at any one time, so when you select the screen you get an extra 12K mapped in containing other data. All you need be concerned about though, is the top 4K area as this is where you will always find the screen.

The RAM containing the screen can be mapped into any of the four 16K locations in the 64K memory map by issuing the correct OUT statement to ports &10-&13, Like this:

```
LD A,67 ; This signifies the screen ram block
OUT (&10),A ; for &0000, or:
OUT (&11),A ; for &4000, or:
OUT (&12),A ; for &8000, or:
OUT (&13),A ; for &C000
```

However, it is strongly recommended that BBC Basic programs should use the final one of these calls in order to map the screen in at &C000 so that BBC Basic's own RAM area is not affected. In fact, as it is only 4K long, the screen is mapped in at &F000 and, because the other 12K is reserved, make sure you do sufficient bounds checking so that screen writes don't stray into it.

In addition, the bank switching registers are write-only. Therefore, for the NC100 to know its current status at any time it must refer to its own copy of the various settings. These are held at locations &B000-&B003. So, before you write to any of the ports you must first read the value from the relevant location and store a copy (perhaps by pushing it on the stack), then write your new value back to this location, and only then write the value to the port, like this:

```
LD A,(&B003)
PUSH AF
LD A,67
LD (&B003),A
OUT (&13),A
```

To put a screen back from where you got it, pop the value off the stack (or get it from where you stored it) and write it to the location before also writing it to the bank switching port, like this:

```
POP AF
LD (&B003),A
OUT (&13),A
RET
```

Courageous users may wish to experiment with using values other than 67 and mapping the NC100's various RAM/ROM blocks somewhere in memory (such as at &4000, so that you can still use the screen) in order to have a peek at how the computer is organised. But this is not recommended for the faint-hearted! You can also examine the RAMs and ROMs using the MM and DU commands available from Protext's command mode, described in Chapter 2.

Anyway, down to the nitty-gritty. Here's some example code for directly accessing the video ram on a pixel level. In order to access a given X,Y location on the screen you have to perform the following steps:

- ❑ Save the old memory block
- ❑ Map in the video memory
- ❑ Multiply the Y pixel address (0 to 63) by 64
- ❑ Add on the X byte address (0 to 60)
- ❑ OR with &70 to convert to address between &F000 and &FFFF
- ❑ Read from or write to the eight pixels pointed to
- ❑ Restore the old memory block

In the following example HL is the Y pixel. H is always zero and L has a value between 0 and 63, inclusive, while DE is the X offset which ranges from 0 to 479. This program will display a single pixel towards the top right-hand side of the display:

```
10 CLS
20 DIM A% &100
30 PROCassemble
40 CLS:CALL A%
50 END
60 DEF PROCassemble
70 FOR PASS=0 TO 3 STEP 3
80 P%=A%
90 [
100 OPT PASS
110 :
120 . start
130 :
140 LD HL,3
150 LD DE,377
160 :
170 ; Save memory, and set video memory
180 :
190 LD A,(&B003)
200 PUSH AF
```

```
210 LD A,67
220 LD (&F003),A
230 OUT (&13),A
240 :
250 ; Multiply HL by 64 (bytes per pixel line)
260 :
270 LD H,0
280 ADD HL,HL
290 ADD HL,HL
300 ADD HL,HL
310 ADD HL,HL
320 ADD HL,HL
330 ADD HL,HL
340 :
350 ; Determine which bit to act on
360 :
370 LD A,E
380 AND 7
390 LD B,A
400 LD A,0
410 SCF
420 :
430 .power
440 :
450 RRA
460 DJNZ power
470 PUSH AF
480 :
490 ; Divide DE by 8 to get pixel address
500 :
510 SRL D
520 RR E
530 SRL D
540 RR E
550 SRL D
560 RR E
570 :
580 ; Add on X address to start of pixel line
590 :
600 ADD HL,DE
610 :
620 ; Convert to range &F000-&FFFF
630 :
640 LD A,H
650 OR &F0
660 LD H,A
670 :
680 ; HL now points at 8 bits of screen memory, so write pixel
690 :
700 POP AF
710 LD B,(HL)
720 OR B
730 LD (HL),A
740 :
750 ; Now clean up
760 :
770 POP AF
780 LD (&B003),A
790 OUT (&13),A
800 RET
```

```
810 ]
820 NEXT
830 ENDPROC
```

Most of this program is pretty self-explanatory, but there are two bits that need further discussion. Take a look at lines 370-470. Here, the E register is copied to A and then ANDed with 7. This leaves it with only the three right-most bits (a number between 0 and 7).

The contents of A are then transferred to B, A is zeroed, the carry flag is set, and the loop called *power* rotates A right the number of times stored in B. This moves the pixel to be set to the correct location. The RRA command moves all the bits in A to the right, at the same time placing the contents of the carry flag in bit 7 (on the left), and the contents of bit 0 into the carry flag.

The value in A is then stored by PUSHing AF on to the stack where it is later retrieved at line 700 and ORed with the contents of the location pointed to by HL. If you wanted to clear the pixel you would first issue an XOR &FF and then AND with B instead.

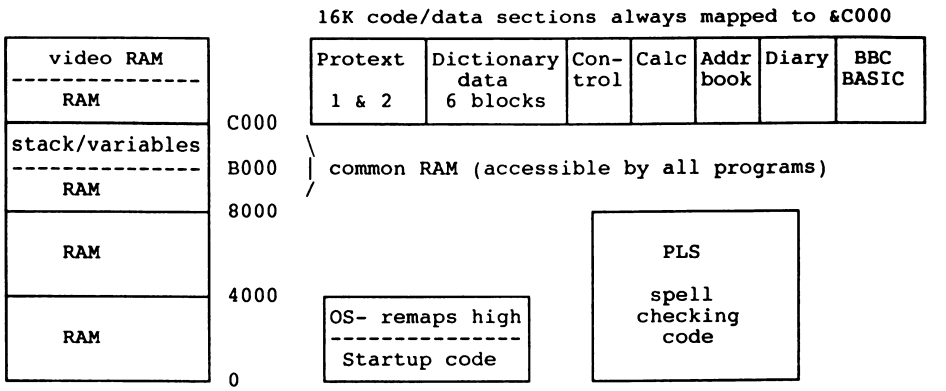
```
16D4 F5           PUSH AF
16D5 3E 43        LD A,67
16D7 32 03 B0     LD (&B003),A
16DA D3 13        OUT (&13),A
16DC              ; Multiply HL by 64 (bytes per pixel line)

Escape at line 270
*
```

Assembling the example pixel setting program

4

THE NOTEPAD'S INPUT/OUTPUT PORTS



The NC100's memory map

You will not often need to make use of the Input/Output ports on the Notepad, but all the details you need are here for when you do, including mapping the video RAM into the core 64K of RAM, determining the battery and memory card status, communications configuration and so on.

&00	**WRITE ONLY**	**START ADDRESS OF DISPLAY MEMORY**
bits 0-3	Not used	
bit 4	Address line	&0C
bit 5	Address line	&0D
bit 6	Address line	&0E
bit 7	Address line	&0F

On reset this is set to 0.

The display memory for the 8-line NC computers consists of a block of 4096 bytes where the first byte defines the state of the pixels in the top left-hand corner of the screen.

A 1 bit set means the pixel is set to black. The first byte controls the first eight dots with bit 7 controlling the bit on the left. The next 59 bytes complete the first raster line of 480 dots.

The bytes which define the second raster line start at byte 64 to make the hardware simpler so bytes 60, 61, 62 and 63 are wasted. There are then another 64 bytes (with the last four unused) which define the second raster line and so on straight down the screen.

So the layout is like this:

	BYTE 00	*BYTE 01*	*BYTE 02*
Bit No.	76543210	76543210	76543210
Pixel No.	00000000	00111111	11112222
	01234567	89012345	67890123
Character No.	0-----0-	----0---	--0-----
(for 1 row)	0-----1-	----2---	--3-----

This continues on for subsequent lines. For example, the second line is the range of bytes 64-127, and line three is 128-191, and so on. You may also have noticed that displayed characters are only six pixels wide, so slightly unusual routines are required to read and write them, although you can use the Jump Block calls to do this for you.

&10-&13 READ/WRITE: MEMORY MANAGEMENT CONTROL

These addresses control the NC 100's bank switch capabilities. Writers of external applications will most like use them for accessing the display RAM for direct screen reading and writing. Port:

10	controls 0000-3FFF
11	controls 4000-7FFF
12	controls 8000-BFFF
13	controls C000-FFFF

On reset all are set to 0. For each address the byte written has the following meaning:

bits 0-5	determine address lines 14-19.
bit 6	selects internal RAM
bit 7	selects card RAM

If neither bit 6 or bit 7 are set then ROM is selected. Therefore:

&00 is the first 16K of ROM
&01 is the second 16K...
&40 is the first 16K of internal RAM,
&41 is the second 16K...
&80 is the first 16K of card RAM
&81 is the second 16K...

So, for example, if you want to switch the third 16K of internal RAM so the processor sees it at &4000-&7FFF you would output the value 42 to I/O address &11. 42 has bits 6 set to 1 and bit 7 to 0, while bits 0-5 are 00010b which is the third 16K of internal RAM.

Therefore, to switch the screen (which is the fourth 16K of internal RAM) into the fourth 16K of mapped RAM so that the processor sees it between &C000 and &FFFF, you would output the value &43 (67 decimal) to port &13.

Here is a broad overview of the NC100's layout and the values required to map each 16K block in to one of the four areas of memory:

&00 ROM - Operating System
&01 ROM - Control code
&02 ROM - Calculator
&03 ROM - Address book
&04 ROM - Diary
&05 ROM - BBC Basic
&06 ROM - Protext
&07 ROM - Protext
&08 ROM - Spell Check Code
&09 ROM - Spell Check Code
&0A ROM - Dictionary
&0B ROM - Dictionary
&0C ROM - Dictionary
&0D ROM - Dictionary
&0E ROM - Dictionary
&0F ROM - Dictionary
&40 RAM - Internal RAM
&41 RAM - Internal RAM
&42 RAM - Internal RAM
&43 RAM - Internal RAM - Including Video RAM
&80- RAM - Card RAM (up to 64 16K blocks)

&20 WRITE ONLY MEMORY CARD WAIT STATE CONTROL

bit 7 = 1 for wait states, 0 for no wait

On reset this is set to 1. The bit should be set if the card RAM/ROM is 200nS or slower.

&30 WRITE ONLY BAUD RATE

bits 0-2 set the baud rate as follows:
000 = 150
001 = 300
010 = 600
011 = 1200
100 = 2400
101 = 4800
110 = 9600
111 = 19200
bit 3 UART clock and reset: 1=off, 0=on
bit 4 uPD4711 line driver: 1=off, 0=on
bit 5 not used
bit 6 parallel interface Strobe signal
bit 7 select card register: 1=common, 0=attribute

On reset all data is set to 1. If programming the UART directly ensure that TxD clock is operating x16.

&40 WRITE ONLY PARALLEL INTERFACE DATA

The byte written here is latched into the parallel port output register. To print it you must then take the Strobe signal (I/O address 30 bit 6) low and then high again. If the printer sends ACK this may generate an IRQ if the mask bit is set in I/O address 60 - IRQ mask.

&50-&53 WRITE ONLY SOUND CHANNELS PERIOD CONTROL

&50 channel A period low
&51 channel A period high
&52 channel B period low
&53 channel B period high

On reset all data is set to &FF. The top bit in the high byte (&51 and &53) switches the respective sound generator on or off: 1=off, 0=on. The frequency generated is determined as:

$$\text{Frequency} = \frac{307,200}{\text{data}}$$

So if the data word programmed into &50 and &51 was &7800 (that is, &50=0, &51=78) then the frequency generated would be:

$$\text{Frequency} = \frac{307,200}{\&7800} = \frac{307,200}{30,720} = 10\text{Hz}$$

&60 WRITE ONLY INTERRUPT REQUEST MASK

bit 0 Rx Ready from UART
bit 1 Tx Ready from UART
bit 2 ACK from parallel interface
bit 3 Key Scan interrupt (every 10mS)
bits 4-7 Not used

On reset all bits are 0. For each bit: 1=allow that interrupt source to produce IRQs, 0=interrupt source is masked.

&70 WRITE ONLY POWER OFF CONTROL

bit 0 1 = no effect, 0 = power off
bits 1-7 Not Used

On reset this is set to 1.

&90 READ/WRITE IRQ STATUS

bit 0 Rx Ready interrupt
bit 1 Tx Ready interrupt
bit 2 ACK from parallel interface
bit 3 Key scan
bits 4-7 Not used

When an interrupt occurs this port should be read to determine its source. The bit will be set to 0 to identify the interrupting device. The interrupt can then be cleared by writing 0 to that bit.

&A0 READ ONLY MEMORY CARD/BATTERY STATUS

bit 0 Parallel interface ACK: 1 if ACK
bit 1 Parallel interface BUSY: 0 if busy
bit 2 Lithium battery: 1 if less than 2.7 Volts
bit 3 Alkaline batteries: 1 if less than 3.2 Volts. (Although tests show this may be nearer to 4.2 volts in practice).

bit 4 RAM card battery: 1 if battery is OK
bit 5 Mains Adaptor: 1 if less than 4 Volts
bit 6 Card write protected: 1 = yes, 0 = no
bit 7 Memory card present: 0 = yes, 1 = no

&B0-&B9 READ ONLY KEYBOARD DATA

Each key of the 64 on the keyboard will set a bit in one of these bytes while pressed.

The gate array scans the keyboard every 10mS and then generates an interrupt. The program should then read these 10 I/O locations to determine which key has been pushed. When I/O address &B9 is read the key scan interrupt is cleared automatically and the next scan cycle will start from &B0.

&C0 READ/WRITE UART CONTROL/DATA

&C0 UART data register
&C1 UART status/control register

The UART is the NEC uPD71051. Programmers are advised to study the data sheet for that chip for more information. The Serial interface requires that the uPD4711 line driver chip be turned on by writing a 0 to bit 4 of I/O address &30. While turned on, power consumption increases so this should only be done when necessary. Calling PADINITSERIAL (&B85A) first will ensure no bytes are lost when writing.

&D0 READ/WRITE REAL TIME CLOCK CHIP (TM8521)

&D0-&DC Data
&DD Control register
&DE Control register (Write only)
&DF Control register (Write only)

See the chip data sheet for more information.

5

THE JUMPBLOCK ENTRIES

The result of calling SELECTFILE (&B8C3)

Most of the following routines return with the carry flag set if successful and, unless otherwise stated, you should assume that AF is corrupt on return and that other registers are preserved.

Where you see *All registers preserved* this includes the flags, but NOT the alternate registers. In fact the alternate register contents can NEVER be assumed to be preserved as they are used as scratch registers in time-critical routines.

To use any one of these routines just load the registers as described and then call the relevant address. Although the running of a routine may involve a different ROM bank being switched in, this mechanism is invisible to the caller. So, for example, to print a capital A you could use the following (pretty useless, but explanatory) example:

```
 10 CLS
 20 txtoutput=&B833
 30 FOR pass=0 TO 3 STEP 3
 40 [
 50 OPT pass
 60 LD A,ASC("A")
 70 CALL txtoutput
 80 RET
 90 ]
100 NEXT
```

KEYBOARD FUNCTIONS

EDITBUF – &B800

Action:
A line editor with options. A zero-terminated string may be passed in buffer (HL). This will display the initial contents.

Entry conditions:

HL:	Pointer to input buffer
B:	Size of buffer (excluding terminating zero)
A:	Flags:
	bit 2 = 1 – Up and down cursor keys terminate input
	bit 3 = 1 – Input not echoed
	bit 4 = 1 – Delete trailing spaces
	bit 5 = 1 – Edit unless characters entered
	bit 6 = 1 – Dotty background (character 176)
	Other bits must be set to zero.

Exit conditions:

c=0 & z=1	[Stop] pressed
c=1 & z=1	Empty string input
c=1 & z=0	At least one character entered
HL	Preserved
BC	Last key token (or –1 if [Stop] used to terminate)

KMCHARRETURN – &B803

Action:
Returns a token to the keyboard buffer. This is useful for determining which token is due next without removing it from the buffer, by first reading it and then returning it.

Entry conditions:

BC	The token

Exit conditions:
All registers preserved

KMREADKBD – &B806

Action:
Gets a key token if there is one. It does not wait but checks put-back characters and expands macros. It also returns tick event tokens, if enabled.

Entry conditions:
None

Exit conditions:
c=1: BC=token (B=0 for simple character)
c=0: No key token available

KMSETEXPAND – &B809

Action:
Defines a macro string.

Entry conditions:
BC: Macro token (between 256 and 383)
HL: Points to new macro string (the first byte is the length, followed by the string, which need not be zero terminated)

Exit conditions:
c=1 Macro defined successfully
c=0 Insufficient room in the buffer (The buffer size is user configurable)

KMSETTICKCOUNT – &B80C

Action:
Enables the ticker event. There are 100 ticks per second. When a ticker event occurs a special value of 941 is returned by KMREADKBD (&B806).

Entry conditions:
HL: Number of ticks before first event
DE: Number of ticks between events

Exit conditions:
All registers preserved

KMWAITKBD – &B80F

Action:
Waits for a key token. It uses KMREADKBD (B806) and checks put-back characters and expands macros. It also returns tick event tokens if enabled.

Entry conditions:
None

Exit conditions:
c=1: BC=Token (B=0 for a simple character)

READBUF – &B812

Action:
A line editor. See also EDITBUF (&B800).

Entry conditions:

HL:	Pointer to input buffer (empty)
B:	Size of buffer (excluding terminating zero)

Exit conditions:

c=0 & z=1:	[Stop] pressed
c=1 & z=1:	Empty string input
c=1 & z=0:	At least one character entered
BC:	Last key token (or –1 if [Stop] used to terminate)
HL:	Preserved

TESTESCAPE – &B815

Action:
Tests whether an Escape key has been pressed (either [Stop] or [Function]). It waits for a key if one is found in the keyboard buffer.

Entry conditions:
None

Exit conditions:

c=1:	No Escape key in buffer, or Escape key in buffer but [Stop] not pressed
c=0:	Escape key in buffer and [Stop] then pressed
A:	Preserved

SCREEN DISPLAY FUNCTIONS

COL1 – &B818

Action:

If the cursor is at the start of a line it does nothing, otherwise it moves the cursor to the start of next line (within the current window).

Entry conditions:
None

Exit conditions:
All registers preserved

COL1TEXT – &B81B

Action:
The same as TEXTOUT (&B81E), but it calls COL1 (&B818) first.

Entry conditions:
None

Exit conditions:
All registers preserved

TEXTOUT – &B81E

Action:
Displays a string.

Entry conditions:
HL: Pointer to a zero-terminated string.
WARNING – HL must not point into an upper ROM!

Exit conditions:
All registers preserved

TEXTOUTCOUNT – &B821

Action:
The same as TEXTOUT (&B81E), but returns a character count in B.

Entry conditions:
None

Exit conditions:
B: Character count

TXTCLEARWINDOW – &B824

Action:
Clears the current window and moves the cursor to the top-left of it.

Entry conditions:
None

Exit conditions:
All registers preserved

TXTCUROFF – &B827

Action:
Removes the cursor from the screen.

Entry conditions:
None

Exit conditions:
All registers preserved

TXTCURON – &B82A

Action:
Displays the cursor on the screen.

Entry conditions:
None

Exit conditions:
All registers preserved

TXTGETCURSOR – &B82D

Action:
Returns the cursor position.

Entry conditions:
None

Exit conditions:

H:	Column (between 0 and 79)
L:	Row (between 0 and 7)

TXTGETWINDOW – &B830

Action:
Returns the window coordinates.

Entry conditions:
None

Exit conditions:

H:	Left column (between 0 and 79)
L:	Top row (between 0 and 7)
D:	Right column (between 0 and 79)
E:	Bottom row (between 0 and 7)
c=0:	Window is whole screen
c=1:	A smaller window has been created

TXTOUTPUT – &B833

Action:
Displays a character or acts on a control code.

Entry conditions:

A:	character: A = 7: Beep A = 10: Line Feed A = 13: Carriage Return All other values are displayed as a character (the same as the PC character set)

Exit conditions:
All registers preserved

TXTSETCURSOR – &B836

Action:
Moves the cursor to a new position.

Entry conditions:

H:	Column (between 0 and 79)
L:	Row (between 0 and 7)

Exit conditions:
All registers preserved

TXTSETWINDOW – &B839

Action:
Defines a new window.

Entry conditions:

H:	Left column (between 0 and 79)
L:	Top row (between 0 and 7)
D:	Right column (between 0 and 79)
E:	Bottom row (between 0 and 7)

Exit conditions:
All registers preserved

TXTWRCHAR – &B83C

Action:
Displays a character. Control codes are also displayed as characters rather than being acted upon.

Entry conditions:

A:	Character. All values are displayed as per the PC character set.

Exit conditions:
All registers preserved

TXTBOLDOFF – &B83F

Action:
Resets the bold attribute. The next time text is written to the screen it will be without this attribute.

Entry conditions:
None

Exit conditions:
All registers preserved

TXTBOLDON – &B842

Action:
Sets the bold attribute. The next time text is written to the screen it will be with this attribute.

Entry conditions:
None

Exit conditions:
All registers preserved

TXTINVERSEOFF – &B845

Action:
Resets the inverse attribute. The next time text is written to the screen it will be without this attribute.

Entry conditions:
None

Exit conditions:
All registers preserved

TXTINVERSEON – &B848

Action:
Sets the inverse attribute. The next time text is written to the screen it will be with this attribute.

Entry conditions:
None

Exit conditions:
All registers preserved

TXTUNDERLINEOFF – &B84B

Action:
Resets the underline attribute. The next time text is written to the screen it will be without this attribute.

Entry conditions:
None

Exit conditions:
All registers preserved

TXTUNDERLINEON – &B84E

Action:
Sets the underline attribute. The next time text is written to the screen it will be with this attribute.

Entry conditions:
None

Exit conditions:
All registers preserved

PARALLEL AND SERIAL PORT FUNCTIONS

MCPRINTCHAR – &B851

Action:
Sends a character to the printer.

Entry conditions:
A: Character

Exit conditions:
c=1: Successful
c=0: Not sent
A: Preserved

MCREADYPRINTER – &B854

Action:
Tests whether the printer is ready.

Entry conditions:
None

Exit conditions:
c=0: Busy
c=1: Ready
A: Preserved

MCSETPRINTER – &B857

Action:
Sets the printer type to be used by MCPRINTCHAR (&B851) and MCREADYPRINTER (&B854).

Entry conditions:
A: Printer type:
0 = Parallel
1 = Serial

Exit conditions:
All registers preserved

PADINITSERIAL – &B85A

Action:
Initialises the serial port using the global configured settings and turns on the UART and 4711. To prolong battery life, do not call this until needed.

Entry conditions:
None

Exit conditions:
All registers preserved

PADINSERIAL – &B85D

Action:
Reads a character from the serial port.

Entry conditions:
None

Exit conditions:

c=1:	Successful, A=character
c=0:	No character read

PADOUTPARALLEL – &B860

Action:
Sends a character to the parallel port.

Entry conditions:

A:	Character

Exit conditions:

c=1:	Successful
c=0:	Not sent
A:	Preserved

PADOUTSERIAL – &B863

Action:
Sends a character to the serial port.

Entry conditions:

A:	Character

Exit conditions:
c=1: Successful
c=0: Not sent
A: Preserved

PADREADYPARALLEL – &B866

Action:
Tests whether the parallel port is ready.

Entry conditions:
None

Exit conditions:
c=0: Busy
c=1: Ready
A: Preserved

PADREADYSERIAL – &B869

Action:
Tests whether the serial port is ready.

Entry conditions:
None

Exit conditions:
c=0: Busy
c=1: Ready
A: Preserved

PADRESETSERIAL – &B86C

Action:
Turns off the UART and 4711. To prolong battery life call this as soon as you have finished using the serial port.

Entry conditions:

None

Exit conditions:
All registers preserved

PADSERIALWAITING – &B86F

Action:
Tests whether there is a character waiting to be read from the serial port.

Entry conditions:
None

Exit conditions:
c=1: Character waiting
c=0: No character waiting

CLOCK FUNCTIONS

PADGETTICKER – &B872

Action:
Returns the address of a four-byte 100Hz ticker.

Entry conditions:
None

Exit conditions:
HL: The address of the least significant byte (first of four)

PADGETTIME – &B875

Action:
Reads the time and date from the Real Time Clock.

Entry conditions:
HL: Points to a seven-byte buffer to use:

Exit conditions:
HL: Preserved. The buffer contains seven bytes of data:
byte 0 = year (low)
byte 1 = year (high)
byte 2 = month
byte 3 = date
byte 4 = hour
byte 5 = minute
byte 6 = second

PADSETALARM – &B878

Action:
Sets the ALARM date and time (within the next month).

Entry conditions:
HL: Points to a three-byte data area:
byte 0 = date
byte 1 = hour
byte 2 = minute

Exit conditions:
All registers preserved

PADSETTIME – &B87B

Action:
Sets the Real Time Clock date and time.

Entry conditions:
HL: Points to a seven-byte data area:
byte 0 = year (low)
byte 1 = year (high)
byte 2 = month
byte 3 = date
byte 4 = hour
byte 5 = minute
byte 6 = second

Exit conditions:
All registers preserved

MEMORY ALLOCATION FUNCTIONS

HEAPADDRESS – &B87E

Action:
Obtains the address of a memory block for a given memory handle.

Entry conditions:
DE: Memory handle

Exit conditions:
HL: Pointer to memory block

HEAPALLOC – &B881

Action:
Allocates a block of memory from the heap.

Entry conditions:
DE: Number of bytes to allocate

Exit conditions:
HL=0: 0 if failed
HL<>0: Memory handle in the range 1 – 63

NOTE: HEAPADDRESS (&B87E) must be used to get a pointer to the memory block Unless the block is locked with HEAPLOCK (&B887). HEAPADDRESS (&B87E) must be called each time the memory block is used as it may have moved!

HEAPFREE – &B884

Action:
Frees a block of memory.

Entry conditions:
DE: Memory handle, returned by HEAPALLOC (B881) or HEAPREALLOC (B88D)

Exit conditions:
HL: Preserved
BC: Preserved

NOTE: The memory handle passed must be a valid handle returned by HEAPALLOC (B881) or HEAPREALLOC (B88D). This is not validated.

HEAPLOCK – &B887

Action:
Locks or unlocks a memory block.

Entry conditions:
DE: Memory handle
BC=0: The block is locked. It will not be moved until unlocked so fixed addresses can be used as pointers into the block
BC<>0: The block is unlocked

HEAPMAXFREE – &B88A

Action:
Returns the largest block size that can be allocated.

Entry conditions:
None

Exit conditions:
HL: Largest free block size in bytes

HEAPREALLOC – &B88D

Action:
Changes the size of an allocated memory block.

Entry conditions:
DE: Memory handle
BC: New size for memory block

Exit conditions:
HL=0: Failed. The old block will not be freed but could have moved.
HL<>0: Successful

NOTE: If the block is being expanded, it must be assumed that the base of the memory block will be moved (even if the block cannot actually be expanded), so HEAPADDRESS (&B87E) must be called afterwards. If the block is being contracted, the base will not move.

FILE I/O FUNCTIONS

FCLOSE – &B890

Action:
Closes a file.

Entry conditions:
DE: File handle

Exit conditions:
c=1: Successful
c=0: Failed

FERASE – &B893

Action:
Erases a file.

Entry conditions:
HL: Zero-terminated filename

Exit conditions:
c=1: Successful
c=0: Error (file not found)

FINBLOCK – &B896

Action:
Reads a block from a file.

Entry conditions:
DE: File handle
HL: Buffer
BC: Number of bytes to read (greater than 0)

Exit conditions:
c=1: End of file not reached
c=0: Eof (or error?)
BC: Number of bytes read
HL: Address after last byte read

FINCHAR – &B899

Action:
Reads a byte from a file.

Entry conditions:
DE: File handle

Exit conditions:
c=1: Successful, A=character
c=0: A corrupt if end of file reached
Other registers preserved

FINDFIRST – &B89C

Action:
Finds the first file. SETDTA (&B8C6) must have been called first.

Entry conditions:
None

Exit conditions:
HL=0: No files
HL<>0: HL points to a file info structure. The first item in the structure is the filename, zero-terminated (up to 12 characters long), offset 13 is the attribute byte – see FGETATTR (&B8CF) for further details on attributes. Offsets 14 and 15 are the file size in bytes (low, high)

FINDNEXT – &B89F

Action:
Finds the next file. FINDFIRST (&B89C) must have been called first.

Entry conditions:
None

Exit conditions:
HL=0: No more files
HL<>0: HL returns info as with FINDFIRST (&B89C)

FOPENIN – &B8A2

Action:
Opens a file for input.

Entry conditions:
HL: Points to zero-terminated filename

Exit conditions:
c=1: Successful, DE=file handle
c=0: Failed (file not found), DE=Corrupt
A: Corrupt
Other registers preserved

FOPENOUT – &B8A5

Action:
Opens a file for output.

Entry conditions:
HL: Points to zero-terminated filename

Exit conditions:

c=1:	Successful, DE=file handle
c=0:	Failed (out of memory/too many files/file exists), DE=Corrupt
A:	Corrupt
	Other registers preserved

FOPENUP – &B8A8

Action:
Opens a file for input and output. The file must already exist.

Entry conditions:

HL:	Points to zero-terminated filename

Exit conditions:

c=1:	Successful, DE=file handle
c=0:	File not found, DE=Corrupt
A:	Corrupt
	Other registers preserved

FOUTBLOCK – &B8AB

Action:
Writes a block to a file.

Entry conditions:

DE:	File handle
HL:	Buffer
BC:	Number of bytes to write (greater than 0)

Exit conditions:

c=1:	Successful
c=0:	Error
BC:	Number of bytes written
HL:	Address after last byte written

FOUTCHAR – &B8AE

Action:
Writes a byte to a file.

Entry conditions:

DE:	File handle
A:	Character

Exit conditions:
c=1: Successful
c=0: A=Corrupt if end of file reached
Other registers preserved

FRENAME – &B8B1

Action:
Renames a file.

Entry conditions:
HL: Points to zero-terminated old filename
DE: Points to zero-terminated new filename

Exit conditions:
c=1: Successful
c=0: Error (file not found)

FSEEK – &B8B4

Action:
Moves the file pointer to a position within a file.

Entry conditions:
DE: File handle
BC: Offset from start of file

Exit conditions:
c=1: Successful
c=0: Offset past end of file (pointer not changed)

FSIZE – &B8B7

Action:
Finds the size of a file.

Entry conditions:
HL: Points to zero-terminated filename

Exit conditions:
c=1: HL=size in bytes
c=0: Not found

FSIZEHANDLE – &B8BA

Action:
Finds the size of an open file.

Entry conditions:
DE: File handle

Exit conditions:
HL: Size in bytes

FTELL – &B8BD

Action:
Returns the value of the file pointer.

Entry conditions:
DE: File handle

Exit conditions:
HL: Current file position

FTESTEOF – &B8C0

Action:
Tests whether the end of a file has been reached.

Entry conditions:
DE: File handle

Exit conditions:
c=1: Not eof
c=0: Eof

SELECTFILE – &B8C3

Action:
Displays the file selector (clearing the screen first), shows all files and allows a selection to be made using the cursor keys and [Return]. In addition [Del->] and [<-Del] can be used to delete files.

An undocumented feature of this function is the ability to press [Shift][Ctrl][H] to override the effect of the Hidden file attribute and make these files instantly visible. See FGETATTR (&B8CF) for further details on attributes.

Entry conditions:
None

Exit conditions:
c=1: A was file selected ([Return] pressed), HL=filename
c=0: [Stop] was pressed

SETDTA – &B8C6

Action:
Sets the memory block to be used by FINDFIRST (&B89C) and FINDNEXT (&B89F).

Entry conditions:
DE: Address of a 36-byte buffer which must be in common RAM (&8000h-&BFFF).

Exit conditions:

All registers preserved

MISCELLANEOUS FUNCTIONS

FDATESTAMP – &B8C9

Action:
Sets a file's date and time to the current date and time.

Entry conditions:
HL: Zero terminated filename

Exit conditions:
c=1: Successful
c=0: File not found

FGETATTR – &B8CF

Action:
Returns the attribute byte of a file.

Entry conditions:
HL: Zero-terminated filename

Exit conditions:
c=1: A=attribute
bit 0 = System (for in-built applications)
bit 1 = Hidden
bit 2 = Basic
bit 3 = Binary
bit 4 = Reserved
bit 5 = Reserved for internal use
bit 6 = Reserved
bit 7 = Reserved

System files are generally those created by the Diary, Address Book and other in-built applications. They are also generally saved as Hidden files except where the user needs to be able to select them.

Protext saves all files without any attributes so that they can be seen and selected by all applications. This allows you to write programs in Protext, then enter BBC Basic and *EXEC them into memory – providing an easier way of editing code.

All BBC Basic programs are saved with the Basic attribute set.

If the user has not configured the NC100 to display file dates and times via the System Setting menu, if a file has the Basic attribute set, under BBC Basic SELECTFILE (&B8C3) will display it but if the Hidden attribute is set, it will not. When not in BBC Basic, the Protext file selector will not display files with a Basic or Hidden attribute, so you can hide selected files from non-BBC Basic applications.

However, if the user has elected to have file dates and times displayed, all files except those with a Hidden attribute will be displayed whether in BBC Basic or not.

c=0: Not found
HL: Preserved

FSETATTR – &B8CC

Action:
Sets the attribute byte for a file opened for output. If the file is open for input only there is no effect

Entry conditions:
DE: File handle
C: Attribute byte:

Exit conditions:
c=1: Successful
c=0: File not found

KMGETYELLOW – &B8D2

Action:
Ascertains whether a Yellow event (so called because the [Function] key is coloured yellow) is pending. A Yellow event occurs:

- When the user has pressed one of the [Function][Key] combinations that cause an immediate context switch ([Function][Red], [Function][Green], [Function][Blue], [Function][Menu]), or
- When the machine is powered up and (because the option to preserve context has not been set) needs to return to the main menu.

Entry conditions:
None

Exit conditions:
c=1: BC=token if a Yellow event is pending. An application should exit conditions: normally as quickly as possible Any unsaved files should be saved automatically!
c=0: BC=0 if no Yellow event is pending

NOTE: Each of the yellow event keys return the [Stop] token (&2FCh). An application should call KMGETYELLOW (&B8D2) whenever an Escape key is read. This distinguishes between a Yellow event and an ordinary Escape.

KMSETYELLOW – &B8D5

Action:
Sets up a Yellow event. Specialised use only.

Entry conditions:
BC: A yellow event token

Exit conditions:
All registers preserved

LAPCAT_RECEIVE – &B8D8

Action:
Reads a character from the parallel port using Lapcat protocol.

Entry conditions:
None

Exit conditions:
c=1: Successful, A=character
c=0: No character read

LAPCAT_SEND – &B8DB

Action:
Sends a character to the parallel port using Lapcat protocol.

Entry conditions:
A: Character

Exit conditions:
c=1: Successful
c=0: Error

PADGETVERSION – &B8DE

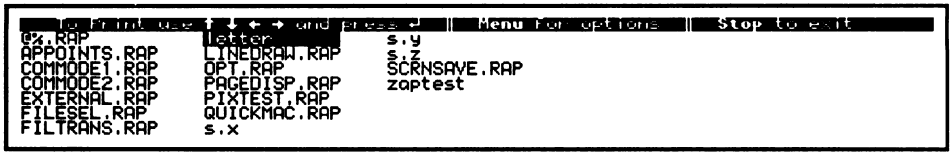
Action:
Gets the firmware version number.

Entry conditions:
None

Exit conditions:
HL: Version number (times 100). So, 1.03 returns 103

6

THE SYSTEM VARIABLES



Poke &B139 with 0 in Basic and lose the file sizes

Following are some of the more important RAM-based variables used by the operating system. Amstrad have expressed an intent always to try and use these locations in subsequent versions of the software, but they are not guaranteeing it. It would be sensible to perform checks by calling firmware routines which return known values to selected addresses and only if the correct values are returned for addresses you wish to use, should you then assume they are available to you.

Alternatively you could contact Amstrad at the following address with any queries relating to newer versions of the NC series. Write to:

Notepad Project Manager, Amstrad Plc, 169 Kings Road, Brentwood, Essex, CM14 4EF.

Many of the addresses shown in this section have little or no explanation other than the name given to them by the program developers. It is entirely up to you to experiment with them and come to your own decision as to their usefulness. Thankfully though, many addresses are fully self-evident and will provide you with a lot of scope for enhancing your own programs.

ADDRESS	NAME	SIZE	COMMENTS
&B000	copyofmmu0	&01	Copy of MMU0
&B001	copyofmmu1	&01	Copy of MMU1
&B002	copyofmmu2	&01	Copy of MMU2
&B003	copyofmmu3	&01	Copy of MMU3
&B03B		&50	A small stack which is only used in initialisation. Therefore, you should be able to use this as a temporary storage area when code space is tight.
&B08D	kbdstate1	&0A	1 bit per key: 1=down, 0=up to correspond to the matrix.
&B097	kbdstate2	&0A	2nd byte of state
&B0A1	padkeybuf	&40	Keyboard buffer
&B0E1	padnextin	&01	Offset into padkeybuf.
&B0E2	padnextout	&01	Next character due out
&B0E3	padbufempty	&01	Non-zero if empty.
&B0E4	lastkbdstate	&02	Saved state
&B0E6	thiskbdstate	&02	This state
&B0E8	caps.state	&01	0=off, &FF=on
&B112	rptdelay	&01	Keyboard repeat Centiseconds.
&B113	rptrate	&01	Keyboard delay Centiseconds.
&B114	rpttimer	&01	Count down timer for key repeat.
&B115	keytorepeat	&01	Key number.
&B116	rptkeystates	&01	Shift states.
&B12C	soundcounter	&01	Non-zero if playing a tune.
&B12D	soundptr	&02	Pointer to array of frequency, duration.
&B132	poweroffminutes	&01	Configured time to power off.
&B133	minutesleft	&01	Minutes left
&B134	minutecounter	&02	Minute counter
&B137	preservecontext	&01	0=return to main screen at power on.
&B138	dontpreservecontext	&01	1=don't preserve (diagnostics/ battery).
&B139	mainprog	&01	6=inbasic, 128=inexternal (foreground program id).
&B13A	currentprinter	&01	0 for parallel, 1 for serial.
&B13D	wasmenusel	&01	After KMWAITCHAR this is 1 if menu used, 0 if not.
&B140	sdumpname	&04	File names s.a, s.b, s.c and so on - for screen dumps.

&B150	d.datebuf	&12	Date buffer
&B162	d.asciitime	&0C	hh:mm:ss
&B16E	currentcfg	&4C	Current configuration parameters
&B1BD	g.pos	&01	Current column number (charout).
&B258	d.calcmode	&01	Non-zero if keyboard in calculator mode.
&B259	d.kmexplen	&01	Expansion string length.
&B25A	d.kmexpptr	&02	Expansion string pointer.
&B25C	d.expbuffer	&02	Address of expansion key buffer.
&B25E	d.expbufptr	&02	Pointer to free byte.
&B260	d.expbufend	&02	Last byte in buffer.
&B2A1	macro_buf	&100	Macro buffer
&B3A7		-	File selector variables...
&B3A7	fs_clicat	&01	Non-zero if Cat command, not Select.
&B3A8	fs_showsizes	&01	Non-zero if showing file sizes (pad default=off).
&B3A9	fs_showsys	&01	Non-zero if showing system files.
&B3AA	fs_curfile	&01	Current file number offset from top left.
&B3AB	fs_topleftfile	&01	File number displayed top left.
&B3AC	fs_numcols	&01	Number of columns
&B3AD	fs_colwidth	&01	Width of columns
&B3AE	fs_numshown	&01	Number of columns shown
&B3AF	fs_maxfiles	&01	Max files that can be shown.
&B3B2	fs_numfilerows	&01	Rows of files in CAT command.
&B3B3	fs_startlist	&02	Start of file list. Zero if doing unsorted list.
&B3B5	fs_startdir	&02	Start of directory entries.
&B3B9	fs_numfiles	&01	Number of files in directory.
&B3BA	fs_lastshown	&01	Last file number currently shown.

BBC BASIC MAIN SYSTEM VARIABLES

ADDRESS	SIZE	COMMENTS
&A000	&100	String accumulator
&A100	&100	String input buffer
&A200	&6C	Static variables @% to Z%
&A2DC	&02	PAGE
&A2DE	&02	TOP
&A2E0	&02	LOMEM
&A2E2	&02	Free space pointer
&A2E4	&02	HIMEM
&A2E6	&02	Current line number
&A2E8	&02	TRACE number
&A2EA	&02	AUTO number
&A2EC	&02	ON ERROR number

7

RECOVERING FROM LOCK-OUTS

If you use the Notepad's BBC Basic assembler facilities you are likely to crash the computer at some point. What usually happens in a crash is you get a complete lock-out and even turning the computer off and on just results in a blank (or sometimes black) screen.

Sometimes you can get out of crashes quite quickly and easily by switching off the Notepad and holding down the [Function] and [Stop] keys while you switch it on again. However, it does have the effect of completely resetting various settings you may have set up, such as Preserve Context or Document Transfer, although the time and date are unaffected.

Unfortunately, there is nothing you can do other than press the [Menu] key and re-enter your preferred defaults. If this doesn't work you may find that the documented reset facility may do so – try switching off, pressing [Function][Stop] [<-Del] and switching on again while holding these keys down. If it does get you out of a lock-up and back into the system, this reset will have entirely erased any files or data held in the Notepad, although all data on any RAM card you may have inserted will remain untouched.

Occasionally a bug may have a peculiar effect that the [Function][Stop] procedure does appear to remedy, in that it returns you to the front menu, but you then find you cannot re-enter Basic by pressing [Function][B] because the screen goes completely blank and nothing happens. However, you may be able to get around this by switching the Notepad on and off yet again and then pressing [Function][B] one more time.

Unfortunately, resetting the Notepad is not always as easy as this because some crashes appear to lock up the Notepad completely so that no combination of key presses or reset commands will restore it. In this eventuality you have no recourse

other than to remove the four AA batteries, disconnect the power supply lead and remove the small lithium battery and any RAM card you may have inserted.

Having done this you should press the on/off switch repeatedly for a minute or two in order to drain any residual power which may be left in the Notepad. Now re-insert all the batteries, power lead and any RAM card you may be using, and switch on. You should then have a fully-functional NC100 again. Remember that this procedure completely erases all data from your computer, including addresses, diary entries and anything you may have stored in the Private area.

A strong word of caution: If you develop any programs yourself or type in any of the listings from this book, it is quite likely that you will introduce one or more bugs and consequently may get a crash that causes you to lose all the data stored in the NC100. Therefore it is very important that you first transfer any programs or documents you need to keep, to another computer using the Lapcat communications lead and software. It's available from Arnor, the NC100's developers (see Appendix 6 for full details). In fact, you would be well advised to regularly back up important files in any event.

But more than that, if you don't have one, you should strongly consider buying a RAM card. These come in sizes from 32Kb up to 1Mb and are essential if you wish to store more than one or two programs or documents at a time. In addition, if you happen to crash the NC100, files stored on the RAM card will almost certainly not be destroyed and, after resetting the computer, you can re-insert the RAM card and start using the stored files immediately.

Page 47 of the NC100 user guide offers further information, including how to format a new RAM card ready for use. The Lapcat communications lead and software and RAM cards for the NC100 (compatible with the industry standard) are available from Arnor (see Appendix 6).

8

THE COMPLETE Z80 INSTRUCTION SET

ADC A,(HL)	The contents of the address pointed to by HL and the carry flag are both added to the contents of A, and the result is then stored in A.
ADC A,(IX+d)	The contents of the address pointed to by IX plus displacement d and the carry flag are both added to the contents of A, and the result is then stored in A.
ADC A,(IY+d)	The contents of the address pointed to by IY plus displacement d and the carry flag are both added to the contents of A, and the result is then stored in A.
ADC A,A	The contents of A and the carry flag are added to A, and the result is stored in A.
ADC A,B	The contents of B and the carry flag are added to A, and the result is stored in A.
ADC A,C	The contents of C and the carry flag are added to A, and the result is stored in A.
ADC A,D	The contents of D and the carry flag are added to A, and the result is stored in A.
ADC A,E	The contents of E and the carry flag are added to A, and the result is stored in A.
ADC A,H	The contents of H and the carry flag are added to A, and the result is stored in A.
ADC A,L	The contents of L and the carry flag are added to A, and the result is stored in A.
ADC A,n	The value n and the carry flag are added to A, and the result is stored in A.
ADC HL,BC	The contents of HL and the carry flag are added to BC, and the result is stored in HL.

ADC HL,DE	The contents of HL and the carry flag are added to DE, and the result is stored in HL.
ADC HL,HL	The contents of HL and the carry flag are added to HL, and the result is stored in HL.
ADC HL,SP	The contents of HL and the carry flag are added to SP, and the result is stored in HL.
ADD A,(HL)	The contents of the address pointed to by HL are added to A, and the result is stored in A.
ADD A,(IX+d)	The contents of the address pointed to by IX plus displacement d are added to A, and the result is stored in A.
ADD A,(IY+d)	The contents of the address pointed to by IY plus displacement d are added to A, and the result is stored in A.
ADD A,A	The contents of A are added to A, and the result is stored in A.
ADD A,B	The contents of A are added to B, and the result is stored in A.
ADD A,C	The contents of A are added to C, and the result is stored in A.
ADD A,D	The contents of A are added to D, and the result is stored in A.
ADD A,E	The contents of A are added to E, and the result is stored in A.
ADD A,H	The contents of A are added to H, and the result is stored in A.
ADD A,L	The contents of A are added to L, and the result is stored in A.
ADD A,n	The value n is added to A, and the result is stored in A.
ADD HL,BC	The contents of HL are added to BC, and the result is stored in HL.
ADD HL,DE	The contents of HL are added to DE, and the result is stored in HL.
ADD HL,HL	The contents of HL are added to HL, and the result is stored in HL.
ADD HL,SP	The contents of HL are added to SP, and the result is stored in HL.
ADD IX,BC	The contents of IX are added to BC, and the result is stored in IX.
ADD IX,DE	The contents of IX are added to DE, and the result is stored in IX.
ADD IX,IX	The contents of IX are added to IX, and the result is stored in IX.
ADD IX,SP	The contents of IX are added to SP, and the result is stored in IX.
ADD IY,BC	The contents of IY are added to BC, and the result is stored in IY.
ADD IY,DE	The contents of IY are added to DE, and the result is stored in IY.
ADD IY,IY	The contents of IY are added to IY, and the result is stored in IY.
ADD IY,SP	The contents of IY are added to SP, and the result is stored in IY.
AND (HL)	The contents of the address pointed to by HL is logically ANDed with A, and the result is stored in A.

AND (IX+d)	The contents of the address pointed to by IX plus displacement d is logically ANDed with A, and the result is stored in A.
AND (IY+d)	The contents of the address pointed to by IY plus displacement d is logically ANDed with A, and the result is stored in A.
AND A	A is logically ANDed with A, and the result is stored in A.
AND B	B is logically ANDed with A, and the result is stored in A.
AND C	C is logically ANDed with A, and the result is stored in A.
AND D	D is logically ANDed with A, and the result is stored in A.
AND E	E is logically ANDed with A, and the result is stored in A.
AND H	H is logically ANDed with A, and the result is stored in A.
AND L	L is logically ANDed with A, and the result is stored in A.
AND n	The value n is logically ANDed with A, and the result is stored in A.
BIT 0,(HL)	Bit 0 of the contents of the location pointed to by HL is tested. The Z flag returns its state.
BIT 0,(IX+d)	Bit 0 of the contents of the location pointed to by IX plus displacement d is tested. The Z flag returns its state.
BIT 0,(IY+d)	Bit 0 of the contents of the location pointed to by IY plus displacement d is tested. The Z flag returns its state.
BIT 0,A	Bit 0 of A is tested. The Z flag returns its state.
BIT 0,B	Bit 0 of B is tested. The Z flag returns its state.
BIT 0,C	Bit 0 of C is tested. The Z flag returns its state.
BIT 0,D	Bit 0 of D is tested. The Z flag returns its state.
BIT 0,E	Bit 0 of E is tested. The Z flag returns its state.
BIT 0,H	Bit 0 of H is tested. The Z flag returns its state.
BIT 0,L	Bit 0 of L is tested. The Z flag returns its state.
BIT 1,(HL)	Bit 1 of the contents of the location pointed to by HL is tested. The Z flag returns its state.
BIT 1,(IX+d)	Bit 1 of the contents of the location pointed to by IX plus displacement d is tested. The Z flag returns its state.
BIT 1,(IY+d)	Bit 1 of the contents of the location pointed to by IY plus displacement d is tested. The Z flag returns its state.
BIT 1,A	Bit 1 of A is tested. The Z flag returns its state.
BIT 1,B	Bit 1 of B is tested. The Z flag returns its state.
BIT 1,C	Bit 1 of C is tested. The Z flag returns its state.

BIT 1,D	Bit 1 of D is tested. The Z flag returns its state.
BIT 1,E	Bit 1 of E is tested. The Z flag returns its state.
BIT 1,H	Bit 1 of H is tested. The Z flag returns its state.
BIT 1,L	Bit 1 of L is tested. The Z flag returns its state.
BIT 2,(HL)	Bit 2 of the contents of the location pointed to by HL is tested. The Z flag returns its state.
BIT 2,(IX+d)	Bit 2 of the contents of the location pointed to by IX plus displacement d is tested. The Z flag returns its state.
BIT 2,(IY+d)	Bit 2 of the contents of the location pointed to by IY plus displacement d is tested. The Z flag returns its state.
BIT 2,A	Bit 2 of A is tested. The Z flag returns its state.
BIT 2,B	Bit 2 of B is tested. The Z flag returns its state.
BIT 2,C	Bit 2 of C is tested. The Z flag returns its state.
BIT 2,D	Bit 2 of D is tested. The Z flag returns its state.
BIT 2,E	Bit 2 of E is tested. The Z flag returns its state.
BIT 2,H	Bit 2 of H is tested. The Z flag returns its state.
BIT 2,L	Bit 2 of L is tested. The Z flag returns its state.
BIT 3,(HL)	Bit 3 of the contents of the location pointed to by HL is tested. The Z flag returns its state.
BIT 3,(IX+d)	Bit 3 of the contents of the location pointed to by IX plus displacement d is tested. The Z flag returns its state.
BIT 3,(IY+d)	Bit 3 of the contents of the location pointed to by IY plus displacement d is tested. The Z flag returns its state.
BIT 3,A	Bit 3 of A is tested. The Z flag returns its state.
BIT 3,B	Bit 3 of B is tested. The Z flag returns its state.
BIT 3,C	Bit 3 of C is tested. The Z flag returns its state.
BIT 3,D	Bit 3 of D is tested. The Z flag returns its state.
BIT 3,E	Bit 3 of E is tested. The Z flag returns its state.
BIT 3,H	Bit 3 of H is tested. The Z flag returns its state.
BIT 3,L	Bit 3 of L is tested. The Z flag returns its state.
BIT 4,(HL)	Bit 4 of the contents of the location pointed to by HL is tested. The Z flag returns its state.
BIT 4,(IX+d)	Bit 4 of the contents of the location pointed to by IX plus displacement d is tested. The Z flag returns its state.

BIT 4,(IY+d)	Bit 4 of the contents of the location pointed to by IY plus displacement d is tested. The Z flag returns its state.
BIT 4,A	Bit 4 of A is tested. The Z flag returns its state.
BIT 4,B	Bit 4 of B is tested. The Z flag returns its state.
BIT 4,C	Bit 4 of C is tested. The Z flag returns its state.
BIT 4,D	Bit 4 of D is tested. The Z flag returns its state.
BIT 4,E	Bit 4 of E is tested. The Z flag returns its state.
BIT 4,H	Bit 4 of H is tested. The Z flag returns its state.
BIT 4,L	Bit 4 of L is tested. The Z flag returns its state.
BIT 5,(HL)	Bit 5 of the contents of the location pointed to by HL is tested. The Z flag returns its state.
BIT 5,(IX+d)	Bit 5 of the contents of the location pointed to by IX plus displacement d is tested. The Z flag returns its state.
BIT 5,(IY+d)	Bit 5 of the contents of the location pointed to by IY plus displacement d is tested. The Z flag returns its state.
BIT 5,A	Bit 5 of A is tested. The Z flag returns its state.
BIT 5,B	Bit 5 of B is tested. The Z flag returns its state.
BIT 5,C	Bit 5 of C is tested. The Z flag returns its state.
BIT 5,D	Bit 5 of D is tested. The Z flag returns its state.
BIT 5,E	Bit 5 of E is tested. The Z flag returns its state.
BIT 5,H	Bit 5 of H is tested. The Z flag returns its state.
BIT 5,L	Bit 5 of L is tested. The Z flag returns its state.
BIT 6,(HL)	Bit 6 of the contents of the location pointed to by HL is tested. The Z flag returns its state.
BIT 6,(IX+d)	Bit 6 of the contents of the location pointed to by IX plus displacement d is tested. The Z flag returns its state.
BIT 6,(IY+d)	Bit 6 of the contents of the location pointed to by IY plus displacement d is tested. The Z flag returns its state.
BIT 6,A	Bit 6 of A is tested. The Z flag returns its state.
BIT 6,B	Bit 6 of B is tested. The Z flag returns its state.
BIT 6,C	Bit 6 of C is tested. The Z flag returns its state.
BIT 6,D	Bit 6 of D is tested. The Z flag returns its state.
BIT 6,E	Bit 6 of E is tested. The Z flag returns its state.
BIT 6,H	Bit 6 of H is tested. The Z flag returns its state.
BIT 6,L	Bit 6 of L is tested. The Z flag returns its state.

BIT 7,(HL)	Bit 7 of the contents of the location pointed to by HL is tested. The Z flag returns its state.
BIT 7,(IX+d)	Bit 7 of the contents of the location pointed to by IX plus displacement d is tested. The Z flag returns its state.
BIT 7,(IY+d)	Bit 7 of the contents of the location pointed to by IY plus displacement d is tested. The Z flag returns its state.
BIT 7,A	Bit 7 of A is tested. The Z flag returns its state.
BIT 7,B	Bit 7 of B is tested. The Z flag returns its state.
BIT 7,C	Bit 7 of C is tested. The Z flag returns its state.
BIT 7,D	Bit 7 of D is tested. The Z flag returns its state.
BIT 7,E	Bit 7 of E is tested. The Z flag returns its state.
BIT 7,H	Bit 7 of H is tested. The Z flag returns its state.
BIT 7,L	Bit 7 of L is tested. The Z flag returns its state.
CALL C,nn	If C (Carry) is set then push the current contents of the program counter onto the stack and call the routine at location nn. When the routine returns using a RET it comes straight back by popping the correct return address off the stack.
CALL M,nn	If M (Minus) is set then push the current contents of the program counter onto the stack and call the routine at location nn. When the routine returns using a RET it comes straight back by popping the correct return address off the stack.
CALL NC,nn	If C is not set (No Carry) then push the current contents of the program counter onto the stack and call the routine at location nn. When the routine returns using a RET it comes straight back by popping the correct return address off the stack.
CALL NZ,nn	If Z is not set (Not Zero) then push the current contents of the program counter onto the stack and call the routine at location nn. When the routine returns using a RET it comes straight back by popping the correct return address off the stack.
CALL P,nn	If P (Plus) is set then push the current contents of the program counter onto the stack and call the routine at location nn. When the routine returns using a RET it comes straight back by popping the correct return address off the stack.
CALL PE,nn	If PE (Even) is set then push the current contents of the program counter onto the stack and call the routine at location nn. When the routine returns using a RET it comes straight back by popping the correct return address off the stack.
CALL PO,nn	If PO (Odd) is set then push the current contents of the program counter onto the stack and call the routine at location nn. When the routine returns using a RET it comes straight back by popping the correct return address off the stack.

CALL Z,nn	If Z (Zero) is set then push the current contents of the program counter onto the stack and call the routine at location nn. When the routine returns using a RET it comes straight back by popping the correct return address off the stack.
CALL nn	Immediately push the current contents of the program counter onto the stack and call the routine at location nn. When the routine returns using a RET it comes straight back by popping the correct return address off the stack.
CCF	Complement the C (Carry). If it is 1 it becomes 0, or vice versa.
CP (HL)	The contents of the location pointed to by HL are subtracted from A and the result is discarded. The flags are then set according to the result.
CP (IX+d)	The contents of the location pointed to IX plus displacement d are subtracted from A and the result is discarded. The flags are then set according to the result.
CP (IY+d)	The contents of the location pointed to by IY plus displacement d are subtracted from A and the result is discarded. The flags are then set according to the result.
CP A	The contents of A are subtracted from A and the result is discarded. The flags are then set according to the result.
CP B	The contents of B are subtracted from A and the result is discarded. The flags are then set according to the result.
CP C	The contents of C are subtracted from A and the result is discarded. The flags are then set according to the result.
CP D	The contents of D are subtracted from A and the result is discarded. The flags are then set according to the result.
CP E	The contents of E are subtracted from A and the result is discarded. The flags are then set according to the result.
CP H	The contents of H are subtracted from A and the result is discarded. The flags are then set according to the result.
CP L	The contents of L are subtracted from A and the result is discarded. The flags are then set according to the result.
CP n	The value n are subtracted from A and the result is discarded. The flags are then set according to the result.
CPD	The contents of the location pointed to by HL are subtracted from A and the result is discarded. Then both HL and BC are decremented. The flags are then set according to the result.
CPDR	The contents of the location pointed to by HL are subtracted from A and the result is discarded. Then both HL and BC are decremented. This instruction then repeats until BC equals 0 or A is the same as the contents of the location pointed to by HL.

CPI	The contents of the location pointed to by HL are subtracted from A and the result is discarded. Then HL is incremented and BC is decremented. The flags are then set according to the result.
CPIR	The contents of the location pointed to by HL are subtracted from A and the result is discarded. This instruction then repeats until BC equals 0 or A is the same as the contents of the location pointed to by HL.
CPL	The contents of A are complemented. All 0 bits become 1s and vice versa.
DAA	Conditionally adds 6 to the right and/or left nibble of a, based on the status register for BCD conversion after maths operations.
DEC (HL)	Decrement the contents of the location pointed to by HL.
DEC (IX+d)	Decrement the contents of the location pointed to by IX plus displacement d.
DEC (IY+d)	Decrement the contents of the location pointed to by IY plus displacement d.
DEC A	Decrement A.
DEC B	Decrement B.
DEC BC	Decrement BC.
DEC C	Decrement C.
DEC D	Decrement D.
DEC DE	Decrement DE.
DEC E	Decrement E.
DEC H	Decrement H.
DEC HL	Decrement HL.
DEC IX	Decrement IX.
DEC IY	Decrement IY.
DEC L	Decrement L.
DEC SP	Decrement SP.
DI	Disable all maskable interrupts.
DJNZ e	B is decremented. If the result is not zero then execution jumps to location e. The new location must be within 126 bytes before and 129 bytes following the current location as this is a relative branch.
EI	Enable all maskable interrupts.
EX (SP),HL	Exchange the contents of the address pointed to by SP with HL.
EX (SP),IX	Exchange the contents of the address pointed to by SP with IX.

EX (SP),IY	Exchange the contents of the address pointed to by SP with IY.
EX AF,AF'	Exchange AF with its alternate AF' register pair.
EX DE,HL	Swap the contents of DE and HL.
EXX	Exchange, BC, DE and HL with the alternate BC', DE' and HL' register pairs.
HALT	Suspend operation and execute NOPs until and interrupt or reset is received.
IM 0	Set interrupt mode 0 in which the interrupting device may insert one instruction onto the bus for execution, the first byte of which must occur during the interrupt acknowledge cycle,
IM 1	Set interrupt mode 1. A RST &38 instruction will be executed when an interrupt occurs.
IM 2	Set interrupt mode. When an interrupt occurs one byte of data must be provided by the peripheral, which is used as a low-order address. The high order address is taken from the I register. This points to a second address in memory which is loaded into the program counter and executed.
IN A,(n)	Load A with a byte from port (n). A supplies bits 8 to 15 of the port address, while n provides 0 to 7.
IN A,(C)	Load A with a byte from port (C). B must contain bits 8 to 15 of the port address, while C contains 0 to 7.
IN B,(C)	Load B with a byte from port (C). B must contain bits 8 to 15 of the port address, while C contains 0 to 7.
IN C,(C)	Load C with a byte from port (C). B must contain bits 8 to 15 of the port address, while C contains 0 to 7.
IN D,(C)	Load D with a byte from port (C). B must contain bits 8 to 15 of the port address, while C contains 0 to 7.
IN E,(C)	Load E with a byte from port (C). B must contain bits 8 to 15 of the port address, while C contains 0 to 7.
IN H,(C)	Load H with a byte from port (C). B must contain bits 8 to 15 of the port address, while C contains 0 to 7.
IN L,(C)	Load L with a byte from port (C). B must contain bits 8 to 15 of the port address, while C contains 0 to 7.
INC (HL)	Increment the contents of the location pointed to by HL.
INC (IX+d)	Increment the contents of the location pointed to by IX plus displacement d.
INC (IY+d)	Increment the contents of the location pointed to by IY plus displacement d.
INC A	Increment A.

INC B	Increment B.
INC BC	Increment BC.
INC C	Increment C.
INC D	Increment D.
INC DE	Increment DE.
INC E	Increment E.
INC H	Increment H.
INC HL	Increment HL.
INC IX	Increment IX.
INC IY	Increment IY.
INC L	Increment L.
INC SP	Increment SP.
IND	The device addressed by C is read into the memory location pointed to by HL. Then both B and HL are decremented.
INDR	The device addressed by C is read into the memory location pointed to by HL. Then both B and HL are decremented. This repeats until B equals 0.
INI	The device addressed by C is read into the memory location pointed to by HL. Then B is decremented and HL is incremented.
INIR	The device addressed by C is read into the memory location pointed to by HL. Then B is decremented and HL is incremented. This repeats until B equals 0.
JP nn	Jump directly to location nn.
JP (HL)	Jump directly to the location pointed to by the contents of HL.
JP (IX)	Jump directly to the location pointed to by the contents of IX.
JP (IY)	Jump directly to the location pointed to by the contents of IY.
JP C,nn	If C (Carry) is set, jump to nn.
JP M,nn	If M (Minus) is set, jump to nn.
JP NC,nn	If C is not set (No Carry), jump to nn.
JP NZ,nn	If Z is not set (Not Zero), jump to nn.
JP P,nn	If P (Plus) is set, jump to nn.
JP PE,nn	If PE (Even) is set, jump to nn.
JP PO,nn	If PO (Odd) is set, jump to nn.
JP Z,nn	If Z (Zero) is set, jump to nn.

JR C,e	If C (Carry) is set, jump relatively to e.
JR NC,e	If C is not set (No Carry), jump relatively to e. The new location must be within 126 bytes before and 129 bytes after the current location.
JR NZ,e	If Z is not set (Not Zero), jump relatively to e. The new location must be within 126 bytes before and 129 bytes after the current location.
JR Z,e	If Z (Zero) is set, jump relatively to e. The new location must be within 126 bytes before and 129 bytes after the current location.
JR e	Jump relatively to e. The new location must be within 126 bytes before and 129 bytes after the current location.
LD (BC),A	Load the location pointed to by BC with the value stored in A.
LD (DE),A	Load the location pointed to by DE with the value stored in A.
LD (HL),A	Load the location pointed to by HL with the value stored in A.
LD (HL),B	Load the location pointed to by HL with the value stored in B.
LD (HL),C	Load the location pointed to by HL with the value stored in C.
LD (HL),D	Load the location pointed to by HL with the value stored in D.
LD (HL),E	Load the location pointed to by HL with the value stored in E.
LD (HL),H	Load the location pointed to by HL with the value stored in H.
LD (HL),L	Load the location pointed to by HL with the value stored in L.
LD (HL),n	Load the location pointed to by HL with the value n.
LD (IX+d),A	Load the location pointed to by IX plus displacement d with the value stored in A.
LD (IX+d),B	Load the location pointed to by IX plus displacement d with the value stored in B.
LD (IX+d),C	Load the location pointed to by IX plus displacement d with the value stored in C.
LD (IX+d),D	Load the location pointed to by IX plus displacement d with the value stored in D.
LD (IX+d),E	Load the location pointed to by IX plus displacement d with the value stored in E.
LD (IX+d),H	Load the location pointed to by IX plus displacement d with the value stored in H.
LD (IX+d),L	Load the location pointed to by IX plus displacement d with the value stored in L.
LD (IX+d),n	Load the location pointed to by IX plus displacement d with the value n.

LD (IY+d),A	Load the location pointed to by IY plus displacement d with the value stored in A.
LD (IY+d),B	Load the location pointed to by IY plus displacement d with the value stored in B.
LD (IY+d),C	Load the location pointed to by IY plus displacement d with the value stored in C.
LD (IY+d),D	Load the location pointed to by IY plus displacement d with the value stored in D.
LD (IY+d),E	Load the location pointed to by IY plus displacement d with the value stored in E.
LD (IY+d),H	Load the location pointed to by IY plus displacement d with the value stored in H.
LD (IY+d),L	Load the location pointed to by IY plus displacement d with the value stored in L.
LD (IY+d),n	Load the location pointed to by IY plus displacement d with the value n.
LD (nn),A	Load the location pointed to by nn with the value stored in A.
LD (nn),BC	Load the two locations pointed to by nn with the two-byte value stored in BC.
LD (nn),DE	Load the two locations pointed to by nn with the two-byte value stored in DE.
LD (nn),HL	Load the two locations pointed to by nn with the two-byte value stored in HL.
LD (nn),IX	Load the two locations pointed to by nn with the two-byte value stored in IX.
LD (nn),IY	Load the two locations pointed to by nn with the two-byte value stored in IY.
LD (nn),SP	Load the two locations pointed to by nn with the two-byte value stored in SP.
LD A,(BC)	Load A with the contents of the location pointed to by BC.
LD A,(DE)	Load A with the contents of the location pointed to by DE.
LD A,(HL)	Load A with the contents of the location pointed to by HL.
LD A,(IX+d)	Load A with the contents of the location pointed to by IX plus displacement d.
LD A,(IY+d)	Load A with the contents of the location pointed to by IY plus displacement d.
LD A,(nn)	Load A with the contents of the location pointed to by nn.
LD A,A	Load A with the contents of A. (Can there be a use for this?).

LD A,B	Load A with the contents of B.
LD A,C	Load A with the contents of C.
LD A,D	Load A with the contents of D.
LD A,E	Load A with the contents of E.
LD A,H	Load A with the contents of H.
LD A,I	Load A with the contents of I (Interrupt register).
LD A,L	Load A with the contents of L.
LD A,n	Load A with the value n.
LD A,R	Load A with the contents of R (Refresh register).
LD B,(HL)	Load B with the contents of the location pointed to by HL.
LD B,(IX+d)	Load B with the contents of the location pointed to by IX plus displacement d.
LD B,(IY+d)	Load B with the contents of the location pointed to by IY plus displacement d.
LD B,(nn)	Load B with the contents of the location pointed to by nn.
LD B,A	Load B with the contents of A.
LD B,B	Load B with the contents of B.
LD B,C	Load B with the contents of C.
LD B,D	Load B with the contents of D.
LD B,E	Load B with the contents of E.
LD B,H	Load B with the contents of H.
LD B,L	Load B with the contents of L.
LD B,n	Load B with the value n.
LD C,(HL)	Load C with the contents of the location pointed to by HL.
LD C,(IX+d)	Load C with the contents of the location pointed to by IX plus displacement d.
LD C,(IY+d)	Load C with the contents of the location pointed to by IY plus displacement d.
LD C,(nn)	Load C with the contents of the location pointed to by nn.
LD C,A	Load C with the contents of A.
LD C,B	Load C with the contents of B.
LD C,C	Load C with the contents of C.
LD C,D	Load C with the contents of D.
LD C,E	Load C with the contents of E.

LD C,H	Load C with the contents of H.
LD C,L	Load C with the contents of L.
LD C,n	Load C with the value n.
LD D,(HL)	Load D with the contents of the location pointed to by HL.
LD D,(IX+d)	Load D with the contents of the location pointed to by IX plus displacement d.
LD D,(IY+d)	Load D with the contents of the location pointed to by IY plus displacement d.
LD D,(nn)	Load D with the contents of the location pointed to by nn.
LD D,A	Load D with the contents of A.
LD D,B	Load D with the contents of B.
LD D,C	Load D with the contents of C.
LD D,D	Load D with the contents of D.
LD D,E	Load D with the contents of E.
LD D,H	Load D with the contents of H.
LD D,L	Load D with the contents of L.
LD D,n	Load D with the value n.
LD E,(HL)	Load E with the contents of the location pointed to by HL.
LD E,(IX+d)	Load E with the contents of the location pointed to by IX plus displacement d.
LD E,(IY+d)	Load E with the contents of the location pointed to by IY plus displacement d.
LD E,(nn)	Load E with the contents of the location pointed to by nn.
LD E,A	Load E with the contents of A.
LD E,B	Load E with the contents of B.
LD E,C	Load E with the contents of C.
LD E,D	Load E with the contents of D.
LD E,E	Load E with the contents of E.
LD E,H	Load E with the contents of H.
LD E,L	Load E with the contents of L.
LD E,n	Load E with the value n.
LD H,(HL)	Load H with the contents of the location pointed to by HL.
LD H,(IX+d)	Load H with the contents of the location pointed to by IX plus displacement d.

LD H,(IY+d)	Load H with the contents of the location pointed to by IY plus displacement d.
LD H,(nn)	Load H with the contents of the location pointed to by nn.
LD H,A	Load H with the contents of A.
LD H,B	Load H with the contents of B.
LD H,C	Load H with the contents of C.
LD H,D	Load H with the contents of D.
LD H,E	Load H with the contents of E.
LD H,H	Load H with the contents of H.
LD H,L	Load H with the contents of L.
LD H,n	Load H with the value n.
LD HL,(nn)	Load HL with the two-byte pair at location nn.
LD HL,nn	Load HL with the two-byte value nn.
LD I,A	Load I (Interrupt register) with the contents of A.
LD IX,(nn)	Load IX with the two-byte value at location nn.
LD IX,nn	Load IX with the two-byte value nn.
LD IY,(nn)	Load IY with the two-byte value at location nn.
LD IY,nn	Load IY with the two-byte value nn.
LD L,(HL)	Load L with the contents of the location pointed to by HL.
LD L,(IX+d)	Load L with the contents of the location pointed to by IX plus displacement d.
LD L,(IY+d)	Load L with the contents of the location pointed to by IY plus displacement d.
LD L,(nn)	Load L with the contents of the location pointed to by nn.
LD L,A	Load L with the contents of A.
LD L,B	Load L with the contents of B.
LD L,C	Load L with the contents of C.
LD L,D	Load L with the contents of D.
LD L,E	Load L with the contents of E.
LD L,H	Load L with the contents of H.
LD L,L	Load L with the contents of L.
LD L,n	Load L with the value n.
LD R,A	Load R (Refresh register) with the contents of A.

LD SP,(nn)	Load SP with the two-byte contents pointed to by nn.
LD SP,HL	Load SP with the value in HL.
LD SP,IX	Load SP with the value in IX.
LD SP,IY	Load SP with the value in IY.
LD SP,nn	Load SP with the value nn.
LDD	The contents of the location pointed to by HL are loaded into the address pointed to by DE. Then BC, DE and HL are all decremented.
LDDR	The contents of the location pointed to by HL are loaded into the address pointed to by DE. Then BC, DE and HL are all decremented. This continues until BC equals 0.
LDI	The contents of the location pointed to by HL are loaded into the address pointed to by DE. Then DE and HL are incremented, while BC is decremented.
LDIR	The contents of the location pointed to by HL are loaded into the address pointed to by DE. Then DE and HL are incremented, while BC is decremented. This continues until BC equals 0.
NEG	The contents of A are subtracted from 0, and the result is stored in A.
NOP	Do nothing for one clock cycle.
OR A	A is logically ORed with A, and the result is stored in A.
OR B	A is logically ORed with B, and the result is stored in A.
OR C	A is logically ORed with C, and the result is stored in A.
OR D	A is logically ORed with D, and the result is stored in A.
OR E	A is logically ORed with E, and the result is stored in A.
OR H	A is logically ORed with H, and the result is stored in A.
OR L	A is logically ORed with L, and the result is stored in A.
Or n	A is logically ORed with the value n, and the result is stored in A.
OTDR	The contents of the location pointed to by HL are output to the device addressed by the C register. Both B and HL are then decremented. This continues until B equals 0. C supplies bits 0 to 7 of the port address, and B (after decrementing) supplies bits 8 to 15
OTIR	The contents of the location pointed to by HL are output to the device addressed by the C register. B is then decremented and HL is incremented. This continues until B equals 0. C supplies bits 0 to 7 of the port address, and B (after decrementing) supplies bits 8 to 15

OUT (C),A	Output the contents of A to port C. C supplies bits 0 to 7 of the port address, while B supplies bits 8 to 15.
OUT (C),B	Output the contents of B to port C. C supplies bits 0 to 7 of the port address, while B supplies bits 8 to 15.
OUT (C),C	Output the contents of C to port C. C supplies bits 0 to 7 of the port address, while B supplies bits 8 to 15.
OUT (C),D	Output the contents of D to port C. C supplies bits 0 to 7 of the port address, while B supplies bits 8 to 15.
OUT (C),E	Output the contents of E to port C. C supplies bits 0 to 7 of the port address, while B supplies bits 8 to 15.
OUT (C),H	Output the contents of H to port C. C supplies bits 0 to 7 of the port address, while B supplies bits 8 to 15.
OUT (C),L	Output the contents of L to port C. C supplies bits 0 to 7 of the port address, while B supplies bits 8 to 15.
OUT (n),A	Output the contents of A to the device addressed by n.
OUTD	The contents of the location pointed to by HL are output to the device addressed by C. Then B and HL are decremented. C supplies bits 0 to 7 of the ports address and B supplies bits 8 to 15 (after decrementing).
OUTI	The contents of the location pointed to by HL are output to the device addressed by C. Then B is decremented and HL is incremented. C supplies bits 0 to 7 of the ports address and B supplies bits 8 to 15 (after decrementing).
POP AF	The two-byte contents of the location pointed to by SP (the Stack Pointer) are loaded into AF and SP is incremented by two.
POP BC	The two-byte contents of the location pointed to by SP (the Stack Pointer) are loaded into BC and SP is incremented by two.
POP DE	The two-byte contents of the location pointed to by SP (the Stack Pointer) are loaded into DE and SP is incremented by two.
POP HL	The two-byte contents of the location pointed to by SP (the Stack Pointer) are loaded into HL and SP is incremented by two.
POP IX	The two-byte contents of the location pointed to by SP (the Stack Pointer) are loaded into IX and SP is incremented by two.
POP IY	The two-byte contents of the location pointed to by SP (the Stack Pointer) are loaded into IY and SP is incremented by two.
PUSH AF	SP (the Stack Pointer) is decremented by two and the two-byte contents of AF are loaded to the location now pointed to by SP.
PUSH BC	SP (the Stack Pointer) is decremented by two and the two-byte contents of BC are loaded to the location now pointed to by SP.

PUSH DE	SP (the Stack Pointer) is decremented by two and the two-byte contents of DE are loaded to the location now pointed to by SP.
PUSH HL	SP (the Stack Pointer) is decremented by two and the two-byte contents of HL are loaded to the location now pointed to by SP.
PUSH IX	SP (the Stack Pointer) is decremented by two and the two-byte contents of IX are loaded to the location now pointed to by SP.
PUSH IY	SP (the Stack Pointer) is decremented by two and the two-byte contents of IY are loaded to the location now pointed to by SP.
RES 0,(HL)	Bit 0 of the location pointed to by HL is reset to 0.
RES 0,(IX+d)	Bit 0 of the location pointed to by IX plus displacement d is reset to 0.
RES 0,(IY+d)	Bit 0 of the location pointed to by IY plus displacement d is reset to 0.
RES 0,A	Bit 0 of A is reset to 0.
RES 0,B	Bit 0 of B is reset to 0.
RES 0,C	Bit 0 of C is reset to 0.
RES 0,D	Bit 0 of D is reset to 0.
RES 0,E	Bit 0 of E is reset to 0.
RES 0,H	Bit 0 of F is reset to 0.
RES 0,L	Bit 0 of G is reset to 0.
RES 1,(HL)	Bit 1 of the location pointed to by HL is reset to 0.
RES 1,(IX+d)	Bit 1 of the location pointed to by IX plus displacement d is reset to 0.
RES 1,(IY+d)	Bit 1 of the location pointed to by IY plus displacement d is reset to 0.
RES 1,A	Bit 1 of A is reset to 0.
RES 1,B	Bit 1 of B is reset to 0.
RES 1,C	Bit 1 of C is reset to 0.
RES 1,D	Bit 1 of D is reset to 0.
RES 1,E	Bit 1 of E is reset to 0.
RES 1,H	Bit 1 of F is reset to 0.
RES 1,L	Bit 1 of G is reset to 0.
RES 2,(HL)	Bit 2 of the location pointed to by HL is reset to 0.
RES 2,(IX+d)	Bit 2 of the location pointed to by IX plus displacement d is reset to 0.

RES 2,(IY+d)	Bit 2 of the location pointed to by IY plus displacement d is reset to 0.
RES 2,A	Bit 2 of A is reset to 0.
RES 2,B	Bit 2 of B is reset to 0.
RES 2,C	Bit 2 of C is reset to 0.
RES 2,D	Bit 2 of D is reset to 0.
RES 2,E	Bit 2 of E is reset to 0.
RES 2,H	Bit 2 of F is reset to 0.
RES 2,L	Bit 2 of G is reset to 0.
RES 3,(HL)	Bit 3 of the location pointed to by HL is reset to 0.
RES 3,(IX+d)	Bit 3 of the location pointed to by IX plus displacement d is reset to 0.
RES 3,(IY+d)	Bit 3 of the location pointed to by IY plus displacement d is reset to 0.
RES 3,A	Bit 3 of A is reset to 0.
RES 3,B	Bit 3 of B is reset to 0.
RES 3,C	Bit 3 of C is reset to 0.
RES 3,D	Bit 3 of D is reset to 0.
RES 3,E	Bit 3 of E is reset to 0.
RES 3,H	Bit 3 of F is reset to 0.
RES 3,L	Bit 3 of G is reset to 0.
RES 4,(HL)	Bit 4 of the location pointed to by HL is reset to 0.
RES 4,(IX+d)	Bit 4 of the location pointed to by IX plus displacement d is reset to 0.
RES 4,(IY+d)	Bit 4 of the location pointed to by IY plus displacement d is reset to 0.
RES 4,A	Bit 4 of A is reset to 0.
RES 4,B	Bit 4 of B is reset to 0.
RES 4,C	Bit 4 of C is reset to 0.
RES 4,D	Bit 4 of D is reset to 0.
RES 4,E	Bit 4 of E is reset to 0.
RES 4,H	Bit 4 of F is reset to 0.
RES 4,L	Bit 4 of G is reset to 0.
RES 5,(HL)	Bit 5 of the location pointed to by HL is reset to 0.

RES 5,(IX+d)	Bit 5 of the location pointed to by IX plus displacement d is reset to 0.
RES 5,(IY+d)	Bit 5 of the location pointed to by IY plus displacement d is reset to 0.
RES 5,A	Bit 5 of A is reset to 0.
RES 5,B	Bit 5 of B is reset to 0.
RES 5,C	Bit 5 of C is reset to 0.
RES 5,D	Bit 5 of D is reset to 0.
RES 5,E	Bit 5 of E is reset to 0.
RES 5,H	Bit 5 of F is reset to 0.
RES 5,L	Bit 5 of G is reset to 0.
RES 6,(HL)	Bit 6 of the location pointed to by HL is reset to 0.
RES 6,(IX+d)	Bit 6 of the location pointed to by IX plus displacement d is reset to 0.
RES 6,(IY+d)	Bit 6 of the location pointed to by IY plus displacement d is reset to 0.
RES 6,A	Bit 6 of A is reset to 0.
RES 6,B	Bit 6 of B is reset to 0.
RES 6,C	Bit 6 of C is reset to 0.
RES 6,D	Bit 6 of D is reset to 0.
RES 6,E	Bit 6 of E is reset to 0.
RES 6,H	Bit 6 of F is reset to 0.
RES 6,L	Bit 6 of G is reset to 0.
RES 7,(HL)	Bit 7 of the location pointed to by HL is reset to 0.
RES 7,(IX+d)	Bit 7 of the location pointed to by IX plus displacement d is reset to 0.
RES 7,(IY+d)	Bit 7 of the location pointed to by IY plus displacement d is reset to 0.
RES 7,A	Bit 7 of A is reset to 0.
RES 7,B	Bit 7 of B is reset to 0.
RES 7,C	Bit 7 of C is reset to 0.
RES 7,D	Bit 7 of D is reset to 0.
RES 7,E	Bit 7 of E is reset to 0.
RES 7,H	Bit 7 of F is reset to 0.

RES 7,L	Bit 7 of G is reset to 0.
RET	PC (the Program Counter) is popped off the stack and execution continues from the new address.
RET C	If C (Carry) is set PC (the Program Counter) is popped off the stack and execution continues from the new address.
RET M	If M (Minus) is set PC (the Program Counter) is popped off the stack and execution continues from the new address.
RET NC	If C is not set (No Carry) PC (the Program Counter) is popped off the stack and execution continues from the new address.
RET NZ	If Z is not set (Not Zero) PC (the Program Counter) is popped off the stack and execution continues from the new address.
RET P	If P (Plus) is set PC (the Program Counter) is popped off the stack and execution continues from the new address.
RET PE	If PE (Even) is set PC (the Program Counter) is popped off the stack and execution continues from the new address.
RET PO	If PO (Odd) is set PC (the Program Counter) is popped off the stack and execution continues from the new address.
RET Z	If Z (Zero) is set PC (the Program Counter) is popped off the stack and execution continues from the new address.
RETI	This is the same as RET but must be used when returning from an interrupt to properly handle nested interrupts. You must execute and EI before issuing a RET.
RETN	This is the same as RET but must be used when returning from a non-maskable interrupt to restore the state of the interrupt flag before the non-maskable interrupt.
RL (HL)	The contents of the location pointed to by HL are shifted to the left by one bit. The contents of the Carry flag is placed in bit 0 and the contents of bit 7 is moved to the Carry flag.
RL (IX+d)	The contents of the location pointed to by IX plus displacement d are shifted to the left by one bit. The contents of the Carry flag are placed in bit 0 and the contents of bit 7 are moved to the Carry flag.
RL (IY+d)	The contents of the location pointed to by IY plus displacement d is shifted to the left by one bit. The contents of the Carry flag is placed in bit 0 and the contents of bit 7 is moved to the Carry flag.
RL A	The contents of A is shifted to the left by one bit. The contents of the Carry flag is placed in bit 0 and the contents of bit 7 is moved to the Carry flag.
RL B	The contents of B is shifted to the left by one bit. The contents of the Carry flag is placed in bit 0 and the contents of bit 7 is moved to the Carry flag.

RL C	The contents of C is shifted to the left by one bit. The contents of the Carry flag is placed in bit 0 and the contents of bit 7 is moved to the Carry flag.
RL D	The contents of D is shifted to the left by one bit. The contents of the Carry flag is placed in bit 0 and the contents of bit 7 is moved to the Carry flag.
RL E	The contents of E is shifted to the left by one bit. The contents of the Carry flag is placed in bit 0 and the contents of bit 7 is moved to the Carry flag.
RL H	The contents of H is shifted to the left by one bit. The contents of the Carry flag is placed in bit 0 and the contents of bit 7 is moved to the Carry flag.
RL L	The contents of L is shifted to the left by one bit. The contents of the Carry flag is placed in bit 0 and the contents of bit 7 is moved to the Carry flag.
RLA	The contents of A is shifted to the left by one bit. The contents of the Carry flag is placed in bit 0 and the contents of bit 7 is moved to the Carry flag.
RLC (HL)	The contents of the location pointed to by HL is rotated left by one bit. The original contents of bit 7 is placed in the Carry flag and also bit 0.
RLC (IX+d)	The contents of the location pointed to by IX plus displacement d is rotated left by one bit. The original contents of bit 7 is placed in the Carry flag and also bit 0.
RLC (IY+d)	The contents of the location pointed to by IY plus displacement d is rotated left by one bit. The original contents of bit 7 is placed in the Carry flag and also bit 0.
RLC A	The contents of A is rotated left by one bit. The original contents of bit 7 is placed in the Carry flag and also bit 0.
RLC B	The contents of B is rotated left by one bit. The original contents of bit 7 is placed in the Carry flag and also bit 0.
RLC C	The contents of C is rotated left by one bit. The original contents of bit 7 is placed in the Carry flag and also bit 0.
RLC D	The contents of D is rotated left by one bit. The original contents of bit 7 is placed in the Carry flag and also bit 0.
RLC E	The contents of E is rotated left by one bit. The original contents of bit 7 is placed in the Carry flag and also bit 0.
RLC H	The contents of H is rotated left by one bit. The original contents of bit 7 is placed in the Carry flag and also bit 0.
RLC L	The contents of L is rotated left by one bit. The original contents of bit 7 is placed in the Carry flag and also bit 0.

RLCA	The contents of A is rotated left by one bit. The original contents of bit 7 is placed in the Carry flag and also bit 0.
RLD	The four low bits of the location pointed to by HL are moved to the four high bits of the same location. The high bits are moved to the four low bits of A, after the four low bits of A have been moved to the four low bits of the original location.
RR (HL)	The contents of the location pointed to by HL are shifted to the right by one bit. The contents of the Carry flag is moved to bit 7 and the contents of bit 0 is moved to the Carry flag.
RR (IX+d)	The contents of the location pointed to by IX plus displacement d are shifted to the right by one bit. The contents of the Carry flag is moved to bit 7 and the contents of bit 0 is moved to the Carry flag.
RR (IY+d)	The contents of the location pointed to by IY plus displacement d are shifted to the right by one bit. The contents of the Carry flag are moved to bit 7 and the contents of bit 0 are moved to the Carry flag.
RR A	The contents of A are shifted to the right by one bit. The contents of the Carry flag are moved to bit 7 and the contents of bit 0 are moved to the Carry flag.
RR B	The contents of B are shifted to the right by one bit. The contents of the Carry flag are moved to bit 7 and the contents of bit 0 are moved to the Carry flag.
RR C	The contents of C are shifted to the right by one bit. The contents of the Carry flag are moved to bit 7 and the contents of bit 0 are moved to the Carry flag.
RR D	The contents of D are shifted to the right by one bit. The contents of the Carry flag are moved to bit 7 and the contents of bit 0 are moved to the Carry flag.
RR E	The contents of E are shifted to the right by one bit. The contents of the Carry flag are moved to bit 7 and the contents of bit 0 are moved to the Carry flag.
RR H	The contents of H are shifted to the right by one bit. The contents of the Carry flag are moved to bit 7 and the contents of bit 0 are moved to the Carry flag.
RR L	The contents of L are shifted to the right by one bit. The contents of the Carry flag are moved to bit 7 and the contents of bit 0 are moved to the Carry flag.
RRA	The contents of A are shifted to the right by one bit. The contents of the Carry flag are moved to bit 7 and the contents of bit 0 are moved to the Carry flag.

RRC (HL) The contents of the location pointed to by HL are rotated to the right by one bit. The contents of bit 0 are moved to the Carry flag and also to bit 7.

RRC (IX+d) The contents of the location pointed to by IX plus displacement d are rotated to the right by one bit. The contents of bit 0 are moved to the Carry flag and also to bit 7.

RRC (IY+d) The contents of the location pointed to by IX plus displacement d are rotated to the right by one bit. The contents of bit 0 are moved to the Carry flag and also to bit 7.

RRC A The contents of A are rotated to the right by one bit. The contents of bit 0 are moved to the Carry flag and also to bit 7.

RRC B The contents of B are rotated to the right by one bit. The contents of bit 0 are moved to the Carry flag and also to bit 7.

RRC C The contents of C are rotated to the right by one bit. The contents of bit 0 are moved to the Carry flag and also to bit 7.

RRC D The contents of D are rotated to the right by one bit. The contents of bit 0 are moved to the Carry flag and also to bit 7.

RRC E The contents of E are rotated to the right by one bit. The contents of bit 0 are moved to the Carry flag and also to bit 7.

RRC H The contents of H are rotated to the right by one bit. The contents of bit 0 are moved to the Carry flag and also to bit 7.

RRC L The contents of L are rotated to the right by one bit. The contents of bit 0 are moved to the Carry flag and also to bit 7.

RRCA The contents of A are rotated to the right by one bit. The contents of bit 0 are moved to the Carry flag and also to bit 7.

RRD The four high order bits of the location pointed to by HL are moved to the four low bits of the same location. The four low order bits are moved to the four low order bits of A, after the four low order bits of A are moved to the four high order bits of the original location.

RST &00 The contents of PC are pushed onto the stack and a jump is made directly to address &0000.

RST &08 The contents of PC are pushed onto the stack and a jump is made directly to address &0008.

RST &10 The contents of PC are pushed onto the stack and a jump is made directly to address &0010.

RST &18 The contents of PC are pushed onto the stack and a jump is made directly to address &0018.

RST &20 The contents of PC are pushed onto the stack and a jump is made directly to address &0020.

RST &28	The contents of PC are pushed onto the stack and a jump is made directly to address &0028.
RST &30	The contents of PC are pushed onto the stack and a jump is made directly to address &0030.
RST &38	The contents of PC are pushed onto the stack and a jump is made directly to address &0038.
SBC A,n	The value n is summed with the Carry flag and then subtracted from the contents of A, and the result is placed in A.
SBC A,(HL)	The contents of the address pointed to by HL are summed with the Carry flag and then subtracted from the contents of A, and the result is placed in A.
SBC A,(IX+d)	The contents of the address pointed to by IX plus displacement d is summed with the Carry flag and then subtracted from the contents of A, and the result is placed in A.
SBC A,(IY+d)	The contents of the address pointed to by IY plus displacement d are summed with the Carry flag and then subtracted from the contents of A, and the result is placed in A.
SBC A,A	The contents of A are summed with the Carry flag and then subtracted from the contents of A, and the result is placed in A.
SBC A,B	The contents of B are summed with the Carry flag and then subtracted from the contents of A, and the result is placed in A.
SBC A,C	The contents of C are summed with the Carry flag and then subtracted from the contents of A, and the result is placed in A.
SBC A,D	The contents of D are summed with the Carry flag and then subtracted from the contents of A, and the result is placed in A.
SBC A,E	The contents of E are summed with the Carry flag and then subtracted from the contents of A, and the result is placed in A.
SBC A,H	The contents of H are summed with the Carry flag and then subtracted from the contents of A, and the result is placed in A.
SBC A,L	The contents of L are summed with the Carry flag and then subtracted from the contents of A, and the result is placed in A.
SBC HL,BC	The contents of BC plus the Carry flag are subtracted from the contents of HL, and the result is placed in HL.
SBC HL,DE	The contents of DE plus the Carry flag are subtracted from the contents of HL, and the result is placed in HL.
SBC HL,HL	The contents of HL plus the Carry flag are subtracted from the contents of HL, and the result is placed in HL.
SBC HL,SP	The contents of SP plus the Carry flag are subtracted from the contents of HL, and the result is placed in HL.

SCF	The Carry flag is set.
SET 0,(HL)	Bit 0 of the location pointed to by HL is set to 1.
SET 0,(IX+d)	Bit 0 of the location pointed to by IX plus displacement d is set to 1.
SET 0,(IY+d)	Bit 0 of the location pointed to by IY plus displacement d is set to 1.
SET 0,A	Bit 0 of A is set to 1.
SET 0,B	Bit 0 of B is set to 1.
SET 0,C	Bit 0 of C is set to 1.
SET 0,D	Bit 0 of D is set to 1.
SET 0,E	Bit 0 of E is set to 1.
SET 0,H	Bit 0 of F is set to 1.
SET 0,L	Bit 0 of G is set to 1.
SET 1,(HL)	Bit 1 of the location pointed to by HL is set to 1.
SET 1,(IX+d)	Bit 1 of the location pointed to by IX plus displacement d is set to 1.
SET 1,(IY+d)	Bit 1 of the location pointed to by IY plus displacement d is set to 1.
SET 1,A	Bit 1 of A is set to 1.
SET 1,B	Bit 1 of B is set to 1.
SET 1,C	Bit 1 of C is set to 1.
SET 1,D	Bit 1 of D is set to 1.
SET 1,E	Bit 1 of E is set to 1.
SET 1,H	Bit 1 of F is set to 1.
SET 1,L	Bit 1 of G is set to 1.
SET 2,(HL)	Bit 2 of the location pointed to by HL is set to 1.
SET 2,(IX+d)	Bit 2 of the location pointed to by IX plus displacement d is set to 1.
SET 2,(IY+d)	Bit 2 of the location pointed to by IY plus displacement d is set to 1.
SET 2,A	Bit 2 of A is set to 1.
SET 2,B	Bit 2 of B is set to 1.
SET 2,C	Bit 2 of C is set to 1.
SET 2,D	Bit 2 of D is set to 1.

SET 2,E	Bit 2 of E is set to 1.
SET 2,H	Bit 2 of F is set to 1.
SET 2,L	Bit 2 of G is set to 1.
SET 3,(HL)	Bit 3 of the location pointed to by HL is set to 1.
SET 3,(IX+d)	Bit 3 of the location pointed to by IX plus displacement d is set to 1.
SET 3,(IY+d)	Bit 3 of the location pointed to by IY plus displacement d is set to 1.
SET 3,A	Bit 3 of A is set to 1.
SET 3,B	Bit 3 of B is set to 1.
SET 3,C	Bit 3 of C is set to 1.
SET 3,D	Bit 3 of D is set to 1.
SET 3,E	Bit 3 of E is set to 1.
SET 3,H	Bit 3 of F is set to 1.
SET 3,L	Bit 3 of G is set to 1.
SET 4,(HL)	Bit 4 of the location pointed to by HL is set to 1.
SET 4,(IX+d)	Bit 4 of the location pointed to by IX plus displacement d is set to 1.
SET 4,(IY+d)	Bit 4 of the location pointed to by IY plus displacement d is set to 1.
SET 4,A	Bit 4 of A is set to 1.
SET 4,B	Bit 4 of B is set to 1.
SET 4,C	Bit 4 of C is set to 1.
SET 4,D	Bit 4 of D is set to 1.
SET 4,E	Bit 4 of E is set to 1.
SET 4,H	Bit 4 of F is set to 1.
SET 4,L	Bit 4 of G is set to 1.
SET 5,(HL)	Bit 5 of the location pointed to by HL is set to 1.
SET 5,(IX+d)	Bit 5 of the location pointed to by IX plus displacement d is set to 1.
SET 5,(IY+d)	Bit 5 of the location pointed to by IY plus displacement d is set to 1.
SET 5,A	Bit 5 of A is set to 1.
SET 5,B	Bit 5 of B is set to 1.

SET 5,C	Bit 5 of C is set to 1.
SET 5,D	Bit 5 of D is set to 1.
SET 5,E	Bit 5 of E is set to 1.
SET 5,H	Bit 5 of F is set to 1.
SET 5,L	Bit 5 of G is set to 1.
SET 6,(HL)	Bit 6 of the location pointed to by HL is set to 1.
SET 6,(IX+d)	Bit 6 of the location pointed to by IX plus displacement d is set to 1.
SET 6,(IY+d)	Bit 6 of the location pointed to by IY plus displacement d is set to 1.
SET 6,A	Bit 6 of A is set to 1.
SET 6,B	Bit 6 of B is set to 1.
SET 6,C	Bit 6 of C is set to 1.
SET 6,D	Bit 6 of D is set to 1.
SET 6,E	Bit 6 of E is set to 1.
SET 6,H	Bit 6 of F is set to 1.
SET 6,L	Bit 6 of G is set to 1.
SET 7,(HL)	Bit 7 of the location pointed to by HL is set to 1.
SET 7,(IX+d)	Bit 7 of the location pointed to by IX plus displacement d is set to 1.
SET 7,(IY+d)	Bit 7 of the location pointed to by IY plus displacement d is set to 1.
SET 7,A	Bit 7 of A is set to 1.
SET 7,B	Bit 7 of B is set to 1.
SET 7,C	Bit 7 of C is set to 1.
SET 7,D	Bit 7 of D is set to 1.
SET 7,E	Bit 7 of E is set to 1.
SET 7,H	Bit 7 of F is set to 1.
SET 7,L	Bit 7 of G is set to 1.
SLA (HL)	The contents of the address pointed to by HL are arithmetically shifted right by one bit. The contents of bit 7 are moved to the Carry flag and bit 0 is loaded with 0.
SLA (IX+d)	The contents of the address pointed to by IX plus displacement d are arithmetically shifted right by one bit. The contents of bit 7 are moved to the Carry flag and bit 0 is loaded with 0.

SLA (IY+d) The contents of the address pointed to by IY plus displacement d are arithmetically shifted left by one bit. The contents of bit 7 are moved to the Carry flag and bit 0 is loaded with 0.

SLA A The contents of A are arithmetically shifted left by one bit. The contents of bit 7 are moved to the Carry flag and bit 0 is loaded with 0.

SLA B The contents of B are arithmetically shifted left by one bit. The contents of bit 7 are moved to the Carry flag and bit 0 is loaded with 0.

SLA C The contents of C are arithmetically shifted left by one bit. The contents of bit 7 are moved to the Carry flag and bit 0 is loaded with 0.

SLA D The contents of D are arithmetically shifted left by one bit. The contents of bit 7 are moved to the Carry flag and bit 0 is loaded with 0.

SLA E The contents of E are arithmetically shifted left by one bit. The contents of bit 7 are moved to the Carry flag and bit 0 is loaded with 0.

SLA H The contents of H are arithmetically shifted left by one bit. The contents of bit 7 are moved to the Carry flag and bit 0 is loaded with 0.

SLA L The contents of L are arithmetically shifted left by one bit. The contents of bit 7 are moved to the Carry flag and bit 0 is loaded with 0.

SRA (HL) The contents of the address pointed to by HL are arithmetically shifted right by one bit. The contents of bit 0 are moved to the Carry flag and bit 7 remains unchanged.

SRA (IX+d) The contents of the address pointed to by IX plus displacement d are arithmetically shifted right by one bit. The contents of bit 0 are moved to the Carry flag and bit 7 remains unchanged.

SRA (IY+d) The contents of the address pointed to by IY plus displacement d are arithmetically shifted right by one bit. The contents of bit 0 are moved to the Carry flag and bit 7 remains unchanged.

SRA A The contents of A are arithmetically shifted right by one bit. The contents of bit 0 are moved to the Carry flag and bit 7 remains unchanged.

SRA B The contents of B are arithmetically shifted right by one bit. The contents of bit 0 are moved to the Carry flag and bit 7 remains unchanged.

SRA C The contents of C are arithmetically shifted right by one bit. The contents of bit 0 are moved to the Carry flag and bit 7 remains unchanged.

SRA D — The contents of D are arithmetically shifted right by one bit. The contents of bit 0 are moved to the Carry flag and bit 7 remains unchanged.

SRA E — The contents of E are arithmetically shifted right by one bit. The contents of bit 0 are moved to the Carry flag and bit 7 remains unchanged.

SRA H — The contents of H are arithmetically shifted right by one bit. The contents of bit 0 are moved to the Carry flag and bit 7 remains unchanged.

SRA L — The contents of L are arithmetically shifted right by one bit. The contents of bit 0 are moved to the Carry flag and bit 7 remains unchanged.

SRL (HL) — The contents of the location pointed to by HL are logically shifted right by one bit. Bit 7 is set to 0 and the contents of bit 0 are moved to the Carry flag.

SRL (IX+d) — The contents of the location pointed to by IX plus displacement d are logically shifted right by one bit. Bit 7 is set to 0 and the contents of bit 0 are moved to the Carry flag.

SRL (IY+d) — The contents of the location pointed to by IY plus displacement d are logically shifted right by one bit. Bit 7 is set to 0 and the contents of bit 0 are moved to the Carry flag.

SRL A — The contents of A are logically shifted right by one bit. Bit 7 is set to 0 and the contents of bit 0 are moved to the Carry flag.

SRL B — The contents of B are logically shifted right by one bit. Bit 7 is set to 0 and the contents of bit 0 are moved to the Carry flag.

SRL C — The contents of C are logically shifted right by one bit. Bit 7 is set to 0 and the contents of bit 0 are moved to the Carry flag.

SRL D — The contents of D are logically shifted right by one bit. Bit 7 is set to 0 and the contents of bit 0 are moved to the Carry flag.

SRL E — The contents of E are logically shifted right by one bit. Bit 7 is set to 0 and the contents of bit 0 are moved to the Carry flag.

SRL H — The contents of H are logically shifted right by one bit. Bit 7 is set to 0 and the contents of bit 0 are moved to the Carry flag.

SRL L — The contents of L are logically shifted right by one bit. Bit 7 is set to 0 and the contents of bit 0 are moved to the Carry flag.

SUB (HL) — The contents of the location pointed to by HL are subtracted from A, and the result is stored in A.

SUB (IX+d) — The contents of the location pointed to by IX plus displacement d are subtracted from A, and the result is stored in A.

SUB (IY+d) — The contents of the location pointed to by IY plus displacement d are subtracted from A, and the result is stored in A.

SUB A	The contents of A are subtracted from A, and the result is stored in A.
SUB B	The contents of B are subtracted from A, and the result is stored in A.
SUB C	The contents of C are subtracted from A, and the result is stored in A.
SUB D	The contents of D are subtracted from A, and the result is stored in A.
SUB E	The contents of E are subtracted from A, and the result is stored in A.
SUB H	The contents of H are subtracted from A, and the result is stored in A.
SUB L	The contents of L are subtracted from A, and the result is stored in A.
SUB n	The value n is subtracted from A, and the result is stored in A.
XOR (HL)	The contents of the location pointed to by HL are exclusive-ORed with A, and the result is stored in A.
XOR (IX+d)	The contents of the location pointed to by IX plus displacement d are exclusive-ORed with A, and the result is stored in A.
XOR (IY+d)	The contents of the location pointed to by IY plus displacement d are exclusive-ORed with A, and the result is stored in A.
XOR A	The contents of A are exclusive-ORed with A, and the result is stored in A.
XOR B	The contents of B are exclusive-ORed with A, and the result is stored in A.
XOR C	The contents of C are exclusive-ORed with A, and the result is stored in A.
XOR D	The contents of D are exclusive-ORed with A, and the result is stored in A.
XOR E	The contents of E are exclusive-ORed with A, and the result is stored in A.
XOR H	The contents of H are exclusive-ORed with A, and the result is stored in A.
XOR L	The contents of L are exclusive-ORed with A, and the result is stored in A.
XOR n	The value n is exclusive-ORed with A, and the result is stored in A.

9

THE UNDOCUMENTED Z80 INSTRUCTIONS

Not many people know that the Z80 microprocessor incorporates a number of undocumented instructions - particularly to do with handling the IX and IY register pairs as four single-byte registers, as you can with AF, BC, DE and HL.

However, because Zilog did not document them you are unlikely to find an assembler that recognises these new mnemonics. Certainly BBC Basic doesn't, so if you want to use them you will have to enter in the raw machine code hex codes (which are shown next to each instruction in the following table).

Thankfully this is quite easy. All you have to do is look up the two-byte pair and enter code in the following manner (as long as you are in an assembler section of your program):

```
DEFB &DD:DEFB &BC
```

In this case the new instruction CP hX will be assembled.

Please remember that because these instructions are undocumented they are not guaranteed to work and the author and publisher of this book will accept no responsibility for your use of them. That said, let's hope you find them useful.

ADC A,hX	DD 8C	The contents of A, the Carry flag and the high byte of the IX register are added to A, and the result is stored in A.
ADC A,lX	DD 8D	The contents of A, the Carry flag and the low byte of the IX register are added to A, and the result is stored in A.
ADC A,hY	FD 8C	The contents of A, the Carry flag and the high byte of the IY register are added to A, and the result is stored in A.
ADC A,lY	FD 8D	The contents of A, the Carry flag and the low byte of the IY register are added to A, and the result is stored in A.

ADD A,hX	DD 84	The contents of A and the high byte of the IX register are added to A, and the result is stored in A.
ADD A,lX	DD 85	The contents of A and the low byte of the IX register are added to A, and the result is stored in A.
ADD A,hY	FD 84	The contents of A and the high byte of the IY register are added to A, and the result is stored in A.
ADD A,lY	FD 85	The contents of A and the low byte of the IY register are added to A, and the result is stored in A.
AND hX	DD A4	The high byte of the IX register is ANDed with A, and the result is stored in A.
AND lX	DD A5	The low byte of the IX register is ANDed with A, and the result is stored in A.
AND hY	FD A4	The high byte of the IY register is ANDed with A, and the result is stored in A.
AND lY	FD A5	The low byte of the IY register is ANDed with A, and the result is stored in A.
CP hX	DD BC	The conten ; of the high byte of IX are subtracted from A and the result is discarded. The flags are then set according to the result.
CP lX	DD BD	The contents of the low byte of IX are subtracted from A and the result is discarded. The flags are then set according to the result.
CP hY	FD BC	The contents of the high byte of IY are subtracted from A and the result is discarded. The flags are then set according to the result.
CP lY	FD BD	The contents of the low byte of IY are subtracted from A and the result is discarded. The flags are then set according to the result.
INC hX	DD 25	The high byte of IX is incremented.
INC lX	DD 2D	The low byte of IX is incremented.
INC hY	FD 25	The high byte of IY is incremented.
INC lY	FD 2D	The low byte of IY is incremented.
LD hX,A	DD 67	Load the high byte of IX with the value in A.
LD hX,B	DD 60	Load the high byte of IX with the value in B.
LD hX,C	DD 61	Load the high byte of IX with the value in C.
LD hX,D	DD 62	Load the high byte of IX with the value in D.

LD hX,E	DD 63	Load the high byte of IX with the value in E.
LD hX,n	DD 26 nn	Load the high byte of IX with the value n.
LD hY,A	FD 67	Load the high byte of IY with the value in A.
LD hY,B	FD 60	Load the high byte of IY with the value in B.
LD hY,C	FD 61	Load the high byte of IY with the value in C.
LD hY,D	FD 62	Load the high byte of IY with the value in D.
LD hY,E	FD 63	Load the high byte of IY with the value in E.
LD hY,n	FD 26 nn	Load the high byte of IY with the value n.
LD lX,A	DD 6F	Load the low byte of IX with the value in A.
LD lX,B	DD 68	Load the low byte of IX with the value in B.
LD lX,C	DD 69	Load the low byte of IX with the value in C.
LD lX,D	DD 6A	Load the low byte of IX with the value in D.
LD lX,E	DD 6B	Load the low byte of IX with the value in E.
LD lX,n	DD 2E nn	Load the low byte of IX with the value n.
LD lY,A	FD 6F	Load the low byte of IY with the value in A.
LD lY,B	FD 68	Load the low byte of IY with the value in B.
LD lY,C	FD 69	Load the low byte of IY with the value in C.
LD lY,D	FD 6A	Load the low byte of IY with the value in D.
LD lY,E	FD 6B	Load the low byte of IY with the value in E.
LD lY,n	FD 2E nn	Load the low byte of IY with the value n.
LD A,hX	DD 7C	Load A with the high byte of IX.
LD B,hX	DD 44	Load B with the high byte of IX.
LD C,hX	DD 4C	Load C with the high byte of IX.
LD D,hX	DD 54	Load D with the high byte of IX.
LD E,hX	DD 5C	Load E with the high byte of IX.
LD A,lX	DD 7D	Load A with the low byte of IX.
LD B,lX	DD 45	Load B with the low byte of IX.
LD C,lX	DD 4D	Load C with the low byte of IX.
LD D,lX	DD 55	Load D with the low byte of IX.
LD E,lX	DD 5D	Load E with the low byte of IX.

LD A,hY	FD 7C	Load A with the high byte of IY.
LD B,hY	FD 44	Load B with the high byte of IY.
LD C,hY	FD 4C	Load C with the high byte of IY.
LD D,hY	FD 54	Load D with the high byte of IY.
LD E,hY	FD 5C	Load E with the high byte of IY.
LD A,lY	FD 7D	Load A with the low byte of IY.
LD B,lY	FD 45	Load B with the low byte of IY.
LD C,lY	FD 4D	Load C with the low byte of IY.
LD D,lY	FD 55	Load D with the low byte of IY.
LD E,lY	FD 5D	Load E with the low byte of IY.
LD hX,lX	DD 65	Load the high byte of IX with the low byte.
LD lX,hX	DD 6C	Load the low byte of IX with the high byte.
LD hY,lY	FD 65	Load the high byte of IY with the low byte.
LD lY,hY	FD 6C	Load the low byte of IX with the high byte.
OR hX	DD B4	The high byte of IX is logically ORed with A, and the result is stored in A.
OR lX	DD B5	The low byte of IX is logically ORed with A, and the result is stored in A.
OR hY	FD B4	The high byte of IY is logically ORed with A, and the result is stored in A.
OR lY	FD B5	The low byte of IY is logically ORed with A, and the result is stored in A.
SBC A,hX	DD 9C	The high byte of IX is summed with the Carry flag and subtracted from A. The result is then stored in A.
SBC A,lX	DD 9D	The low byte of IX is summed with the Carry flag and subtracted from A. The result is then stored in A.
SBC A,hY	FD 9C	The high byte of IY is summed with the Carry flag and subtracted from A. The result is then stored in A.
SBC A,lY	FD 9D	The low byte of IX is summed with the Carry flag and subtracted from A. The result is then stored in A.
SLL A	CB 37	The contents of A are logically shifted left. Bit 0 is set to 0 and the contents of bit 7 are moved to the Carry flag.
SLL B	CB 30	The contents of B are logically shifted left. Bit 0 is set to 0 and the contents of bit 7 are moved to the Carry flag.

SLL C	CB 31	The contents of C are logically shifted left. Bit 0 is set to 0 and the contents of bit 7 are moved to the Carry flag.
SLL D	CB 32	The contents of D are logically shifted left. Bit 0 is set to 0 and the contents of bit 7 are moved to the Carry flag.
SLL E	CB 33	The contents of E are logically shifted left. Bit 0 is set to 0 and the contents of bit 7 are moved to the Carry flag.
SLL H	CB 34	The contents of H are logically shifted left. Bit 0 is set to 0 and the contents of bit 7 are moved to the Carry flag.
SLL L	CB 35	The contents of L are logically shifted left. Bit 0 is set to 0 and the contents of bit 7 are moved to the Carry flag.
SLL (HL)	CB 36	The contents of the location pointed to by HL are logically shifted left. Bit 0 is set to 0 and the contents of bit 7 are moved to the Carry flag.
SUB hX	DD 94	The high byte of IX is subtracted from A and the result is stored in A.
SUB lX	DD 95	The low byte of IX is subtracted from A and the result is stored in A.
SUB hY	FD 94	The high byte of IY is subtracted from A and the result is stored in A.
SUB lY	FD 95	The low byte of IY is subtracted from A and the result is stored in A.
XOR hX	DD AC	The high byte of IX is exclusive-ORed with A and the result is stored in A.
XOR lX	DD AD	The low byte of IX is exclusive-ORed with A and the result is stored in A.
XOR hY	FD AC	The high byte of IY is exclusive-ORed with A and the result is stored in A.
XOR lY	FD AD	The low byte of IY is exclusive-ORed with A and the result is stored in A.

SECTION 3

APPENDICES

APPENDIX 1

NC100 JUMPBLOCK ENTRY POINTS

COL1	&B818
COL1TEXT	&B81B
EDITBUF	&B800
FCLOSE	&B890
FDATESTAMP	&B8C9
FERASE	&B893
FGETATTR	&B8CF
FINBLOCK	&B896
FINCHAR	&B899
FINDFIRST	&B89C
FINDNEXT	&B89F
FOPENIN	&B8A2
FOPENOUT	&B8A5
FOPENUP	&B8A8
FOUTBLOCK	&B8AB
FOUTCHAR	&B8AE
FRENAME	&B8B1
FSEEK	&B8B4
FSETATTR	&B8CC
FSIZE	&B8B7
FSIZEHANDLE	&B8BA
FTELL	&B8BD
FTESTEOF	&B8C0
HEAPADDRESS	&B87E

HEAPALLOC	&B881
HEAPFREE	&B884
HEAPLOCK	&B887
HEAPMAXFREE	&B88A
HEAPREALLOC	&B88D
KMCHARRETURN	&B803
KMGETYELLOW	&B8D2
KMREADKBD	&B806
KMSETEXPAND	&B809
KMSETTICKCOUNT	&B80C
KMSETYELLOW	&B8D5
KMWAITKBD	&B80F
LAPCAT_RECEIVE	&B8D8
LAPCAT_SEND	&B8DB
MCPRINTCHAR	&B851
MCREADYPRINTER	&B854
MCSETPRINTER	&B857
PADGETTICKER	&B872
PADGETTIME	&B875
PADGETVERSION	&B8DE
PADINITSERIAL	&B85A
PADINSERIAL	&B85D
PADOUTPARALLEL	&B860
PADOUTSERIAL	&B863
PADREADYPARALLEL	&B866
PADREADYSERIAL	&B869
PADRESETSERIAL	&B86C
PADSERIALWAITING	&B86F
PADSETALARM	&B878
PADSETTIME	&B87B
READBUF	&B812
SELECTFILE	&B8C3
SETDTA	&B8C6
TESTESCAPE	&B815
TEXTOUT	&B81E
TEXTOUTCOUNT	&B821

TXTBOLDOFF	&B83F
TXTBOLDON	&B842
TXTCLEARWINDOW	&B824
TXTCUROFF	&B827
TXTCURON	&B82A
TXTGETCURSOR	&B82D
TXTGETWINDOW	&B830
TXTINVERSEOFF	&B845
TXTINVERSEON	&B848
TXTOUTPUT	&B833
TXTSETCURSOR	&B836
TXTSETWINDOW	&B839
TXTUNDERLINEOFF	&B84B
TXTUNDERLINEON	&B84E
TXTWRCHAR	&B83C

APPENDIX 2

INPUT/OUTPUT PORTS (&0000 - &00FF)

&0000	Display memory start	Write only
&0010-&0013	Memory management	Read/Write
&0020	Card wait control	Write only
&0030	Baud rate	Write only
&0040	Parallel port data	Write only
&0050-&0053	Speaker frequency	Write only
&0060	IRQ Mask	Write only
&0070	Power on/off control	Write only
&0080-&008F	Not Used	-
&0090	IRQ request status	Read/Write
&00A0	Card Status	Read only
&00B0-&00B9	Key data in	Read only
&00C0-&00C1	UART (uPD71051)	Read/Write
&00D0-&00DF	RTC (TC8521)	Read/Write
&00E0-&00FF	Not Used	-

APPENDIX 3

KEYBOARD SCAN CODES

(As reported by the program INKEY.BAS)

KEY	NORMAL	SHIFT	CONTROL	SYMBOL	SHFT/CTRL	SHFT/SMBL
[Stop]	&000	&000	-	-	-	-
[Tab]	&009	&2E4	&2E1	-	-	-
[Return]	&00D	&2EC	&2EC	-	-	-
[Space]	&020	&220	&2EB	-	-	-
[!]	-	&021	-	-	-	-
["]	-	&022	-	-	-	-
[#]	-	&023	-	-	-	-
[$]	-	&024	-	-	-	-
[%]	-	&025	-	-	-	-
[&]	-	&026	-	-	-	-
[']	&027	-	-	&33A	-	-
[(]	-	&028	-	-	-	-
[)]	-	&029	-	-	-	-
[*]	-	&02A	-	-	-	-
[+]	-	&02B	-	-	-	-
[,]	&02C	-	-	&0AE	-	-
[-]	&02D	-	-	-	-	-
[.]	&02E	-	-	&0AF	-	-
[/]	&02F	-	-	&0A8	-	-
[0]	&030	-	&2E1	-	-	-
[1]	&031	-	&211	&0AD	-	-
[2]	&032	-	&209	&33C	-	-
[3]	&033	-	&2E6	-	-	-
[4]	&034	-	&2E1	-	-	-
[5]	&035	-	&2D9	&33E	&2D2	-
[6]	&036	-	&2E0	&33D	&2D7	-
[7]	&037	-	&342	-	-	-
[8]	&038	-	&355	-	-	-
[9]	&039	-	&2DC	-	-	-
[:]	-	&03A	-	-	-	-
[;]	&03B	-	-	-	-	-
[<]	-	&03C	-	-	-	-
[=]	&03D	-	-	-	-	-
[>]	-	&03E	-	-	-	-
[?]	-	&03F	-	-	-	-
[@]	-	&040	&200	-	-	-

[A]	&041	&061	&201	&084	-	&08E
[B]	&042	&062	&202	-	-	-
[C]	&043	&063	&203	&087	&309	&080
[D]	&044	&064	&204	&314	&38E	-
[E]	&045	&065	&205	&091	-	&092
[F]	&046	&066	&206	-	-	-
[G]	&047	&067	&207	-	-	-
[H]	&048	&068	&2E3	&0AB	&3DC	-
[I]	&049	&069	&209	-	-	-
[J]	&04A	&06A	&20A	-	-	-
[K]	&04B	&06B	&20B	-	-	-
[L]	&04C	&06C	&20C	-	&308	-
[M]	&04D	&06D	&20D	&0E6	&306	-
[N]	&04E	&06E	&20E	&0A4	-	&0A5
[O]	&04F	&06F	&20F	&094	-	&099
[P]	&050	&070	&3DE	&014	&210	-
[Q]	&051	&071	&211	&0AC	-	-
[R]	&052	&072	&212	-	-	-
[S]	&053	&073	&213	&0E1	&370	&0E1
[T]	&054	&074	&29F	&315	-	-
[U]	&055	&075	&215	&081	-	&09A
[V]	&056	&076	&216	-	-	-
[W]	&057	&077	&217	-	-	-
[X]	&058	&078	&218	-	-	-
[Y]	&059	&079	&219	-	-	-
[Z]	&05A	&07A	&21A	-	-	-
[[]	&05B	-	-	-	-	-
[\]	&05C	-	-	&33B	-	-
[]]	&05D	-	-	-	-	-
[^]	-	&05E	-	-	-	-
[_]	-	&05F	-	-	-	-
[{]	-	&07B	-	-	-	-
[|]	-	&07C	-	-	-	-
[}]	-	&07D	-	-	-	-
[~]	-	&07E	-	-	-	-
[_]	-	&09C	-	-	-	-
[Del->]	&221	&2E5	&205	-	-	-
[<-Del]	&27F	&2D3	&2D4	-	-	-
[Up]	&2F0	&2F4	&2F8	&018	-	-
[Down]	&2F1	&2F5	&2F9	&019	-	-
[Left]	&2F2	&2F6	&2FA	&01B	-	-
[Right]	&2F3	&2F7	&2FB	&01A	-	-
[Menu]	&386	&3E3	&3D4	&393	-	-

APPENDIX 4

THE COMPLETE SET OF Z80 INSTRUCTION CODES

OPCODES	MNEMONICS
8E	ADC A,(HL)
DD 8E 05	ADC A,(IX+d)
FD 8E 05	ADC A,(IY+d)
8F	ADC A,A
88	ADC A,B
89	ADC A,C
8A	ADC A,D
8B	ADC A,E
8C	ADC A,H
8D	ADC A,L
CE 20	ADC A,n
ED 4A	ADC HL,BC
ED 5A	ADC HL,DE
ED 6A	ADC HL,HL
ED 7A	ADC HL,SP
86	ADD A,(HL)
DD 86 05	ADD A,(IX+d)
FD 86 05	ADD A,(IY+d)
87	ADD A,A
80	ADD A,B
81	ADD A,C
82	ADD A,D
83	ADD A,E
84	ADD A,H
85	ADD A,L
C6 20	ADD A,n
09	ADD HL,BC
19	ADD HL,DE
29	ADD HL,HL
39	ADD HL,SP
DD 09	ADD IX,BC
DD 19	ADD IX,DE
DD 29	ADD IX,IX

DD 39	ADD IX,SP
FD 09	ADD IY,BC
FD 19	ADD IY,DE
FD 29	ADD IY,IY
FD 39	ADD IY,SP
A6	AND (HL)
DD A6 05	AND (IX+d)
FD A6 05	AND (IY+d)
A7	AND A
A0	AND B
A1	AND C
A2	AND D
A3	AND E
A4	AND H
A5	AND L
E6 20	AND n
CB 46	BIT 0,(HL)
DD CB 05 46	BIT 0,(IX+d)
FD CB 05 46	BIT 0,(IY+d)
CB 47	BIT 0,A
CB 40	BIT 0,B
CB 41	BIT 0,C
CB 42	BIT 0,D
CB 43	BIT 0,E
CB 44	BIT 0,H
CB 45	BIT 0,L
CB 4E	BIT 1,(HL)
DD CB 05 4E	BIT 1,(IX+d)
FD CB 05 4E	BIT 1,(IY+d)
CB 4F	BIT 1,A
CB 48	BIT 1,B
CB 49	BIT 1,C
CB 4A	BIT 1,D
CB 4B	BIT 1,E
CB 4C	BIT 1,H
CB 4D	BIT 1,L
CB 56	BIT 2,(HL)
DD CB 05 56	BIT 2,(IX+d)
FD CB 05 56	BIT 2,(IY+d)
CB 57	BIT 2,A
CB 50	BIT 2,B
CB 51	BIT 2,C
CB 52	BIT 2,D
CB 53	BIT 2,E
CB 54	BIT 2,H
CB 55	BIT 2,L
CB 5E	BIT 3,(HL)
DD CB 05 5E	BIT 3,(IX+d)
FD CB 05 5E	BIT 3,(IY+d)
CB 5F	BIT 3,A

CB 58	BIT 3,B
CB 59	BIT 3,C
CB 5A	BIT 3,D
CB 5B	BIT 3,E
CB 5C	BIT 3,H
CB 5D	BIT 3,L
CB 66	BIT 4,(HL)
DD CB 05 66	BIT 4,(IX+d)
FD CB 05 66	BIT 4,(IY+d)
CB 67	BIT 4,A
CB 60	BIT 4,B
CB 61	BIT 4,C
CB 62	BIT 4,D
CB 63	BIT 4,E
CB 64	BIT 4,H
CB 65	BIT 4,L
CB 6E	BIT 5,(HL)
DD CB 05 6E	BIT 5,(IX+d)
FD CB 05 6E	BIT 5,(IY+d)
CB 6F	BIT 5,A
CB 68	BIT 5,B
CB 69	BIT 5,C
CB 6A	BIT 5,D
CB 6B	BIT 5,E
CB 6C	BIT 5,H
CB 6D	BIT 5,L
CB 76	BIT 6,(HL)
DD CB 05 76	BIT 6,(IX+d)
FD CB 05 76	BIT 6,(IY+d)
CB 77	BIT 6,A
CB 70	BIT 6,B
CB 71	BIT 6,C
CB 72	BIT 6,D
CB 73	BIT 6,E
CB 74	BIT 6,H
CB 75	BIT 6,L
CB 7E	BIT 7,(HL)
DD CB 05 7E	BIT 7,(IX+d)
FD CB 05 7E	BIT 7,(IY+d)
CB 7F	BIT 7,A
CB 78	BIT 7,B
CB 79	BIT 7,C
CB 7A	BIT 7,D
CB 7B	BIT 7,E
CB 7C	BIT 7,H
CB 7D	BIT 7,L
DC 84 05	CALL C,nn
FC 84 05	CALL M,nn
D4 84 05	CALL NC,nn

C4 84 05	CALL NZ,nn
F4 84 05	CALL P,nn
EC 84 05	CALL PE,nn
E4 84 05	CALL PO,nn
CC 84 05	CALL Z,nn
CD 84 05	CALL nn
3F	CCF
BE	CP (HL)
DD BE 05	CP (IX+d)
FD BE 05	CP (IY+d)
BF	CP A
B8	CP B
B9	CP C
BA	CP D
BB	CP E
BC	CP H
BD	CP L
FE 20	CP n
ED A9	CPD
ED B9	CPDR
ED B1	CPIR
ED A1	CPI
2F	CPL
27	DAA
35	DEC (HL)
DD 35 05	DEC (IX+d)
FD 35 05	DEC (IY+d)
3D	DEC A
05	DEC B
0B	DEC BC
0D	DEC C
15	DEC D
1B	DEC DE
1D	DEC E
25	DEC H
2B	DEC HL
DD 2B	DEC IX
FD 2B	DEC IY
2D	DEC L
3B	DEC SP
F3	DI
10 2E	DJNZ e
FB	EI
E3	EX (SP),HL
DD E3	EX (SP),IX
FD E3	EX (SP),IY

08	EX AF,AF'
EB	EX DE,HL
D9	EXX
76	HALT
ED 46	IM 0
ED 56	IM 1
ED 5E	IM 2
ED 78	IN A,(C)
ED 40	IN B,(C)
ED 48	IN C,(C)
ED 50	IN D,(C)
ED 58	IN E,(C)
ED 60	IN H,(C)
ED 68	IN L,(C)
34	INC (HL)
DD 34 05	INC (IX+d)
FD 34 05	INC (IY+d)
3C	INC A
04	INC B
03	INC BC
0C	INC C
14	INC D
13	INC DE
1C	INC E
24	INC H
23	INC HL
DD 23	INC IX
FD 23	INC IY
2C	INC L
33	INC SP
DB 20	IN A,n
ED AA	IND
ED BA	INDR
ED A2	INI
ED B2	INIR
C3 84 05	JP nn
E9	JP (HL)
DD E9	JP (IX)
FD E9	JP (IY)
DA 84 05	JP C,nn
FA 84 05	JP M,nn
D2 84 05	JP NC,nn
C2 84 05	JP NZ,nn
F2 84 05	JP P,nn
EA 84 05	JP PE,nn
E2 84 05	JP PO,nn
CA 84 05	JP Z,nn

38 2E	JR C,e
30 2E	JR NC,e
20 2E	JR NZ,e
28 2E	JR Z,e
18 2E	JR e
02	LD (BC),A
12	LD (DE),A
77	LD (HL),A
70	LD (HL),B
71	LD (HL),C
72	LD (HL),D
73	LD (HL),E
74	LD (HL),H
75	LD (HL),L
36 20	LD (HL),n
DD 77 05	LD (IX+d),A
DD 70 05	LD (IX+d),B
DD 71 05	LD (IX+d),C
DD 72 05	LD (IX+d),D
DD 73 05	LD (IX+d),E
DD 74 05	LD (IX+d),H
DD 75 05	LD (IX+d),L
DD 36 05 20	LD (IX+d),n
FD 77 05	LD (IY+d),A
FD 70 05	LD (IY+d),B
FD 71 05	LD (IY+d),C
FD 72 05	LD (IY+d),D
FD 73 05	LD (IY+d),E
FD 74 05	LD (IY+d),H
FD 75 05	LD (IY+d),L
FD 36 05 20	LD (IY+d),n
32 84 05	LD (nn),A
ED 43 84 05	LD (nn),BC
ED 53 84 05	LD (nn),DE
22 84 05	LD (nn),HL
DD 22 84 05	LD (nn),IX
FD 22 84 05	LD (nn),IY
ED 73 84 05	LD (nn),SP
0A	LD A,(BC)
1A	LD A,(DE)
7E	LD A,(HL)
DD 7E 05	LD A,(IX+d)
FD 7E 05	LD A,(IY+d)
3A 84 05	LD A,(nn)
7F	LD A,A
78	LD A,B
79	LD A,C
7A	LD A,D
7B	LD A,E
7C	LD A,H
ED 57	LD A,I
7D	LD A,L

3E 20	LD A,n
ED 5F	LD A,R
46	LD B,(HL)
DD 46 05	LD B,(IX+d)
FD 46 05	LD B,(IY+d)
47	LD B,A
40	LD B,B
41	LD B,C
42	LD B,D
43	LD B,E
44	LD B,H
45	LD B,L
06 20	LD B,n
ED 4B 84 05	LD BC,(nn)
01 84 05	LD BC,nn
4E	LD C,(HL)
DD 4E 05	LD C,(IX+d)
FD 4E 05	LD C,(IY+d)
4F	LD C,A
48	LD C,B
49	LD C,C
4A	LD C,D
4B	LD C,E
4C	LD C,H
4D	LD C,L
0E 20	LD C,n
56	LD D,(HL)
DD 56 05	LD D,(IX+d)
FD 56 05	LD D,(IY+d)
57	LD D,A
50	LD D,B
51	LD D,C
52	LD D,D
53	LD D,E
54	LD D,H
55	LD D,L
16 20	LD D,n
ED 5B 84 05	LD DE,(nn)
11 84 05	LD DE,nn
5E	LD E,(HL)
DD 5E 05	LD E,(IX+d)
FD 5E 05	LD E,(IY+d)
5F	LD E,A
58	LD E,B
59	LD E,C
5A	LD E,D
5B	LD E,E
5C	LD E,H
5D	LD E,L
1E 20	LD E,n
66	LD H,(HL)
DD 66 05	LD H,(IX+d)
FD 66 05	LD H,(IY+d)
67	LD H,A

60	LD H,B
61	LD H,C
62	LD H,D
63	LD H,E
64	LD H,H
65	LD H,L
26 20	LD H,n
2A 84 05	LD HL,(nn)
21 84 05	LD HL,nn
ED 47	LD I,A
DD 2A 84 05	LD IX,(nn)
DD 21 84 05	LD IX,nn
FD 2A 84 05	LD IY,(nn)
FD 21 84 05	LD IY,nn
6E	LD L,(HL)
DD 6E 05	LD L,(IX+d)
FD 6E 05	LD L,(IY+d)
6F	LD L,A
68	LD L,B
69	LD L,C
6A	LD L,D
6B	LD L,E
6C	LD L,H
6D	LD L,L
2E 20	LD L,n
ED 4F	LD R,A
ED 7B 84 05	LD SP,(nn)
F9	LD SP,HL
DD F9	LD SP,IX
FD F9	LD SP,IY
31 84 05	LD SP,nn
ED A8	LDD
ED B8	LDDR
ED A0	LDI
ED B0	LDIR
ED 44	NEG
00	NOP
B6	OR (HL)
DD B6 05	OR (IX+d)
FD B6 05	OR (IY+d)
B7	OR A
B0	OR B
B1	OR C
B2	OR D
B3	OR E
B4	OR H
B5	OR L
F6 20	OR n
ED 8B	OTDR
ED B3	OTIR

ED 79	OUT (C),A
ED 41	OUT (C),B
ED 49	OUT (C),C
ED 51	OUT (C),D
ED 59	OUT (C),E
ED 61	OUT (C),H
ED 69	OUT (C),L
D3 20	OUT (n),A
ED AB	OUTD
ED A3	OUTI
F1	POP AF
C1	POP BC
D1	POP DE
E1	POP HL
DD E1	POP IX
FD E1	POP IY
F5	PUSH AF
C5	PUSH BC
D5	PUSH DE
E5	PUSH HL
DD E5	PUSH IX
FD E5	PUSH IY
CB 86	RES 0,(HL)
DD CB 05 86	RES 0,(IX+d)
FD CB 05 86	RES 0,(IY+d)
CB 87	RES 0,A
CB 80	RES 0,B
CB 81	RES 0,C
CB 82	RES 0,D
CB 83	RES 0,E
CB 84	RES 0,H
CB 85	RES 0,L
CB 8E	RES 1,(HL)
DD CB 05 8E	RES 1,(IX+d)
FD CB 05 8E	RES 1,(IY+d)
CB 8F	RES 1,A
CB 88	RES 1,B
CB 89	RES 1,C
CB 8A	RES 1,D
CB 8B	RES 1,E
CB 8C	RES 1,H
CB 8D	RES 1,L
CB 96	RES 2,(HL)
DD CB 05 96	RES 2,(IX+d)
FD CB 05 96	RES 2,(IY+d)
CB 97	RES 2,A
CB 90	RES 2,B
CB 91	RES 2,C

CB 92	RES 2,D
CB 93	RES 2,E
CB 94	RES 2,H
CB 95	RES 2,L
CB 9E	RES 3,(HL)
DD CB 05 9E	RES 3,(IX+d)
FD CB 05 9E	RES 3,(IY+d)
CB 9F	RES 3,A
CB 98	RES 3,B
CB 99	RES 3,C
CB 9A	RES 3,D
CB 9B	RES 3,E
CB 9C	RES 3,H
CB 9D	RES 3,L
CB A6	RES 4,(HL)
DD CB 05 A6	RES 4,(IX+d)
FD CB 05 A6	RES 4,(IY+d)
CB A7	RES 4,A
CB A0	RES 4,B
CB A1	RES 4,C
CB A2	RES 4,D
CB A3	RES 4,E
CB A4	RES 4,H
CB A5	RES 4,L
CB AE	RES 5,(HL)
DD CB 05 AE	RES 5,(IX+d)
FD CB 05 AE	RES 5,(IY+d)
CB AF	RES 5,A
CB A8	RES 5,B
CB A9	RES 5,C
CB AA	RES 5,D
CB AB	RES 5,E
CB AC	RES 5,H
CB AD	RES 5,L
CB B6	RES 6,(HL)
DD CB 05 B6	RES 6,(IX+d)
FD CB 05 B6	RES 6,(IY+d)
CB B7	RES 6,A
CB B0	RES 6,B
CB B1	RES 6,C
CB B2	RES 6,D
CB B3	RES 6,E
CB B4	RES 6,H
CB B5	RES 6,L
CB BE	RES 7,(HL)
DD CB 05 BE	RES 7,(IX+d)
FD CB 05 BE	RES 7,(IY+d)
CB BF	RES 7,A
CB B8	RES 7,B

CB B9	RES 7,C
CB BA	RES 7,D
CB BB	RES 7,E
CB BC	RES 7,H
CB BD	RES 7,L
C9	RET
D8	RET C
F8	RET M
D0	RET NC
C0	RET NZ
F0	RET P
E8	RET PE
E0	RET PO
C8	RET Z
ED 4D	RETI
ED 45	RETN
CB 16	RL (HL)
DD CB 05 16	RL (IX+d)
FD CB 05 16	RL (IY+d)
CB 17	RL A
CB 10	RL B
CB 11	RL C
CB 12	RL D
CB 13	RL E
CB 14	RL H
CB 15	RL L
17	RLA
CB 06	RLC (HL)
DD CB 05 06	RLC (IX+d)
FD CB 05 06	RLC (IY+d)
CB 07	RLC A
CB 00	RLC B
CB 01	RLC C
CB 02	RLC D
CB 03	RLC E
CB 04	RLC H
CB 05	RLC L
07	RLCA
ED 6F	RLD
CB 1E	RR (HL)
DD CB 05 1E	RR (IX+d)
FD CB 05 1E	RR (IY+d)
CB 1F	RR A
CB 18	RR B
CB 19	RR C
CB 1A	RR D
CB 1B	RR E
CB 1C	RR H
CB 1D	RR L
1F	RRA

CB 0E	RRC (HL)
DD CB 05 0E	RRC (IX+d)
FD CB 05 0E	RRC (IY+d)
CB 0F	RRC A
CB 08	RRC B
CB 09	RRC C
CB 0A	RRC D
CB 0B	RRC E
CB 0C	RRC H
CB 0D	RRC L
0F	RRCA
ED 67	RRD
C7	RST 00H
CF	RST 08H
D7	RST 10H
DF	RST 18H
E7	RST 20H
EF	RST 28H
F7	RST 30H
FF	RST 38H
DE 20	SBC A,n
9E	SBC A,(HL)
DD 9E 05	SBC A,(IX+d)
FD 9E 05	SBC A,(IY+d)
9F	SBC A,A
98	SBC A,B
99	SBC A,C
9A	SBC A,D
9B	SBC A,E
9C	SBC A,H
9D	SBC A,L
ED 42	SBC HL,BC
ED 52	SBC HL,DE
ED 62	SBC HL,HL
ED 72	SBC HL,SP
37	SCF
CB C6	SET 0,(HL)
DD CB 05 C6	SET 0,(IX+d)
FD CB 05 C6	SET 0,(IY+d)
CB C7	SET 0,A
CB C0	SET 0,B
CB C1	SET 0,C
CB C2	SET 0,D
CB C3	SET 0,E
CB C4	SET 0,H
CB C5	SET 0,L
CB CE	SET 1,(HL)
DD CB 05 CE	SET 1,(IX+d)
FD CB 05 CE	SET 1,(IY+d)

CB CF	SET 1,A
CB C8	SET 1,B
CB C9	SET 1,C
CB CA	SET 1,D
CB CB	SET 1,E
CB CC	SET 1,H
CB CD	SET 1,L
CB D6	SET 2,(HL)
DD CB 05 D6	SET 2,(IX+d)
FD CB 05 D6	SET 2,(IY+d)
CB D7	SET 2,A
CB D0	SET 2,B
CB D1	SET 2,C
CB D2	SET 2,D
CB D3	SET 2,E
CB D4	SET 2,H
CB D5	SET 2,L
CB DE	SET 3,(HL)
DD CB 05 DE	SET 3,(IX+d)
FD CB 05 DE	SET 3,(IY+d)
CB DF	SET 3,A
CB D8	SET 3,B
CB D9	SET 3,C
CB DA	SET 3,D
CB DB	SET 3,E
CB DC	SET 3,H
CB DD	SET 3,L
CB E6	SET 4,(HL)
DD CB 05 E6	SET 4,(IX+d)
FD CB 05 E6	SET 4,(IY+d)
CB E7	SET 4,A
CB E0	SET 4,B
CB E1	SET 4,C
CB E2	SET 4,D
CB E3	SET 4,E
CB E4	SET 4,H
CB E5	SET 4,L
CB EE	SET 5,(HL)
DD CB 05 EE	SET 5,(IX+d)
FD CB 05 EE	SET 5,(IY+d)
CB EF	SET 5,A
CB E8	SET 5,B
CB E9	SET 5,C
CB EA	SET 5,D
CB EB	SET 5,E
CB EC	SET 5,H
CB ED	SET 5,L
CB F6	SET 6,(HL)
DD CB 05 F6	SET 6,(IX+d)
FD CB 05 F6	SET 6,(IY+d)

CB F7	SET 6,A
CB F0	SET 6,B
CB F1	SET 6,C
CB F2	SET 6,D
CB F3	SET 6,E
CB F4	SET 6,H
CB F5	SET 6,L
CB FE	SET 7,(HL)
DD CB 05 FE	SET 7,(IX+d)
FD CB 05 FE	SET 7,(IY+d)
CB FF	SET 7,A
CB F8	SET 7,B
CB F9	SET 7,C
CB FA	SET 7,D
CB FB	SET 7,E
CB FC	SET 7,H
CB FD	SET 7,L
CB 26	SLA (HL)
DD CB 05 26	SLA (IX+d)
FD CB 05 26	SLA (IY+d)
CB 27	SLA A
CB 20	SLA B
CB 21	SLA C
CB 22	SLA D
CB 23	SLA E
CB 24	SLA H
CB 25	SLA L
CB 2E	SRA (HL)
DD CB 05 2E	SRA (IX+d)
FD CB 05 2E	SRA (IY+d)
CB 2F	SRA A
CB 28	SRA B
CB 29	SRA C
CB 2A	SRA D
CB 2B	SRA E
CB 2C	SRA H
CB 2D	SRA L
CB 3E	SRL (HL)
DD CB 05 3E	SRL (IX+d)
FD CB 05 3E	SRL (IY+d)
CB 3F	SRL A
CB 38	SRL B
CB 39	SRL C
CB 3A	SRL D
CB 3B	SRL E
CB 3C	SRL H
CB 3D	SRL L
96	SUB (HL)
DD 96 05	SUB (IX+d)

FD 96 05	SUB (IY+d)
97	SUB A
90	SUB B
91	SUB C
92	SUB D
93	SUB E
94	SUB H
95	SUB L
D6 20	SUB n
AE	XOR (HL)
DD AE 05	XOR (IX+d)
FD AE 05	XOR (IY+d)
AF	XOR A
A8	XOR B
A9	XOR C
AA	XOR D
AB	XOR E
AC	XOR H
AD	XOR L
EE 20	XOR n

APPENDIX 5

NEW NOTEPAD MODELS

As this book was going to press it was revealed that a new model called the NC150 is retailing in France and Italy and should be available soon in Britain. It has 512K ROM and 128K RAM (rather than the 256K ROM and 64K RAM of the NC100). The extra ROM contains a powerful spreadsheet and an arcade/action game similar in play to a popular game in which you have to fit falling shapes of different sizes into the smallest possible space. It is intended that the NC150 will eventually replace the NC100 as the entry-level model.

In addition, an NC200 should be launched by the time you read this. This machine will be further enhanced to include not only the additional features of the NC150 but also a PC-compatible 3.5in floppy disk drive and an increase in screen size from 8 to 16 lines by 80 characters - along with a back light to make it easier to read the display.

The back light can be toggled on and off but files cannot be written to or read directly from the floppy disk. Rather, the external drive has been designed as a backup mechanism for storing files or transferring them to a PC.

To do this, an extra option has been added to the menu displayed when you press [Function][L] to list files. This allows you to tag files using [Space] and then transfer them in bulk to and from a floppy disk. However, the floppy disk transfer functions are not available when the file selector is called from BBC Basic.

The main difference that programmers will have to cater to is direct screen addressing and Basic programs that only use eight lines (or 64 pixels depth). With regard to the memory map, when paging in the display ram, just remember that the bottom eight screen lines are effectively the same as the standard eight lines on the NC100.

To address the top eight lines you would start reading and writing to address &E000, as the screen now takes up addresses &E000-&FFFF when the 16K RAM display memory block is mapped in at &C000. Will we see addresses &C000-&DFFF used for future increases in screen resolution? Let's hope so. Particularly seeing as the NC200 folds open like a laptop and there should now be room for the larger screen area.

APPENDIX 6

EXTRAS

GET CONNECTED WITH LAPCAT

In order for you to transfer and backup programs between your Notepad and a desktop computer, the Lapcat communications software and lead is now available for the following computers:

- ❑ IBM and PC Compatibles
- ❑ Commodore Amiga
- ❑ Atari ST and TT
- ❑ Amstrad PCW
- ❑ A version for the Archimedes is planned

The price is £40 (valid throughout Europe)

EXPAND YOUR NOTEPAD WITH A RAM CARD

Now you can increase the storage area available on your Notepad by up to a megabyte with a RAM card. These are available in 64K, 128K, 256K, 512K, or 1Mb sizes. Please call for the current prices.

SAVE THE WEAR AND TEAR ON YOUR FINGERS – ORDER THE DISK OF THE BOOK

If you would like a copy of all the programs featured in this book, they are available for a range of computers on floppy disk for just £10. But remember you will need to have a copy of the Lapcat software and lead in order to transfer them to your Notepad.

ORDER FORM (may be photocopied)

Please send me:

[ ] The Lapcat software and lead, (£40.00)

[ ] The program disk of this book, (£10.00)

For the following format:

[ ] IBM and Compatible

[ ] Commodore Amiga

[ ] Atari ST and TT

[ ] Amstrad PCW

Disk Size: [ ] 3.5" [ ] 5.25"

[ ] I enclose a Cheque/PO for £_____ . ____

[ ] Please charge my Access/Visa card

Credit card number:

Expiry date: | / |

Name: _____ Signature: _____

Address: _____

_____

Post Code: _____Telephone: _____

Send to: Notepad Offer, Arnor Ltd., 611 Lincoln Road, Peterborough, Cambs, PE1 3HA. Tel: 0733 68909 (24 hours). Fax: 0733 67299.

Index

General

Music and New Technology – Georghiades and Jacobs – £12.95
Getting the Best from your Amstrad Notepad – Wilson – £12.95
Computers and Chaos (Atari and Amiga editions) – Bessant – £12.95
Computers in Genealogy – Isaac – £12.95
Multimedia, CD-ROM and Compact Disc – Botto – £14.95
Advanced Manufacturing Technology – Zairi – £14.95

Networks

$25 Network User Guide – Sinclair – £12.95
Integrated Digital Networks – Lawton – £24.95
Novell Netware Companion – Croucher – £16.95

PC Operating Systems and Architecture

Working with Windows 3.1 – Sinclair – £16.95
Servicing and Supporting IBM PCs and Compatibles – Moss – £16.95
The DR DOS Book – Croucher – £16.95
MS-DOS Revealed – Last – £12.95
PC Architecture and Assembly Language – Kauler – £16.95
Programmer's Technical Reference – Williams – £19.95
MS-DOS File and Program Control – Sinclair – £12.95
Mastering DesqView – Sinclair – £12.95

Programming

C Applications Library – Pugh – £16.95
Starting MS-DOS Assembler – Sinclair – £12.95
Understanding Occam and the transputer – Ellison – £12.95
Programming in ANSI Standard C – Horsington – £14.95
Programming in Microsoft Visual Basic – Penfold – £16.95
For **LOGO**, *see Amstrad PCW*

UNIX and mainframes

UNIX – The Book – Banahan and Rutter – £11.95
UNIX – The Complete Guide – Manger – £19.95
RPG on the IBM AS/400 – Tomlinson – £24.95

HOW TO ORDER

Prices correct for 1993.
Order these books from your usual bookshop, or direct from:

SIGMA PRESS,
1 SOUTH OAK LANE,
WILMSLOW, CHESHIRE, SK9 6AR

PHONE: 0625 – 531035; FAX: 0625 – 536800

PLEASE ADD £1 TOWARDS POST AND PACKING FOR ONE BOOK.
POSTAGE IS FREE FOR TWO OR MORE BOOKS.
OVERSEAS ORDERS: please pay by credit card; we will add airmail postage at actual cost

CHEQUES SHOULD BE MADE PAYABLE TO **SIGMA PRESS.**

ACCESS AND VISA WELCOME – 24 HOUR ANSWERPHONE SERVICE.

"PACKED WITH PROGRAMS – FULL OF INFORMATION"

The *"Advanced User Guide"* takes over from where the manual left off. It tells you absolutely everything there is to know about the inner workings of the Amstrad Notepad range, including:

- ★ professional hints, tips and "undocumented" commands
- ★ grabbing Notepad screens and displaying them on a PC
- ★ writing programs on a PC and running them on a Notepad

All this, plus complete lists of Z80 assembler instructions and dozens of program listings, ready to type in. And if that isn't enough, there's a disk offer inside this book to save time and effort.

This is **Robin Nixon's** second book for Sigma Press, and Sigma's second book for the Notepad. For even more hints, tips and programs – be sure to buy *"How to Program the Amstrad NC100 Notepad"*, by **Patrick Hall**, also published by Sigma.

About Sigma Press:

We publish a wide range of books on all aspects of computing. Please write or phone for a complete catalogue:

Sigma Press,
1 South Oak Lane,
Wilmslow,
Cheshire SK9 6AR

Phone: 0625 - 531035

We welcome new authors.

ISBN 1-85058-515-6

SIGMA