14 Advanced Programming Techniques

To give your programs that really professional quality it is essential to make them *user friendly*. This is one of the few pieces of advertising jargon that actually bears any relation to reality: it is essential to make programs easy to use, not just for yourself but for other people. We have all returned to programs written in a hurry three months previously, only to find that they are so badly structured/ commented that we cannot understand them. It is good programming practice to comment all but the most trivial programs, as well as to make their output self-explanatory. Mode 7 listings enable you to introduce colour codes that highlight sections of the program. REMarks do take up a great deal of space in the memory, but you must distinguish between a listing for general distribution and heavily used working programs. If you take a sensible approach at the beginning, and plan your programs, you can save a great deal of time and effort later on. For example, our placing of all BASIC statements on lines ending in 0 and REMs on lines ending in 9 makes the REM stripper (AUT09, 10) a simple way to turn a readable program into a memory-wise efficient one. Do not take the need to save store to ludicrous extremes; space out statements and never place too many BASIC statements on any one line – this makes programs incomprehensible. Ensure that the prompts displayed while you are actually RUNning the program are clear and concise. Another simple way of providing help is to include an introductory instruction routine, as found on many video games.

In programs where a set of routines may be used in any order or combination, the usual method of providing for selection between options is the menu (such as the CHARACTER GENERATOR 1 program, listing 5.6). Provided that the prompts are appropriate to the actions that they initiate, then this method is especially useful for people who do not understand the details of the program and are using it only as a drawing tool. Common-sense plays its part in deciding what prompts should be issued. Avoid such classic misprompts as PRESS 1 FOR DUPLICATE DATA OR 2 FOR SINGLE DATA. If possible use cursor keys for movements about the screen (see option 3 of CHARACTER GENERATOR 1); this will seem natural to any regular user of the BBC micro.

As a rule it is best to write programs in modules. In this way each module can be tested individually and errors can be traced more quickly. Placing a coloured REMark before each procedure, routine or function allows us to isolate sections of the program and then read, correct and adapt them with ease.

A common-sense practical approach saves a great deal of time in the long run. It is also very useful to understand exactly how the operating system stores and deals with BASIC programs. This is dealt with later in this chapter but first we introduce a simple disassembler which can be used to look at the storage of such programs. This disassembler is also extremely useful for rescuing programs that haw become corrupted, for debugging assembly language programs or for understanding the operating system.

Listing 14.1

```
  10 OSBYTE=&FFF4 : OSWORD=&FFF1 : OSASCI=&FFE3
  20 OSRDCH=&FFE0 : CLI=&FFF7
  30 IF ?&8015=49 THEN HEXSP=&856A : HEX=&8570 ELSE HEXSP=&B562: HEX=&B545
  40 OUT=&70 : PUT=&71 : SP=&72 : BPT=&74
  50 CONBLK=&76 : HI=&81 : LO=&80 : DISP=&82 : DH=&83
  60 UP=&85 : STK=&86 : ACR=&84
  70 OLO=&7B : OHI=&7C
  80 HA=&8D : HB=&8E : HC=&8F
  90 DLINES=&7D : NUMB=&7E : TEMPSTACK=&7F
 100 FOR OP=0 TO 3 STEP 3
 110 P%=&7000
 120 [OPT OP
 130 TSX : STX TEMPSTACK
 140 \ make sure BASIC rom is paged in since we will use it \
 150 LDX#0:.ROM STX &FE30:LDA &8009:INX: CMP #ASC("B"):BNE ROM
 160 \ set &600 as address for string for OSWORD with A=0 \
 170 LDA #0 : STA CONBLK : LDA #6 : STA CONBLK+1
 180 \ set display mode to Dump mode \
 190 LDA #ASC("D") : STA DISP
 200 \ ST1 is string which sets mode 7 and prints instructions \
 210 LDX #(ST1 MOD 256)
 220 LDY #(ST1 DIV 256)
 230 JSR OUTPUT
 240 \ *FX4,1 sets cursor keys to produce ascii codes \
 250 LDA #4 : LDX #1 : JSR OSBYTE
 260
 270
 280 \ main loop, update display then deal with commands \
 290 .L5 : JSR DISPLAY
 300 \ get key, if it's a command then execute else try again \
 310 .L7 : JSR KEYIN
 320 \ deal with commands common to both modes \
 330 CMP #ASC("D") : BNE NOND
 340 \ set display mode to either D or L \
 350 .STD : STA DISP : JMP L5
 360 .NOND : CMP #ASC("L") : BEQ STD
 370
 380 \ if H command then input hex address \
 390 CMP #ASC("H") : BNE NONH
 400 \ begin H command, PRINT TAB(15,2); \
 410 .HGIN : LDX #15 : LDY #2 : JSR SETAB
 420 \ input 4 characters in range 0-F \
 430 LDA #4 : LDX #ASC("0") : LDY #ASC("F") : JSR STRIN
 440 \ convert first two chars to one byte, if error then reinput \
 450 JSR DHEX : BCC HGIN : LDA DH : STA HI
 460 \ convert next two chars to one byte \
 470 LDA &602 : STA &600 : LDA &603 : STA &601 : JSR DHEX
 480 \ if error then reinput \
 490 BCC HGIN : LDA DH : STA LO : JMP L5
 500
 510 \ check display mode if it's L then do L COMmands \
 520 .NONH : LDX DISP : CPX #ASC("L") : BNE NONF : JMP LCOM
 530
 540 \ commands only available in Dump mode \
 550 \ check for cursor keys and alter address accordingly \
 560 .NONF : CMP #&88 : BNE NLEFT
 570 \ cursor left, take one off \
 580 LDA LO : BNE DE1 : DEC HI : .DE1 : DEC LO : JMP L5
 590
 600 .NLEFT  : CMP #&89 : BNE NRT
 610 \ cursor right, add one on \
 620 INC LO : BNE DE2 : INC HI : .DE2 : JMP L5
 630
 640 .NRT : CMP #&8A : BNE NDOWN
 650 \ cursor down, add eight on \
 660 LDA LO : CLC : ADC #8 : STA LO
 670 LDA HI : ADC #0 : STA HI : JMP L5
 680
 690 .NDOWN : CMP #&8B : BNE NMOVE
 700 \ cursor up, take eight off \
 710 LDA LO : SEC : SBC #8 : STA LO
 720 LDA HI : SBC #0 : STA HI : JMP L5
 730
 740 \ not a cursor movement \
 750 .NMOVE : CMP #ASC("A") : BNE NONA
 760 \ start of A command to ALTER a byte \
 770 .AGIN : LDY #14
 780 \ ACR is no of bytes ACRoss cursor is on display \
 790 LDA ACR : ASL A : CLC : ADC ACR : CLC : ADC #6
 800 \ set text position TAB(ACR*3+6,14); \
 810 TAX : JSR SETAB
 820 \ get two characters between 0 and F \
 830 LDA #2 : LDX #ASC("0") : LDY #ASC("F") : JSR STRIN
 840 \ convert chars to byte value, store if valid or reinput \
 850 JSR DHEX : BCC AGIN : LDA DH : LDY #0 : STA (LO),Y
 860 JMP L5
 870
 880
 890 \ characters to hex conversion routine \
 900 .DHEX : LDA &600 : CMP #ASC("9")+1 : BCS AL1
 910 AND #&0F : BCC TIP
 920 .AL1 : CMP #ASC("A") : BCC ERHEX : SBC #ASC("A")-10
 930 \ put value of first hex digit in top half of byte \
 940 .TIP : ASL A : ASL A : ASL A : ASL A : STA DH
 950 LDA &601 : CMP #ASC("9")+1 : BCS AL2
 960 AND #&0F : BCC BOT
 970 .AL2 : CMP #ASC("A") : BCC ERHEX : SBC #ASC("A")-10
 980 \ mix value of second digit in as bottom half \
 990 .BOT : ORA DH : STA DH : SEC
1000 \ if error detected then carry flag is unset \
1010 .ERHEX : RTS
1020
1030
1040 \ back to checking commands \
1050 .NONA : CMP #ASC("S") : BNE NONS
1060 \ start of S command, set text position to ascii dump \
1070 .SGIN : LDY #14 : LDA ACR : CLC : ADC #30 : TAX : JSR SETAB
1080 \ get up to 255 characters between blank and alpha white \
1090 LDA #255 : LDX #ASC(" ") : LDY #&87 : JSR STRIN
1100 \ copy all characters from buffer to memory \
1110 LDX #0 : LDY #0
1120 \ stop when we find a carriage return \
1130 .LOP : LDA &600,X : CMP #&0D : BEQ FINLOP
1140 STA (LO),Y : INY : INX : BNE LOP
1150 .FINLOP : JMP L5
1160
1170 \ arrive here if keypress was nonsense \
1180 .NONS
1190 JMP L7
1200
1210
1220 \ deal with commands for listing mode \
1230 .LCOM : CMP #&0D : BNE NONS
1240 \ only return is valid, reset start to one line lower \
1250 LDA OLO : STA LO : LDA OHI : STA HI : JMP L5
1260
1270
1280 \ disassemble routine used in Listing mode \
1290 .DIS : LDX #(ST2 MOD 256)
1300 LDY #(ST2 DIV 256)
1310 \ ST2 removes cursor and sets print position to (0,7) \
1320 JSR OUTPUT
1330 \ set number of lines counter \
1340 LDA #16 : STA DLINES
1350 .LINLOP
1360 \ output address + space \
1370 LDA HI : JSR HEX : LDA LO : JSR HEXSP
1380 \ copy following three bytes to buffer \
1390 \ three bytes in question are now HA,HB,HC \
1400 LDY #0 : LDX #3
1410 .HALOP : LDA (LO),Y : STA HA,Y : INY : DEX : BNE HALOP
1420 \ from opcode type find number of bytes in instruction \
1430 LDX HA : LDA TYPE,X : TAX : LDA NUM,X
1440 \ move address on to next instruction \
1450 STA NUMB : CLC : ADC LO : STA LO : LDA #0 : ADC HI : STA HI
1460 \ output bytes of instruction then... \
1470 LDX NUMB : LDY #3 : LDA HA : JSR HEXSP : DEY : DEX : BEQ PAD
1480 LDA HB : JSR HEXSP : DEY : DEX : BEQ PAD
1490 LDA HC : JSR HEXSP : DEX : BEQ MNEM
1500 \ pad with spaces if necessary \
1510 .PAD : TYA : PHA
1520 LDX #(ST4 MOD 256) : LDY #(ST4 DIV 256) : JSR OUTPUT
1530 PLA : TAY : DEY : BNE PAD
1540 \ add one more space before mnemonic \
1550 .MNEM : LDA #ASC(" ") : JSR OSASCI
1560 \ find which mnemonic starts instruction \
1570 LDX HA : LDA HASH,X : ASL A : ASL A : STA OUT
1580 \ multiply by 4 to find address in string of mnemonics \
1590 LDA #(STRING DIV 256) : STA PUT : LDX #4 : LDY #0
1600 \ output four chars, first one is colour code \
1610 .MNMLOP : LDA (OUT),Y : JSR OSASCI : INY : DEX : BNE MNMLOP
1620 LDA #ASC(" ") : JSR OSASCI
1630 \ form rest of instruction depending on type \
1640 LDX HA : LDA TYPE,X : CMP #1 : BNE NOD1
1650 \ type 1, immediate so format #&HH \
1660 LDA #ASC("#") : JSR OSASCI : LDA #ASC("&") : JSR OSASCI
1670 \ Y is no. of spaces to make format 10 wide \
1680 LDA HB : JSR HEXSP : LDY #5 : JMP NDLINE
1690
1700 .NOD1 : CMP #2 : BNE NOD2
1710 \ type 2, absolute     format &HHHH \
1720 LDA #ASC("&") : JSR OSASCI : LDA HC : JSR HEX
```

```
1730 LDA HB : JSR HEXSP : LDY #4 : JMP NDLINE
1740
1750 .NOD2 : CMP #3 : BNE NOD3
1760 \ type 3, zero page    format &HH \
1770 LDA #ASC("&") : JSR OSASCI : LDA HB : JSR HEXSP
1780 LDY #6 : JMP NDLINE
1790
1800 .NOD3 : CMP #4 : BNE NOD4
1810 \ type 4, accumulator  format A \
1820 LDA #ASC("A") : JSR OSASCI : LDY #9 : JMP NDLINE
1830
1840 .NOD4 : CMP #5 : BNE NOD5
1850 \ type 5, implied      format   \
1860 LDY #10 : JMP NDLINE
1870
1880 .NOD5 : CMP #6 : BNE NOD6
1890 \ type 6, indirect x   format (&HH,X) \
1900 LDA #ASC("(") : JSR OSASCI : LDA #ASC("&") : JSR OSASCI
1910 LDA HB : JSR HEX : LDA #ASC(",") : JSR OSASCI
1920 LDA #ASC("X") : JSR OSASCI : LDA #ASC(")") : JSR OSASCI
1930 LDY #3 : JMP NDLINE
1940
1950 .NOD6 : CMP #7 : BNE NOD7
1960 \ type 7, indirect y   format (&HH),Y \
1970 LDA #ASC("(") : JSR OSASCI : LDA #ASC("&") : JSR OSASCI
1980 LDA HB : JSR HEX : LDA #ASC(")") : JSR OSASCI
1990 LDA #ASC(",") : JSR OSASCI : LDA #ASC("Y") : JSR OSASCI
2000 LDY #3 : JMP NDLINE
2010
2020 .NOD7 : CMP #8 : BNE NOD8
2030 \ type 8, zero page x  format &HH,X \
2040 LDA #ASC("&") : JSR OSASCI : LDA HB : JSR HEX
2050 LDA #ASC(",") : JSR OSASCI : LDA #ASC("X") : JSR OSASCI
2060 LDY #5 : JMP NDLINE
2070
2080 .NOD8 : CMP #9 : BNE NOD9
2090 \ type 9, absolute x   format &HHHH,X \
2100 LDA #ASC("&") : JSR OSASCI : LDA HC : JSR HEX
2110 LDA HB : JSR HEX : LDA #ASC(",") : JSR OSASCI
2120 LDA #ASC("X") : JSR OSASCI
2130 LDY #3 : JMP NDLINE
2140
2150 .NOD9  : CMP #&A : BNE NODA
2160 \ type A, absolute y   format &HHHH,Y \
2170 LDA #ASC("&") : JSR OSASCI : LDA HC : JSR HEX
2180 LDA HB : JSR HEX : LDA #ASC(",") : JSR OSASCI
2190 LDA #ASC("Y") : JSR OSASCI
2200 LDY #3 : JMP NDLINE
2210
2220 .NODA  : CMP #&B : BNE NODB
2230 \ type B, relative     format &HHHH \
2240 LDA #ASC("&") : JSR OSASCI
2250 \ for relative must calculate address from displacement \
2260 LDA HB : CMP #&7F : BCS BACK
2270 \ calculation for forward branch, leave lo-byte on stack \
2280 CLC : ADC LO : PHA : LDA #0 : ADC HI : CLC : BCC REL
2290 \ calculation for backward branch, leave lo-byte on stack \
2300 .BACK : EOR #255 : ADC #0 : STA &70 : LDA LO : SEC : SBC &70
2310 PHA : LDA HI : SBC #0
2320 \ output hi-byte then get lo-byte and print that \
2330 .REL : JSR HEX : PLA : JSR HEXSP
2340 LDY #4 : JMP NDLINE
```

```
2350
2360 .NODB : CMP #&C : BNE NODC
2370 \ type C, indirect     format (&HHHH) \
2380 LDA #ASC("(") : JSR OSASCI : LDA #ASC("&") : JSR OSASCI
2390 LDA HC : JSR HEX : LDA HB : JSR HEX
2400 LDA #ASC(")") : JSR OSASCI : LDY #3 : JMP NDLINE
2410
2420 .NODC : CMP #&D : BNE NODD
2430 \ type D, zero page y  format &HH,Y \
2440 LDA #ASC("&") : JSR OSASCI : LDA HB : JSR HEX
2450 LDA #ASC(",") : JSR OSASCI : LDA #ASC("Y") : JSR OSASCI
2460 LDY #5 : JMP NDLINE
2470
2480 .NODD : LDY #10
2490
2500 .NDLINE : LDA #ASC(" ") : JSR OSASCI : DEY : BNE NDLINE
2510 \ output alpha white control code \
2520 LDA #&87 : JSR OSASCI
2530 \ output ascii equivalents of bytes in instruction \
2540 LDX NUMB : LDY #0
2550 .DISPLOP LDA HA,Y : JSR CHOUT :INY : DEX : BNE DISPLOP
2560 \ if necessary pad with spaces to standard width \
2570 LDA #ASC(" ") : .PAD2 : JSR OSASCI : INY : CPY #4 : BNE PAD2
2580 \ output sufficient spaces to get ready for next line \
2590 LDX #5 : .PAD3 : JSR OSASCI : DEX : BNE PAD3
2600 \ if this was the first line then the address is where we \
2610 \ want to start next time if return is pressed \
2620 LDA DLINES : CMP #16 : BNE OVOLD
2630 \ so store it somewhere \
2640 LDA LO : STA OLO :  LDA HI : STA OHI
2650 \ keep doing lines until 16 done \
2660 .OVOLD : DEC DLINES : BEQ FINLIN : JMP LINLOP
2670 .FINLIN : RTS
2680
2690
2700 \ display routine calls either disassembler or dump \
2710 .DISPLAY : LDY #2 : LDX #15 : JSR SETAB
2720 \ update address display \
2730 LDA HI : JSR HEX : LDA LO : JSR HEXSP
2740 \ check whether Listing mode or not \
2750 LDA DISP : CMP #ASC("L") : BEQ DDISP
2760 \ Dump mode just shows contents of memory \
2770 JSR SHOW : RTS
2780 \ Listing mode produces disassembled display \
2790 .DDISP : JSR DIS : RTS
2800
2810
2820 \ show routine \
2830 .SHOW : LDX #(ST2 MOD 256)
2840 LDY #(ST2 DIV 256)
2850 \ ST2 gets rid off cursor and sets print position to (0,7) \
2860 JSR OUTPUT
2870 \ calculate byte at top so that active byte is central \
2880 LDA LO : AND #&F8 : SEC : SBC #64 : STA UP
2890 LDA HI : SBC #0 : STA STK
2900 \ find how many across to active byte \
2910 LDA LO : AND #7 : STA ACR
2920 \ for X=0 to 15 , address=address+8 \
2930 LDX #0 : .L4 : STX STK+1 : LDA #8 : CLC : ADC UP : STA UP
2940 LDA #0 : ADC STK : STA STK
2950 \ output address plus space \
2960 JSR HEX : LDA UP : JSR HEXSP : LDA #ASC(" ") : JSR OSASCI
```

```
2970 \ for XX=0 to 7 \
2980 LDX #0 : .L2 : STX STK+2 : TXA
2990 TAY
3000 \ output each byte across line \
3010 LDA (UP),Y : JSR HEXSP
3020 \ next XX \
3030 LDX STK+2 : INX : CPX #8 : BNE L2
3040 \ for XX=0 to 7 \
3050 LDX #0 : .L3 :STX STK+2 : TXA
3060 \ output each byte as ascii equivalent \
3070 TAY
3080 LDA (UP),Y : JSR CHOUT
3090 \ next XX \
3100 LDX STK+2 : INX : CPX #8 : BNE L3
3110 \ move on to next line \
3120 LDA #ASC(" ") : JSR OSASCI : JSR OSASCI
3130 \ next X \
3140 LDX STK+1 : INX : CPX #16 : BNE L4
3150 \ calculate position of active byte in display \
3160 LDY #14 : LDA ACR : ASL A : CLC : ADC ACR
3170 ADC #5 : STA STK+1 : TAX : JSR SETAB
3180 \ put an alpha green in front of it \
3190 LDA #&82 : JSR OSASCI
3200 LDA STK+1 : CLC : ADC #3 : TAX : LDY #14 : JSR SETAB
3210 \ put an alpha white after it \
3220 LDA #&87 : JSR OSASCI
3230 RTS
3240
3250
3260 \ chout routine outputs ascii equivalent of byte \
3270 .CHOUT : CMP #ASC(" ") : BCC DUF
3280 \ control codes or codes above &88 are not printed \
3290 CMP #&88 : BCS DUF
3300 \ delete is not printed \
3310 CMP #127 : BNE OK
3320 \ replace unprintables codes with a dot \
3330 .DUF : LDA #ASC(".")
3340 .OK : JSR OSASCI : RTS
3350
3360
3370 \ set tab position by printing ST3 with X,Y in it \
3380 .SETAB : STY ST3+2 : STX ST3+1
3390 LDX #(ST3 MOD 256) : LDY #(ST3 DIV 256)
3400 JSR OUTPUT : RTS
3410
3420
3430 \ keyin clears the buffer and waits for a key \
3440 .KEYIN : LDX #0 : LDA #21 : JSR OSBYTE
3450 JSR OSRDCH : BCS KEYERR : RTS
3460 \ if an error has occurred then reset cursor keys \
3470 .KEYERR : STA STK+1
3480 LDX #(ST6 MOD 256) : LDY #(ST6 DIV 256) : JSR CLI
3490 LDA STK+1 : CMP #&1B : BNE KERR
3500 \ acknowledge escape \
3510 LDA #&7E : JSR OSBYTE
3520 .KERR
3530 \ set the print position, reset the stack \
3540 LDX #0 : LDY #22 : JSR SETAB : LDX TEMPSTACK : TXS : BRK
3550 \ use break to print error message \
3560 ]
3570 $(P%+1)="End of DIS" : P%=P%+11
3580 [OPT OP
```

```
3590 \ end of error marked by break \
3600 BRK
3610
3620
3630 \ output a string of codes ending with an &FF \
3640 .OUTPUT : STX &70 : STY &71
3650 LDY #0
3660 .LOOP : LDA (&70),Y : CMP #&FF : BEQ OUTEND
3670 JSR OSASCI : INY : BNE LOOP
3680 .OUTEND : RTS
3690
3700
3710 \ store parameters in control block and use OSWORD \
3720 \ with A=1 to get a string \
3730 .STRIN : STA CONBLK+2 : STX CONBLK+3 : STY CONBLK+4
3740 LDX #(ST5 MOD 256)
3750 LDY #(ST5 DIV 256) : JSR OUTPUT
3760 LDA #0 : LDX #CONBLK : TAY : JSR OSWORD
3770 BCS ESC1 : RTS
3780 .ESC1 : LDA #&1B : JMP KEYERR
3790 ]
3800 ST1=P%+1 : !ST1=&0D0C0716 : P%=P%+5
3810 $P%=" BBC Disassembler    D Dump memory      "
3820 $(P%+40)=" H Hex addr. : 0000  L Listing          "
3830 $(P%+80)="                     A Alter byte       "
3840 $(P%+120)="                     S String"
3850 P%=P%+LEN($P%) : ?P%=&FF
3860 ST2=P%+1 : !ST2=&00000117 : ST2!4=0 : P%=P%+9
3870 !P%=&001F0000 : P%=P%+4 : !P%=&FF07 : P%=P%+2
3880 ST3=P% : !P%=&0000001F : P%!4=&FF000000 : P%=P%+8
3890 ST4=P% : !P%=&FF202020 : P%=P%+4
3900 ST5=P% : !P%=&00010117 : ST5!4=0 : P%=P%+8
3910 !P%=&FF000000 : P%=P%+4
3920 ST6=P% : $ST6="FX4,0" : P%=P%+6
3930 RESTORE
3940 NUM=P% : FOR I=0 TO 14 : READ N% : P%?I=N% : NEXT I
3950 P%=P%+15
3960 P%=(P% DIV 256)+1 : P%=P%*256
3970 HASH=P% : TYPE=P%+256 : P%=P%+512 : STRING=P%
3980 NEXT OP
3990 FOR I%=0 TO 255 STEP 4 : HASH!I%=0 : TYPE!I%=0 : STRING!I%=0
: NEXT I%
4000 REPEAT
4010 READ A$ : IF A$="*" THEN 4050
4020 IF ASC(A$)=ASC("-") THEN I=ABS(VAL(A$)) : GOTO 4010
4030 M=EVAL("&"+LEFT$(A$,1)) : MN=EVAL("&"+RIGHT$(A$,2))
4040 HASH?MN=I : TYPE?MN=M
4050 UNTIL A$="*"
4060 A$=" ___ ADC AND ASL BCC BCS BEQ BIT BMI BNE"
4070 A$=A$+" BPL BRK BVC BVS CLC CLD CLI CLV CMP CPX"
4080 A$=A$+" CPY DEC DEX DEY EOR INC INX INY JMP JSR"
4090 A$=A$+" LDA LDX LDY LSR NOP ORA PHA PHP PLA PLP"
4100 A$=A$+" ROL ROR RTI RTS SBC SEC SED SEI STA STX"
4110 A$=A$+" STY TAX TAY TSX TXA TXS TYA"
4120 $STRING=A$
4130 END

4140 DATA1,2,3,2,1,1,2,2,2,3,3,2,3,2,1
4150 DATA-1,169,26D,365,661,771,875,97D,A79
4160 DATA-2,129,22D,325,621,731,835,93D,A39
4170 DATA-3,20E,306,40A,816,91E
4180 DATA-4,B90
```

```
4190 DATA-5,BB0
4200 DATA-6,BF0
4210 DATA-7,22C,324
4220 DATA-8,B30
4230 DATA-9,BD0
4240 DATA-10,B10
4250 DATA-11,500
4260 DATA-12,B50
4270 DATA-13,B70
4280 DATA-14,518
4290 DATA-15,5D8
4300 DATA-16,558
4310 DATA-17,5B8
4320 DATA-18,1C9,2CD,3C5,6C1,7D1,8D5,9DD,AD9
4330 DATA-19,1E0,2EC,3E4
4340 DATA-20,1C0,2CC,3C4
4350 DATA-21,2CE,3C6,7D6,9DE
4360 DATA-22,5CA
4370 DATA-23,588
4380 DATA-24,149,24D,345,641,751,855,95D,A59
4390 DATA-25,2EE,3E6,7F6,9FE
4400 DATA-26,5E8
4410 DATA-27,5C8
4420 DATA-28,24C,C6C
4430 DATA-29,220
4440 DATA-30,1A9,2AD,3A5,6A1,7B1,8B5,9BD,AB9
4450 DATA-31,1A2,2AE,3A6,ABE,DB6
4460 DATA-32,1A0,2AC,3A4,8B4,9BC
4470 DATA-33,24E,346,44A,856,95E
4480 DATA-34,5EA
4490 DATA-35,109,20D,305,601,711,815,91D,A19
4500 DATA-36,548
4510 DATA-37,508
4520 DATA-38,568
4530 DATA-39,528
4540 DATA-40,22E,326,42A,836,93E
4550 DATA-41,26E,366,46A,876,97E
4560 DATA-42,540
4570 DATA-43,560
4580 DATA-44,1E9,2ED,3E5,6E1,7F1,8F5,9FD,AF9
4590 DATA-45,538
4600 DATA-46,5F8
4610 DATA-47,578
4620 DATA-48,28D,385,681,791,895,99D,A99
4630 DATA-49,28E,386,D96
4640 DATA-50,28C,384,894
4650 DATA-51,5AA
4660 DATA-52,5A8
4670 DATA-53,5BA
4680 DATA-54,58A
4690 DATA-55,59A
4700 DATA-56,598,*
```

The Disassembler

It is often more reliable to do the job of rescuing programs by hand but it is always useful to have a disassembler that gives a convenient way ofexamining the memory and altering it. Running listing 14.1 creates a machine-code version ofthe disassembler program and stores it, starting at the location given by variable P% (line 80), which we set at &7000. It also stores this code in a file called DIS. You can run the code directly by typing CALL &7000 or by typing *DIS which loads the code from backing store and runs it.

The micro immediately switches to teletext mode and displays a simple menu (coloured blue), the hexadecimal address of the *currently active byte* (in green) and a dump (in white). This dump is in the standard format generated by the *DUMP command of the disk filing system. It shows the contents of 128 consecutive locations as 16 rows, each row containing (a) an address, (b) the contents of eight consecutive bytes of memory and (c) the characters equivalent to these bytes when treated as ASCII codes (a dot denotes an unprintable code). The address, which is always a multiple of 8, is the location of the first byte in the row. The currently active byte is coloured green and appears in the middle of the dump. We can return to the dump at any time by typing D.

The currently active byte can be changed by typing H, which places a flashing cursor on the screen, thus signifying a request for four hexadecimal digits (it must be exactly four). A new dump centred at this new byte is then displayed. Your can also change the active byte via the cursor keys; the screen scrolls where necessary to keep the active byte in the middle of the screen.

You can alter the value of the currently active byte (and the equivalent ASCII character) by typing A followed by two hexadecimal digits, or by typing S which allows you to replace the ASCII dump with a new string; naturally the machine makes the equivalcnt alterations to the bytes.

Option L changes the machine code into assembler mnemonics and gives a listing of 16 assembly codes; the code for the currently active byte is at the top of the screen. Immediately the currently active byte is redefined to be the byte that follows the last instruction that was displayed on the screen. Type RETURN and the code that contains this byte scrolls up on to the screen and the currently active byte is changed to one byte after the last code displayed. Typing L again displays the code that contains the currently active byte at the top of the screen and then changes its values in the same way as before so that it is just off the screen. H can be used to alter the currently active byte directly. If the currently active byte is not the first byte ofa valid assembler instruction then three light-blue dashes are displayed when it is brought on to the screen. The locations that start at &7800 hold the table of mnemonics that are used by the List option of the disassembler. Each mnemonic is stored as four bytes, one for a colour code and three for the mnemonic characters. JMP has been set to red, JSR to green, branching instructions to yellow, BRK to purple, invalid bytes and NOP to light blue, RTI and RTS to dark blue. This allows us to colour code the mnemonics and makes it easy fur us to recognise these codes when Listed. You can change these colour codes if you wish by using the disassembler – you can even add flash.

Exercise 14.1
Experiment with this disassembler. LOAD a simple BASIC program at PAGE = &1900 and then type *DIS (having previously RUN listing 14.1). Set the currently active byte to &1900 and read through the BASIC program byte by byte to see how such a program is stored. Also see the sample input at the end of the chapter.

The Structure of BASIC Programs

The previous exercise will have raised a number of questions about the way the BBC microcomputer stores BASIC programs. You will have noticed that the programs are stored line by line, starting with a cursor-return code (13) at location PAGE. Each line starts with the line number (16 bits) which takes up two locations, the hi-byte followed by the lo-byte; then comes a description of the line as a sequence of characters or TOKENs (see user guide) each of which takes up one byte, and finally a cursor-return (13). Apart from the line number all other numeric values are stored in a standard lo-hi format.

Example 14.1
To illustrate this idea we enter the trivial program that consists of a single line 10 REM and use the indirection operator ‘?'. If we type

```
PRINT ? PAGE, ? (PAGE + 1), ? (PAGE + 2)
```

we shall see that the values 13, 0 and 10 are stored in locations PAGE, PAGE + 1 and PAGE + 2 respectively. If we replace line 10 with another trivial statement, 34 REM, we shall see that the representation of this new number is now stored in the latter two locations.

The third byte for a line gives the length of the present line (the number of bytes including the line number, the third byte itself, the codes for the line and the final cursor-return) and it can be used to find the start of the next program line without traversing the whole ofthe present line. Whenever a reference is made to another line (for example, in a GOTO or THEN) then this line number is not stored directly but in a devious way. It is given as four bytes: a token &8D to indicate a reference and three 8-bit digits which are manipulated to give the required line number. The end of a program is indicated by ASCII 255 (&FF) following th$ ASCII 13 ofthe final line.

We can get the operating system to print out the equivalent keyword for each token (or a character if not a token) by calling the routine at &B53A in the ROM which accesses a table stored after location &806D. You should use the disassembler to investigate this table.

A Search/Replace and Listing Program

We have all written programs to which on completion we would have preferred that different variable names had been given. Also when debugging a program it is difficult to locate the occurrence of specific variables. By using the information about how BASIC programs are stored we can write a program (listing 14.2) that will list any program, including itself. It also searches out text strings in the program and if required replaces them with alternative strings.

Listing 14.2

```
 10 MODE 7
 20 INPUT"SEARCH ",A$ : SEARCH=(A$<>"")
 30 IF NOT SEARCH THEN REPLACE=FALSE : GOTO 50
 40 INPUT"REPLACE ",B$ : REPLACE=(B$<>"")
 49 REM use BASIC ROM routine to get keywords from tokens
 50 IF ?&8015=49 THEN DECRUNCH=&B53A ELSE DECRUNCH=&B50E
 60 INPUT"LOCATION OF PROGRAM ",P$
 70 IF LEFT$(P$,1)="&" THEN P%=EVAL(P$) ELSE P%=VAL(P$)
 79 REM assume disk system is used
 80 IF P%=0 THEN P%=&1900
 89 REM find end of program
 90 Q%=P% : REPEAT : Q%=Q%+1 : UNTIL ?Q%=&FF
 99 REM if first byte of line number is 255 then stop
100 P%=P%+1 : I=?P% : IF I=255 THEN END
110 P%=P%+1 : I=I*256+?P%
119 REM pad line number with blanks for printing
120 C$=STR$(I) : IF LEN C$<6 THEN REPEAT C$=" "+C$ : UNTIL LEN
C$=6
130 C$=C$+" " : PRINTC$;
140 P%=P%+1
149 REM L% points to byte where lines length is stored
150 L%=P%
159 REM set flag to decide whether characters are inside quotes
160 QUOTE=FALSE
169 REM list each token of line
170 REPEAT
180 P%=P%+1
190 A%=?P% : PROClist
200 UNTIL P%=L%+?L%-3
210 PRINT
220 GOTO100

300 REM list
310 DEF PROClist
319 REM first section deals with line numbers stored in code
320 IF A%<>&8D THEN 390
330 A%=?(P%+1)*4 : B%=?(P%+2) : C%=?(P%+3)
340 S%=A% AND &C0 : B%=B% EOR S%
350 S%=(A%*4) AND &C0 : C%=C% EOR S%
360 PRINT"";B%+C%*256;
370 P%=P%+3
380 ENDPROC
389 REM deal with all other tokens
```

```
390 T$=LEFT$($P%,LEN(A$))
399 REM highlight string if searching
400 IF T$=A$ AND SEARCH AND NOT REPLACE THEN VDU 133 :
PROCout(A$) : VDU 135 : P%=P%+LEN(A$)-1 : ENDPROC
409 REM replace string if it isn't inside quotes
410 IF T$=A$ AND REPLACE AND NOT QUOTE THEN PROCreplace : ENDPROC
419 REM if quote symbol encountered reverse quotes flag
420 IF A%=&22 THEN QUOTE=NOT QUOTE
429 REM if character is inside quotes just print it out else
        use decrunch routine
430 IF QUOTE THEN VDU A% ELSE CALL DECRUNCH
440 ENDPROC

500 REM replace
510 DEF PROCreplace
519 REM if the strings are different lengths the rest of the
        program must be moved
520 DIFF=LEN A$-LEN B$
530 IF DIFF>0 THEN PROCdown(DIFF,P%)
540 IF DIFF<0 THEN PROCup(DIFF,P%)
549 REM temporarily save the byte thats overwritten
        by the carriage return
550 T%=?(P%+LEN B$)
559 REM replace string and restore following byte
560 $P%=B$ : ?(P%+LEN B$)=T%
570 A%=0:PROCout(B$):P%=P%+LEN(B$)-1
580 ENDPROC

600 REM down
610 DEF PROCdown(N%,M%)
619 REM move program from M% to end down by N% bytes
620 FOR I%=M% TO Q%
630 ?I%=I%?N%
640 NEXT I%
649 REM change length of current line and whole program
650 ?L%=?L%-N% : Q%=Q%-N%
660 ENDPROC

700 REM up
710 DEF PROCup(N%,M%)
719 REM move program from M% to end up by N% bytes
720 N%=ABS(N%)
730 FOR I%=Q% TO M% STEP -1
740 N%?I%=?I%
750 NEXT I%
759 REM change length of current line and whole program
760 ?L%=?L%+N% : Q%=Q%+N%
770 ENDPROC

800 REM out
810 DEF PROCout(C$)
820 LOCAL I%
829 REM output a string with decrunch
830 FOR I%=1 TO LEN(C$)
840 A%=ASC(MID$(C$,I%,1)) :  CALL DECRUNCH
850 NEXT I%
860 ENDPROC
```

You must load the program that you are investigating into the store at &1900 and then load listing 14.2 into the store somewhat above it (use TOP to find the location of the end of your first program). Now RUN the search program, which will first ask you for a search string. Typing a RETURN means you wish only to list the program. If you do specify a search string it will ask you for a replacement string. You can search/replace BASIC keywords by redefining the soft keys with their corresponding token and then entering them. If you RETURN without a replacement it will place a purple code in front of the search string. The listing then follows, starting at a specified location (RETURN means &1900). You can then SAVE the program in its adjusted form.

Example 14.2
Add the line

```
315 1F A% = 58 THEN PRINT'SPC(7);: ENDPROC
```

to the search program and repeat the above procedure. Note that now each individual BASIC statement of your program appears on a new line of a listing that is produced by this program.

Scrolling

In chapter 13 we saw the need for machine-code routines for rapid transfer ofdata (the ‘movie'). BASIC is just not fast enough to be ofany practical use where such animation is required. In any mode other than mode 7 it takes tens of seconds to reorganise the screen memory. As an example we give both a BASIC and assembly language program (listings 14.3 and 14.4) for scrolling the whole screen in (mode 0) character-size blocks in any of eight directions (up, down, 1eft, right, or diagonally up + left, down + 1eft, up + right, down + right). The direction ofmovement is passed to the routine via the X-register from the variable X%, and the size of the block to be moved is defined by the four values in locations &70 to &73 . In this implementation of software scrolling a line that goes off the top of the screen will reappear on the bottom (and vice versa) and a line that goes off one edge will reappear on the other side, that is we imagine the screen as a torus.

Compare the time taken to run the BASIC procedure with that for the machinecode routine. You will find the machine code is hundreds of times faster. Even so it is often useful to write your routines first in BASIC in order to check that your algorithm works before embarking on the assembly language stage of development.

Listing 14.3

```
 10 REM BASIC version of scroll
 20 MODE 2 : HIMEM=&2D00
 30 HI=&2D00 : TEMP=&2D80
 40 PROCtable
 50 FOR I=1 TO 31 : FOR J=1 TO 20 : VDU ((I+J)MOD 27)+63 : NEXT J,I
 59 REM check cursor keys
 60 REPEAT
 70 IF INKEY(-122) THEN PROCscroll(1)
 80 IF INKEY(-58) THEN PROCscroll(2)
 90 IF INKEY(-26) THEN PROCscroll(3)
100 IF INKEY(-42) THEN PROCscroll(4)
110 UNTIL FALSE
120 END
200 REM scroll
210 DEF PROCscroll(X)
220 IF X=1 THEN PROCright
230 IF X=2 THEN PROCup
240 IF X=3 THEN PROCleft
250 IF X=4 THEN PROCdown
260 ENDPROC
300 REM left
310 DEF PROCleft
319 REM for each row save the first eight bytes
320 FOR I%=0 TO 31
330 ADT=TEMP
340 ADF=(HI?I%)*256+(I% MOD 2)*&80
350 PROCtran(8)
359 REM then move the transfer the rest to the left
360 ADT=ADF : ADF=ADT+8
370 PROCtran(632)
380 ADF=TEMP
389 REM then retrieve the first eight bytes at the end of the row
390 ADT=HI?I%*256+(I% MOD 2)*&80+632
400 PROCtran(8)
410 NEXT I%
420 ENDPROC
500 REM up
510 DEF PROCup
519 REM store the whole top row
520 ADT=TEMP
530 ADF=(HI?0)*256
540 PROCtran(640)
549 REM set the address-to pointer to the current row
        and the address-from pointer to the next row
550 FOR I%=0 TO 30
560 ADT=(HI?I%)*256+(I% MOD 2)*&80
570 N%=I%+1
580 ADF=(HI?N%)*256+(N% MOD 2)*&80
589 REM transfer a whole row
590 PROCtran(640)
600 NEXT I%
609 REM retrieve the top row from temporary store to the bottom
610 ADF=TEMP
620 ADT=(HI?31)*256+&80
630 PROCtran(640)
640 ENDPROC
```

```
 700 REM down
 710 DEF PROCdown
 719 REM save bottom row
 720 ADT=TEMP
 730 ADF=(HI?31)*256+&80
 740 PROCtran(640)
 749 REM transfer current row to the row below
 750 FOR I%=30 TO 0 STEP -1
 760 ADF=(HI?I%)*256+(I% MOD 2)*&80
 770 N%=I%+1
 780 ADT=(HI?N%)*256+(N% MOD 2)*&80
 790 PROCtran(640)
 800 NEXT I%
 809 REM retrieve row for top
 810 ADF=TEMP
 820 ADT=(HI?0)*256
 830 PROCtran(640)
 840 ENDPROC
 900 REM right
 910 DEF PROCright
 920 FOR I%=0 TO 31
 929 REM for each row save the final eight bytes
 930 ADT=TEMP
 940 ADF=(HI?I%)*256+(I% MOD 2)*&80+632
 950 PROCtran(8)
 959 REM set a pointer to the start of the row
 960 TT=(HI?I%)*256+(I% MOD 2)*&80
 969 REM for each block of eight bytes from right to left
         move the block to the right
 970 FOR K%=78 TO 0 STEP -1
 980 ADF=TT+8*K%
 990 ADT=ADF+8
1000 PROCtran(8)
1010 NEXT K%
1019 REM retrieve the bytes for the start of the line
1020 ADT=TT
1030 ADF=TEMP
1040 PROCtran(8)
1050 NEXT I%
1060 ENDPROC
1100 REM tran
1110 DEF PROCtran(A)
1119 REM move A bytes from pointer ADF to pointer ADT
1120 FOR J%=0 TO A-1
1130 ADT?J%=ADF?J%
1140 NEXT J%
1150 ENDPROC
1200 REM table
1210 DEF PROCtable
1219 REM construct table of hibytes of addresses for screen
1220 M%=&3000
1230 FOR I%=0 TO 31
1240 M=M%+640*I%
1250 HI?I%=M DIV 256
1260 NEXT I%
1270 ENDPROC
```

Listing 14.4

```
 10 REM M/C code version of scroll
 20 MODE 2 : HIMEM=&2A00
 30 HI=&2D00 : TEMP=&2D80
 40 PROCtable : PROCassemble
 50 FOR I=1 TO 31 : FOR J=1 TO 20 : VDU ((I+J)MOD 27)+63 : NEXT J,I
 59 REM check cursor keys
 60 REPEAT
 70 IF INKEY(-122) THEN A%=1 : CALL scroll
 80 IF INKEY(-58) THEN A%=2 : CALL scroll
 90 IF INKEY(-26) THEN A%=3 : CALL scroll
100 IF INKEY(-42) THEN A%=4 : CALL scroll
110 UNTIL FALSE
120 END
200 REM assemble
210 DEF PROCassemble
220 VDU 22,7,14
230 ADFL=&70 : ADFH=&71
240 ADTL=&72 : ADTH=&73
250 TTL=&74 : TTH=&75
260 TAL=&76 : TAH=&77
270 FOR I%=0 TO 3 STEP 3
280 P%=&2A00
290 [ OPT I%
300 \ scroll routine \
310
320 .scroll
330 CMP #1 : BNE NO1 : JSR right : RTS
340 .NO1
350 CMP #2 : BNE NO2 : JSR up : RTS
360 .NO2
370 CMP #3 : BNE NO3 : JSR left : RTS
380 .NO3
390 CMP #4 : BNE NO4 : JSR down
400 .NO4
410 RTS
420
430
440 \ left routine \
450
460 .left
470 LDA #0
480 .LOOP1 : PHA
490 LDA #(TEMP MOD 256) : STA ADTL : LDA #(TEMP DIV 256) : STA ADTH
500 PLA : PHA : JSR calc
510 LDA TAL : STA ADFL : LDA TAH : STA ADFH
520 LDA #0 : LDX #8 : JSR tran
530 LDA ADFL : STA ADTL : LDA ADFH : STA ADTH
540 LDA ADTL : CLC : ADC #8 : STA ADFL
550 LDA #2 : LDX #120 : JSR tran
560 LDA #(TEMP MOD 256) : STA ADFL : LDA #(TEMP DIV 256) : STA ADFH
570 PLA : PHA : JSR calc
580 LDA TAL : CLC : ADC #120 : STA ADTL : LDA TAH : ADC #2 : STA ADTH
590 LDA #0 : LDX #8 : JSR tran
600 PLA : CLC : ADC #1 : CMP #32 : BNE LOOP1
610 RTS
620
```

```
 630
 640 \ up routine \
 650
 660 .up
 670 LDA #(TEMP MOD 256) : STA ADTL : LDA #(TEMP DIV 256) : STA ADTH
 680 LDA HI : STA ADFH : LDA#0 : STA ADFL
 690 LDA #2 : LDX #128 : JSR tran
 700 LDA #0
 710 .LOOPU1 : PHA
 720 JSR calc : LDA TAL : STA ADTL : LDA TAH : STA ADTH
 730 PLA : PHA : CLC : ADC #1
 740 JSR calc : LDA TAL : STA ADFL : LDA TAH : STA ADFH
 750 LDA #2: LDX #128 : JSR tran
 760 PLA : CLC : ADC#1 : CMP #31 : BNE LOOPU1
 770 LDA #(TEMP MOD 256) : STA ADFL : LDA #(TEMP DIV 256) : STA ADFH
 780 LDX #31 : LDA HI,X : STA ADTH : LDA #&80 : STA ADTL
 790 LDA #2: LDX #128 : JSR tran
 800 RTS
 810
 820
 830 \ down routine \
 840
 850 .down
 860 LDA #(TEMP MOD 256) : STA ADTL : LDA #(TEMP DIV 256) : STA ADTH
 870 LDX #31 : LDA HI,X : STA ADFH : LDA #&80 : STA ADFL
 880 LDA #2: LDX #128 : JSR tran
 890 LDA #30
 900 .LOOPD1 : PHA
 910 JSR calc : LDA TAL : STA ADFL : LDA TAH : STA ADFH
 920 PLA : PHA : CLC : ADC #1 : JSR calc
 930 LDA TAL : STA ADTL : LDA TAH : STA ADTH
 940 LDA #2: LDX #128 : JSR tran
 950 PLA : SEC : SBC #1 : BPL LOOPD1
 960 LDA #(TEMP MOD 256) : STA ADFL : LDA #(TEMP DIV 256) : STA ADFH
 970 LDA HI : STA ADTH : LDA #0 : STA ADTL
 980 LDA #2 : LDX #128 : JSR tran
 990 RTS
1000
1010
1020 \ right routine \
1030
1040 .right
1050 LDA #0
1060 .LOOPR1 : PHA
1070 LDA #(TEMP MOD 256) : STA ADTL : LDA #(TEMP DIV 256) : STA ADTH
1080 PLA : PHA : JSR calc
1090 LDA TAL : CLC : ADC #120 : STA ADFL : LDA TAH : ADC #2 : STA ADFH
1100 LDA #0 : LDX #8 : JSR tran
1110 LDA #0
1120 .LOOPR2 : PHA
1130 LDA ADFL: STA ADTL : SEC : SBC #8 : STA ADFL : LDA ADFH
     : STA ADTH : SBC #0 : STA ADFH
1140 LDA #0 : LDX #8 : JSR tran
1150 PLA : CLC : ADC #1 : CMP #79: BNE LOOPR2
1160 LDA TAL : STA ADTL : LDA TAH : STA ADTH
1170 LDA #(TEMP MOD 256) : STA ADFL : LDA #(TEMP DIV 256) : STA ADFH
1180 LDA #0 : LDX #8 : JSR tran
1190 PLA : CLC : ADC #1 : CMP #32 : BNE LOOPR1
1200 RTS
1210
1220
```

```
1230 \ tran routine \
1240
1250 .tran
1260 TAY : TXA : PHA : TYA : BEQ NOHI
1270 .LOOPT1
1280 PHA : LDY #0
1290 .LOOPT2
1300 LDA (ADFL),Y
1310 STA (ADTL),Y
1320 INY : BNE LOOPT2
1330 INC ADTH : INC ADFH
1340 PLA : SEC : SBC #1 : BNE LOOPT1
1350 .NOHI : PLA : TAX
1360 .LOOPT3
1370 LDA (ADFL),Y
1380 STA (ADTL),Y
1390 INY : DEX : BNE LOOPT3
1400 RTS
1410
1420
1430 \ calc routine performs \
1440 \ TA=(HI?A%)*256+(A% MOD 2)*&80 \
1450
1460 .calc
1470 TAX : LDA HI,X : STA TAH
1480 TXA : AND #1 : CLC : ROR A : ROR A : STA TAL
1490 RTS
1500 ]
1510 NEXT I%
1520 VDU 22,2
1530 ENDPROC
1600 REM table
1610 DEF PROCtable
1619 REM construct table of hibytes of addresses for screen
1620 M%=&3000
1630 FOR I%=0 TO 31
1640 M=M%+640*I%
1650 HI?I%=M DIV 256
1660 NEXT I%
1670 ENDPROC
```

Exercise 14.2

Write a BASIC program that specifies a fixed window on the screen and then software scrolls the contents of that window as above, while the outside of the window stays fixed.

Animation: hardware scrolling

Software scrolling can only achieve fast animation on relatively small areas. To move large areas of screen about we need to use hardware scrolling. The simplest way to get this without learning the complexities of programming the 6845 CRT display controller is to force the operating system to scroll the screen for you. This is done by moving the text cursor to either the top or bottom of the screen and by printing the appropriate control code (control K or control J) to force the cursor off the screen and thus to initiate a scroll. We use this method in a simple game (listing 14.5). The object ofthe game is to catch randomly appearing apples in an ever-increasing shower of bricks. If a brick hits your hand then the game is over.

To get the speed of animation we hardware scroll the whole screen and then PRINT (or perhaps software scroll) stationary text and characters back in position. Machine code is needed for more complicated tasks and this requires a great deal more thought and study. A good book on the subject and preferably specifically written for the BBC Micro should be consulted; see, for example, Birnbaum (1982) and Zaks (1978).

Listing 14.5

```
 10 ENVELOPE 1,2,5,10,15,5,3,1,64,4,0,-126,100,126
 20 MODE 5
 29 REM get rid of cursor and define characters for game
 30 VDU 23,1,0;0;0;0;
 40 VDU 23,128,&FF,&BD,&C3,&DB,&DB,&C3,&BD,&FF
 50 VDU 23,129,&18,&10,&7C,&FE,&FE,&FE,&7C,&38
 60 VDU 23,130,&10,&54,&55,&D5,&FF,&FE,&7C,&3C
 70 VDU 19,3,6,0,0,0 : VDU 19,2,2,0,0,0
 79 REM reset keyboard to give better response
 80 *FX4,1
 90 *FX11,15
100 *FX12,13
110 X=10
120 COLOUR 3 : PRINT TAB(X,20);:VDU 130
130 DEAD=FALSE : C=0 : SCORE=0
140 REPEAT
150 HARD=(SCORE DIV 20)
160 COLOUR1
169 REM put some bricks on top row
170 FOR I%=1 TO RND(HARD)
180 PRINT TAB(RND(20)-1,0);: VDU 128
190 NEXT I%
200 COLOUR 2
209 REM add an apple
210 PRINT TAB(RND(20)-1,0);: VDU 129
220 NX=X
229 REM check for left or right cursor
230 A$=INKEY$(0)
240 IF ASC(A$)=&88 AND X>0 THEN NX=X-1
250 IF ASC(A$)=&89 AND X<19 THEN NX=X+1
259 REM erase hand before scrolling
260 PRINT TAB(X,20);" ";
269 REM force scroll down with home and cursor up
270 VDU 26,11
279 REM prepare to reprint hand
280 X=NX : PRINT TAB(X,20);
289 REM read character at hands new position
290 A%=135 : C=(USR(&FFF4) DIV &FF) AND &FF
299 REM was it a brick
300 IF C=55 THEN DEAD=TRUE
309 REM score one for an apple
310 IF C=56 THEN SCORE=SCORE+1 : SOUND 1,1,60,4
320 COLOUR 3 : IF NOT DEAD THEN VDU 130
330 PRINT TAB(0,31);"SCORE ";SCORE;
340 T=TIME : REPEAT UNTIL TIME>T+4
```

```
350 UNTIL DEAD
359 REM crunching noise
360 FOR I%=1 TO 6 : SOUND 0,-15+I%*2,4,3 : NEXT I%
369 REM use text window to display game over message
370 VDU 28,4,10,16,6
380 COLOUR 129 : CLS : COLOUR 3
390 PRINT''"  GAME OVER"
399 REM return keyboard to normal
400 *FX4,0
410 *FX12,0
420 *FX15,1
430 COLOUR 128 : COLOUR 2
440 VDU 26 : END
```

Unusual Displays

The BBC micro is a very sophisticated machine that allows you to perform complicated manipulations on the screen. It would be impossible to give all the techniques, so to give you a flavour we introduce two such methods: (a) changing the print vector, and (b) synchronous display.

(a) Changing the print vector
The print vector (locations &20E and &20F) contains the address ofthat section of the PRINT routine that places characters on the screen. We can write our own routine and let the print vector refer to it instead. Subsequent calls to PRINT inside a BASIC program will use the revised output routine. Take the simple example that is given in listing 14.6. If you run this program it will place a machine-code routine which prints a character and two backspace codes at location &C00. Soft key 1 is defined to change the print vector to this location, soft key 0 returns it to normal. If you now load another program and LIST it, having pressed f1 you will see that the listing appears backwards. Press f9 and L1ST and a normal listing appears.

Exercise 14.3
You can adjust the ‘prynt’ routine ofchapter 5 and make it available to PRINT by changing the print vector. Note that ‘prynt’ could draw only simple strings. By using this adjusted method quite complex strings of multi-coloured characters can be printed, with all the complex evaluation being done by the BASIC PRINT statement before it calls your new output routine.

(b) Syncrhonous Display
The television display is completely redrawn every 1/50th ofa second (1/60th in the U.S.A.). *FX19 is used to wait until the start ofthe next frame, and thus gives us a method of starting instructions at a fixed time relative to each refresh of the display. If we change the logical-actual colour relationship partway through the drawing of the display, then for that frame only the top part of the screen

Listing 14.6

```
 10 FOR I=0 TO 3 STEP 3
 20 OUTPUT=!&20E AND &FFFF
 30 P%=&C00
 40 [OPTI
 50 CMP #10 : BNE OD : JSR OUTPUT
 60 .OD
 70 CMP #32 : BCC OF
 80 JSR OUTPUT
 90 LDA #8 : JSR OUTPUT
100 LDA #8
110 .OF
120 JSR OUTPUT
130 RTS
140 ]
150 NEXT I
159 REM f1 changes vector to routine
160 *KEY1?&20E=0 : ?&20F=&C|M
169 REM f0 changes vector back
170 O%=OUTPUT DIV 256 : P%=OUTPUT MOD 256
180 *KEY0?&20E=P% : ?&20F=O%|M
```

will show one actual colour while the remainder of the screen shows another. Obviously we could use *FX19 to wait fbr the start ofthe next frame and repeat the process. Provided that we keep repeating the same actions, this will result in the display of a multi-colour steady(ish!) picture, even in mode 0! Listing 14.7 draws a simple picture in mode 0 and then redefines the background colour continuously. There is a little flickering at the junctions of the colours so we cover this up by placing blocks of foreground colour across the screen. Type any key and the coloured bands move because we have interrupted the time balance.

Listing 14.7

```
 10 MODE 0
 20 A%=19 : Z%=0
 30 GOSUB 130
 40 GCOL0,1
 50 FOR Y=75 TO 1025 STEP 165
 60 MOVE 0,Y : MOVE 0,40+Y
 70 PLOT 85,1280,Y : PLOT 85,1280,40+Y
 80 NEXT Y
 90 REPEAT
100 FOR I%=1 TO 6 : VDU A%,Z%,I%,Z%,Z%,Z% : NEXT I%
110 *FX19
120 UNTIL FALSE
129 REM draw lace curtains
130 FOR I=0 TO 1280 STEP 16
140 MOVE 0,I : DRAW 1280-I,0
150 MOVE 1280,I : DRAW 1280-I,1024
160 MOVE 0,1024-I : DRAW 1280-I,1024
170 MOVE 1280,1024-I : DRAW 1289-I,0
180 NEXT I
190 RETURN
```

Complete Programs

Sample input will be underlined.

I Listing 14.1. RUN the program and then CALL &7000. Then type H and 8000 which will bring you to the start of the BASIC ROM. Now H and 7C00 will bring you to the start of the disassembler. H and 7C00; the start of the mode 7 screen. Type S and What's up Doc?. Type H, 78DC, which will get you into the table that holds mnemonics, specifically code TXS. A and 81 will change the value ofthe currently active byte from 87 to 81. Now type H and 7000 and List through the program to locate TXS code which will be in red.

II Listing 14.2. First load any program (such as listing 13.2) at PAGE = &1900. Then load listing 14.2 at PAGE = &3000 and RUN. Try the following sample data.

Search?RETURN , , location?RETURN

Search?J , replace?RETURN , location?&1900

Search?J , replace?K , location?&1900

III Listing 14.3 or 14.4. RUN and use the cursor keys to scroll the screen.

IV Listing 14.5. The apple-catching game. RUN to start, move the 'hand' left or right by means of the cursor keys.

V Listing 14.6. RUN, press f1 and LIST, then f0 and LIST.

VI Listing 14.7. Type any key.