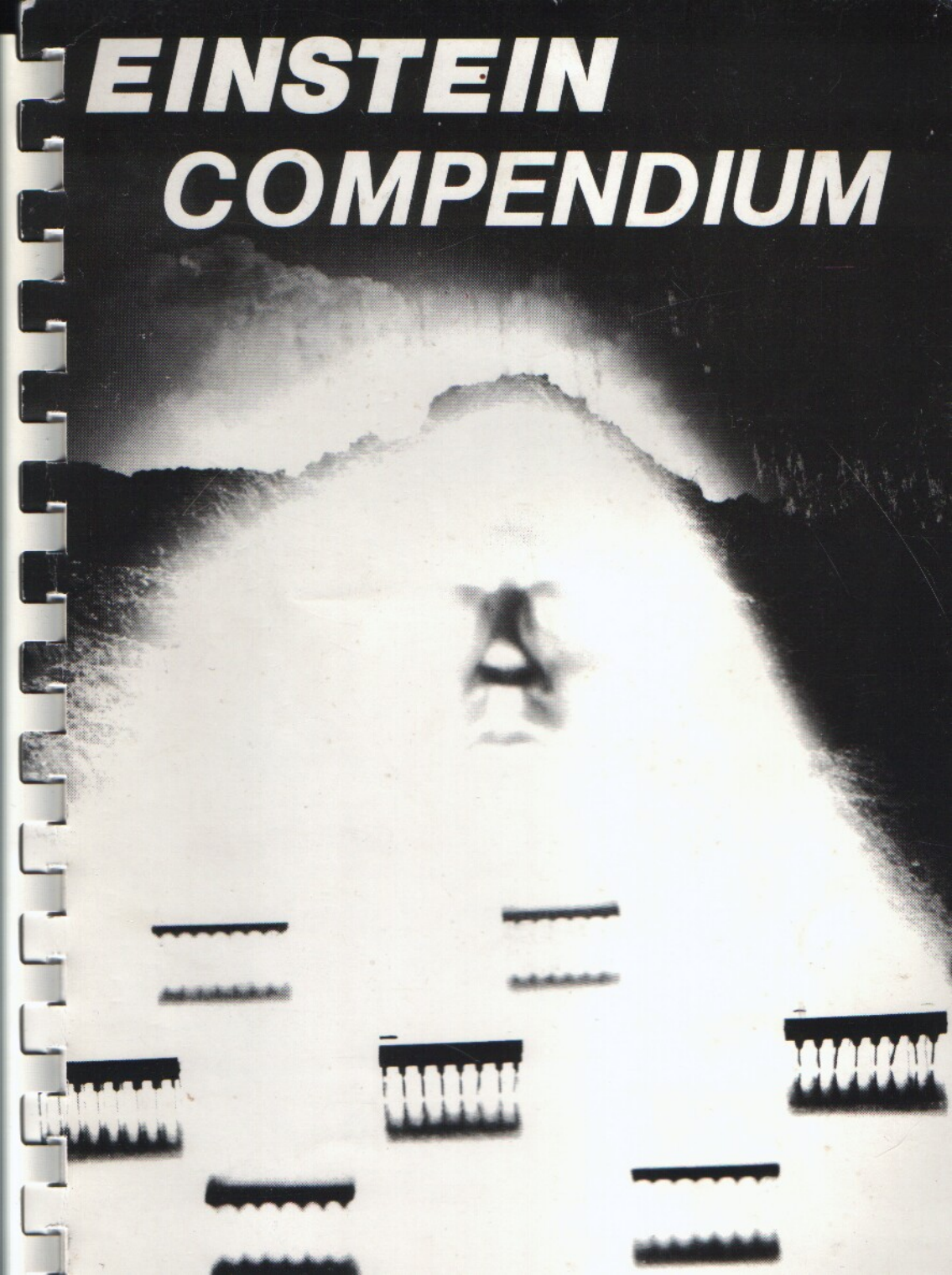
# EINSTEIN COMPENDIUM



OUR SPECIAL THANKS TO OUR WIVES, CAROLINE AND PAT. WITHOUT
WHOSE UNDERSTANDING THIS BOOK WOULD NOT HAVE BEEN POSSIBLE.

# THE EINSTEIN COMPENDIUM

COMPILED BY MIKE SMALLMAN & GRAHAM BETTANY

This book contains a selection of programs and information for the Tatung Einstein computer collected by the U.K. Einstein User Group.

Within these pages you will find a collection of programs ranging from simple one liners in Basic to machine code utilities.

Also included are explanations on the Einstein video display processor (VDP), and the disc operating system. These are included to give the reader a greater insight into the workings of the Einstein.

All programs within this book are listings taken from working copies. If you experience problems with any of the programs the U.K.E.U.G. are prepared to answer queries provided a S.A.E. is included, address as below.

For those of you who find typing in long listings a daunting task, a disc copy of all the programs is available on a 3" disc for £10 including P+P from;

U.K.E.U.G.,
80 DALES ROAD,
IPSWICH,
SUFFOLK,
IP1 4JR.

Please make any cheques/postal orders payable to U.K.E.U.G.



Whilst every effort has been made to guard against faults or errors, the U.K.E.U.G. will not be held liable for any consequential damage resulting from the use of the programs within these pages.

CONTENTS

# PROGRAMMING IN BASIC

Beginners All-purpose Symbolic Instruction Code, better known as BASIC is one of the most widely used programming languages in the world. It was invented in the USA in 1964 for the purpose of enabling people to write computer programs in a language that was quick and easy to learn.

Unfortunately over the years many variations of BASIC have been produced for different machines, this has created a situation where a BASIC program on one machine will not transfer readily to another. XBAS, developed by Crystal Research is the version of BASIC supplied as standard with the Einstein.

Unlike most home micro-computers the Einstein does not have a language resident in the machine, BASIC being loaded from disc. This has several advantages;

a) Improved versions of BASIC can be supplied on disc.
b) Additions to the BASIC are possible.
c) If BASIC is not loaded there is more memory available for machine code programming.
d) Other languages can be loaded from disc.

There are many ways of writing a BASIC program, the overall aim should be to produce a program that works!! there are however a number of other factors to take into consideration. BASIC is a high level language, this means that it uses instructions or statements similar to the English language. This requires a piece of software called an 'interpreter' which converts each BASIC line into its machine code equivalent. A poorly written BASIC program will run more slowly and use up more memory, here are a few tips that will help you write more efficient programs;

a) REM statements are essential aids to developing programs, however they take up memory space and although not acted upon still have to be interpreted thus slowing down the operation of the program. Whilst a program is under development it is useful to use many REMs, after completion this copy of the program can be saved as a library copy and the program to be used can have all its REMs removed. If modification is required in the future the library copy can be recalled and the same process repeated.

b) When using sub-routines, i.e. GOSUB, XBAS searches for the line number of the sub-routine starting from the GOSUB line number. Therefore keeping the sub-routines as close as possible to the calling line will ensure that as little time as possible is wasted in searching for the sub-routine.

c) The most commonly used variable should be assigned first, this is due to the way that XBAS allocates memory space for variables. The first assigned variable is located at the top of the variable list and will obviously be found before a variable further down the list.

d) Every variable you assign in a program occupies memory even if it is only used once. Therefore try to use the same variable throughout the program. i.e. Separate FOR/NEXT loops can use the same variable.

e) Since XBAS arrays are numbered from 0 e.g. A$(0) always use the zero element.

f) Using multiple statements on one line seperatef by a colon results in faster programs because for each extra line number XBAS uses five extra bytes. This does cause problems with readability so it should be assesed whether the extra speed and less memory is needed for the particular application in hand.

g) Using A%, i.e. integer variables instead of A, real variables, saves three bytes of memory. Using % after a variable name denotes that the variable will contain only whole numbers, if you need to store numbers containing fractions then use just the variable name.

Another factor that should be considered when writing a program is the overall structure. With short programs this is not really relevant but for complicated applications some form of planning and structure will make the task far easier. It is more than likely that some modifications will need to be made as a large program progresses, if the program has some form of structure it will make for an easier task. Planning can be in the form of a flow chart or structured diagram or just a list of what happens.

Programming structure could take up a complete book on its own but just one thought on the subject, try to write your next program without using the command GOTO, if you can then it must have some form of structure. If someone else can look at a listing of your program and understand it with comparative ease then again it is likely to have good structure.

Typing in listings is an excellent way of learning programming skills. Everyone makes typing mistakes and it is in finding and correcting these erorrs that most progress is made. It is a known fact that after a certain time viewing a monitor a type of word blindness affects people, glaring mistakes that would have been noticed when first starting the programming session are not apparent. The cure is of course to have a break from the VDU and come back to the problem with a refreshed mind.

## ONE LINERS

Trying to write a program in one line really makes you think and the results can prove most rewarding. The normal line length for BASIC on the Einstein is 127 characters so use of the ? insted of PRINT is essential. When typing in the following 1 liners use the ? command insted of PRINT at all times. Remember though, if you list the program then PRINT will be displayed and may cause the end of the line to be lost.

```
1 FORD=0TO9:CLS:FORA=65TO90:PRINTCHR$(A);:NEXT:B=RND(26):
  PRINT@B,1;CHR$(94):BEEP20-D:C=KBD:IFC<>B+65THENPRINTD:
  ELSENEXT:PRINTD
2 REM HOLD DOWN KEY INDICATED
```

```
1 FORT=0TO360STEP9:X=15*COS(RAD(T)):Y=15*SIN(RAD(T)):DRAW99
  ,96 TO 99+X1,96+Y1,1:DRAW99,96TO99+X,96+Y:X1=X:Y1=Y:NEXT:
  GOTO1
2 REM by DAVE HARVEY
```

```
1 A=KBD:IFA<>90THEN1:ELSELETB=RND(99):IFB<>5THEN1:ELSEBEEP
  :FORC=0TO40:PRINT@C,9;"]":B=KBD:IFB=77THENPRINTC:ELSENEXT
  :PRINT"SLOWE
2 REM HOLD DOWN Z PRESS M WHEN ARROWS SEEN
```

```
1 TCOL15,2:BCOL2:CLS40:R=100:FORA=0TO1000STEP0.1:PLOTR*COS
  (A) *SIN(A*0.98)+125,R/1.5*SIN(A)+100 :NEXTA
2 REM BALL
```

```
1 BCOL1:R=80:B=125:C=95:FORL=1TO6STEP.5:CLS:GCOLL+7:FORA=0
  TO2*PISTEPPI/C:X=R*COS(A)+B:Y=R*SIN(L*A)+C:DRAWB,CTOX,Y:
  NEXT A,L
2 REM by STEVE COOPER
```

```
1 CLS:FORC=0TO20:PRINT@0,0;S:X=RND(20)+10:Z=RND(26)+65:
  PRINT@X,C;CHR$(Z):BEEP25-C:A=KBD:IFA<>ZTHENS=S-1:NEXT:
  ELSEPRINT@X,C;" ":NEXT
2 REM PRESS KEY INDICATED
```

```
1 CLS:V=.8+RND(1)*3:PRINTV:FORT=0TO200*VSTEPV:R=T/V:X=R*COS
  (T):Y=R*SIN(T):DRAW125,98TO125+X,98+Y:NEXT:FORP=1TO999:
  GOTO1
2 REM by DAVE HARVEY
```

```
1 FORF=6144TO8192STEP8:C=0:FORG=0TO7:A(G)=VPEEK(F+G):NEXTG:
  FORG=7TOOSTEP-1:VPOKE(F+C),A(G):C=C+1:NEXTG,F:BEEP5
2 REM TAKES TIME..WAIT FOR BEEPS
```

```
1 CLS:FORC=1TORND(50):BEEP:NEXTC:FORF=0TO99:TCOLF,1:E=KBD:
  PRINT@F,5;" ";:IFE<>81THENNEXTFELSEPRINT:PRINT"YOU SCORE"
  ;F:A=INCH:GOTO1
2 REM PRESS Q WHEN BEEP STOPS ... NO CHEATING
3 REM GRAPH/SPACE BETWEEN QUOTES
```

```
1 PRINT:A=INCH:IFA<>27THENPRINT""CHR$(A)"=code("A")":GOTO1
  :ELSEINPUTA:IFA<33ORA>255THENPRINT"n/a!":GOTO1:ELSEPRINT"
  code("A")="CHR$(A):GOTO1
2 REM by STEVE COOPER  prints code for typed character.
  Press escape, then prints character for typed code.
```

```
1 INPUT"ANGLEINC.VALUE";A:B=133:C=100:CLS:PLOT-1,512,386:
  MOVEO,O:FORI=0TO36000STEPA:J=2*I:X=J/B*COS(RAD(I)):Y=J/B*
  SIN(RAD(I)):DRAWX,Y:NEXTI:Z=GET:GOTO1
2 REM by KEVIN LYONS.
3 REM BBC BASIC.
```

```
1 BCOLO:ORIGIN99,85:A=3:B=6:CLS:FORC=2TO15:FORI=0TO250:XX=Y
  -SGN(X)*SQR(ABS(B*X-C)):YY=A-X:X=XX:Y=YY:PLOTX*5,Y*5:GCOL
  C:NEXTI,C
```

```
1 A=KBD:A=A-48:X=X+(A=5)-(A=8):Y=Y+(A=6)-(A=7):PLOTX,Y:FORF
  =1TO20:NEXT:IFA=1THENCLS:GOTO1:ELSEGOTO1
2 REM 5..LEFT 8..RIGHT 6..DOWN 7..UP 1..CLS
```

```
1 CLS:FORA=0TO14914:PLOT128+(50+30*SIN(A/39))*SIN(A/40),88+
  (30+50*COS(A/41))*COS(A/40):NEXT
```

This next routine would very nearly fit on to one line but it has been spaced out for clarity. It requires the use of a joystick and the volume control turned up.

```
 10 TCOL1,1:CLS32
 20 PSG8,&31
 30 PSG7,&70
 40 N=ADC(0):M=BTN(0)
 50 PSG0,N/2
 60 PSG2,N/4
 70 PSG5,N/3
 80 PSG6,18:PSG9,31:PSG8,31
 90 PSG10,31
100 IF M=1 THENPSG&C,10:PSG&D,0
110 IF M=0 THENPSG&C,2:PSG&D,4
120 TCOLN,1:PRINT@RND(29)+1,RND(21)+1;"*"
130 SPEED240:GOTO 40
```

## KALEIDESCOPE

Type this program in, run it and sit back and watch your screen dance to the ever changing patterns and colours.

```
 10 TCOL1,0:CLS32
 20 FOR A=50 TO 1 STEP-1
 30 FOR B=0 TO 20
 40 FOR C=B TO 20:D=INT(B*C/A+(B+C)/A+A)
 50 CL=D-15*INT(D/15):TCOLCL+4,CL
 60 PRINT@C,B;"*";@31-C,B;"*";@C,21-B;"*";@31-C,21-B;"*"
 70 PRINT@B,C;"*";@31-B,C;"*";@B,21-C;"*";@31-B,21-C;"*"
    ;@31-B,21-C;"*"
 80 NEXT C
 90 NEXT B
100 NEXT A
```

LISTING NOTES:
LINE 50 calculates the TCOL foreground and background colours.
LINES 60 and 70 do the actual printing to screen, the position of the print depending on the values of A,B and C.
NOTE: the three loops are 'NESTED' each successive loop lies within the previous loop. It is important when writing programs which contain more than one loop to ensure that the loop with the first 'FOR' statement is also the loop with the last 'NEXT' otherwise the program will crash.

## ROTATE

The DRAW command is fast enough for small drawings to give an appearance of a continuously moving picture. The ROTATE program draws what appears to be a square rotating into the screen.

```
 10 TCOL15,1:GCOL15:BCOL1:CLS
 20 A=0:B=10:C=40:GOTO70
 30 DRAW D,90 TO E,90 TO E,G TO D,G TO D,90
 40 DRAW D,90 TO E,90,1 TO E,G,1 TO D,G,1 TO D,90,1
 50 A=A+B
 60 IF A=C OR A<0 THEN B=-B
 70 D=90+.5*A:E=D+C-A:G=90+C
 80 GOTO 30
```

LISTING NOTES:
LINES 10 @ 20 initialize variables.
LINE 30 DRAWS the square.
LINE 40 DRAWS the same square but this time erasing it, this is accomplished by adding the third parameter to the DRAW statement. (see page 67 in the Basic reference manual) When using the DRAW command the third parameter should not be added to the first pair of co-ordinates only to the second and subsequent pairs.
LINES 50-80 perform the modification to the square to give the impression of rotation.
LINE 90 go back and draw it again.

## COLLISION DETECTION

When writing BASIC games using sprites, how can you tell when your intergalactic hyperdrive star cruiser is crossing the path of the dreaded Gurtvoguls stratoblaster? Well the answer lies in the VDP. Register 8 in the VDP is the status register. Bit 5 in the status register is the coincidence detector bit, this is normaly a 0, but whenever two sprites cross each other it is set to a 1.

The following basic listing gives one method of detecting collisions.

```
  5 RST:MAG0:TCOL15,1:CLS
 10 X=10:XX=250
 20 SPRITE1,X,50,15,65
 30 SPRITE2,XX,50,15,66
 40 A=INP(9):A$=BIN$(A,8):A$=MID$(A$,3,1):IF A$="1" THEN
    PRINT@20,20;"CROSSING"
 50 IF  A$="0"  THEN  PRINT@20,20;"           "
 60 IF F=1ANDX>10 THEN X=X-1:XX=XX+1:GOTO 80
 70 IF F=0ANDX<250 THEN X=X+1:XX=XX-1
 80 IF X=250 THEN F=1
 90 IF X=10 THEN F=0
100 GOTO 20
```

LISTING NOTES
LINES 20-30 set up sprites, we are using the A and B from the normal character set.
LINE 40 this is the line that reads the status register, A=INP(9) reads the register into A. Next we change it into a Binary string then we check to see if bit 5 is a 1, bit 5 being the coincidencf bit and the 3rd bit from the left in the Binary string. If it is a 1 we print "CROSSING".
LINE 50 just blanks out the "CROSSING" once the sprites have passed each other.
LINES 60-90 adjust the positions of the sprites.
LINE 100 go back and do it again.
When using this method it is only possible to detect when ANY two sprites coincide.

# PIN TABLE

A pinball type game. LEFT = Q RIGHT = P.

```
 10 RST:MAG0:GOSUB260
 20 SPRITE1,X,Y,8,161
 30 R=PEEK(64326)
 40 IF R=80ANDXX<18THENXX=XX+1:PRINT@XX-1,21;" "
 50 IF R=81ANDXX>12THENXX=XX-1:PRINT@XX+1,21;" "
 60 TCOL1,4:PRINT@XX,21;"h":TCOL15,4:REM GRAPH H
 70 IFXFLAG=0ANDPOINT(X-1,Y-4)=1THENGOSUB180:
    IFRND(2)=1THENX=X+8:XFLAG=1
 80 IFXFLAG=1ANDPOINT(X+8,Y-4)=1THENGOSUB180:IFRND(2)=1
    THENX=X-8:XFLAG=0
 90 IFYFLAG=0ANDPOINT(X,Y-9)=1THENGOSUB180:IFRND(2)=1
    THENY=Y+8:YFLAG=1
100 IFYFLAG=1ANDPOINT(X+4,Y+1)=1THENGOSUB180:IFRND(2)=1
    THENY=Y-8:YFLAG=0
110 IFXFLAG=0THENIFPOINT(X-1,Y-4)=0THENX=X-8:ELSEXFLAG=1
120 PRINT@8,0;SC
130 IFXFLAG=1THENIFPOINT(X+8,Y-4)=0THENX=X+8:ELSEXFLAG=0
140 IFYFLAG=0THENIFPOINT(X,Y-9 )=0THENY=Y-8:ELSEYFLAG=1
150 IFYFLAG=1THENIFPOINT(X+4,Y+1)=0THENY=Y+8:ELSEYFLAG=0
160 IFY=<20THENGOSUB200
170 GOTO20
180 PSG0,RND(150)+35:PSG13,0:SC=SC+2
190 RETURN
200 LIVES=LIVES-1:PRINT@26,0;LIVES:FORF=10TO112:PSG0,F:
    PSG13,0:NEXTF:Y=(RND(10)+1)*8+32:IF LIVES<>0THENYFLAG=1:
    RETURN
210 GOSUB250
220 SPRITEOFF1:
230 PRINT@0,21;" DO YOU WISH TO PLAY AGAIN  ";:A=INCH:IF
    A<>89THENSTOP
240 RUN
250 FORF=10TO90:FORG=90TO10:PSG0,F:PSG13,0:PSG0,G:PSG13,0
    :NEXTG,F:RETURN
260 FOR F=0TO3:READN$:SHAPE161+F,N$:NEXT
270 DATA"00183C7E7E3C1800","FF81A5A5A5A581FF","8080808080
    808080","183C66DBDB663C18"
280 BCOL4:CLS32:FORF=8TO22:PRINT@F,1;" ":NEXT:FORF=1TO21:
    PRINT@8,F;" ";@22,F;" ":NEXT:FORF=8TO22:PRINT@F,21;" "
    :NEXT:REM GRAPH+SPACE BETWEEN ALL QUOTES ON LINE 280
290 PRINT@12,21;"        "    300 PRINT@11,4;""      ""
310 PRINT@13,6;"  "":REM ITALIC QUOTES ARE GRAPH+SHIFT+2
320 PRINT@11,8;""       ""
330 PRINT@11,10;"     "     "
340 PRINT@11,12;"  "   "  "
350 PRINT@11,14;"     ""
360 PRINT@11,17;""       ""
370 TCOL15,4:PRINT@14,21;"h":REM GRAPH+H
380 XX=14:X=160:Y=96:LIVES=3
390 PSG7,&40:PSG8,31:PSG12,6
400 PRINT@1,0;"SCORE:                LIVES:";LIVES
410 RETURN
```

Since the dawn of mankind, man has used codes and ciphers in an attempt to prevent his enemies from reading his messages. Probably the simplest method of encryption is by the use of a displaced alphabet. This is simply a matter of, for instance displacing the letters of the alphabet by say 3 therefore a 'B' would be encrypted as an 'E'. So for a message that ran 'VHH BRX VRRQ' as long as we knew the displacement, it would be a simple matter to work out that it should read 'SEE YOU SOON'. Of course there are many other forms of encryption, we could substitute graphics or numerals for the letters in our message, however the important point is that the receiver of the encrypted message must know the key used in the encryption.
The computer of course is an ideal medium for cipher encryption and for the breaking of ciphers. If you have used PSW(password) in Basic to protect your programs from prying eyes you will know the nesseccity of knowing the code used in encrypting a message, since if you forget your password for a particular program you'll have a hard time trying to use that program.
Below we have a program by Robert Whitrow, in this program a key word is entered and this key word is used to encrypt the message you enter. Messages encrypted using the program may also be decrypted by the program as long as the key word used is known.

```
  5 CLS
 20 INPUT "ENTER CODEWORD : ";W$
 30 LW=LEN(W$)
 40 DIM W(LW)
 50 FOR I=1 TO LW:W(I)=ASC (MID$ (W$,I,1))-65:NEXT I
 60 REM STORE NUMBER EQUIV. OF LETTERS IN CODEWORD IN
    ARRAY W()
 70 REM
 80 INPUT "ENCRYPT OR DECRYPT (E/D): ";E$
 90 E=1:IF E$="D" THEN E=-1
100 PRINT "ENTER PLAINTEXT OR CIPHERTEXT,LETTERS ONLY:"
110 INPUT C$
120 P$=""
130 FOR I=1 TO LEN(C$):X=ASC (MID$ (C$,I,1))-65
140 Y=(I-1)-LW*INT (I-1)/LW
150 X=X+E*W(Y+1)
160 IF X>25 THEN X=X-26
170 IF X=<0 THENX=X+26:REM 180 AND 190 SET X=X MOD 26
180 P$=P$+CHR$(X+65)
190 NEXT I
200 PRINT:PRINT:INPUT "OUTPUT TO SCREEN OR PRINTER
    (S/P)";O$
210 IF O$="P" OR O$="p" THEN PRINT£1
220 PRINT P$:REM P$ IS CIPHERTEXT OR PLAINTEXT
```

As you will see if you run this program and encrypt a message you will get quite unreadable text. This can only be deciphered if ran through the same program and only then if the codeword is known. However a lot of enjoyment can be had by trying to work out what a cipher message says without using the decoding potion of the program and without knowing the codeword. To do this you need to know a little about the occurance of letters in English words.

Here is a table that may help.

| Letter | Frequency per 1000 words | Letter | Frequency per 1000 words |
|---|---|---|---|
| E | 591 | M | 114 |
| T | 473 | U | 111 |
| A | 368 | G | 90 |
| O | 360 | Y | 89 |
| N | 320 | P | 89 |
| R | 308 | W | 68 |
| I | 286 | B | 65 |
| S | 275 | V | 41 |
| H | 237 | K | 19 |
| D | 171 | X | 7 |
| L | 153 | J | 6 |
| F | 132 | Q | 5 |
| C | 124 | Z | 3 |

As one may expect E is the most commonly used letter in the English language. A few more useful facts would go a long way to helping us decipher an encoded message. The most common double letters are, in order, LL, EE ,SS ,OO, TT, FF, RR, NN, PP, CC, MM, GG. The most common two letter combinations are; TH, HE, AN, RE, ER, IN, ON, AT, ND, ST, ES, EN, OF, TE.
The most common three letter combinations are; THE, ING, CON, ENT, ERE, ERS, EVE, FOR, HER, TED, TER, TIO, VER. Finally it is worth noting that 50% of English words end in E, S, D or T and more than 50% of all English words begin with T, A, O, S or W.

## RISING SUN

The music capabalities of the EINSTEIN have never been fully exploited. This short program shows just a little of what the PSG is capable of.

```
 10 REM RSUN
 20 VOICE 0,31,10,2,60,5
 30 B$= "R5 A7 A5 C7 F5 bB7 bB5 bB7 bB5 A7 A5 A7 C5 E9 A1
    A1 A7 A5 A7 F5 -D5 -G5 -G5 -G5 -F7 -F5 -A7 -A5 -E7
    -E5 -A7 R"
 40 REM SPACE
 50 C$= "R5 F7 F5 A7 A5 D7 G5 F7 F5 F7 F5 F7 F5 #C9 E1 E1
    F7 D5 C7 C5 -bB5 -bB5 -bB5 -bB5 -bB7 -bB5 -F7 -F5
    -#C7 -#C5 -F7 R"
 60 REM SPACE
 70 A$= "-A5 D7 E5 F7 A5 G7 D5 D7 +D5 +D7 +D5 +C7 A5 A9
    +D1 +D1 +D7 E5 F7 A5 G5 D5 D5 D5 D7 D5 D7 D5 #C7 #C5
    D7 R"
 80 TEMPO 5
 90 MUSIC "VO"+A$,B$,C$
100 GOTO 90
```

## SOMBRERO

Plotting to the screen can produce some interesting effects. This program plots a mathematical function that gives a '3-D' image. The program is short but be prepared for a long wait once run, it takes around 12 minutes to complete plotting.

```
 10 TCOL1,15:BCOL15:GCOL4:CLS32
100 FOR X=40 TO 215
110 B=999:T=0
120 FORY=16TO144STEP4
130 R=SQR((X-127)*(X-127)+(Y-80)*(Y-80))/15
140 Z=INT(Y+90*EXP(-R/3)*COS(R))
150 IFZ<BORZ>TTHENPLOTX,Z
160 IFZ<BTHENB=Z
170 IFZ>TTHENT=Z
180 NEXTY
190 NEXTX
```

## BIGPRINT

There now follows several programs for printing large characters. The first of these prints a vertically scrolling message to the screen. The character set is held in VRAM from address 6144. Calculating the address of where a particular character is stored is simply a matter of taking the characters ASCII number, multiplying this by 8, (each character is made up of 8 bytes), then adding this figure to the start address 6144. Thus 'E' who's ASCII value is 69 is stored in VRAM from; 69*8 = 552 + 6144 = 6696.

```
100 PRINT"ENTER MESSAGE":INPUT"";M$
110 FORA=1TOLEN(M$)
120 G$=MID$(M$,A,1)
130 G=ASC(G$):G=G*8+6144
140 FORC=0TO7:X(C)=VPEEK(G+C):NEXTC
150 FORF=1TO8:
160 B=X(F):A$=BIN$(B,8):
170 FORC=1TO8:PRINTTAB(15):G$=MID$(A$,C,1):IFG$="1"THEN
    PRINT " ";:ELSEPRINT" ";
180 REM GRAPH/SPACE BETWEEN QUOTES IN LINE 170
190 NEXTC
200 PRINT:NEXTF:PRINT
210 NEXTA
```

## SIDE PRINT

This program is similar to BIGPRINT only this time it outputs a message to the printer, sideways on, thus you can produce a message as tall as the printer paper is wide and as long as you want.

```
100 PRINT"ENTER MESSAGE":INPUT"";MSG$
110 PRINT"WIDTH":INPUT"";WID$
115 W=VAL(WID$)
120 FORA=1TOLEN(MSG$)
130 L$=MID$(MSG$,A,1)
140 L=ASC(L$):L=L*8+6144
160 FORO=0TO7:X(8-O)=VPEEK(L+O):NEXTO
200 N=128
210 FORO=0TO7
220 FORE=1TOW
230 FORF=1TO8:IFX(F)<NTHEN260
240 PRINT#1:PRINTTAB(4*F-4);"    ";:IFE=WTHENX(F)=X(F)-N
250 REM GRAPH/SPACE BETWEEN QUOTES IN LINE 240
260 NEXTF:PRINT
270 NEXTE:N=N/2
280 NEXTO
290 NEXTA
```

## DOUBLE HEIGHT

Double Height by David Hambly produces double height characters on the screen by pokeing the VRAM directly. The process from BASIC is fairly slow but does produce an interesting effect.

```
 10 CLS40
 20 INPUT"ENTER A MESSAGE UPTO 28 CHARACTERS";A$
 30 Z=5
 40 FOR A=1 TO LEN(A$)
 50 D=8176:B=0
 60 B$=MID$(A$,A,1)
 70 C=(ASC(B$)-32)*8+6400+B
 80 E=VPEEK(C)
 90 VPOKE D,E:VPOKE D+1,E
100 PRINT @Z,10;CHR$(254):PRINT @Z,11;CHR$(255)
110 B=B+1
120 D=D+2:IF D>8191 THEN 130:ELSE 70
130 Z=Z+1:NEXT A
140 END
```

## DOUBLE HEIGHT 2

This version of Double Height by Chris Giles is a little longer a near instant effect. Your own program may be inserted to take advantage of the Double Height subroutine.

```
 10 REM  Subroutine for DOUBLE HEIGHT characters
 20 REM
 30 REM
 40 REM        MAIN PROGRAM GOES HERE
 50 REM
 60 REM
 70 BCOL1:TCOL6:PRINT CHR$(20):CLS:REM Set background
    and text colours.
 80 REM  Cursor off and clear screen.
 90 REM
100 REM
110 REM  Before calling the subroutine, WD$ must be
    declared as the string
120 REM  to be printed in double height characters.
130 REM  X% and Y% must be set to the horizontal and
    vertical text cursor
140 REM positions where the string is to be printed.
150 REM
160 REM
170 WD$="U.K. EINSTEIN USER GROUP":X%=6:Y%=12:GOSUB 250
180 REM
190 REM
200 PRINT CHR$(30);:END:REM    Cursor home, then end.
210 REM
220 REM
230 REM
```

```
240 REM THE SUBROUTINE BEGINS HERE
250 PRINT@X%,Y%,"";:REM    Position cursor.
260 FORJ%=1TOLEN(WD$):REM For each character in the
    string,
270 I%=ASC(MID$(WD$,J%,1)):REM determine the ASCII value.
280 A%=I%*8+6144:REM Convert ASCII value to text pattern
    table address
290 B$=HEX$(VPEEK(A%),2):C$=HEX$(VPEEK(A%+1),2):D$=HEX$
    (VPEEK(A %+2),2):E$ =HEX$(VPEEK(A%+3),2)
300 REM Assign first four bytes of text pattern to four
    strings, as hex numbers
310 REM
320 SHAPE128,B$+B$+C$+C$+D$+D$+E$+E$
330 REM Redefine free text pattern as upper half of
    double height
340 REM    character.
350 PRINT CHR$(128);CHR$(8);CHR$(10);
360 REM  Print redefined character, then backspace and
    linefeed.
370 B$=HEX$(VPEEK(A%+4),2):C$=HEX$(VPEEK(A%+5),2):D$=
    HEX$(VPEEK (A%+6),2): E$=HEX$(VPEEK(A%+7),2)
380 REM Assign second four bytes of text pattern to four
    strings, as hex numbers
390 REM
400 SHAPE128,B$+B$+C$+C$+D$+D$+E$+E$
410 REM    Redefine free text pattern as lower half of
    double height character
420 REM
430 PRINT CHR$(128);CHR$(11);
440 REM Print redefined character, then 'cursor up'.
450 NEXT:REM  Next character in the string. If last one
    printed
460 RETURN:REM return to main program.
```

Bomb the wall before it squashes you. NOTE: Lines 50, 70, 1020, 10020 and 10050 where numbers appear between quotes, press GRAPH/SHIFT and the number indicated all together.

```
   10 CLS32
   20 SPEED248:GOSUB10000
   30 SPEED255
   40 IFX=30THENX=0:PRINT@29,2;"     ":C=C+1:IFC=2THENPRINT@
      2,23;"":IFB>5THENB=B-1
   50 PRINT@X,2;"12":PSG12,4:PSG13,4:POWER=POWER+1
   60 K=KBD:IFK<>0ANDPOWER>4THENGOSUB1000
   70 CH$=MID$(SCRN$(2),X+3,1):IFCH$="3"THENGOTO2000
   80 PRINT@12,0;"SCORE:";SC
   90 IFC=2THENC=0
  100 X=X+1:PRINT@X-1,2;" "
  110 GOTO40
 1000 REM *** FIRE ***
 1010 FORF=3TOB+RND(8):TCOL5,3:CH$=MID$(SCRN$(F),X+1,1):IF
      CH$="3"THEN SC=SC+5:PRINT@12,0;"SCORE:";SC
 1020 PRINT@X,F;"4":TCOL15,3:PSG0,F*10:PSG13,0:PRINT@X,F;" "
 1030 NEXTF
 1040 IFSC=1680THENFORF=15TO1STEP-1:BCOLF:PSG0,F*16:PSG13,0:
      PRINT@0,F;" YOU HAVE DEFEATED THE WALL    ":NEXT:A=INCH
      :RUN
 1050 POWER=0:RETURN
 2000 REM *** END ***
 2010 B$="YOU HAVE BEEN DEFEATED ":C$="DO YOU WISH TO PLAY
      AGAIN?    "
 2020 FORF=31TO1STEP-1:PSG0,F*7:PSG8,31:PSG12,12:PSG7,&78:
      PSG13,0
 2030 PRINT@F,16;LEFT$(B$,32-F)
 2040 NEXTF
 2050 SPEED100:FOR F=0TO500:NEXTF:SPEED255:TCOL1,3
 2060 FORF=31TO1STEP-1:PSG0,F*7:PSG8,31:PSG12,12:PSG7,&78:
      PSG13,0
 2070 PRINT@1,14;"THERE WERE";(1680-SC)/5;"BRICKS LEFT
       "
 2080 FORF=31TO1STEP-1:PSG0,F*7:PSG8,31:PSG12,12:PSG7,&78:
      PSG13,0
 2090 PRINT@F,16;LEFT$(C$,32-F)
 2100 NEXTF
 2110 POKE&013D,0
 2120 K=KBD
 2130 IFK=0THEN2120
 2140 IF K=89THENRUN
 2150 STOP
10000 FORF=0TO3:READN$:SHAPE161+F,N$:NEXTF
10010 DATA"000103E66C3D1309", "0080C06736BCC890", "FF8181818
      18181FF","555 5363 E3E3E1C08"
10020 A$="3333333333333333333333333333"
10030 TCOL15,3:BCOL3:GCOL3:CLS32:B=15
10040 PSG8,31:PSG12,12:PSG7,78
10050 FORF=11TO22:FORG=2TO29:TCOL15,F-10:PRINT@G,F;"3":
      PSG0,F*4:PSG2,G: PSG1 3,0:NEXTG,F:TCOL15,3
10060 RETURN
```

# DRAWING WITH THE EINSTEIN

There are several commercial drawing programs available for your EINSTEIN, these range from PICPEN, given free on your master disc, to the more comprehensive programs such as SCREENPLUS and GRAPHDRAW. The following two programs are very basic but do provide ideas which you may care to expand on.

## JOY DRAW

One of the many competitions held by the U.K.E.U.G. was to write a draw program in just 10 lines. JOY DRAW by Stuart Marshall won the competition, this program allows you to draw your picture using a joystick and dump it to a printer, the graphic dump program is required to be in memory for dumping to printer, instructions on how to add this facility are given in the printer graphics article towards the rear of the book.

```
 1 CLEAR&E260:CLS:PRINT@0,0;"  JOYSTICK FOR UP/DOWN,LEFT/
   RIGHT":PRINT@0,2;"FIRE BUTTON TO ERASE LINE":PRINT@0,4
   ;"C .. TO ERASE"
 2 PRINT@0,6;"   P .. FOR PRINTER DUMP(WILL ONLY WORK
   IF DUMP INSTALLED IN DOS)":PRINT@0,10;"          SPACE
   BAR TO START"
 3 S=KBD:IFS=32THEN GOTO4:ELSEGOTO3
 4 CLS32:SHAPE140,"E0C0A09008040000":GCOL15,0:BCOL4:
   X=256/2:Y= 192/2:A=1/ 300
 5 X%=ADC(0)-112:Y%=(ADC(1)-112)*.75:X=X+X%*A:Y=Y+Y%*A:
   SPRITE0 ,X,Y,1,140
 6 IFX<0THENX=0:ELSEIFX>255THENX=255
 7 IFY<4THENY=4:ELSEIFY>192THENY=192
 8 IFBTN(0)=1THENPLOTX,Y:ELSEIFBTN(0)=0THEN UNPLOTX,Y
 9 S=KBD:IFS=80THENCALL&E270
10 IFS=67THENGOTO4:ELSEGOTO5
```

## WIRE DRAW

This program, was first published in Popular Computing Weekly, here modified by Peter Leadbeater. It illustrates how to make simple wire drawings that can be saved to disc, re-called and modified as required. There is scope for further alterations to the program. i.e. rotation and scaling of the drawings would be nice.

```
 10 RST:TCOL15,1:BCOL1:GCOL15,1
 20 GOSUB370:GOSUB530
 30 IOM5,0:TCOL1,15:PRINT@0,0,CHR$(20)"MOVE CURSOR X";
    @0,1 ;H$;@18,0,"SX=";X;@25,0,"SY=";Y;@18,1,"FX=";XX;
    @25,1, "FY=";YY
 40 SPRITE0,X-4,Y+4,15,130
 50 PRINT@17,1:A=INCH
 60 IFA=4THENX=X+SP:IFX=>254THENX=254
 70 IFA=8THENX=X-SP:IFX<=1  THENX=1
 80 IFA=10THENY=Y-SP:IFY<=1  THENY=1
 90 IFA=11THENY=Y+SP:IFY=>174THENY=174
100 IFA=72 ORA=104THENGOSUB380
110 IFA<>13THEN30
120 TCOL1,15:PRINT@0,0,CHR$(20)"MOVE CURSOR +";@0,1;H$;
    @18,0,"SX=";X;@25,0,"SY=";Y;@18,1,"FX=";XX;@25,1,
    "FY=";YY
130 SPRITE1,XX-4,YY+4,15,131
140 PRINT@17,1:A=INCH
150 IFA=4THENXX=XX+SP:IFXX=>254THENXX=254
160 IFA=8THENXX=XX-SP:IFXX<=1  THENXX=1
170 IFA=10THENYY=YY-SP:IFYY<=1  THENYY=1
180 IFA=11THENYY=YY+SP:IFYY=>174THENYY=174
190 IFA=72 ORA=104THENGOSUB380
200 IFA<>13THEN120
210 TCOL15,1:PRINT@0,0;"SELECT OPERATION";@0,1,"USE
    COMMAND KEYS":TCOL1,15:PRINT@17,1;:A=INCH:IFA=>97
    THENA=A-32
220 IFA=>49ANDA<=57THENSP=A-48:GOTO30
230 IFA=67THENX=XX:Y=YY
240 IFA=68THENDRAWX,YTOXX,YY:PX(C)=X:PX(C+1)=Y:PX(C+2)=XX
    :PX(C+ 3)=YY:C=C+4
250 IFA=69THENDRAWX,YTOXX,YY,1:C=C-4
260 IFA=72THENGOSUB380
270 IFA=76THENTCOL15,1:RUN540
280 IFA=79THENSPRITEOFF:A=INCH
290 IFA=83THENGOSUB670
300 IFA=84THENTCOL15,1:RUN
310 IFA=88THENSPRITEOFF:TCOL15,1:PRINTCHR$(17):CLS40:END
320 GOTO30
330 B=0:DOKE&8000,C:FORF=&8002TO&8002+C-1:POKEF,PX(B):
    B=B+1:NEXT:RETURN
340 IFPX(1)=0THENRETURN
350 C=0:N=DEEK(&8000):FORA=&8002TO&8002+N-1:PX(C)=
    PEEK(A):C=C+1 :NEXT
360 FORA=0TOC-1STEP4:DRAWPX(A),PX(A+1)TOPX(A+2),PX(A+3):
    NEXT:RETURN
```

```
370 DIMPX(1000):C=0
380 TCOL15,1:SPRITEOFF
390 IFA=72ORA=104THENBEEP:GOSUB330
400 CLS32:PRINT@4,0,"GRAPHIC LINE-DRAW UTILITY";@4,1,
    MUL$("-",25) ;@5,4,"To Position Cursors Use"
410 PRINT@5,5,"The Cursor Control Keys";@6,8,"COMMAND
    ACTION";@24,8,"KEY"
420 PRINT@6,9,MUL$("-",14);@24,9,"---";@6,10,"Cursor
    Speed";@24,10,"1-9"
430 PRINT@6,12,"Cursor Off";@25,12,"O";@6,13,"Cursor
    Overlay";@25,13,"C";@6,14,"Draw from x to +"
440 PRINT@25,14,"D";@6,15,"Erase from X to +";@25,15,"E";
    @6,16,"Load Drawing";@25,16,"L";@6,17,"Save Drawing";
    @25,17,"S"
450 PRINT@6,18,"Scratch & Re-start";@25,18,"T";@6,19,
    "Return to DOS";@25,19,"X";@2,22,"(Press SPACE-BAR to
    continue)"
460 K=KBD:IF K<>32THEN 460
470 IFA=72ORA=104 THEN:GOSUB530:GOSUB340
480 X=127:Y=100:XX=X:YY=Y:SP=1
490 H$="(H for HELP)     "
500 SHAPE 130,"8142241818244281"
510 SHAPE 131,"0808087F08080800"
520 RETURN
530 TCOL15,1:BCOL1:CLS32:TCOL1,15:PRINTSPC(64):TCOL15,1
    :RETURN
540 RST:CLEAR&8000:PRINTCHR$(20):DIMPX(1000):GOSUB480:
    CLS32:TCOL1,15:PRINTSPC(64)
550 ON ERR GOTO630
560 PRINT@0,0,"PRESS <ENTER> FOR FILE DIRECTORY"
570 PRINT@0,1,CHR$(17)"LOAD FILE - NAME ?";@19,1;:
    INPUT"";F$:TCOL15,1
580 IF F$="NO"THEN620
590 IF LEN(F$)>8THENF$=LEFT$(F$,8)
600 LOADF$+".OBJ":N=DEEK(&8000):FOR A=&8002TO&8002+N-1
    :PX(C)=PEEK(A):C=C+1:NEXT
610 FORA=0TOC-1STEP4:DRAWPX(A),PX(A+1)TOPX(A+2),PX(A+3)
    :NEXT:BEEP
620 OFF ERR: TCOL1,15:PRINT@0,0,SPC(64):GOTO30
630 CLS40:PRINT@4,0,"THE FOLLOWING FILES ARE AVAILABLE";
    @12,2,"PLEASE SELECT ONE";-0,4;
640 DIR"*.OBJ":PRINT@4,20,"(File Name 'NO' for cursor
    control)";@8,22,"Press SPACE BAR to continue"
650 A=KBD:IF A<>32THEN 650
660 GOTO540
670 REM
680 GOSUB330
690 TCOL1,15:PRINT@0,0,CHR$(17)"SAVE DRAWING BY "
700 PRINT@0,1,"NAME ? ";@7,1;:INPUT"";F$:TCOL15,1:IF
    F$="" THEN 690
710 IF F$="NO"THEN RETURN
720 IF LEN(F$)>8THENF$=LEFT$(F$,8)
730 SAVE F$+".OBJ",&8000,&8000+C+2:RUN
```

## BIORYTHM

An old favorite with some interesting additions.

LISTING NOTES:
LINES 240-260 - Before the words Physical, Emotional and Intellectual type GRAPH/3 twice.
LINE 280 - GRAPH/LEFT CURSOR and GRAPH/RIGHT CURSOR

```
 10 REM*****************************
 20 REM*   BIORYTHM FOR EINSTEIN   *
 30 REM*          BY               *
 40 REM*    PETER P. HEFFERNAN     *
 50 REM*****************************
 60 RST:ORIGIN1,0:MAG2:BCOL4
 70 SHAPE128,"8080808080808080808080808080808000000000000
    0000000000000000000000"
 80 GOSUB710:CLS40:P=-8:GCOL15:BCOL4:D=0:Z=23:
    TCOL15,6:W=W-0
 90 PRINT@0,2;N$" will be";T;"days old today;"
100 PRINT@0,3;" and is a"
110 REM
120 PRINT@0,6;"it is";W;"days to XMAS"
130 TCOL15,12
140 PRINT@0,5;D(1);"/"M(1);"/"Y(1)"    "
150 GOSUB1220
160 IFW=0THENPRINT@0,6;"** MERRY CHRISTMAS *":TCOL15,6:
    PRINT@20,6;"   "
170 REM
180 REM
190 GOSUB1410
200 TCOL14,4
210 DRAW1,65TO248,65
220 GCOL15
230 DEF FN A(X)=SIN(RAD(360*(X/Z)))*60
240 TCOL15,13:PRINT@25,4;"33 Physical    "
250 TCOL3,13:PRINT@25,5;"33 Emotional    "
260 TCOL11,13:PRINT@25,6;"33 Intellectual"
270 TCOL15,4
280 PRINT@0,0 ;"to move day-line use 'C'forand 'B'for"
290 PRINT@0,1;" Press 'space'to EXIT:'enter' to re-run"
300 FORX=T-12 TOT+20STEP.125
310 P=P+1
320 LETY=FNA(X)
330 PLOTP+8,Y+65
340 NEXTX
350 D=D+1:P=-8:X=96
360 IFD=1THENZ=28:GCOL3:GOTO230
370 IFD=2THENZ=33:GCOL11:GOTO230
380 FORJ=63TO67
390 FORI=0TO248 STEP8
400 PLOT I,J :NEXTI
410 NEXTJ
420 TCOL15,12
430 A$=KBD$
440 IFA$=CHR$(13)THENRST:RUN
```

```
450 IFA$=CHR$(32)THENRST:END
460 IFA$="B"THENX=X+8:GOSUB1140
470 IFA$="C"THENX=X-8:GOSUB1050
480 IFX<0THENX=0
490 IFX>248THENX=248
500 SPRITE4,X  ,100,15,128
510 SPRITE1,X  ,84 ,15,128
520 SPRITE2,X  ,68 ,15,128
530 SPRITE7,X  ,36 ,15,128
540 SPRITE8,X  ,20 ,15,128
550 SPRITE5,X  ,116,15,128
560 SPRITE3,X  ,52 ,15,128
570 SPRITE6,X  ,132,15,128
580 PRINT@0,5;D(1);"/"M(1);"/"Y(1)"    "
590 TCOL15,6
600 PRINT@0,2;N$" will be";T;"days old today;"
610 PRINT@0,6;"it is";W;"days to XMAS"
620 TCOL6,15
630 IFD(1)=LANDM(1)=MTHENPRINT@0,7;"HAPPY BIRTHDAY ";N$;"
    YOU ARE";Y(1)-R;"TODAY"
640 TCOL15,0
650 IFD(1)<>LORM(1)<>MTHENPRINT@0,7;"
                    "
660 TCOL6,15
670 IFW=0THENPRINT@0,6;"** MERRY CHRISTMAS *":TCOL
    15,6:PRINT@20,6;"  "
680 TCOL15,6
690 GOSUB1220
700 GOTO430
710 POKE&D000,00,31,29
720 POKE&D003,31,30,31,30,31,31,31,31,30,31
730 CLS
740 PRINT@2,4;"ENTER YOUR NAME";
750 INPUTN$
760 IFLEN(N$)>8THENBEEP:GOTO730
770 PRINT@2,10;"YEAR CHART REQD.FORie(1973)";
780 INPUTY(1)
790 GOSUB1930
800 IFY(1)<1582ORY(1)>3000THENBEEP:GOTO730
810 PRINT@2,11;"MONTH CHART FOR(1-12).....";
820 INPUTM(1)
830 IFM(1)<1ORM(1)>12THENBEEP:GOTO730
840 IF Y(1)MOD4>0THENPOKE&D002,28
850 PRINT@2,12;"DATE CHART REQD.FOR.....";
860 INPUTD(1)
870 IFD(1)<1ORD(1)>31THENBEEP:GOTO730
880 IFY(1)MOD4=0ANDM(1)=2ANDD(1)>28THENBEEP:GOTO730
890 IFM(1)>2THENGOSUB1670
900 IFM(1)=1ORM(1)=2THENGOSUB1590
910 PRINT@2,16;"YEAR OF BIRTH ie(1955).....";
920 INPUTY:R=Y
930 IFY>Y(1)THENBEEP:GOTO730
940 IFY<1582ORY>3000THENBEEP:GOTO730
950 PRINT@2,17;"MONTH OF BIRTH(1-12).....";
```

```
 960 INPUTM:K=M
 970 IFM<1ORM>12THENBEEP:GOTO730
 980 PRINT@2,18;"DATE.....";
 990 INPUTD:L=D
1000 IFD<1ORD>31THENBEEP:GOTO730
1010 IFYMOD4=0ANDM=2ANDD>28THENBEEP:GOTO730
1020 IFM>2THENGOSUB1840
1030 IFM=1ORM=2THENGOSUB1760
1040 RETURN
1050 K=PEEK(&D000+M(1))
1060 IFX>248ORX<0THENRETURN
1070 D(1)=D(1)-1
1080 IFD(1)<1ANDM(1)=1THEND(1)=31:Y(1)=Y(1)-1:M(1)=12:
     GOTO1100
1090 IFD(1)<1THEND(1)=((PEEK(&D000+(M(1)-1)))):M(1)=M(1)-1
1100 T=T-1:F=F-1:W=W+1:IF(Y(1)+1)MOD4=0ANDW=366THENW=0
1110 IF(Y(1)+1)MOD4>0ANDW=365THENW=0
1120 GOSUB1220
1130 RETURN
1140 K=PEEK(&D000+M(1))
1150 IFX>248ORX<0THENRETURN
1160 D(1)=D(1)+1
1170 IFD(1)>31ANDM(1)=12THEND(1)=1:Y(1)=Y(1)+1:M(1)=1:
     GOTO1190
1180 IFD(1)>KTHEND(1)=1:M(1)=M(1)+1:IFM(1)=13THENM(1)=1
1190 T=T+1:F=F+1:W=W-1:IFW<1THENGOSUB1970
1200 GOSUB1220
1210 RETURN
1220 REM
1230 TCOL15,12
1240 IFF=8THENF=1
1250 IFF=0THENF=7
1260 ONFGOTO1270,1290,1310,1330,1350,1370,1390
1270 PRINT@1,4;"Sunday    "
1280 RETURN
1290 PRINT@1,4;"Monday    "
1300 RETURN
1310 PRINT@1,4;"Tuesday   "
1320 RETURN
1330 PRINT@1,4;"Wednesday"
1340 RETURN
1350 PRINT@1,4;"Thursday  "
1360 RETURN
1370 PRINT@1,4;"Friday    "
1380 RETURN
1390 PRINT@1,4;"Saturday  "
1400 RETURN
1410 REM
1420 TCOL11,6
1430 ONF(3)GOTO1440,1460,1480,1500,1520,1540,1560
1440 PRINT@10,3;"'Sundays child'"
1450 RETURN
1460 PRINT@10,3;"'Mondays child'"
1470 RETURN
```

```
1480 PRINT@10,3;"'Tuesdays child'"
1490 RETURN
1500 PRINT@10,3;"'Wednesdays child'"
1510 RETURN
1520 PRINT@10,3;"'Thursdays child'"
1530 RETURN
1540 PRINT@10,3;"'Fridays child'"
1550 RETURN
1560 PRINT@10,3;"'Saturdays child'"
1570 RETURN
1580 REM NO.OF DAYS
1590 A=365*Y(1)+D(1)
1600 A=A+(31*(M(1)-1))
1610 A=A+INT((Y(1)-1)/4)
1620 A=A-INT(.75*(INT(((Y(1)-1)/100)+1)))
1630 O=A
1640 Q=A+(INT(-A/7)*7)
1650 F=Q+7
1660 RETURN
1670 A=365*Y(1)+D(1)
1680 A=A+(31*(M(1)-1))
1690 A=A-INT((.4*M(1))+2.3)
1700 A=A+INT(Y(1)/4)
1710 A=A-INT(.75*(INT(Y(1)/100)+1))
1720 O=A
1730 U=A+(INT(-A/7)*7)
1740 F=U+7
1750 RETURN
1760 A=365*Y+D
1770 A=A+(31*(M-1))
1780 A=A+INT((Y-1)/4)
1790 A=A-INT(.75*(INT(((Y-1)/100)+1)))
1800 Q=A+(INT(-A/7)*7)
1810 T=O-A
1820 F(3)=Q+7
1830 RETURN
1840 A=365*Y+D
1850 A=A+(31*(M-1))
1860 A=A-INT((.4*M)+2.3)
1870 A=A+INT(Y/4)
1880 A=A-INT(.75*(INT(Y/100)+1))
1890 U=A+(INT(-A/7)*7)
1900 T=O-A
1910 F(3)=U+7
1920 RETURN
1930 M(1)=12:D(1)=25
1940 GOSUB1670
1950 W=A
1960 RETURN
1970 IFW=0THENPRINT@0,6;"** MERRY CHRISTMAS *"
1980 IF(Y(1)+1)MOD4=0THENJ=366
1990 IF(Y(1)+1)MOD4>0THENJ=365
2000 IFW<0THENW=J+W
2010 RETURN
```

If we use sprites in a Basic program we get smooth movement, however the more sprites we have to move the slower the program runs. In 'A Day At The Races' there are 6 horses which have to moved at once, each horse having two shapes to give the appearance of animation. There are also the furlong markers to be moved and so, to ensure a more realistic effect, machine code is employed.

Before entering the game program, the machine code must be entered, so type in the following loader program having first put a disc, with space for the object file which will be produced, into the current drive. As there are a lot of data statements, a quick way of entering them is to type in the first data line, (line 50), then once entered, move the cursor to alter this line number to the subsequent line numbers of the DATA statements, (60-850). Now list the program and you should have lines 50-850 containing the same DATA. Now move the cursor and change the value of the DATA statements in lines 60-850 to the values as printed, remember to press enter after each line is completed.

```
 10 CLEAR &8000
 15 FOR F=&8000 TO &8282
 20 READ A:POKE F,A
 25 NEXT F
 30 SAVE "HORSE.OBJ",&8000,&8282
 40 END
 50 DATA &3E,&47,&F6,&40,&4F,&06,&07,&CD
 60 DATA &38,&82,&01,&1F,&08,&CD,&38,&82
 70 DATA &01,&1F,&09,&CD,&38,&82,&01,&1F
 80 DATA &0A,&CD,&38,&82,&01,&01,&0C,&CD
 90 DATA &38,&82,&C3,&67,&81,&21,&64,&80
100 DATA &01,&80,&02,&7E,&D3,&09,&79,&D3
110 DATA &09,&0C,&23,&10,&F6,&C9,&D5,&CF
120 DATA &C1,&D1,&C9,&01,&00,&1D,&CD,&36
130 DATA &80,&06,&06,&C5,&E5,&06,&40,&7E
140 DATA &D3,&08,&23,&05,&E3,&E3,&C2,&47
150 DATA &80,&E1,&C1,&10,&EE,&06,&40,&11
160 DATA &40,&00,&19,&7E,&D3,&08,&23,&E3
170 DATA &E3,&10,&F8,&C9,&02,&C2,&F4,&00
180 DATA &00,&00,&00,&03,&02,&00,&00,&00
190 DATA &01,&03,&04,&08,&11,&20,&00,&00
200 DATA &00,&00,&01,&73,&FF,&FF,&FF,&FF
210 DATA &FF,&71,&40,&80,&00,&00,&00,&70
220 DATA &63,&C7,&FF,&EE,&FE,&FF,&FF,&FE
230 DATA &FF,&FF,&40,&00,&00,&00,&00,&60
240 DATA &F8,&DE,&80,&00,&00,&00,&00,&00
250 DATA &F8,&C4,&22,&11,&08,&04,&00,&00
260 DATA &01,&01,&02,&02,&01,&01,&00,&00
270 DATA &FF,&08,&08,&08,&08,&08,&08,&C0
280 DATA &20,&20,&10,&10,&20,&20,&C0,&C0
290 DATA &FF,&C0,&C0,&C0,&C0,&C0,&C0,&00
300 DATA &01,&01,&02,&02,&01,&01,&00,&00
310 DATA &FF,&08,&08,&08,&08,&08,&08,&C0
```

```
320 DATA &20,&20,&10,&10,&20,&20,&C0,&C0
330 DATA &FF,&C0,&C0,&C0,&C0,&C0,&C0,&00
340 DATA &00,&00,&00,&02,&03,&00,&01,&01
350 DATA &01,&00,&00,&00,&00,&01,&02,&00
360 DATA &00,&00,&00,&01,&73,&FF,&FF,&FF
370 DATA &FF,&FC,&F0,&F0,&A0,&40,&80,&00
380 DATA &30,&31,&F3,&EF,&CF,&FF,&FF,&FE
390 DATA &FE,&FF,&43,&02,&05,&0A,&04,&00
400 DATA &60,&F0,&F8,&8E,&00,&00,&00,&00
410 DATA &00,&00,&80,&80,&00,&00,&00,&00
420 DATA &01,&01,&02,&02,&01,&01,&00,&00
430 DATA &FF,&08,&08,&08,&08,&08,&08,&C0
440 DATA &20,&20,&10,&10,&20,&20,&C0,&C0
450 DATA &FF,&C0,&C0,&C0,&C0,&C0,&C0,&00
460 DATA &01,&01,&02,&02,&01,&01,&00,&00
470 DATA &FF,&08,&08,&08,&08,&08,&08,&C0
480 DATA &20,&20,&10,&10,&20,&20,&C0,&C0
490 DATA &FF,&C0,&C0,&C0,&C0,&C0,&C0,&00
500 DATA &CD,&25,&80,&06,&01,&C5,&21,&67
510 DATA &80,&CD,&3B,&80,&06,&02,&C5,&01
520 DATA &0B,&06,&CD,&38,&82,&01,&01,&0D
530 DATA &CD,&38,&82,&21,&4A,&82,&01,&00
540 DATA &3B,&CD,&36,&80,&01,&0B,&06,&CD
550 DATA &38,&82,&01,&01,&0D,&CD,&38,&82
560 DATA &06,&0E,&C5,&06,&04,&7E,&D3,&08
570 DATA &23,&E5,&E1,&10,&F8,&C1,&10,&F2
580 DATA &21,&E7,&80,&CD,&1E,&82,&01,&0F
590 DATA &06,&CD,&38,&82,&01,&05,&0D,&CD
600 DATA &38,&82,&CD,&3B,&80,&C1,&10,&B6
610 DATA &3A,&00,&83,&FE,&F0,&20,&0B,&E5
620 DATA &21,&7B,&82,&BE,&E1,&20,&03,&C3
630 DATA &DD,&81,&3A,&7B,&82,&D6,&08,&32
640 DATA &7B,&82,&32,&7F,&82,&C1,&10,&8D
650 DATA &CD,&E4,&81,&C9,&21,&4B,&82,&CD
660 DATA &09,&82,&21,&53,&82,&CD,&09,&82
670 DATA &21,&5B,&82,&CD,&09,&82,&21,&63
680 DATA &82,&CD,&09,&82,&21,&6B,&82,&CD
690 DATA &09,&82,&21,&73,&82,&CD,&09,&82
700 DATA &C9,&ED,&5F,&F6,&FC,&2F,&5F,&7E
710 DATA &83,&77,&23,&23,&23,&23,&7E,&83
720 DATA &CD,&3F,&82,&77,&C9,&C9,&C5,&E5
730 DATA &D5,&06,&3F,&C5,&06,&FF,&11,&01
740 DATA &00,&21,&00,&00,&19,&38,&FD,&10
750 DATA &F8,&C1,&10,&EF,&D1,&E1,&C1,&C9
760 DATA &78,&D3,&02,&79,&D3,&03,&C9,&FE
770 DATA &64,&D8,&F5,&3E,&F0,&32,&00,&83
780 DATA &F1,&C9,&10,&08,&A0,&01,&10,&18
790 DATA &A4,&01,&28,&08,&A8,&08,&28,&18
800 DATA &AC,&08,&40,&08,&B0,&04,&40,&18
810 DATA &B4,&04,&58,&08,&B8,&0D,&58,&18
820 DATA &BC,&0D,&70,&08,&C0,&0E,&70,&18
830 DATA &C4,&0E,&88,&08,&C8,&07,&88,&18
840 DATA &CC,&07,&00,&28,&D0,&0F,&98,&28
850 DATA &D4,&0F,&00,&FF
```

Although the loader program is not needed, as once run it creates an OBJect file for use by the main program, it would be wise to save it just in case a mistake has been made. Run the above program to create the file HORSE.OBJ. You can test the machine code with the following short program.

```
10 TCOL1,15
20 CLS
30 CLEAR &8000
40 LOAD "HORSE.OBJ"
50 FOR F=1 TO 60
60 CALL &8000
70 NEXT F
80 END
```

When run you should have six horses galloping across the screen from left to right, plus two furlong markers moving from right to left. If not reload the loader program and check the data. If all is o.k. reset the machine go back into BASIC and type in the main program.

```
  50 RST
 100 CLEAR&8000
 150 LOAD"HORSE.OBJ"
 200 GOSUB 4700
 250 GOTO1050
 300 CALL&8000
 350 FOR F=0 TO150:CALL&8167:A=PEEK(&8222):IF A>&27THEN
     A=A-1
 400 IFPEEK(&8300)<>0THENPOKE(&820C),&F8
 450 POKE(&8222),A
 500 FORG=&824FTO&8277STEP8:IFPEEK(G)>237THENGOTO3700
 550 NEXTG
 600 NEXT
 650 FOR F=&824BTO&8273 STEP8
 700 POKE(F),8:NEXT
 750 FOR F=&824FTO&8277 STEP8
 800 POKE(F),24:NEXT
 850 POKE(&8222),&3F
 900 POKE(&8300),0:POKE(&820C),&FC
 950 POKE(&827B),40:POKE(&827F),40
1000 GOTO 300
1050 REMSTART
1100 TCOL1,2:BCOL2:CLS40
1150 GOSUB4200
1200 GOSUB4550
1250 FOR A=1TO6
1300 TP$=J$(RND(6)+1)
1350 R$(A)=TP$
1400 FORG=0TOA-1:IF TP$=R$(G)THEN1300
1450 NEXTG
1500 NEXTA
1550 FOR A=1TO6
1600 TP$=H$(RND(6)+1)
```

```
1650 S$(A)=TP$
1700 FORG=0TOA-1:IF TP$=S$(G)THEN1600
1750 NEXTG
1800 NEXTA
1850 RESTORE1900:FORF=1TO6:READO$(F):NEXTF
1900 DATA"2/1","5/4","3/1","7/4","9/2","5/2"
1950 FORF=1TO6
2000 TP$=O$(RND(6)+1):OD$(F)=TP$
2050 FORG=0TOF-1:IFTP$=OD$(G)THEN2000
2100 NEXTG,F
2150 FORF=1TO6:HO$(F)=STR$(F)+S$(F)+R$(F)+OD$(F):NEXTF
2200 TCOL1,2:PRINT@1,2;"              NEXT RACE
     "
2250 TCOL15,2:N=5:FORF=1TO6
2300 PRINT@1,N;F;@6,N;S$(F);@20,N;R$(F);@34,N;OD$(F):N=N+2
2350 NEXTF
2400 PRINT@9,18;"PRESS G TO START"
2450 TCOL15,2:PRINT@15,18;:K=INCH:IFK<>71THEN2450
2500 PRINT@5,18;"THEY'RE UNDER STARTERS ORDERS!"
2550 VOICE0,30,15,2,50,5:TEMPO7
2600 M$="G7B7+D5+D5+D5+D5R7B7B7B7R8G7B7G7D8R":MUSIC "VO"
     +M$
2650 GCOL15,12:TCOL15,12:BCOL12:CLS40
2700 DRAW0,181 TO 256,181
2750 DRAW 0,29 TO 256,29
2800 FOR F=4TO 252STEP 8
2850 DRAW F,181 TO F,175
2900 DRAW F,29 TO F,23
2950 NEXT:MAG2
3000 SPRITE0,8,56,7,&A0
3050 SPRITE1,24,56,7,&A4
3100 SPRITE2,08,80 ,14,&A0
3150 SPRITE3,24,80,14,&A4
3200 SPRITE4,08,104,13,&A0
3250 SPRITE5,24,104,13,&A4
3300 SPRITE6,08,128,4,&A0
3350 SPRITE7,24,128,4,&A4
3400 SPRITE8,08,152,8,&A0
3450 SPRITE9,24,152,8,&A4
3500 SPRITE10,08,176,1,&A0
3550 SPRITE11,24,176,1,&A4
3600 M$="+D5+D5+D5+D5R7B7B7B7B7R8D7D7D7G9R"
3650 MUSIC "VO"+M$:PRINT@10,0;"THEY'RE OFF":FORF=1TO300:
     NEXTF :PRINT@10,0;SPC(12):GOTO650
3700 REM
3750 TCOL15,1:PRINT@11,0;"RESULT TO FOLLOW"
3800 C=1:FORF=&824FTO&8277STEP8:HO$(C)=HO$(C)+STR$(PEEK(F))
     :C=C+1:NEXTF
3850 N=6:FORJ=1TO(N-1):FORI=(J+1)TON:L=N+J-I+1
3860 W=VAL(RIGHT$(HO$(L),3)):V=VAL(RIGHT$(HO$(J),3))
3870 IFW>=VTHENT$=HO$(L):HO$(L)=HO$(J):HO$(J)=T$
3900 NEXTI:NEXTJ
3950 FORG=1TO8:TCOL15,2:FOR F=1TO200:NEXTF:PRINT@11,0;
     "RESULT TO FOLLOW":TCOL2,15:FORF=1TO200:NEXTF
```

```
3960 TCOL2,15:PRINT@11,0;"RESULT TO FOLLOW":NEXTG:TCOL2,2:
     SPRITEOFF:CLS32
4000 TCOL4,2:FORF=20TO4STEP-1:PRINT@5,F;CHR$(235);@24,F;CHR$
     (235) :NEXTF:TCOL1,2:FORF=0TO3:PRINT@5,F;MUL$(CHR$(219)
     ,20):NEXTF
4050 PRINT@11,1;" RESULTS ";@13,2;"FRAME"
4060 N=4:TCOL1,15:FORF=1TO3:PRINT@6,F+N;LEFT$(HO$(F),2);"
     "MID$(HO$(F),3,10);" ";MID$(HO$(F),23,3)" ":N=N+1:
     NEXTF
4100 PRINT@3,22;"PRESS A KEY FOR NEXT RACE"
4150 PRINT@14,22;:K=INCH:TCOL2,2:CLS40:GOTO1200
4200 REM
4250 RESTORE4300:FORA=1TO6:READH$(A):NEXTA
4300 DATA"HENRYBUCK","SPYCATCHER","QUATERMASS","PIED
     PIPER","BLACKADDER","MAGGIE MAY"
4350 RESTORE4400:FORF=1TO6:READJ$(F):NEXTF
4400 DATA"M. BIRCH  ","A.BOND     ","P. EDDERY "
4450 DATA"G. SEXTON ","W. CARSON ","S. CAUTHEN"
4500 RETURN
4550 TCOL4,15:PRINT@6,0;"A  D A Y  A T  T H E  R A C E S"
4600 TCOL14,2:PRINT@3,4;"    HORSES        JOCKEYS     ODDS"
4650 RETURN
4700 REMSETUPANDINFO
4750 DIMHO$(27)
4800 REM
4850 SHAPE160,"0000000003020000"
4900 SHAPE161,"0001030408112000"
4950 SHAPE162,"0000000173FFFFFF"
5000 SHAPE163,"FFFF714080000000"
5050 SHAPE164,"7063C7FFEEFEFFFF"
5100 SHAPE165,"FEFFFF4000000000"
5150 SHAPE166,"60F8DE8000000000"
5200 SHAPE167,"00F8C42211080400"
5250 SHAPE168,"0000000002030001"
5300 SHAPE169,"000000000173FFFF"
5350 SHAPE170,"003031F3EFCFFFFF"
5400 VOICE 0,31,15,2,100,2:TCOL15,2:BCOL2:CLS40:N$="
     WELCOME TO A DAY AT THE RACES          ":L=1
5450 M$="G4G4G7E7G8A6G6E7RE8D8RE8D8G4G4G7E7G7A7RG7E7D7D7E7
     D7C7C7C6E7G7+C9A7A7+C7A7G7RG4G4G7E7G7G7G7A7G5E7D6D5D7E
     7D6D6C8R "
5500 TEMPO7: SPEED245:PRINT@5,1;N$
5550 SPEED 255:DRAW32,175TO223,175:SPEED245
5600 PRINT@0,3;"ALL YOU HAVE TO DO IN THIS GAME IS SIT
     WITH YOUR FRIENDS OR FAMILY ";@28,4;"AND ENJOY   ALL
     THE THRILLS"
5650 PRINT@16,5;"AND EXCITEMENT OF
         A  D A Y  A T  T H E  R A C E S   "
5700 PRINT@0,9;"AS YOU ENJOY A FRIENDLY BET OR JUST SIT
     BACK AND WATCH AS TOP HORSE AND JOCKEYS ARE PUT
     THROUGH THEIR PACES"
5750 SPEED 255:TCOL1,15: PRINT@12,17;"G O O D  L U C K"
5800 MUSIC "VO"+M$
5850 RETURN
```

```
3960 TCOL2,15:PRINT@11,0;"RESULT TO FOLLOW":NEXTG:TCOL2,2:
     SPRITEOFF:CLS32
4000 TCOL4,2:FORF=20TO4STEP-1:PRINT@5,F;CHR$(235);@24,F;CHR$
     (235) :NEXTF:TCOL1,2:FORF=0TO3:PRINT@5,F;MUL$(CHR$(219)
     ,20):NEXTF
4050 PRINT@11,1;" RESULTS ";@13,2;"FRAME"
4060 N=4:TCOL1,15:FORF=1TO3:PRINT@6,F+N;LEFT$(HO$(F),2);"
     "MID$(HO$(F),3,10);" ";MID$(HO$(F),23,3)" ":N=N+1:
     NEXTF
4100 PRINT@3,22;"PRESS A KEY FOR NEXT RACE"
4150 PRINT@14,22;:K=INCH:TCOL2,2:CLS40:GOTO1200
4200 REM
4250 RESTORE4300:FORA=1TO6:READH$(A):NEXTA
4300 DATA"HENRYBUCK","SPYCATCHER","QUATERMASS","PIED
     PIPER","BLACKADDER","MAGGIE MAY"
4350 RESTORE4400:FORF=1TO6:READJ$(F):NEXTF
4400 DATA"M. BIRCH  ","A.BOND     ","P. EDDERY "
4450 DATA"G. SEXTON ","W. CARSON ","S. CAUTHEN"
4500 RETURN
4550 TCOL4,15:PRINT@6,0;"A  D A Y  A T  T H E  R A C E S"
4600 TCOL14,2:PRINT@3,4;"    HORSES        JOCKEYS     ODDS"
4650 RETURN
4700 REMSETUPANDINFO
4750 DIMHO$(27)
4800 REM
4850 SHAPE160,"0000000003020000"
4900 SHAPE161,"0001030408112000"
4950 SHAPE162,"0000000173FFFFFF"
5000 SHAPE163,"FFFF714080000000"
5050 SHAPE164,"7063C7FFEEFEFFFF"
5100 SHAPE165,"FEFFFF4000000000"
5150 SHAPE166,"60F8DE8000000000"
5200 SHAPE167,"00F8C42211080400"
5250 SHAPE168,"0000000002030001"
5300 SHAPE169,"000000000173FFFF"
5350 SHAPE170,"003031F3EFCFFFFF"
5400 VOICE 0,31,15,2,100,2:TCOL15,2:BCOL2:CLS40:N$="
     WELCOME TO A DAY AT THE RACES          ":L=1
5450 M$="G4G4G7E7G8A6G6E7RE8D8RE8D8G4G4G7E7G7A7RG7E7D7D7E7
     D7C7C7C6E7G7+C9A7A7+C7A7G7RG4G4G7E7G7G7G7A7G5E7D6D5D7E
     7D6D6C8R "
5500 TEMPO7: SPEED245:PRINT@5,1;N$
5550 SPEED 255:DRAW32,175TO223,175:SPEED245
5600 PRINT@0,3;"ALL YOU HAVE TO DO IN THIS GAME IS SIT
     WITH YOUR FRIENDS OR FAMILY ";@28,4;"AND ENJOY   ALL
     THE THRILLS"
5650 PRINT@16,5;"AND EXCITEMENT OF
         A  D A Y  A T  T H E  R A C E S   "
5700 PRINT@0,9;"AS YOU ENJOY A FRIENDLY BET OR JUST SIT
     BACK AND WATCH AS TOP HORSE AND JOCKEYS ARE PUT
     THROUGH THEIR PACES"
5750 SPEED 255:TCOL1,15: PRINT@12,17;"G O O D  L U C K"
5800 MUSIC "VO"+M$
5850 RETURN
```

Once typed in and run this program offers you the plenty of scope to be your own detective. A face will appear together with a comprehensive list of features which you are able to alter. Once satisfied with the results you can save it and recover it for future reference or alteration.

```
 10 REM++++++++++++++++++++++++++++
 20 REM+                          +
 30 REM+  FACES  FOR  UKEUG       +
 40 REM+  by PETE HEFFERNAN       +
 50 REM++++++++++++++++++++++++++++
 60 CLEAR&B000:PRINT CHR$(20):MAGO
 70 IOMO,0
 80 ON ERR   GOTO3110
 90 REM
100 REM
110 CLS
120 SHAPE140,"78C0A09080000000"
130 ORIGIN0,0
140 Q=.66:W=.88
150 GOTO370:REM DIMENTION
160 REM
170 RST:END
180 CLS
190 FORI=1TO186
200 READ X(I)
210 X(I)=X(I)-60
220 X(I)=X(I)*Q
230 N(I)=X(I)
240 NEXTI
250 FOR I=1TO186
260 READ Y(I)
270 Y(I)= Y(I)*W
280 M(I)=Y(I)
290 NEXTI
300 GOTO600:REM DRAW
310 REM
320 REM
330 REM
340 REM
350 REM
360 REM
370 DIM F(40):DIM X(186):DIM Y(186):F(1)=5:F(2)=5
380 DIM M(186):DIM N(186)
390 X=3:Y=10:Z=3:GOSUB 530:X=11:Y=16:Z=6:GOSUB530:X=19:
    Y=26:Z=7:GOSUB530:X=30:Y=37:Z=3:GOSUB530:X=34:Y=36:
    Z=2:GOSUB530
400 F(17)=4:F(18)=4:F(23)=3:F(24)=3
410 F(27)=11:F(28)=13:F(29)=13
420 DIM X$(186):DIM Y$(186)
430 PRINT@5,10;"(N)orm or (F)ile";
440 K$=INCH$
450 IFK$="F"THENGOTO2930
460 IFK$="N"THEN180
```

```
 470 GOTO440
 480 REM
 490 REM
 500 REM
 510 REM
 520 REM
 530 FOR I=XTOY:F(I)=Z:NEXTI
 540 RETURN
 550 REM
 560 REM
 570 REM
 580 REM
 590 REM
 600 ORIGIN86-((X(2)-X(1))+X(1)),92-Y(1)
 610 ELLIPSEX(1),Y(1),.8
 620 ELLIPSEX(2),Y(2),.8
 630 I=2
 640 FORJ=1TO37
 650 I=I+1
 660 L=1
 670 IFL=F(J)THEN GOTO710
 680 DRAW X(I),Y(I)TO X(I+1),Y(I+1)
 690 L=L+1:I=I+1
 700 GOTO 670
 710 NEXTJ
 720 IFS<>5THENGOTO1030
 730 RETURN
 740 REM
 750 REM
 760 TCOL15,6
 770 PRINT@23,0;"  'esc' to EXIT "
 780 TCOL15,4:FORF=1TO20:PRINT@23,F;SPC(16):NEXTF
 790 PRINT@23,1;"L/IRIS  R/IRIS "
 800 PRINT@23,2;"B/L LID B/R LID"
 810 PRINT@23,3;"B/L EYE B/R EYE"
 820 PRINT@23,4;"T/L EYE T/R EYE"
 830 PRINT@23,5;"L/EYE L R/EYE L"
 840 PRINT@23,6;"L/NOSE  R/NOSE "
 850 PRINT@23,7;"L/NOST  R/NOST "
 860 PRINT@23,8;"T/LBROW T/RBROW"
 870 PRINT@23,9;"B/LBROW B/RBROW"
 880 PRINT@23,10;"T/U LIP B/U LIP"
 890 PRINT@23,11;"T/L LIP B/L LIP"
 900 PRINT@23,12;"L/FACE  R/FACE "
 910 PRINT@23,13;"L/EAR   R/EAR  "
 920 PRINT@23,14;"JAW     HAIR   "
 930 PRINT@23,15;"TP HEAD L/CH LN"
 940 PRINT@23,16;"R/CH LN L/CH BN"
 950 PRINT@23,17;"R/CH BN L/U/LIP"
 960 PRINT@23,18;"R/U/LIP C/CLEFT"
 970 PRINT@23,19;"C/LINE  WIDTH  "
 980 PRINT@23,20;"HEIGHT  SAVE   "
 990 RETURN
1000 REM
```

```
1010 REM
1020 REM
1030 GOSUB760
1040 C=1:R=23:TCOL1,4:GOSUB1560
1050 GOTO 1140
1060 SPRITE OFF
1070 A$=INCH$
1080 IF A$=CHR$(10)THENC=C+1:GOSUB1530
1090 IF A$=CHR$(11)THENC=C-1:GOSUB 1260
1100 IFA$=CHR$(4)THENR=31:GOSUB1350
1110 IFA$=CHR$(8)THENR=23:GOSUB1440
1120 IFA$=CHR$(13)THENGOSUB2150
1130 IFA$=CHR$(27)THENGOTO170
1140 TCOL15,4:PRINT@23,21;SPC(15)
1150 PRINT@25,21;F(H-2);"Points"
1160 TCOL1,4
1170 PRINT@23,22;"SELECT FEATURE "
1180 PRINT@23,23;"  TO  ALTER    ";
1190 TCOL15,4
1200 GOTO1060
1210 REM
1220 REM
1230 REM
1240 REM
1250 REM
1260 IFC<1 THEN BEEP:C=1 :A$="":RETURN
1270 TCOL15,4:PRINT@R,(C+1);MID$(SCRN$(C+1),R+1,7):
     TCOL1,4
1280 REM
1290 PRINT CHR$(23)
1300 PRINT@R ,C;MID$(SCRN$(C),R+1,7)
1310 PRINT CHR$(23)
1320 IFR=23THENH=(C*2)+1
1330 IFR=31THENH=(C*2)+2
1340 RETURN
1350 R=31
1360 TCOL15,4:PRINT@23,(C  );MID$(SCRN$(C  ),24,7)
1370 REM
1380 PRINT CHR$(23)
1390 TCOL1,4:  PRINT@31,C;MID$(SCRN$(C),32,7)
1400 PRINT CHR$(23)
1410 IFR=23THENH=(C*2)+1
1420 IFR=31THENH=(C*2)+2
1430 RETURN
1440 R=23
1450 TCOL15,4: PRINT@31,(C  );MID$(SCRN$(C  ),32,7)
1460 REM
1470 PRINT CHR$(23)
1480 TCOL1,4:PRINT@R ,C;MID$(SCRN$(C),R+1,7)
1490 PRINT CHR$(23)
1500 IFR=23THENH=(C*2)+1
1510 IFR=31THENH=(C*2)+2
1520 RETURN
1530 IFC>20THEN BEEP:C=20:A$="":RETURN
```

```
1540 TCOL15,4:PRINT@R,(C-1);MID$(SCRN$(C-1),R+1,7):
     TCOL1,4
1550 REM
1560 PRINT CHR$(23)
1570 PRINT@R ,C;MID$(SCRN$(C),R+1,7)
1580 PRINT CHR$(23)
1590 IFR=23THENH=(C*2)+1
1600 IFR=31THENH=(C*2)+2
1610 RETURN
1620 REM
1630 REM
1640 REM
1650 REM
1660 REM
1670 DATA135,190,134,128,133,140,135,190,184,189,196,190,
     119,133,147,177,190,203,121,133,147,177,191,201
1680 DATA118,132,148,176,191,204,127,135,144,178,187,196,
     156,156,156,154,156,161,166,166,166,168,167,161
1690 DATA150,147,146,148,153,161,173,176,177,174,170,163,
     112,113,125,139,150,152,171,173,186,199,208,211
1700 DATA112,124,138,152,171,187,200,210,137,149,156,162,
     168,177,187,138,148,156,163,170,178,186,138,149,156,
     163,170,177,186
1710 DATA141,148,155,163,171,179,185,103,101,104,219,222,
     218,99,92,88,90,94,99,104,224,231,234,232,230,224,
     219,104,108,115
1720 DATA129,147,162,180,196,207,215,219,101,107,114,120,
     131,146,160,174,188,201,210,217,222
1730 DATA93,78,76,82,99,129,158,188,217,236,245,250,233,
     145,139,135,178,185,190,105,109,112,218,214,211,159,
     159,165, 165,162
1740 DATA162,153,162,173
1750 REM************************
1760 DATA105,105,109,106,101,106,109,109,106,101,106,109,
     103,110,104,103,109,103,103,100,104,103,100,102,107,
     113,108,107,113
1770 DATA107,96,97,100,99,96,96,110,97,85,78,71,68,110,
     97,84,78,71,68,81,77,72,68,71,68
1780 DATA 8178, 72, 68, 71, 68
1790 DATA113,118,123,122,119,114,114,118,121,122,118,113,
     112,118,116,114,114,116,118,113
1800 DATA47,51,54,51,53,51,48
1810 DATA47,47,48,47,48,47,48
1820 DATA47,47,48,47,48,48,47
1830 DATA46,43,40,39,40,43,47
1840 DATA109,90,69,110,91,71,100,106,101,90,76,63,66,101,
     106,99,90,77,65,66
1850 DATA69,51,36,22,10,7,11,22,35,51,72,106,121,136,146,
     155,158,157,155,154,147,136,124,107
1860 DATA46,77,108,149,180,204,206,205,186,156,116,72,50,
     75,68,60,74,67,59,72,66,60,72,67,61,64,57,64,57,18
1870 DATA12,32,34,31
1880 FORI=1TO186
1890 PLOTX(I),Y(I)
```

```
1540 TCOL15,4:PRINT@R,(C-1);MID$(SCRN$(C-1),R+1,7):
     TCOL1,4
1550 REM
1560 PRINT CHR$(23)
1570 PRINT@R ,C;MID$(SCRN$(C),R+1,7)
1580 PRINT CHR$(23)
1590 IFR=23THENH=(C*2)+1
1600 IFR=31THENH=(C*2)+2
1610 RETURN
1620 REM
1630 REM
1640 REM
1650 REM
1660 REM
1670 DATA135,190,134,128,133,140,135,190,184,189,196,190,
     119,133,147,177,190,203,121,133,147,177,191,201
1680 DATA118,132,148,176,191,204,127,135,144,178,187,196,
     156,156,156,154,156,161,166,166,166,168,167,161
1690 DATA150,147,146,148,153,161,173,176,177,174,170,163,
     112,113,125,139,150,152,171,173,186,199,208,211
1700 DATA112,124,138,152,171,187,200,210,137,149,156,162,
     168,177,187,138,148,156,163,170,178,186,138,149,156,
     163,170,177,186
1710 DATA141,148,155,163,171,179,185,103,101,104,219,222,
     218,99,92,88,90,94,99,104,224,231,234,232,230,224,
     219,104,108,115
1720 DATA129,147,162,180,196,207,215,219,101,107,114,120,
     131,146,160,174,188,201,210,217,222
1730 DATA93,78,76,82,99,129,158,188,217,236,245,250,233,
     145,139,135,178,185,190,105,109,112,218,214,211,159,
     159,165, 165,162
1740 DATA162,153,162,173
1750 REM************************
1760 DATA105,105,109,106,101,106,109,109,106,101,106,109,
     103,110,104,103,109,103,103,100,104,103,100,102,107,
     113,108,107,113
1770 DATA107,96,97,100,99,96,96,110,97,85,78,71,68,110,
     97,84,78,71,68,81,77,72,68,71,68
1780 DATA 8178, 72, 68, 71, 68
1790 DATA113,118,123,122,119,114,114,118,121,122,118,113,
     112,118,116,114,114,116,118,113
1800 DATA47,51,54,51,53,51,48
1810 DATA47,47,48,47,48,47,48
1820 DATA47,47,48,47,48,48,47
1830 DATA46,43,40,39,40,43,47
1840 DATA109,90,69,110,91,71,100,106,101,90,76,63,66,101,
     106,99,90,77,65,66
1850 DATA69,51,36,22,10,7,11,22,35,51,72,106,121,136,146,
     155,158,157,155,154,147,136,124,107
1860 DATA46,77,108,149,180,204,206,205,186,156,116,72,50,
     75,68,60,74,67,59,72,66,60,72,67,61,64,57,64,57,18
1870 DATA12,32,34,31
1880 FORI=1TO186
1890 PLOTX(I),Y(I)
```

```
1900 NEXTI
1910 A=2
1920 FOR I=1TOH-2
1930 A=A+(F(I))
1940 NEXTI
1950 I=A-(F(H-2)-1)
1960 D=1
1970 IFD=F(H-2)THEN GOTO2020
1980 DRAW N(I),M(I) TO N(I+1),M(I+1),1
1990 N(I)=X(I):M(I)=Y(I)
2000 D=D+1:I=I+1
2010 GOTO 1970
2020 REM
2030 REM
2040 I=A-(F(H-2)-1)
2050 D=1
2060 IFD=F(H-2)THEN GOTO2100
2070 DRAW X(I),Y(I) TO X(I+1),Y(I+1)
2080 D=D+1:I=I+1
2090 GOTO 2060
2100 RETURN
2110 A=2
2120 FOR I=1TOH-2
2130 A=A+(F(I))
2140 NEXTI
2150 IFR=23ANDC=20THENGOTO2420
2160 IFR=31ANDC=19THENGOTO2600
2170 IFR=31ANDC=20THENGOTO2770
2180 A=2
2190 FOR I=1TOH-2
2200 A=A+(F(I))
2210 NEXTI
2220 I=A-(F(H-2)-1)
2230 D=1
2240 IFD=F(H-2)+1 THEN GOTO2410
2250 SPRITE1,X(I)  ,Y(I)  ,1,140
2260 TCOL1,15
2270 PRINT@23,22;"POSITION  ARROW"
2280 PRINT@24,23;"AND PRESS RET ";
2290 TCOL15,4
2300 Q$=INCH$
2310 IFQ$=CHR$(4)THENX(I)=X(I)+1
2320 IFQ$=CHR$(8)THENX(I)=X(I)-1
2330 IFQ$=CHR$(11)THENY(I)=Y(I)+1
2340 IFQ$=CHR$(10)THENY(I)=Y(I)-1
2350 SPRITE1,X(I)  ,Y(I)  ,1,140
2360 IFQ$=CHR$(13)THENGOTO2380
2370 GOTO2300
2380 PLOTX(I),Y(I)
2390 D=D+1:I=I+1
2400 GOTO 2240
2410 GOTO1910
2420 TCOL1,4:     PRINT@23,21;SPC(15)
2430 PRINT@23,22;SPC(15)
```

```
2440 PRINT@23,23;SPC(15);
2450 PRINT@23,21;"Hgt ratio";W
2460 A=W
2470 PRINT@23,22;"New ratio";
2480 INPUTW
2490 TCOL15,4:  FORI=1TO186
2500 Y(I)=(Y(I)/A)*W
2510 M(I)=Y(I)
2520 NEXTI
2530 FORI=0TO23
2540 PRINT@0,I;SPC(40)
2550 NEXTI:GOSUB760
2560 S=5:GOSUB600
2590 RETURN
2600 PRINT@23,21;"                "
2611 PRINT@23,22;"                "
2612 PRINT@23,23;"               ";
2615 PRINT@23,21;"Wid ratio";Q
2620 A=Q
2630 PRINT@23,22;"New ratio";
2640 INPUTQ
2650 FORI=1TO186
2660 X(I)=(X(I)/A)*Q
2670 N(I)=X(I)
2680 NEXTI
2690 FORI=0TO23
2700 TCOL15,4:PRINT@0,I;SPC(40)
2710 NEXTI:GOSUB760
2720 SPRITEOFF
2730 S=5:GOSUB600
2740 PRINT@23,21;"                "
2750 PRINT@23,22;"                "
2760 RETURN
2770 PRINT@23,21;"                "
2780 PRINT@23,22;"                "
2790 PRINT@23,21;"FNAME";
2800 INPUTF$
2810 PRINT@23,21;"                "
2820 FORI=1TO186
2830 POKE&AFFF+I,((X(I)/Q)+60)
2840 NEXTI
2850 FORI=1TO186
2860 POKE&AFFF+186+I,(Y(I)/W )
2870 NEXTI
2880 W=W*100:Q=Q*100
2890 POKE&B000+373,W
2900 POKE&B000+374,Q
2910 SAVE F$+".OBJ",&B000,(&B000+374)
2920 RETURN
2930 CLS
2940 PRINT@23,21;"F/NAME";
2950 INPUTF$
2960 LOAD F$+".OBJ"
2970 W= PEEK(&B000+373)
```

```
2980 Q= PEEK(&B000+374)
2990 W=W/100:Q=Q/100
3000 FORI=1TO186
3010 X(I)=PEEK(&AFFF+I)
3020 X(I)=(X(I)-60)*Q
3030 N(I)=X(I)
3040 NEXTI
3050 FORI=1TO186
3060 Y(I)=PEEK(&AFFF+186+I)
3070 Y(I)=Y(I)*W
3080 M(I)=Y(I)
3090 NEXTI
3100 S=0:GOTO600
3110 CLS
3120 PRINT@5,5;"THE FILE ";F$;" DOES NOT EXIST"
3130 PRINT@5,7;"  FILES ON THIS DISK ARE :"
3140 PRINT:DIR"*.OBJ"
3150 PRINT@3,16;"USE ONLY LETTERS TO THE LEFT OF THE ."
3160 PRINT@10,20;"Hit a key to continue"
3170 A$=INCH$
3180 IFA$<>""THEN RST: RUN
3190 GOTO3170
```

A game of logic combined with speed of thought, the object being to arrange a route around the screen to reach 8 numbered locations without backtracking or crossing over your path. Instructions for the game are in the program. The 'world' was originally written by Martin Page and uses a large amount of data statements, if you find the thought of entering all these data statements daunting, make Line 7020 GOTO 7290 and change 7290 to read PRINT@0,8;" HURRAH YOU'VE SAVED THE WORLD", but you will miss out on a rather spectacular drawing.

```
  100 GOSUB 10000:REM***INSTRUCTIONS***
  110 GOSUB 1000:REM***INITIALIZATION***
  120 GOSUB 2000:REM***DISPLAY***
  130 GOSUB 3000:REM***GAME***
 1000 REM***INITIALIZATION***
 1010 SHAPE 130,"FEFE383F3F38FEFE"
 1020 SHAPE 131,"7F7F1CFCFC1C7F7F"
 1030 SHAPE 132,"C3C3FFFFFFDDDD18"
 1040 SHAPE 133,"1818DDFFFFFFC3C3"
 1050 LET T=0000:REM *** TIME ***
 1060 LET XT=0:LET YT=1:REM *** TANK ***
 1065 LET OXT=0:OYT=1
 1070 LET D=0:LET G=0:LET F=0
 1080 LET RODS=01
 1090 SHAPE 142,"1028444428100001"
 1100 SHAPE 143,"1030101010380002"
 1110 SHAPE 144,"38440438407C0003"
 1120 SHAPE 145,"3844180444380004"
 1130 SHAPE 146,"1828487C08080005"
 1140 SHAPE 147,"7C40780444380006"
 1150 SHAPE 148,"3840784444380007"
 1160 SHAPE 149,"7C04081020200008"
 1170 SHAPE 150,"3844384444380009"
 1180 REM IOM3,0
 1900 RETURN
 2000 REM***DISPLAY***
 2010 TCOL15,4:BCOL4
 2020 CLS32
 2030 PRINT @ 25,3,"TIME"
 2040 PRINT @ 25,19,"CODE"
 2050 DRAW 0,15 TO 0,191 TO 191,191 TO 191,15 TO 0,15
 2060 DRAW 7,23 TO 7,184 TO 184,184 TO 184,23 TO 7,23
 2070 FILL 20,20,9
 2080 IOM 5,0
 2090 IOM 4,0
 2100 TCOL15,6
 2110 FOR A=0 TO 8
 2120 LET X=RND (20)+2:LET Y=RND (18)+2
 2130 IF VPEEK(X*8+8192+Y*256)= 246 THEN GOTO 2120
 2140 PRINT @ X,Y,CHR$(142+A)
 2150 NEXT A
 2160 PRINT @ 0,1,CHR$(131)
```

```
2170 TCOL15,1
2180 PRINT @ 25,20,"     "
2190 RETURN
3000 REM *** GAME ***
3010 TCOL15,1
3020 PRINT @ 25,4,T
3030 M=KBD
3040 IF M=75 THEN OXT=XT:OYT=YT
3050 IF M=76 THEN OXT=XT:OYT=YT
3060 IF M=83 THEN OXT=XT:OYT=YT
3070 IF M=87 THEN OXT=XT:OYT=YT
3080 IF M=75 AND D=2 THEN G=10
3090 IF M=75 AND D=3 THEN G=26
3100 IF M=76 AND D=2 THEN G=11
3110 IF M=76 AND D=3 THEN G=27
3120 IF M=83 AND D=1 THEN G=26
3130 IF M=87 AND D=0 THEN G=11
3140 IF M=87 AND D=1 THEN G=10
3150 IF M=83 AND D=0 THEN G=27
3160 IF M=75 AND D=0 THEN G=0
3170 IF M=76 AND D=1 THEN G=0
3180 IF M=83 AND D=2 THEN G=4
3190 IF M=87 AND D=3 THEN G=4
3200 IF M=75 THEN LET XT=XT-1:F=0:D=0:ELSE IF M=76 THEN
     LET XT=XT+1:F=1:D=1
3210 IF M=87 THEN LET YT=YT-1:F=2:D=3:ELSE IF M=83 THEN
     LET YT=YT+1:F=3:D=2
3220 IF M = 76 OR M =75 OR M =83 OR M =87 THEN 3230ELSE
     3290
3230 IF VPEEK(8195+XT*8+YT*256)=241 THEN 5000
3240 IF VPEEK(8195+XT*8+YT*256)=144 THEN 5000
3250 IF VPEEK(7+XT*8+YT*256)<> 0   THEN GOSUB 4000
3260 IF XT<0 THEN GOTO 5000
3270 PRINT @ XT,YT,CHR$(130+F)
3280 PRINT @ OXT,OYT,CHR$(164+G)
3290 LET T=T+1:IF T=1000 THEN GOTO 5000
3300 GOTO 3000
4000 REM *** CODE ***
4005 BEEP
4010 IF VPEEK(7+XT*8+YT*256)<>RODS THEN 5000
4020 LET CODE=RND(8999)+1000
4030 PRINT @ 25,20,CODE
4040 FOR A=0 TO 100:NEXT A
4050 PRINT @ 25,20,"     "
4060 PRINT
4070 INPUT A
4080 IF A=CODE THEN PRINT @ 25,20,"     ":ELSE GOTO 5000
4090 PRINT@ 0,22,"      "
4100 IF RODS =9  THEN GOTO 6000
4110 LET RODS=RODS+1:RETURN
5000 FORA=1TO 20
5010 BCOL1:TCOL15,9
5020 PRINT@ 10,10,"MELT DOWN"
5030 BCOL9
```

```
5035 CLS
5040 PSG6,31:PSG7,71
5050 PSG8,16:PSG9,16:PSG10,16
5060 PSG12,100
5070 PSG13,0
5080 NEXT A
5090 GOTO 7310
6000 REM *** WINNER ***
6010 FOR Z=0 TO 25
6020 CLS40:TCOL15,RND(15):PRINT @ 5,10; "YOU HAVE SAVED
     THE WORLD":PRINT @ 5,12;"IN "T" HALF LIFE PERIODS"
6030 NEXT Z
6040 RESTORE
7000 REM WORLD
7010 CLS40:   TCOL15,7: BCOL7:GCOL12:CLS40
7020 FOR X=0 TO 1 STEP 0
7030 READ N:IF N=0 THEN X=1:GOTO7060
7040 READA,B:REM USSR,AFRICA,INDIA
7050 FORJ=1TO N:READC,D:DRAW A,B TO C,D:A=C:B=D:NEXTJ
7060 NEXT X
7070 REM EVEN SMALLER ISLANDS
7080 PLOT248,76:PLOT251,75:PLOT250,74
7090 PLOT213,94:PLOT214,95:PLOT215,97
7100 PLOT58,154
7110 PLOT133,120:PLOT134,120
7120 PLOT113,124:PLOT114,124:PLOT113,125:PLOT113,127
     :PLOT127,121
7130 PLOT117,124
7140 PLOT203,105:PLOT204,105:PLOT204,106
7150 FILL125,48,12:REM AFRICA/EUROPE
7160 FILL117,144,12:REM SCANDINAVIA
7170 FILL63,25,12:REM AMERICA
7180 FILL80,148,12:REM GREENLAND
7190 FILL227,41,12:REM AUSTRALIA
7200 FILL206,77,12:REM INDONESIA
7210 FILL232,82,12:REM INDONESIA
7220 FILL211,81,12:REM INDONESIA
7230 FILL146,58,12:REM MALAGASY
7240 FILL69,151,12:FILL67,160,12:FILL50,163,12:
     REM CANADIAN ISLAND
7250 FILL66,166,12
7260 GCOL1:ELLIPSE128,96,170,0.56:GCOL12
7270 FILL239,0,1
7280 FILL239,191,1
7290 FILL0,191,1
7300 GOSUB 11000
7310 CLS40:PRINT@5,10;"STANDBY FOR YOUR NEXT MISSION";
7320 A=INCH:RST:RUN
7330 DATA323
7340 DATA105,135,113,140,113,142,115,140,119,140,121,143,
     123,143,123,145,124,145,125,146
7350 DATA124,147,122,148,125,152,125,153,121,153,121,151,
     119,149,119,145,117,143,116,143
```

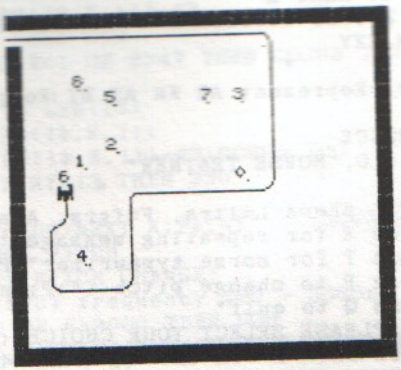
```
7360 DATA116,144,114,146,111,145,111,149,113,149,120,156,
     122,157,124,156,128,156,126,154
7370 DATA128,152,129,154,132,155,133,155,134,154,136,154,
     137,155,139,154,145,157,145,160
7380 DATA148,157,151,162,153,162,153,158,155,154,156,155,
     155,162,160,159,159,162,157,163
7390 DATA159,163,158,164,164,165,165,167,172,166,172,165,
     169,161,173,164,177,163,182,164
7400 DATA184,164,186,165,189,163,190,165,194,165,195,164,
     205,164,206,163,207,164,218,164
7410 DATA219,163,224,163,226,161,219,161,218,160,219,158,
     221,158,223,157,223,156,220,153
7420 DATA223,151,223,147,224,145,218,150,218,153,215,156,
     214,156,212,154,214,153,212,152
7430 DATA211,152,210,153,207,151,206,147,207,146,211,146,
     212,145,213,140,214,140,214,137
7440 DATA213,136,214,135,213,134,212,134,211,133,211,130,
     214,125,212,125,208,130,207,129
7450 DATA206,130,205,131,203,128,204,127,207,127,207,125,
     211,119,210,115,210,113,207,111
7460 DATA206,110,204,108,203,109,201,109,200,108,200,105,
     204,100,204,97,202,95,201,97
7470 DATA197,99,197,94,201,88,200,88,195,94,201,88,200,88,
     196,94,195,94,195,99,193,102,191,101,188,109,183,109
7480 DATA183,106,178,102,178,92,176,94,174,99,174,100,170,
     107,167,107,169,109,168,110
7490 DATA167,109,168,110,168,111,169,112,162,112,160,111,
     157,112,156,113,147,116,146,116
7500 DATA151,112,151,110,152,109,155,110,157,110,157,105,
     150,99,146,98,142,105,135,114
7510 DATA134,115,133,114,138,106,144,97,148,95,152,97,149,
     95,152,96,150,95,152,95,143,82,140,75,142,73
7520 DATA143,70,141,66,137,64,137,59,131,50,125,47,121,48,
     120,52,118,54,119,56
7530 DATA117,60,115,63,115,68,117,70,117,72,115,74,115,78,
     112,81,112,85,113,87
7540 DATA113,88,112,89,109,89,107,91,104,89,103,89,102,90,
     100,89,95,89,94,90
7550 DATA94,91,91,94,91,95,88,98,89,104,89,108,94,113,95,
     113,97,115,97,117
7560 DATA100,120,104,120,109,122,113,122,115,118,117,118,
     119,116,120,116,122,115,124,119
7570 DATA128,116,130,117,131,116,132,117,136,117,136,122,
     135,123,134,122,132,122,131,123
7580 DATA130,123,128,125,130,126,132,127,133,128,134,127,
     140,127,140,129,139,130,138,130
7590 DATA137,131,138,133,135,133,133,131,135,131,133,133,
     130,132,129,131,129,127,127,128
7600 DATA126,127,126,124,125,123,118,131,115,130,115,129,
     118,126,119,126,119,124,118,123
7610 DATA119,124,113,130,110,130,109,129,108,129,107,128,
     107,127,105,125,105,124,104,123
7620 DATA102,123,100,122,98,123,98,127,99,128,99,129,101,
     129,102,128,105,131,105,132
```

```
7630 DATA103,134,105,135,107,136,106,137,105,138,106,139,
     104,141,104,143,103,144,102,144
7640 DATA103,143,103,141,104,140,104,139,102,138,101,139,
     101,140,100,140,99,139,99,138
7650 DATA101,139,102,138,103,137,101,136,106,137,103,138,
     105,139
7660 DATA189:REM  AMERICAS
7670 DATA69,23,67,24,66,23,66,24,62,24,65,25,63,27,63,30,
     64,32,61,32
7680 DATA62,33,61,34,62,35,62,36,61,37,62,38,63,39,63,40,
     66,41,66,42
7690 DATA65,44,66,45,67,44,69,48,70,48,71,52,71,55,74,58,
     78,59,78,70
7700 DATA81,73,81,76,77,80,76,79,74,81,73,80,70,82,68,80,
     67,81,67,82
7710 DATA68,83,66,82,65,83,67,85,67,88,65,88,63,90,61,89,
     59,92,58,92
7720 DATA57,95,54,94,51,95,45,96,44,94,40,94,36,98,37,99,
     37,100,34,100
7730 DATA33,101,33,105,35,107,34,107,32,104,29,104,28,105,
     28,107,27,108,27,111
7740 DATA30,114,30,115,31,116,33,116,35,118,37,118,41,114,
     41,111,42,110,42,111
7750 DATA43,112,43,119,46,119,46,120,49,123,49,124,54,129,
     56,129,56,130,58,132
7760 DATA63,134,61,131,65,132,66,133,61,134,63,135,63,136,
     57,136,61,138,69,138
7770 DATA71,139,71,141,67,145,68,147,65,146,64,147,65,149,
     62,151,60,149,60,147
7780 DATA55,142,55,141,53,139,52,140,53,143,50,144,48,146,
     48,150,55,155,57,157
7790 DATA62,157,62,158,61,157,60,158,58,158,58,161,57,161,
     56,158,55,157,54,158
7800 DATA52,158,51,159,49,158,47,159,46,158,44,160,41,160,
     40,161,35,161,33,160
7810 DATA30,162,29,162,27,164,24,163,23,162,17,162,16,160,
     12,160,14,158,7,156
7820 DATA7,153,4,151,8,151,9,152,11,152,12,153,14,153,15,
     154,21,150,21,147
7830 DATA19,145,20,145,20,142,19,141,19,137,15,130,14,129,
     14,120,15,121,15,116
7840 DATA17,111,16,119,17,120,19,118,20,111,22,104,26,101,
     29,101,33,98,37,93
7850 DATA43,91,43,89,39,79,43,68,50,64,51,63,57,28,62,24,
     65,24,6 6,23
7860 DATA38:REM GREENLAND
7870 DATA80,146,85,151,90,153,93,156,94,155,97,155,99,157,
     100,157,99,158,99,159
7880 DATA102,162,105,167,106,167,108,169,105,169,104,170,
     102,170,101,171,97,171,95,169
7890 DATA92,169,91,168,89,170,87,168,86,169,85,169,84,168,
     82,170,79,170,80,170
7900 DATA77,166,75,166,77,165,79,165,80,161,78,165,79,162,
     80,159,79,146
```

```
7910 DATA42:REM AUSTRALIA
7920 DATA208,45,211,45,213,47,219,47,221,45,221,44,222,45,
     224,45 ,222,42,223,40
7930 DATA226,39,229,39,233,43,234,43,237,46,237,47,239,49,
     239,50,240,51,240,57
7940 DATA239,58,239,59,238,60,238,61,237,62,237,69,236,70,
     232,62,229,66,230,70
7950 DATA226,70,223,67,222,68,214,60,210,60,207,56,207,55,
     206,54,206,47,205,46
7960 DATA 205,45,206,44,207,44
7970 DATA16:REM NEW ZEALAND
7980 DATA232,27,247,36,247,37,246,38,246,40,245,35,246,37,
     243,36,244,34,242,33
7990 DATA240,33,240,34,231,28,232,28,240,33,238,29,236,30
8000 DATA17:REM INDONESIA
8010 DATA193,90,196,87,196,85,198,82,205,75,214,74,218,75,
     220,74,218,75,213,74,210,75,205,80,205,81
8020 DATA204,80,203,81,200,85,197,87,194,90
8030 DATA 23
8040 DATA236,73,238,75,241,72,243,72,240,75,241,72,239,75,
     240,76,241,77,240,77,236,81,231,83
8050 DATA231,80,229,80,229,83,228,79,228,82,227,81,231,79,
     233,77,232,75,233,75
8060 DATA234,74,235,74
8070 DATA21
8080 DATA206,85,207,82,207,81,210,80,212,80,213,84,213,85,
     219,86,217,85,215,84,214,80
8090 DATA216,83,217,79,216,82,218,83,215,78,215,85,213,85,
     213,91,214,90,212,89,207,86
8100 DATA17:REM DEAD SEA
8110 DATA143,133,144,130,145,129,147,127,146,125,147,125,
     148,124,151,123,151,125,150,128
8120 DATA151,129,150,130,149,130,148,131,150,131,150,133,
     149,134,144,133
8130 DATA9:REM MALAGASY
8140 DATA147,57,149,69,146,66,145,66,145,64,146,63,146,61,
     145,60,145,58,147,57
8150 DATA23:REM JAPAN
8160 DATA218,125,219,125,219,124,220,125,221,128,222,128,
     222,130,221,131,221,133,220,134
8170 DATA221,135,221,136,220,136,218,138,218,137,220,135,
     219,135,220,134,221,133,221,129
8180 DATA219,127,219,126,220,126,220,127
8190 DATA26:REM CANADIAN ISLANDS
8200 DATA70,149,70,150,71,151,71,153,74,151,72,155,72,156,
     70,158,70,159,67,162
8210 DATA66,161,64,161,63,162,64,164,63,160,65,159,68,158,
     69,157,68,156,69,155
8220 DATA69,154,69,153,66,153,66,154,66,152,68,150,69,150
8230 DATA16
8240 DATA66,165,68,165,69,166,70,166,71,167,72,168,72,168,
     73,169,74,170
8250 DATA74,171,72,170,71,169,70,170,66,169,67,167,65,167,
     65,166
```

```
 8260 DATA13
 8270 DATA48,162,49,163,50,162,51,161,52,162,53,163,54,163,
      55,164,53,166,52,165
 8280 DATA51,164,50,164,49,165,47,163
 8290 DATA2,44,163,45,163,46,165
 8300 DATA1,53,167,55,168
 8310 DATA1,56,166,59,170
 8320 DATA1,58,166,59,167
 8330 DATA1,61,166,62,167
 8340 DATA2,62,170,64,170,65,171
 8350 DATA1,70,163,71,163
 8360 DATA10,95,148,96,147,99,149,98,150,97,149,96,149,96,
      150,95,149,96,148,97,148,98,149
 8370 DATA5,224,33,226,36,226,35,224,37,224,34,223,33
 8380 DATA3,243,77,244,77,246,79,244,80
 8390 DATA4,252,71,253,70,253,73,252,73,252,72
 8400 DATA13,215,106,215,101,216,100,217,100,216,99,217,99,
      218,98,217,97,220,98,218,96,220,96,218,94,218,92,219,
      94
 8410 DATA3,219,94,221,92,221,93,220,94
 8420 DATA2,180,91,180,94,181,92
 8430 DATA6,38,107,39,108,43,107,44,106,43,105,43,106,
      40,107
 8440 DATA6,38,107,39,108,43,107,44,106,43,105,43,106,
      40,107
 8450 DATA0
 8460 DATA0
10000 REM***INSTRUCTIONS***
10010 GOSUB  12000:SPEED235
10020 PRINT@11,10;"HELLO SUPER-HERO"
10030 PRINT@2,12;"DO YOU REQUIRE INSTRUCTION?";:A=INCH
10050 IF A=89 OR A=121 THEN 10100:ELSE SPEED255:RETURN
10100 GOSUB 12000
10108 PRINT
10110 PRINT"TIME IS RUNNING OUT FOR MANKIND. THERE  IS A
      STATE OF EMERGENCY AT SELASCALE THELARGEST NUCLEAR
      POWER STATION IN THE "   10120 PRINT"WORLD. THE CORE
      HAS GONE 'CRITICAL' AND SHOULD MELTDOWN OCCUR THERE
      WILL RESULT,A CHAIN REACTION THAT WILL RIP APART  "
10130 PRINT"THE EARTHS CRUST LEAVING NO MAN ALIVE TORECORD
      HIS FOLLIES."
10135 PRINT
10140 PRINT"YOUR MISSION SHOULD YOU WISH TO ACCEPT  IT, IS
      TO REMOVE THE RODS FROM THE CORE OF THE REACTOR
      BEFORE       IT'S TOO LATE
10141 FOR F=1TO1200:NEXTF:GOSUB12000
10142 PRINT
10145 PRINT"YOU CANNOT ENTER THE AREA, SO YOU MUST  GUIDE A
      'REMOTE' TO EACH ROD. BEFORE A  ROD CAN BE WITHDRAWN
      ITS CODE WHICH    10146 PRINT"WILL BE FLASHED ON THE
      LOWER RIGHT OF    YOUR MONITOR MUST BE ENTERED
      CORRECTLY."
10147 PRINT:PRINT"EACH ROD MUST BE WITHDRAWN IN THE CORRECT
      SEQUENCE."
```

```
10148 PRINT"DUE TO THE UNSTABLE NATURE OF THE AREA  SHOULD
      THE REMOTE CROSS ITS TRAILING    CABLES OR TOUCH THE
      WALLS, MELTDOWN"
10149 PRINT"WILL BE THE INEVITABLE RESULT."
10160 PRINT:PRINT"TIME IS CRITICAL YOU HAVE 1000 HALF-
      LIFEPERIODS --GOOD LUCK!"
10161 FOR F=1TO1200:NEXTF:GOSUB12000:PRINT@11,4;"YOUR
      CONTROLS ARE"
10162 PRINT@19,6;"U";@19,8;"W";@14,10;"L-K      L-R"
      ;@19,12;"S";@19,14;"D"
10165 FOR F=1TO1200:NEXTF:PRINT@4,17;"DON'T DELAY PRESS A
      KEY NOW..";
10170 A=INCH:SPEED255
10180 RETURN
11000 REM PRETTY TUNE
11010 VOICE 0,31,10,3,90,50
11020 A$="C5C5-A5C8-A5-F8C5D7C6C5C5-A5-bB7-bB7-G5-E8-bB5C7-
      bB8-A5-A5-bB5C7-A5-F8C5D7C6C5C5-A5C7C7-bB7-G7-F9"
11030 TEMPO 5
11040 MUSIC "V0"+A$,"R","R"
11050 PSG7,&7F
11060 RETURN
12000 CLS40:BCOL2:TCOL4,15:PRINT@13,1;"ATOMIC RESCUE":
      TCOL15,2
12020 RETURN
```



## MORSE TRAINER

For those just learning Morse code or for those who would like to brush up on their knowledge 'MORSE TRAINER' is well worth typing in.

```
10 REM MORSE TRAINER
20 REM BY DAVE WEST
30 REM  INITIALIZE
40 DIM M$(47):FOR X=1 TO 47:READ M$(X):NEXT X
50 E=10:PO=125:P1=0:F=1000
60 PSGO,PO:PSG 1,P1:PSG 1,0:PSG 8,15:PSG 7,127
70 ON ERR GOTO 330
80 GOTO 1040
90 REM SUB ROUTINES
100 CH$=INCH$:IF CH$="" THEN 100:REM READ KEY BOARD
110 RETURN
120 PSG 7,120:FOR PS=1TO200:NEXT:PSG7,127:RETURN
130 REM PRINT ROUTINE
140 FOR K=1 TO LEN(I$)
150 P$=MID$(I$,K,1):PRINT P$;
160 R=ASC(P$)-43
170 GOSUB 200
180 NEXT K
190 RETURN
200 REM SOUND ROUTINE
210 IF R=-11 THEN FOR Z=1 TO 4000000/E^4:NEXT Z:RETURN
220 FOR M= 1 TO 6
230 SD=VAL(MID$(M$(R),M,1))
240 IF SD=0 THEN 280
250 PSG 7,120:FOR X=1 TO SD 4.5:NEXTX
260 PSG 7,127
270 FOR Z=1 TO 20:NEXT Z
280 NEXT M:FOR Z=1 TO 1000000/E^4:NEXT Z:RETURN
290 REM PROSIGN KEY
300 PRINT"                "
310 PRINT"<,>,@,=Represent AR KN AS BT Respectively":PRINT
320 RETURN
330 REM  GET CHOICE
340 CLS:PRINT@11,0,"MORSE TRAINER"
350 PRINT@8,1;"                    "
360 PRINT:PRINT"  press L=ltrs, F=fgrs, A=all":PRINT
370 PRINT"  press R for repeating message":PRINT
380 PRINT"  press T for morse typewriter":PRINT
390 PRINT"  press P to change pitch (";F;"hz)":PRINT
400 PRINT"  press Q to quit"
410 PRINT@6,18;"PLEASE SELECT YOUR CHOICE";:GOSUB 100
420 IF CH$="L" OR CH$="F" OR CH$="A" THEN 480
430 IF CH$="T" THEN 660
440 IF CH$="R" THEN 760
450 IF CH$="P" THEN 910
460 IF CH$="Q" THEN 980
470 GOSUB 120:GOTO 330
480 REM  RANDOM FIVE CHR GROUPS
490 CLS:PRINT"     RANDOM GROUPS"
500 PRINT"<RET> to quit or any other to hold"
```

```
510 IF CH$="A" THEN GOSUB 290
520 T$=",^./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ"
530 GP=0
540 I$="":FOR X=1 TO 5
550 GP=GP+1:IF GP>=31 THEN PRINT"     ";:GP=0
560 IF CH$="L" THEN R$=MID$(T$,RND(26)+22,1)
570 IF CH$="F" THEN R$=MID$(T$,RND(10)+5,1)
580 IF CH$="A" THEN R$=MID$(T$,RND(47)+1,1)
590 IF R$=" " THEN 560
600 I$=I$+R$
610 KY=KBD:IF KY=13 THEN GOTO 330
620 IF KY<>0 THEN GOSUB 100
630 NEXT X :I$=I$+" "
640 GOSUB 130
650 GOTO 540
660 REM TYPE MORSE
670 CLS:PRINT"TYPE FOR IMMEDIATE MORSE OUTPUT":PRINT"
    PRESS <RET> TO QUIT "
680 GOSUB 290
690 GOSUB 100
700 LET R=ASC(CH$)-43
710 IF R=-30 THEN 330
720 IF R=-11 THEN PRINT" ";:GOTO 690
730 IF R=2 OR R<1 OR R>47 THEN GOSUB 120
740 PRINTCH$;:CH$=""
750 GOSUB 200:GOTO 690
760 REM  REPEATING MESSAGE
770 CLS:PRINT"      REPEATING MESSAGE"
780 GOSUB 290:PRINT"PRESS <RET> TO QUIT"
790 INPUT "ENTER MESSAGE ";I$
800 FOR K=1 TO LEN(I$)
810 R=ASC(MID$(I$,K,1))-43
820 IF R=-11 THEN 840
830 IF R=2 OR R<1 OR R>47 THEN GOSUB 120
840 NEXT K
850 FOR K=1 TO LEN(I$)
860 PRINT MID$(I$,K,1);
870 R=ASC(MID$(I$,K,1))-43:GOSUB 200
880 KY=KBD:IF KY=13 THEN 330
890 NEXT K
900 FOR K=1TO200:NEXT K:PRINT" ";:GOTO850
910 REM CHANGE PITCH
920 CLS:PRINT"PITCH CHANGE"
930 INPUT "INPUT frequency 500 - 3000hz ";F
940 IF F<500 OR F>3000 THEN 930
950 TP=2E6/(16*F):P1=INT(TP/256):P0=TP-P1
960 PSG 0,P0:PSG 1,P1
970 GOTO 330
980 REM QUIT
990 CLS:END
1000 REM  MORSE DATA
1010 DATA 131313,,111111,31131,33333,13333,11333,11133,
     11113,11111,31111,33111,33311,33331,331111,313131,
```

```
1020 DATA 113311,13111,13,3111,3131,311,1,1131,331, 1111,
     11,1333,313,1311,33,31,333,1331,3313,131,111,3,113,
     1113,133, 3113
1030 DATA 3133,3311
1040 CLS:PRINT" MORSE TRAINER DEMO";@9,21;"PRESS ENTER FOR
     MENU":PRINT@0,2;:
1050 I$="A PROGRAM WRITTEN BY DAVE WEST 1 JUL 86."
1060 GOTO 850
```

# RUBIKS CUBE

This version of Rubiks cube on the Einstein was written for the U.K.E.U.G. by Peter Heffernan.

```
 10 MAGO:SPRITEOFF:CLS:BCOLO
 20 PRINTCHR$(20):REM Turn cursor off
 30 PRINT@5,3;"*******************************"
 40 PRINT@5,4;"*     INSTRUCTIONS            *"
 50 PRINT@5,5;"*     FOR THE CUBE            *"
 60 PRINT@5,6;"*******************************"
 70 PRINT@3,8;"ONLY THREE FACES OF THE CUBE WILL"
 80 PRINT@3,9;"BE DISPLAYED AT ANY TIME"
 90 PRINT@3,10;"THE LETTERS AROUND THE CUBE INDICATE THE
    ROW TO BE ADJUSTED.AFTER THE PROMT (WHAT MOVE) YOU
    INPUT THE"
100 PRINT@3,13;"LETTER OF YOUR CHOICE,AT WHICH POINT
    YOU WILL BE ASKED FOR A DIRECTION       ie UP ↑ DOWN:
    RIGHT) LEFT(."
110 PRINT@3,17;"NOTE THE DIRECTION KEYS ARE NOT
    THE     CURSOR KEYS"
120 PRINT@3,21;"PRESS ANY KEY TO START"
130 K=INCH
140 IFK>OTHEN GOTO150
150 ORIGIN0,0
160 MAGO
170 REM Set up initial colours
180 F1=15:F2=15:F3=15:F4=15:F5=15:F6=15:F7=15:F8=15
    :F9=15
190 L1=7:L2=7:L3=7:L4=7:L5=7:L6=7:L7=7:L8=7:L9=7
200 R1=4:R2=4:R3=4:R4=4:R5=4:R6=4:R7=4:R8=4:R9=4
210 S1=12:S2=12:S3=12:S4=12:S5=12:S6=12:S7=12:S8=12:S9=12
220 T1=10:T2=10:T3=10:T4=10:T5=10:T6=10:T7=10:T8=10:T9=10
230 B1=6:B2=6:B3=6:B4=6:B5=6:B6=6:B7=6:B8=6:B9=6
240 REM Set up alpha letters around the cube
250 SPRITE1,84,88,3,65
260 SPRITE2,84,57,5,66
270 SPRITE3,84,28,9,67
280 SPRITE4,100,8,11,68
290 SPRITE5,127,8,6,69
300 SPRITE6,154,8,15,70
310 SPRITE7,177,12,4,71
320 SPRITE8,193,24,12,72
330 SPRITE9,209,36,14,73
340 CLS
350 PRINT@1,9;"PRESS"
360 PRINT@1,11;"(R)TO RESET"
370 PRINT@1,13;"(X)TO EXIT"
380 PRINT@1,15;"(T)TO TURN"
390 PRINT@1,16;"WHOLE CUBE"
400 Q=0
410 REM Draw and colour front face
420 GCOLF1
430 ORIGIN0,0: GOSUB620
440 GCOLF2
450 ORIGIN27,0:GOSUB620
```

# RUBIKS CUBE

This version of Rubiks cube on the Einstein was written for the U.K.E.U.G. by Peter Heffernan.

```
 10 MAGO:SPRITEOFF:CLS:BCOLO
 20 PRINTCHR$(20):REM Turn cursor off
 30 PRINT@5,3;"*******************************"
 40 PRINT@5,4;"*     INSTRUCTIONS            *"
 50 PRINT@5,5;"*     FOR THE CUBE            *"
 60 PRINT@5,6;"*******************************"
 70 PRINT@3,8;"ONLY THREE FACES OF THE CUBE WILL"
 80 PRINT@3,9;"BE DISPLAYED AT ANY TIME"
 90 PRINT@3,10;"THE LETTERS AROUND THE CUBE INDICATE THE
    ROW TO BE ADJUSTED.AFTER THE PROMT (WHAT MOVE) YOU
    INPUT THE"
100 PRINT@3,13;"LETTER OF YOUR CHOICE,AT WHICH POINT
    YOU WILL BE ASKED FOR A DIRECTION       ie UP ↑ DOWN:
    RIGHT) LEFT(."
110 PRINT@3,17;"NOTE THE DIRECTION KEYS ARE NOT
    THE     CURSOR KEYS"
120 PRINT@3,21;"PRESS ANY KEY TO START"
130 K=INCH
140 IFK>OTHEN GOTO150
150 ORIGIN0,0
160 MAGO
170 REM Set up initial colours
180 F1=15:F2=15:F3=15:F4=15:F5=15:F6=15:F7=15:F8=15
    :F9=15
190 L1=7:L2=7:L3=7:L4=7:L5=7:L6=7:L7=7:L8=7:L9=7
200 R1=4:R2=4:R3=4:R4=4:R5=4:R6=4:R7=4:R8=4:R9=4
210 S1=12:S2=12:S3=12:S4=12:S5=12:S6=12:S7=12:S8=12:S9=12
220 T1=10:T2=10:T3=10:T4=10:T5=10:T6=10:T7=10:T8=10:T9=10
230 B1=6:B2=6:B3=6:B4=6:B5=6:B6=6:B7=6:B8=6:B9=6
240 REM Set up alpha letters around the cube
250 SPRITE1,84,88,3,65
260 SPRITE2,84,57,5,66
270 SPRITE3,84,28,9,67
280 SPRITE4,100,8,11,68
290 SPRITE5,127,8,6,69
300 SPRITE6,154,8,15,70
310 SPRITE7,177,12,4,71
320 SPRITE8,193,24,12,72
330 SPRITE9,209,36,14,73
340 CLS
350 PRINT@1,9;"PRESS"
360 PRINT@1,11;"(R)TO RESET"
370 PRINT@1,13;"(X)TO EXIT"
380 PRINT@1,15;"(T)TO TURN"
390 PRINT@1,16;"WHOLE CUBE"
400 Q=0
410 REM Draw and colour front face
420 GCOLF1
430 ORIGIN0,0: GOSUB620
440 GCOLF2
450 ORIGIN27,0:GOSUB620
```

```
460 GCOLF3
470 ORIGIN54,0:GOSUB620
480 GCOLF5
490 ORIGIN27,-31:GOSUB620
500 GCOLF4
510 ORIGIN0,-31:GOSUB620
520 GCOLF6
530 ORIGIN54,-31:GOSUB620
540 GCOLF7
550 ORIGIN0,-62:GOSUB620
560 GCOLF8
570 ORIGIN27,-62:GOSUB620
580 GCOLF9
590 ORIGIN54,-62:GOSUB620
600 ORIGIN0,0
610 GOTO650
620 DRAW93,96TO113,96TO113,72TO93,72TO93,96:FILL100,80
630 RETURN
640 REM Draw and colour top face
650 GCOLT7
660 ORIGIN-6,2:GOSUB840
670 GCOLT8
680 ORIGIN22,2:GOSUB840
690 GCOLT9
700 ORIGIN50,2:GOSUB840
710 GCOLT4
720 ORIGIN8,16:GOSUB840
730 GCOLT5
740 ORIGIN36,16:GOSUB840
750 GCOLT6
760 ORIGIN65,16:GOSUB840
770 GCOLT1
780 ORIGIN23,30:GOSUB840
790 GCOLT2
800 ORIGIN51,30:GOSUB840
810 GCOLT3
820 ORIGIN79,30:GOSUB840
830 ORIGIN0,0:GCOL7:GOTO870
840 DRAW102,102TO112,112TO132,112TO122,102TO102,102
    :FILL110,106
850 RETURN
860 REM Draw and colour side face
870 GCOLS1
880 ORIGIN 5,-2:GOSUB1060
890 GCOLS4
900 ORIGIN5,-33:GOSUB1060
910 GCOLS7
920 ORIGIN5,-65:GOSUB1060
930 GCOL S2
940 ORIGIN20,11:GOSUB1060
950 GCOL S5
960 ORIGIN20,-20:GOSUB1060
970 GCOLS8
980 ORIGIN20,-51:GOSUB1060
```

```
 990 GCOLS3
1000 ORIGIN35,26:GOSUB1060
1010 GCOL S6
1020 ORIGIN35,-6:GOSUB1060
1030 GCOLS9
1040 ORIGIN35,-37:GOSUB1060
1050 ORIGINO,0:GOTO1080
1060 DRAW169,100TO169,76TO178,85TO178,109TO169,100
     :FILL175,90
1070 RETURN
1080 PRINT@1,2;"WHAT MOVE(A-I)"
1090 P=INCH
1100 IFP=66THEN1370:REM B MOVE
1110 IFP=70THEN1880:REM F MOVE
1120 IFP=69THEN1760:REM E MOVE
1130 IFP=72THEN2160:REM H MOVE
1140 IFP=68THEN1630:REM D MOVE
1150 IFP=71THEN2020:REM G MOVE
1160 IFP=65THEN1230:REM A MOVE
1170 IFP=67THEN1490:REM C MOVE
1180 IFP=73THEN2280:REM I MOVE
1190 IFP=84THEN2440:REM T MOVE
1200 IFP=88THEN2420:REM X EXIT PROG
1210 IFP=82THEN CLS:GOTO150:REM RESET R
1220 GOTO1090
1230 REM A MOVE
1240 PRINT@1,3;"LEFT([)OR RIGHT(])"
1250 S=INCH
1260 IFS=91THEN1330
1270 IFS=93THEN1290
1280 GOTO1250
1290 REM RIGHTA
1300 SWAPT7,T1:SWAPT1,T3:SWAPT3,T9:SWAPT4,T2:SWAPT2,T6
     :SWAPT6,T8
1310 SWAPF1,L3:SWAPL3,R3:SWAPR3,S1:SWAPF2,L2:SWAPL2,R2
     :SWAPR2,S2:SWAPF3,L1:SWAPL1,R1:SWAPR1,S3
1320 IFQ=5THEN1460ELSE340
1330 REM  LEFT A
1340 SWAPT7,T9:SWAPT9,T3:SWAPT3,T1:SWAPT8,T6:SWAPT6,T2
     :SWAPT2,T4
1350 SWAPF1,S1:SWAPS1,R3:SWAPR3,L3:SWAPF2,S2:SWAPS2,R2
     :SWAPR2,L2:SWAPF3,S3:SWAPS3,R1:SWAPR1,L1
1360 IFQ=5THEN1430ELSE340
1370 REM B MOVE
1380 PRINT@1,3;"LEFT([)OR RIGHT(])"
1390 S=INCH
1400 IFS=91THEN1430
1410 IFS=93THEN1460
1420 GOTO1390
1430 REM LEFT B
1440 SWAPF4,S4:SWAPS4,R6:SWAPR6,L6:SWAPF5,S5:SWAPS5,R5
     :SWAPR5,L5:SWAPF6,S6:SWAPS6,R4:SWAPR4,L4
1450 IFQ=5THEN1550ELSE340
1460 REM RIGHT B
```

```
1470 SWAPF4,L6:SWAPL6,R6:SWAPR6,S4:SWAPF5,L5:SWAPL5,R5
     :SWAPR5,S5:SWAPF6,L4:SWAPL4,R4:SWAPR4,S6
1480 IFQ=5THEN1590ELSE340
1490 REM C MOVE
1500 PRINT@1,3;"LEFT([)OR RIGHT(])"
1510 S=INCH
1520 IFS=91THEN1550
1530 IFS=93THEN1590
1540 GOTO1390
1550 REM RIGHT C
1560 SWAPB7,B9:SWAPB9,B3:SWAPB3,B1:SWAPB8,B6:SWAPB6,B2
     :SWAPB2,B4
1570 SWAPF7,S7:SWAPS7,R9:SWAPR9,L9:SWAPF8,S8:SWAPS8,R8
     :SWAPR8,L8:SWAPF9,S9:SWAPS9,R7:SWAPR7,L7
1580 GOTO 340
1590 REM LEFT C
1600 SWAPB7,B1:SWAPB1,B3:SWAPB3,B9:SWAPB4,B2:SWAPB2,B6
     :SWAPB6,B8
1610 SWAPF9,L7:SWAPL7,R7:SWAPR7,S9:SWAPF8,L8:SWAPL8,R8
     :SWAPR8,S8:SWAPF7,L9:SWAPL9,R9:SWAPR9,S7
1620 GOTO340
1630 REM D MOVE
1640 PRINT@1,3;"UP(^)OR DOWN(:)"
1650 S=INCH
1660 IFS=94THEN1720
1670 IFS=58THEN1690
1680 GOTO1650
1690 SWAPL7,L1:SWAPL1,L3:SWAPL3,L9:SWAPL4,L2:SWAPL2,L6
     :SWAPL6,L8
1700 SWAPF1,T1:SWAPT1,R7:SWAPR7,B7:SWAPF4,T4:SWAPT4,R4
     :SWAPR4,B4:SWAPF7,T7:SWAPT7,R1:SWAPR1,B1
1710 IFQ=5THEN1820ELSE340
1720 REM D UP
1730 SWAPL7,L9:SWAPL9,L3:SWAPL3,L1:SWAPL8,L6:SWAPL6,L2
     :SWAPL2,L4
1740 SWAPF1,B7:SWAPB7,R7:SWAPR7,T1:SWAPF4,B4:SWAPB4,R4
     :SWAPR4,T4:SWAPF7,B1:SWAPB1,R1:SWAPR1,T7
1750 IFQ=5THEN1850ELSE340
1760 REM E MOVE
1770 PRINT@1,3;"UP(^)OR DOWN(:)"
1780 S=INCH
1790 IFS=94THEN1850
1800 IFS=58THEN1820
1810 GOTO1780
1820 REM E DOWN
1830 SWAPF2,T2:SWAPT2,R8:SWAPR8,B8:SWAPF5,T5:SWAPT5,R5
     :SWAPR5,B5:SWAPF8,T8:SWAPT8,R2:SWAPR2,B2
1840 IFQ=5THEN1980ELSE340
1850 REM E UP
1860 SWAPF2,B8:SWAPB8,R8:SWAPR8,T2:SWAPF5,B5:SWAPB5,R5
     :SWAPR5,T5:SWAPF8,B2:SWAPB2,R2:SWAPR2,T8
1870 IFQ=5THEN1940ELSE340
1880 REM F MOVE
1890 PRINT@1,3;"UP(^)OR DOWN(:)"
```

```
1900 S=INCH
1910 IFS=94THEN1940
1920 IFS=58THEN1980
1930 GOTO1900
1940 REM F UP
1950 SWAPS7,S9:SWAPS9,S3:SWAPS3,S1:SWAPS8,S6:SWAPS6,S2
     :SWAPS2,S4
1960 SWAPF3,B9:SWAPB9,R9:SWAPR9,T3:SWAPF6,B6:SWAPB6,R6
     :SWAPR6,T6:SWAPF9,B3:SWAPB3,R3:SWAPR3,T9
1970 GOTO 340
1980 REM F DOWN
1990 SWAPS7,S1:SWAPS1,S3:SWAPS3,S9:SWAPS4,S2:SWAPS2,S6
     :SWAPS6,S8
2000 SWAPF3,T3:SWAPT3,R9:SWAPR9,B9:SWAPF6,T6:SWAPT6,R6
     :SWAPR6,B6:SWAPF9,T9:SWAPT9,R3:SWAPR3,B3
2010 GOTO 340
2020 REM G MOVE
2030 PRINT@1,3:"UP(^)OR DOWN(:)"
2040 S=INCH
2050 IFS=94THEN2080
2060 IFS=58THEN2120
2070 GOTO2040
2080 REM G UP
2090 SWAPF7,F1:SWAPF1,F3:SWAPF3,F9:SWAPF4,F2:SWAPF2,F6
     :SWAPF6,F8
2100 SWAPS1,B9:SWAPB9,L7:SWAPL7,T7:SWAPS4,B8:SWAPB8,L4
     :SWAPL4,T8:SWAPS7,B7:SWAPB7,L1:SWAPL1,T9
2110 GOTO340
2120 REM G DOWN
2130 SWAPF7,F9:SWAPF9,F3:SWAPF3,F1:SWAPF8,F6:SWAPF6,F2
     :SWAPF2,F4
2140 SWAPS1,T7:SWAPT7,L7:SWAPL7,B9:SWAPS4,T8:SWAPT8,L4
     :SWAPL4,B8:SWAPS7,T9:SWAPT9,L1:SWAPL1,B7
2150 GOTO340
2160 REM H MOVE
2170 PRINT@1,3:"UP(^)OR DOWN(:)"
2180 S=INCH
2190 IFS=94THEN2220
2200 IFS=58THEN2250
2210 GOTO2180
2220 REM H UP
2230 SWAPS2,B6:SWAPB6,L8:SWAPL8,T4:SWAPS5,B5:SWAPB5,L5
     :SWAPL5,T5:SWAPS8,B4:SWAPB4,L2:SWAPL2,T6
2240 GOTO340
2250 REM H DOWN
2260 SWAPS2,T4:SWAPT4,L8:SWAPL8,B6:SWAPS5,T5:SWAPT5,L5
     :SWAPL5,B5:SWAPS8,T6:SWAPT6,L2:SWAPL2,B4
2270 GOTO340
2280 REM I MOVE
2290 PRINT@1,3;"UP(^)OR DOWN(:)"
2300 S=INCH
2310 IFS=94THEN2340
2320 IFS=58THEN2380
2330 GOTO2300
```

```
2340 REM I UP
2350 SWAPR7,R1:SWAPR1,R3:SWAPR3,R9:SWAPR4,R2:SWAPR2,R6
     :SWAPR6,R8
2360 SWAPS3,B3:SWAPB3,L9:SWAPL9,T1:SWAPS6,B2:SWAPB2,L6
     :SWAPL6,T2:SWAPS9,B1:SWAPB1,L3:SWAPL3,T3
2370 GOTO340
2380 REM I DOWN
2390 SWAPR7,R9:SWAPR9,R3:SWAPR3,R1:SWAPR8,R6:SWAPR6,R2
     :SWAPR2,R4
2400 SWAPS3,T1:SWAPT1,L9:SWAPL9,B3:SWAPS6,T2:SWAPT2,L6
     :SWAPL6,B2:SWAPS9,T3:SWAPT3,L3:SWAPL3,B1
2410 GOTO340
2420 REM X (EXIT PROG)
2430 CLS:SPRITEOFF:RST:BCOL4:END
2440 REM TURN CUBE
2450 Q=5
2460 PRINT@1,3;"LEFT([)OR RIGHT(])"
2470 PRINT@1,5;"UP(^)OR DOWN(:)"
2480 S=INCH
2490 IFS=91THEN1330
2500 IFS=93THEN1290
2510 IFS=94THEN1720
2520 IFS=58THEN1690
2530 GOTO2460
```

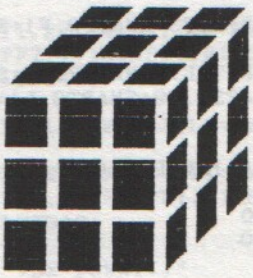
```
WHAT MOVE(A-I)

PRESS

(R)TO RESET

(X)TO EXIT

(T)TO TURN
WHOLE CUBE
```



Discs are known as random access devices, you can compare a disc to a record player, the head of the disc being the needle of the record player. Data is read serially i.e. one bit after another, just like the stylus of the record player following the groove in the record. (Disc tracks are of course concentric rings unlike the spiral grove in a record.) The random access refers to being able to position the read/write head anywhere on the disc, just as you can select any track on the record by moving the stylus arm, any disc track can be selected by moving the disc head. This is what makes discs so much quicker than cassette.

EINSTEIN has a capability of using four drives, the reason only four drives can be used is due to the design of the FDC, (Floppy Disc Controller).Two internal 3" drives can fitted and external drives can be used via the external expansion port. Any external drives require their own power supply, the data cabling being connected to M004 on the rear of the machine. Any Shugartt interface drive, 3", 3.5", or 5.25" will work. (Shugartt interface is a standard that defines the signals used by the drive and the FDC). The drives are run in a daisy chain configuration, which means data is sent to all drives on a common line, the daisy chain, and a select signal gates the data to the selected drive. The same principle applies to a read from disc. This of course means that only one drive can be in use at any moment in time.

The installed 3" drive is single sided, 40 track with the media being reversable and used on both sides. The use of double sided or 80 track drives requires an upgraded DOS such as System 5 from Crystal Research. The recording media used is a magnetic oxide coating on a circular flexible disc, hence floppy disc. When in use the disc is in contact with the read/write head due to slight pressure from a head pad that rests on the opposite side of the disc, (With double sided discs two heads are used). Data is written to the disc in a binary pattern when a current is passed through the head magnetising the disc media. A disc read uses the magnetic changes on the media to create small current pulses in the disc head which are converted back to binary data.

The actual media is enclosed in a plastic case which gives good protection to the disc itself, and offers a rigid mounting to the drive. On the edge of the case that is inserted into the drive is a small notch, this is used by the drive to indicate whether side A or B is in use, via the red or green LED, (Light Emitting Diode), on the front of the drive.

Under the control of the FDC the disc head moves in or out along the disc surface as the disc rotates. The position of the head is controlled by a steper motor. This is a precision motor that moves in steps, each step being a track on the disc surface. Each track on the disc is divided into 10 portions known as sectors.

If you examine a 3" disc, near to the center of the protective case is a small hole. The disc media also has a small hole, this is the index hole, a sensor within the drive uses this hole to output a pulse for each revolution of the disc creating the index pulses. These pulses are used by the FDC to check the drive is ready and as a refrencf point for positional information. A special pattern is written on the disc to define between sectors, the FDC can detect this sector pattern. By combining the information from the index pulses, the track number from the steper motor and the sector pattern the position of the disc head can be calculated. This type of sectoring is called soft sectored as it is the software that is formatting the tracks into 10 sectors.(Hard sectored discs have as many index holes as sectors).

Before we actually go on to use our discs it is good practice to use the following rules;

1..DO NOT SWITCH ON OR OFF WITH A DISC IN THE DRIVE.
2..DO NOT EJECT, OR INSERT A DISC WHEN THE BUSY LIGHT IS ON.
3..DO NOT OPEN THE PROTECTIVE CASING.
4..DO NOT STORE DISCS NEAR TO STRONG MAGNETIC FIELDS, SUCH AS HI-FI SPEAKERS.
5..DO MAKE A COPY OF ANY SYSTEM DISC, TAKE REGULAR SECURITY COPIES OF IMPORTANT DATA DISCS.

All the above recommendations will help to keep your discs in good working order and will avoid corruption as far as is possible. Provided security copies are available not too much work will have to be re-done to recover if a corruption happens. Remember Murphy's Law; "If it is possible, it will happen". So be warned!

As we are using soft sectoring a new disc requires formatting before use, a simple procedure using the BACKUP utility. Formatting the disc writes the sector pattern, writes the character E5 across the storage area of the disc, and copies the system DOS to tracks 0&1. The capacity of the 3" disc before formatting is 250k bytes. (Remember 1k=1024 bytes). After formatting the usable storage is 188K bytes, so where does the missing 62k go? As stated before there are 40 tracks each divided into 10 sectors. Each sector holds 512 bytes giving 40*10*512=200k bytes storage, 50k bytes being used for the soft sectoring. The fist two tracks on the disc are occupied by the DOS taking 10k bytes, so now we are down to 190k. 2k of the third track is used for directory entries, leaving 188K for storage of data.

Each directory entry uses 32 bytes giving a maximum of 64 DIR entries, (64*32=2k).
We can examine a directory entry by inserting our backup copy of the master disc and doing the following. (Use a copy of the system disc as the following examples could corrupt the disc in use)

1..Load the disc
2..MOS<E>
3..R 8000 8800 0002<E>
4..T 8000 809F<E>

The command R will read into memory from disc, the data is read into location 8000H to 8800H from track 2 sector 0. ie the 2K directory is now in memory. Using the tabulate command T we can examine the first five DIR's ;
XBAS.COM, BACKUP.COM, COPY.COM, LOGO.COM, LOGO.COM
The disc DIR bytes are a follows:

```
BYTE
: 0  :1-8 :  9-11    :  12  :13-14:    15    :     16-31          :
:USER:FILE:EXTENSION:EXTENT:NOT  :128 BYTE:2k SECTOR BLOCK :
:AREA:NAME:          :      :USED :RECORDS :LOCATIONS          :
```

The DIR entry is copied into memory when the file is loaded and the information is used to form a File Control Block, FCB. Although the DIR entry is 32 bytes long, 36 bytes are used by the system when a file is in memory. These extra 4 bytes are only needed when the file is active, and therefore do not need to be held on disc. They are as follows; Byte 32 holds the record nubef of the next 128 byte record to be used if sequential access is in operation. Bytes 33 and 34 are used for holding the random record number when using random access files, the last byte is an overflow indicator.

The first byte of the FCB is the user area, this is always 00 as XTAL DOS does not implement user areas. The next eleven bytes should be quite familiar being the file name and extension, the file name is filled with spaces if less than eight bytes long. Byte 12 is the extent and is used to distinguish between FCB's for large files that use more than one FCB e.g. LOGO.COM. Byte 15 is the number of 128 byte records, for XBAS 7A*128=15616 bytes. The remaining bytes indicate the actual location of the file in 2k sector blocks.Each block occupies four sectors, (4*512 bytes = 2k). The memory copy of the FCB is updated as the file is altered and this FCB is written back to disc on saving the file. XBAS is on blocks 01-08 giving a total block size of 16k.

From this it can be seen that the minimum disc space allocated to a file is 2k bytes. The first block is track 02, sector 04 to track 02, sector 07.The first block of XBAS can be loaded into memory from MOS by:
1..R 8000 8800 0402<E> 2..T 8000 8800<E> (800H being 2k)

Holding down the break key will stop the display scrolling and escape will re-enter MOS. To work out how the disc block number converts to track and sector use the following formulae; convert the block number from HEX to DECIMAL, multiply by 4, divide by 10 and add 2. B*4/10+2. The whole number is the track and the remainder the sector.

After formatting a new disc the directory sectors contain the character E5. It is this character in byte 00 of the FCB that is used to check whether a file exists .To examine this, find the first DIR entry, XBAS, if the copy of the master disc is being used, and:

1..ERA XBAS.COM<E> 2..DIR<E>

The file XBAS.COM should not be displayed, now do:

1..MOS<E>
2..R 8000 8800 0002<E>
3..T 8000 809F<E>

The first byte of the first DIR entry is now E5 , replacing the number 00 that was originally there. The file is still present on disc but because the DIR entry is not valid the system cannot access the file. If we now do:

1..M 8000<E>
2..00.<E>
3..W 8000 8800 0002<E>
4..Y<E>
5..DIR<E>

XBAS should now be showing in the directory. It is only possible to regain files in total if no disc writes have occured on the sector blocks originally allocated to the file. Also if the FCB is overwritten the only way to recover files is by reading all the sector blocks and examing the contents for the file, presuming you can recognize what to look for!

It is most likely that at some time the DOS on a disc will become corrupt. A common cause is a mains supply spike. One solution is to reformat the disc but all data will be lost, a better method is to rewrite the DOS tracks and see if the disc will function normally. The reason the disc may not work even after rewritting the DOS tracks is that some of the sector marks are corrupt and without these the FDC cannot position the read/write head to the required sectors. The usual error message for a loss of DOS tracks is "DISC NO SECTOR". To rewrite the system tracks:

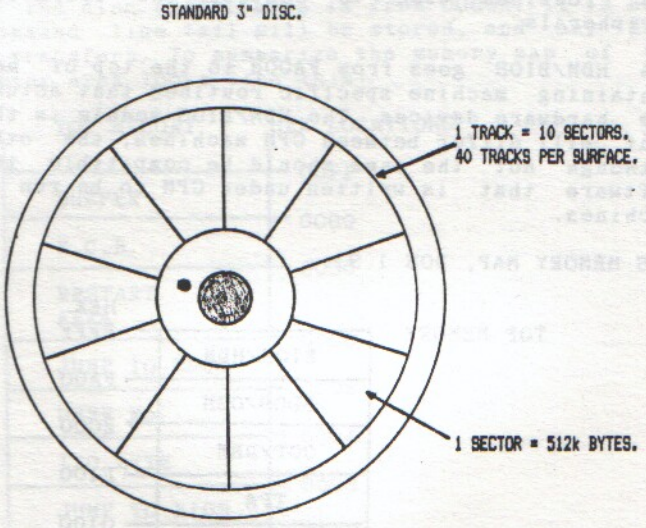
1..Load a good disc
2..MOS<E>
3..R 4000 6800<E>
4..Remove good disc, insert corrupt disc
5..W 4000 6800<E>

The above procedure reads the system tracks from a good disc into memory 4000H to 6800H, (remember that the system tracks 0+1 take 10k=2800 Hex), and then copies the memory image of the DOS to the corrupt disc.

Files that are not altered, or rarely changed can be Locked so accidental overwritting will not happen. System files such as XBAS should be locked, this is done using the DOS command LOCK. A locked file will show in the directory prefixed with an asterisk. The way DOS recognize that a file is locked is contained within that file's DIR entry. We can examine this by doing the following:.

```
1..Load system disc
2..MOS<E>
3..R 8000 8800 0002<E>
4..T 8000 809F<E>
```

Memory 8000H onwards is now displaying the first five FCB's as in the previous example. Bytes 9-11 of the FCB are the extension, COM for a command file such as XBAS. The Hex representation for COM is 434F4D, the actual Hex showing in bytes 9-11 of the FCB for XBAS is C34F4D. XBAS on the original disc is supplied as a Locked file, to make a locked file 80H is added to the first character of the extension. So for a COM file C(43H) becomes C(C3H).



On switching on EINSTEIN and pressing control-break DOS is loaded into memory, page 57 of the DOS/MOS handbook gives a memory map of the RAM at this stage. Xtal DOS is loaded as a CPM operating system would be. Xtal DOS uses different names for the modules in use compared to CPM. The DMM (Dos Monitor Module) equates to the CCP (Console Command Processor), the OSM (Operating System Module) to the BDOS (Basic Disc Operating System) and the HDM (Hardware Dependent Module) to the BIOS (Basic Input Output System), the TPA (Transient Program Area) being used for both systems. Locations 0000 to 00FFH are reserved for the System Parameters, and contain jump vectors to the actual DOS, or MOS, system information, and a buffer area for disc usage.

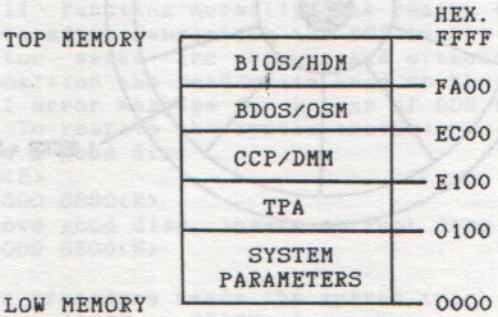
The TPA is the area of RAM that will hold any programs and data that are being executed or used, it extends from location 0100H to the start of the DMM at E100H.

The DMM/CCP contains the routines that monitor the console, and the DOS utilities such as DIR, ERA, etc. once a program is loaded this area of RAM can be overwritten, as the loaded program controls console activity. The DMM/CCP is reloaded on a warm start along with the OSM/BDOS.

The OSM/BDOS is located from EC00H to F9FFH, and contains the routines used to control the operation of the peripherals.

The HDM/BIOS goes from FA00H to the top of memory FFFFH, containing machine specific routines that actually control the hardware devices. The HDM/BIOS module is the one module that will differ between CPM machines, the other routines although not the same should be compatible thus enabling software that is written under CPM to be run on all CPM machines.

DOS MEMORY MAP, DOS 1.31.



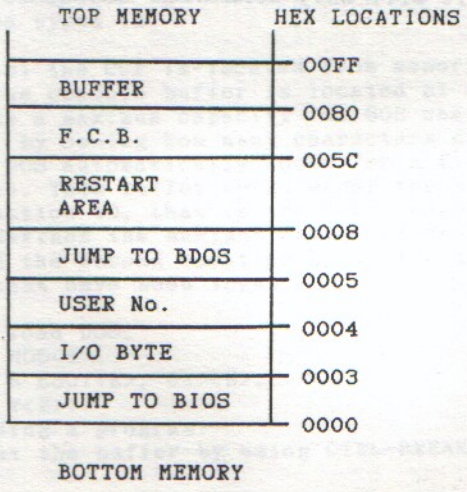
The OSM/BDOS and HDM/BIOS go together to form the FDOS (Full Disc Operating System). If we load DOS, then enter MOS we can examine the system parameter area of memory by:

```
1..T 0000 0080<E>
```

The first three bytes contain C303FA hex, meaning of course jump to location FA03H this is the jump vector to the HDM/BIOS warm start location. Execution from location FA03H will cause a fresh copy of the DMM/CCP and OSM/BDOS to be written into memory. The vectors to OSM/BDOS and HDM/BIOS are also re-initialized, (locations 0000, and 0005) The HDM/BIOS and TPA regions of memory are not altered on a warm start. The fourth byte of the system parameters is the IOBYTE which holds information on the peripherals, console,printer. Byte five contains the current user area. The next three bytes hold the vector to the OSM/BDOS, if we look at the first locations again:

```
1..MOS<E>  2..T 0000 0080<E>
```

Location 0005 holds a jump to EC00H this being the OSM/BDOS entry point. The next addresses are the restart locations for RST 8 through to RST 38. The region of memory from 005CH is the default File Control Block (FCB) area. Details from the FCB, of the file to be used, from the disc directory are entered here. The disc buffer area is from 0080H, it is here that the command line tail will be stored, and any data during disc transfers. To summarize the memory map of the system parameter area looks like this:



We can explore the BIOS vectors by examining the area of memory from FA00 as follows:

| | | | | | |
|---|---|---|---|---|---|
| BIOS | FA00 | Cold | Start | Vector | C3FA5C |
| BIOS+3 | FA03 | Warm | Start | Vector | C3FAC9 |
| Bios+6 | FA06 | Console | Status | Vector | C3FA80 |
| BIOS+9 | FA09 | Console | Input | Vector | C3FA34 |
| BIOS+12 | FA0C | Console | Output | Vector | C3FA37 |
| BIOS+15 | FA0F | Printer | Output | Vector | C3FA3B |
| BIOS+18 | FA12 | RS232 | Output | vector | C3FA3F |
| BIOS+21 | FA15 | RS232 | Input | vector | C3FA43 |
| BIOS+23 | FA18 | Vector to disc routines | | | C3FA49 |

If you are using a different version to DOS 1.31 it is a simple matter to work out the locations, remembering that the vector from location 0000 points to the second vector in the BIOS, (Warm Start). It can be seen that is is an easy matter now to redirect the devices connected to the machine. If we wanted to make the centronics printer output go to the serial port, all that is needed is to load DOS, go back to MOS and modify using the M command the vector at BIOS+15 (FA0F) to read the same as the vector at BIOS+18 (FA12). Another method would be to modify the memory that is called by the vector. This will only work for programs that use the DOS calls to output to the peripherals, some programs contain the code to output directly to the ports concerned and if this is the case you cannot patch the operating system to re-direct their output.

NOTE: Xtal DOS does not implement user areas as with a true CPM system, it will only work with the user number set to zero.

Auto-booting is the process of loading a program and executing it on pressing CTRL-BREAK as if loading DOS. This is accomplished by modifying the system tracks of the boot disc, (DOS tracks 0+1), to load the auto-boot program and run it.

If you normally work in BASIC then by modifying your system disc to auto-boot XBAS, Albert will automatically come up in BASIC. This facility is also very useful for setting the RS232 port or programming the function keys.

When running under the control of the DOS, a part of the operating system called the CCP, (Console Command Processor), handles the monitoring of the keyboard. When you type in a command, the CCP will process what you have input. The CCP expects to see a DOS command or a filename, if your input is not a valid DOS command, (DIR,ERA,DISP etc), the CCP presumes a file name has been entered and will go off and try to find, and load the requested program into memory from location 0100Hex onwards.

When you typed your entry into the machine the characters were stored in the console buffer, sometimes refered to as the command buffer. On loading DOS the console buffer area will contain 00H, and Albert will await your input. On loading a DOS that has been modified the memory area of the console buffer will be written into. Instead of containing 00H as would be expected on first loading DOS it now contains a file name or instruction, as per the modification. The CCP pressume# this command has just been input and instructs the machine to process what it thinks has just been typed in.

Under DOS 1.31 the CCP is located from memory location E100H to EC00H, the console buffer is located at memory location E307H and has a maximum capacity of 80H characters. You can prove this by seeing how many characters can be typed in before the DOS automatically looks for a file name, (80H = 128 Decimal). This buffer works under the same principle as the DOS function 10, that is the first memory location in the buffer defines the maximum amount of characters that can be input and the second location holds the actual number of characters that have been input.

```
Try this; load DOS,
          MOS<E>,
          M E307<E>, 01.<E>,
          Y<E>.
Now try loading a program!
You can reset the buffer by using CTRL-BREAK.
```

Once the DOS has loaded a program the control of the machines ports, i.e. keyboard etc, is handled by the loaded program, this means you cannot modify the DOS to auto-boot more than one program unless the first program calls the second, this will not usually be true for two or more .COM files but will work for XBAS and a second BASIC file. If you use BASIC then modifying the system tracks to auto-boot will save time and give a professional touch to the system.

If you work in Basic and create a file that sets the function keys this file can also be auto-called so we can now automatically bring up BASIC with the function keys programmed. If you intend to auto-boot more than one file the file names must be seperate# by a space, so to use XBAS and a function key setting program called, say, KEYS, the required input for the auto-boot program would be;
XBAS KEYS<E>.

DOS commands such as DIR can also be auto-booted as well as programs. If you auto-boot a file or program it must be present on the disc otherwise a NO FILE error will occur.

Below is a program written in XBAS that was adapted from Einstein User Vol 2,4 that will enable you to modify your system tracks for auto-booting. BEWARE this program is doing direct disc writes therefore you are strongly advised to have a copy of the disc you are modifying just in case something goes wrong and you end up corrupting your disc.

```
 10 REM *** AUTO-BOOT FOR DOS 1.31 ***
 20 RST:BCOL1
 30 CLEAR &8000
 40 POKE &8000,&3E,&00,&21,&00,&90,&11,&7F,&A9,
    &01,&00,&00,&CF,&A4,&C9
 50 PRINT "AUTO-BOOT MODIFICATION":PRINT
 60 PRINT "INSERT DISC TO BE MODIFIED INTO DRIVE 0"
 70 INPUT "ENTER NAME OF AUTO-BOOT PROGRAM/S ";N$
 80 CALL &8000
 90 FOR A=1 TO LEN(N$)
100 POKE &9208+A,ASC(MID$(N$,A,1))
110 NEXT A
120 POKE &9208,A-1
130 POKE &9208+A,0
140 POKE &800C,&A5
150 CALL &8000
160 PRINT "AUTO-BOOT MODIFICATION COMPLETE"
```

In the above program the DATA statements are used to write a small machine code program that is called from within Basic. The program reads into memory the DOS tracks, then modifies the command buffer area and re-writes the modified DOS tracks back to disc. Pressing CTRL-BREAK using this disc in drive 0 will now run the auto-boot modification.

The program uses ROM calls A4, disc read, and A5, disc write. Both ROM calls use the same parameters these being;

```
DRIVE No      ... A Register
DISC SECTOR   ... B Register
TRACK No      ... C Register
```

Data is then read/written into/from memory starting from the address held in the HL Register pair to the finish address held in the DE Register pair.

If we disassemble line 40 we can see these parameters being set up.

```
3E  00    ... LD A,00       ;Load the A register with 0, i.e.
                            ;use drive 0.
21 00 90  ... LD HL,9000    ;Load the HL register pair with
                            ;address 9000hex, i.e. set up start
                            ;address for disc operations at
                            ;9000Hex.
11 7F A9  ... LD DE,A97F    ;Load the DE register pair with
                            ;address A97FHex, i.e. set up
                            ;finish address. The DOS tracks
                            ;being 2800H bytes in length.
01 00 00  ... LD BC,0000    ;Disc track 0, sector 0. Physical
                            ;location of DOS on the disc.
CF A4     ... RST 8 A4      ;ROM call A4, disc read.
C9        ... RET           ;Return to XBAS.
```

After reading in the DOS tracks and modifying them, all that is required is to change the ROM call from a disc read to a disc write. This is done in line 150 by POKEing location 800CHex with A5 and then re-calling the routine to write to disc the modified command buffer.

## SETTING THE FUNCTION KEYS

As mentioned previously we can create a small program to set the function keys and this can be called from the auto-boot routine.
(Key command; page 124 of the Basic reference manual)
e.g.

```
10 KEY 0,"LIST c/r"
20 KEY 1,"CLS32 c/r"
30 KEY 2,"CLS40 c/r"
40 KEY 3,"TCOL15,4:CLS c/r"
50 KEY 4,"DRIVE1 c/r"
60 KEY 5,"LISTP c/r"
70 etc.
80 NEW
```

Adding a NEW command to the last line will clear the program from Basic once it has been run.

This program can then be saved to disc as KEYS, and called by the auto-boot routine. As stated in the manual the c/r is obtained by pressing the GRAPH and ENTER keys together. Any of the screen control codes can be programmed into the function keys by using the CHR$() command, these can be found in the DOS/MOS handbook pages 3-5. So to program a function key to give a text screen dump would be;

```
KEY 0,"CHR$(A)c/r".
```

This has the same effect as pressing CTRL-A.
The function key data is held in VRAM and a maximum of 128 characters can be used.

PROGRAM NOTES;
As the Auto-Boot program stands there is no error checking, therefore if the disc to be modified has it's write protect tab enabled the program will appear to work but of course is unable to write to the disc, hence no modification.

The DISP command from DOS enables the user to display ASCII files on the screen, CTRL-S being used to stop/start the display scrolling. READ is a small machine code utility that enables text files to be displayed page by page or line by line, with the option of a screen dump.

The program makes use of the DOS calls, (rear of DOS/MOS manual), to access and display a file. Pressing the space bar will display the next page, pressing the enter key will display the next line and using the escape key will dump the screen to the printer. Any other key will return to DOS. The following syntax should be used;
READ FILENAME.EXT<E> if READ is on the same drive as the program or READ X:FILENAME.EXT<E> where X is the drive containing the file to read.

```
ORG 0100H
LOAD 4100H
FCB:EQU 5CH          ;Default FCB
BDOS:EQU 5           ;BDOS entry
BUFF:EQU 80H         ;Default buffer
LINE:EQU 0E000H      ;Scratchpad
FUNCT:EQU 0E001H     ;Scratchpad
PARAM:EQU 0E002H     ;Scratchpad
RETVAL:EQU 0E004H    ;Scratchpad
LD A,(FCB+1)         ;Check for
CP 20H               ;File name
JP NZ,COMP           ;
LD DE,IPMESS         ;
LD (PARAM),DE        ;
LD A,9               ;DOS call 9
LD (FUNCT),A         ;
CALL SYSTEM          ;
RET                  ;RET TO DOS
START:LD DE,FCB      ;
CALL OPEN            ;
LD A,(RETVAL)        ;
CP 255               ;
JP Z,NOFILE          ;
CALL READ            ;
LD HL,BUFF           ;
BEGIN:LD B,24        ;Line count
CALL CLS             ;
PRINT:LD A,(HL)      ;
PUSH AF              ;
LD A,L               ;
CP 00                ;
JP Z,NEXT            ;
INC HL               ;
POP AF               ;
CP 1AH               ;EOF marker
JR Z,EXIT            ;Finished
CP 0AH               ;
JR Z,DEC             ;
```

PROGRAM NOTES; READ first checks that a filename has been given to 'read', this is done by checking the default FCB at location 005C+1 Hex, (the first location at 005C Hex will be 00 as this defines the user area). If a character other than 00 is found then a filename to 'read' has been input, and the program will begin. If a 00 is found then a message to inform the user of the correct syntax required is displayed. The program makes use of the fact that the DOS will automatically start a FCB, (at location 005C Hex), for the name of any file input. Having typed in READ FILENAME.EXT the DOS will start a FCB for READ, find the program in the directory and fill out the rest of the FCB. It will then load the program into

```
OUT:LD (PARAM),A      ;
LD A,2                ;
LD (FUNCT),A          ;
CALL SYSTEM           ;
JR PRINT              ;
SYSTEM:PUSH AF        ;System
PUSH BC               ;Subroutine
PUSH DE               ;
PUSH HL               ;
LD A,(FUNCT)          ;
LD C,A                ;
LD DE,(PARAM)         ;
CALL BDOS             ;
LD (RETVAL),A         ;
POP HL                ;
POP DE                ;
POP BC                ;
POP AF                ;
RET                   ;
DEC:DJNZ OUT          ;
NEWPGE:LD A,1         ;
LD (FUNCT),A          ;
CALL SYSTEM           ;
LD A,(RETVAL)         ;
CP 32                 ;Space bar
JR Z,BEGIN            ;
CP ODH                ;Enter key
JP Z,NLINE            ;
CP 1BH                ;Escape key
JP Z,SCPRT            ;
CALL CLS              ;
EXIT:JP 0000          ;Cold start
NLINE:LD B,1          ;
LD A,2                ;DOS call 2
LD (FUNCT),A          ;
LD A,OAH              ;
LD (PARAM),A          ;
CALL SYSTEM           ;
LD A,ODH              ;
LD (PARAM),A          ;
CALL SYSTEM           ;
JR PRINT              ;
SCPRT:LD A,1          ;Screen dump
LD (PARAM),A          ;
LD A,2                ;
LD (FUNCT),A          ;
CALL SYSTEM           ;
JP NEWPGE             ;
COMP:LD A,(FCB+9)     ;.COM file?
CP "C"                ;
JP NZ,START           ;
LD A,(FCB+10)         ;
CP "O"                ;
JP NZ,START           ;
```

memory from location 0100 Hex and overwrite location 005C Hex with the details of the FILENAME.EXT that followed READ on the command line. DOS will now 'goto' location 0100 Hex and execute the program READ. DOS calls, as per the rear of the DOS/MOS handbook, are used throughout the program. A 'system' call is used where all the registers are pushed onto the stack before calling DOS at location 0005 Hex, and poped on return. Scratchpad locations are used to hold the DOS function number, any paramaters that need to be passed to the DOS and, any returned values having called DOS. There are small subroutines to load the scratchpads with the required values and then a call to the 'system' subroutine is made. To display a message DOS function 9 is used, the start location of the message is placed in the DE register pair, 9 in the A register and a call to 0005 is made, (The message is terminated with the $ sign). Provided a filename has been given a check is made on the .EXTension to ensure it is not a .COM file, if a .COM extension is found

```
LD A,(FCB+11)          ;
CP "M"                 ;
JP NZ,START            ;
CALL CLS               ;
LD DE,COMMES           ;
CALL DISP              ;
RET;RET TO DOS         ;
DISP:LD A,9            ;Display
LD (FUNCT),A           ;Message
LD (PARAM),DE          ;
CALL SYSTEM            ;
RET                    ;
CLS:LD A,12            ;Clear screen
LD (PARAM),A           ;
LD A,2                 ;
LD (FUNCT),A           ;
CALL SYSTEM            ;
RET                    ;
NEXT:LD DE,FCB         ;Next record
CALL READ              ;
LD HL,BUFF             ;
JP PRINT               ;
OPEN:LD (PARAM),DE     ;Open file
LD A,OFH               ;
LD (FUNCT),A           ;
CALL SYSTEM            ;
RET                    ;
READ:LD (PARAM),DE     ;
LD A,14H               ;DOS call 20
LD (FUNCT),A           ;
CALL SYSTEM            ;
RET                    ;
NOFILE:CALL CLS        ;No file
LD DE,NOFMES           ;Message
CALL DISP              ;
RET                    ;
IPMESS:DB"CORRECT FORMAT IS: "
DB"READ FILENAME.EXT$"
NOFMES:DB"FILE NOT PRESENT$"
COMMES:DB"TO READ A COM FILE "
DB"LOAD FROM DOS THEN "
DB"  ENTER MOS AND TABULATE$"
NOP
END
```

then function 9 is again used to display a message to inform the user that READ will not work on .COM files. Provided all as is well the routine at START uses function 15 to open the file. The rest of the FCB will now be completed, provided the file is present on disc. Having opened the file we can now read in a 128 byte record to the default buffer at location 80Hex using function 20. The screen is cleared by using function 2, console output, by sending the ASCII code for CTRL-L, 12. It is now a matter of printing the buffer contents to the screen, again using function 2. A check is made for the end of file marker, 1AHex, and if found READ returns to DOS via a jump to location 0000, the cold start vector. After the first screen has been filled a check is made for the codes of the space, enter, and escape keys. A count is held in the HL register pair of how many characters have been displayed and after 80Hex the next record is read from disc into the buffer and the process continued until the end of file is found.

On the next page follows a Hex dump of the program.

```
Hex dump of READ program;
To enter from MOS;
1 .. MOS<E>
2 .. M 0100<E>
3 .. Enter the following code, finish with a full stop.
4 .. CTRL-BREAK to go back to DOS.
5 .. SAVE 2 READ.COM.
```

```
0100 3A 5D 00 FE 20 C2 B1 01 11 1E 02 ED 53 02 E0 3E
0110 09 32 01 E0 CD 50 01 C9 11 5C 00 CD FA 01 3A 04
0120 E0 FE FF CA 14 02 CD 07 02 21 80 00 06 18 CD E0
0130 01 7E F5 7D FE 00 CA EE 01 23 F1 FE 1A 28 46 FE
0140 0A 28 24 32 02 E0 3E 02 32 01 E0 CD 50 01 18 E1
0150 F5 C5 D5 E5 3A 01 E0 4F ED 5B 02 E0 CD 05 00 32
0160 04 E0 E1 D1 C1 F1 C9 10 DA 3E 01 32 01 E0 CD 50
0170 01 3A 04 E0 FE 20 28 B4 FE 0D CA 88 01 FE 1B CA
0180 A1 01 CD E0 01 C3 00 00 06 01 3E 02 32 01 E0 3E
0190 0A 32 02 E0 CD 50 01 3E 0D 32 02 E0 CD 50 01 18
01A0 90 3E 01 32 02 E0 3E 02 32 01 E0 CD 50 01 C3 69
01B0 01 3A 65 00 FE 43 C2 18 01 3A 66 00 FE 4F C2 18
01C0 01 3A 67 00 FE 4D C2 18 01 CD E0 01 11 54 02 CD
01D0 D3 01 C9 3E 09 32 01 E0 ED 53 02 E0 CD 50 01 C9
01E0 3E 0C 32 02 E0 3E 02 32 01 E0 CD 50 01 C9 11 5C
01F0 00 CD 07 02 21 80 00 C3 31 01 ED 53 02 E0 3E 0F
0200 32 01 E0 CD 50 01 C9 ED 53 02 E0 3E 14 32 01 E0
0210 CD 50 01 C9 CD E0 01 11 43 02 CD D3 01 C9 43 4F
0220 52 52 45 43 54 20 46 4F 52 4D 41 54 20 49 53 3A
0230 20 52 45 41 44 20 46 49 4C 45 4E 41 4D 45 2E 45
0240 58 54 24 46 49 4C 45 20 4E 4F 54 20 50 52 45 53
0250 45 4E 54 24 54 4F 20 52 45 41 44 20 41 20 43 4F
0260 4D 20 46 49 4C 45 20 4C 4F 41 44 20 46 52 4F 4D
0270 20 44 4F 53 20 54 48 45 4E 20 20 20 45 4E 54 45
0280 52 20 4D 4F 53 20 41 4E 44 20 54 41 42 55 4C 41
0290 54 45 24 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Ever lost any data by erasing a file that was not meant to be? if you have then this useful utility can help. Written in BASIC and using some ROM calls to access the disc you can now UNERASE any files that have been lost.

If you have read the article on discs, the principle behind this program will already be understood. For those who have not here is a quick refresh; the operating system uses the first 2 tracks of the disc, 2k of the third track, which is actually track 2 as we number from zero, contains directory information. Each file has 32 bytes of directory space, this information is used to create the FCB when a file is loaded into memory. The directory data holds the name of the file, pointers to the actual location of the data on the disc, and also a byte to indicate whether the file has been erased.

When a file is erased the data on the disc is not actually overwritten, only the first byte of the directory entry on the disc is altered. The first byte contains the user area the file was saved under, in Xtal DOS this is always 0, if it has been erased it will contain the Hex character E5. This is the same character that is used to format the disc, and if this character is present in byte 1 then the system will ignore that directory entry. To reclaim an erased file all that has to be done is to reset this byte to a 00 and we can re-access the file on disc. An important note is that we can only regain files that have not been written over, so it may be possible to unerase the directory entry but all the information pointed to may not relate to that file as the disc space may have been used by another program. Also unerasing files tends to confuse the operating system and disc space available figures may appear higher than they should. The best solution is to unerase the disc and transfer the contents to another disc and re-format the original.

The following program uses ROM calls A2, and A3. These being disc sector read and write respectively. It makes use of the default scratchpad locations for these calls, as labelled in the listing. As the directory occupies 4 sectors of 512K bytes we need to do 4 sector reads and writes, lines 130 to 220. Each sector is read into the default memory buffer from FE00H, byte 0 for each directory entry is then changed to 0 and the sector written back to disc. The program prompts the user for the required action, pressing any other key than "S" to start will leave the program and give a directory listing.

```
 10 RST
 20 PRINT"UNERASE BY U.K.E.U.G."
 30 PRINT:PRINT"ENTER DRIVE TO BE USED"
 40 INPUTA:IF A>4 THEN BEEP:GOTO 10
 50 PRINT:PRINT"INSERT DISC IN DRIVE";A
 60 PRINT:PRINT"PRESS S TO START":PRINT"ANY OTHER TO
    EXIT": B$=INCH$
 70 IF B$<>"S" OR B$<>"s" THEN 230
 80 CLEAR &8000
 90 POKE&FB50,A: REM * DRIVE NUMBER  *
100 POKE&FB51,2: REM * TRACK NUMBER *
110 DOKE&FB53,&FE00:REM * BUFFER ADDRESS *
120 POKE&8000,&CF,&A2,&C9:REM * ROM CALL DISC READ *
130 FOR F=0 TO 3
140 :  POKE&FB52,F:REM * SECTOR NUMBER *
150 :  CALL&8000
160 :     FORI=0TO511STEP32
170 :       IF PEEK(&FE00+I)=0THEN190
180 :       POKE&FE00+I,0:REM * UNERASE *
190 :     NEXT I
200 :  POKE&8001,&A3:REM * WRITE DISC SECTOR *
210 :  CALL&8000:POKE&8001,&A2
220 NEXTF
230 DIR
```

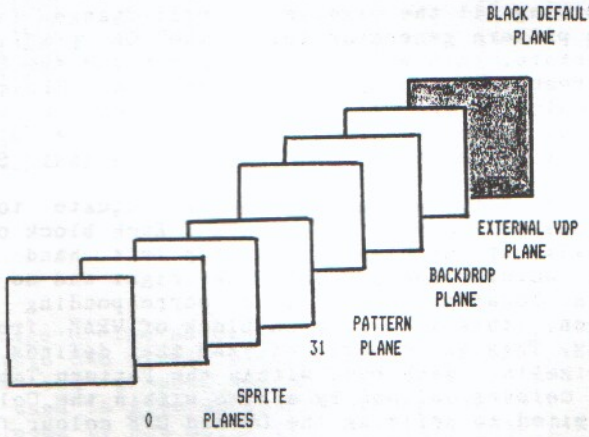
LISTING NOTES:
As the program stands there is no error checking, so if the disc to unerase has its write protect tab set the program will appear to work but as it is unable to write to the disc will not unerase.

It is a fact that some software houses use a development disc to work with whilst writing software and on completion erase everything except the final program. So it is always worth unerasing any discs you buy just in case! If possible backup first.

All screen operations other than 80 column work are handled by the Texas Instruments Video Display Processor, the TMS 9129. This integrated circuit or chip is a microprocessor in it's own right and has nine internal registers that control its operation. It is also coupled to a block of 16k bytes of Video RAM, (Random Access Memory), the combination of the settings in the registers and the data in the VRAM go together to make up the screen display. The following notes apply to the default settings of a Graphic mode II display. The Einstein always works in Graphic mode II unless the programmer initializes the VDP to one of the other 3 modes.

The VDP and the VRAM are initialized on power up by the MOS, i.e. the Machine Operating System or ROM, (Read Only Memory), contains the required software to initialize the VDP to the default settings. The screen display can be imagined as a set of 36 screens or planes that are stacked together, the nearest plane having priority. There are 32 sprite planes, a text plane, a backdrop plane, an external VDP input plane, and a black default plane. The external VDP plane uses another VDP to input a video signal and the two images are merged, to our knowledge this facility has never been used on the Einstein. If all planes are transparent then the the display defaults to black.

The diagram below shows how the planes go together to form a complete image.



The image that we see is created with a series of dots known as pixells, this principle is used to create newspaper pictures, as the eye cannot distinguish between the pixells they appear to be a complete image. A pixell is the smallest amount of information that can be displayed on the screen, the resolution of a display is usually measured in pixells, the higher the figures the better the resolution. Alberts resolution is 256*192, this means we have 49,152 pixells to play with. If you look at the memory map you will see that locations 0000 to 1800Hex are used for the Pattern Generator Table, this is the area of VRAM that holds the information on whether a pixell is being displayed or not. So how does 6k bytes of memory hold all the pixell information? As I'm sure you are aware there are 8 bits to a byte, therefore in 6k of memory there are 6,144*8 bits available, and that equals 49,152, (48k), the same amount of pixells we can access. Each pixell equates to a specific bit within a memory location, the top left hand corner of the screen is VRAM location 0000, see diagram. To turn a pixell on we set it's corresponding bit to a 1, to turn it off set to a zero. This small BASIC program illustrates how the VRAM locations equate to screen locations.

Pattern Table:

```
10 RST
20 BEEP 10
30 FOR A=0 TO &17FF
40 VPOKE A,255
50 NEXT A
```

The value of 255 in line 40 is setting all the pixells in the pattern generator to the 1 state, this will fill the screen with the current TCOL colour. Try using a different value to produce different effects.

Colour Table:

```
10 RST
20 BEEP 10
30 FOR A=&2000 TO &37FF
40 VPOKE A,&36
50 NEXT A
```

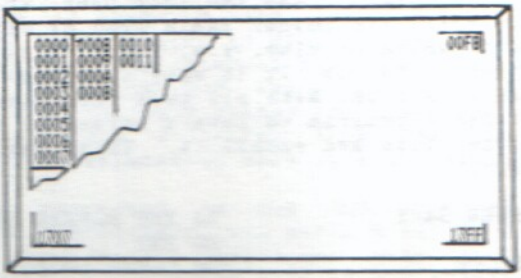
The value of &36 in line 40 will change the colour of the ON pixells to light green and the OFF pixells to dark red. The colours are as per the colour chart, the first value, 3 is ON pixells and the last, 6 OFF pixells.

The VRAM Pattern Table locations equate to the screen locations as shown in the diagram. Each block of 8*8 pixells is built up starting at the top left hand corner, then moving across the screen to the right and so on. For each pattern location there is a corresponding colour table location, this is also a 6K block of VRAM, from 2000Hex to 37FFHex. This is the area of VRAM that defines the colour of the pixells, each byte within the Pattern Table has it's pixell colours defined by a byte within the Colour Table. We are limited to defining the ON and OFF colour for a row of 8 pixells, this limitation can cause problems when creating pictures in colour due to pixells of one colour overlapping those of a different colour. This effect is easily seen when using Picpen.

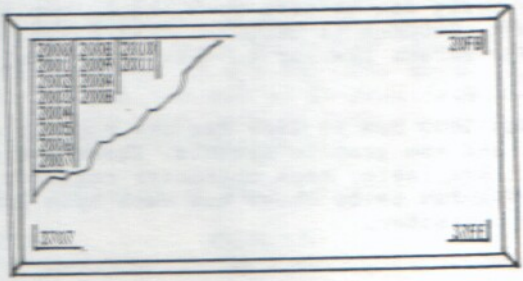
# VIDEO DISPLAY

Video memory locations relative to screen position;

PATTERN TABLE



COLOUR TABLE



The full image of the screen is 256*192 pixells, this is broken up into 8*8 pixell blocks giving 32*24 character positions in the graphic modes. In text mode 6*8 pixell blocks are used to give 40 characters across the screen, this is achieved by not using the two right hand pixels of the character block, if using graphic symbols the 32 column mode must be used otherwise some graphics will be truncated.

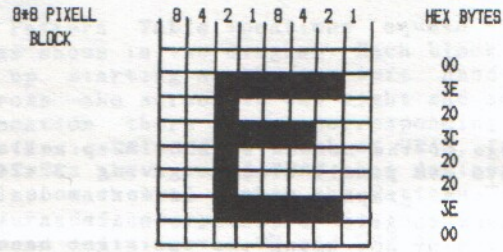
So how do we use a byte of memory to define 16 colours? As said before we have 8 bits to a byte, we use 4 bits for the foreground or ON pixells and 4 bits for the background or OFF pixells . As there are 16 possible combinations from 4 bits all the 16 colours available can be defined. The four MSBs are the ON pixells and the four LSBs are the OFF pixells, if we VPEEK the Colour Table area of VRAM in its default state of white on blue, i.e. PRINT VPEEK(&3000) we will find a value of F4 Hex. It is easy to decode this, F or 15 is white and 4 is blue. With all this information we can write a small Basic program to save a screen as an .OBJect, or machine code, file and recall it. This takes several seconds in Basic.

```
 10 REM SCREEN SAVE
 20 RST
 30 CLEAR &8000
 40 ELLIPSE 100,100,50
 50 C=&8000
 60 FOR A=0 TO &17FF
 70 B=VPEEK(A)
 80 POKE C,B
 90 C=C+1
100 NEXT A
110 SAVE"SCREEN.OBJ",&8000,&97FF
```

```
 10 REM SCREEN LOAD
 20 RST
 30 CLEAR &8000
 40 LOAD"SCREEN.OBJ"
 50 C=0
 60 FOR A=&8000 TO &97FF
 70 B=PEEK(A)
 80 VPOKE C,B
 90 C=C+1
100 NEXT A
```

Obviously you will draw something more spectacular than line 30! The same thing can be done with the colour table, using VRAM locations 2000Hex to 3800Hex. This could be used to create a loader screen similar to that when using a cassette based machine.

VRAM locations 1800 Hex to 19FF Hex are used to define the character set and the graphic symbols. There are a possible 256 characters available, each character requires 8 bytes of memory, the diagram below shows how each byte goes to make up a complete character.



To set the control bits for a read, i.e. the two MSB bits of the MSB address byte to 00, we AND with 3FHex. If we look at the binary of 3FHex all should become clear;

```
Hex      :       3       :       F       :
Binary   : 0 : 0 : 1 : 1 : 1 : 1 : 1 : 1 :
```

ANDing with zero always produces a zero, ANDing with a one will produce a one if the other bit of the AND is a one or a zero if the other bit is a zero. Therefore the first two bits will always end up as a 0 and the following bits will be the same as they were. To set the control bits to 01 as for a write all that is needed is to OR with 40Hex after ANDing with 3FHex as above. ORing always produces a 1 if either or both bits are a 1, a zero is produced if both bits are zero.

The VDP requires a delay of 8 micro seconds between successive reads or writes to VRAM, this is usually not a problem due to the code being used between the operation of reading or writing but if in line code is used a delay must be introduced using say a PUSH and a POP or similar instructions.

VDP REGISTERS:
There are 8 write only registers and 1 read only or STATUS register. To write to any of registers 0-7 requires a two byte transfer. The first byte contains the data to be written into the register and the second byte contains the register number we wish the data to be input to. The register number is held in the 3 LSB's, the MSB must be a 1 with all other bits set to ZERO.

```
                    MSB                                 LSB
BYTE 1/DATA ..... : ? : ? : ? : ? : ? : ? : ? : ? :
BYTE 2/REGISTER.. : 1 : 0 : 0 : 0 : 0 :  REG. No. :
```

To read the STATUS register we read the value input from port 9.

The VDP mode of operation is controlled by the settings of the Mode bits in registers 0 and 1.

| | MB1 | MB2 | MB3 |
|---|---|---|---|
| GRAPHICS MODE 1 ... | 0 | 0 | 0 |
| GRAPHICS MODE 2 ... | 0 | 0 | 1 |
| MULTICOLOUR ..... | 0 | 1 | 0 |
| TEXT MODE ..... | 1 | 0 | 0 |

VDP REGISTERS:

| Register | MSB | | | | | | LSB |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | MB3 | EV |
| 1 | 1 | BL | IE | MB1 | MB2 | 0 | SIZE | MAG |
| 2 | 0 | 0 | 0 | 0 | PATTERN NAME TABLE | | |
| 3 | PATTERN COLOUR TABLE BASE ADDRESS | | | | | | |
| 4 | 0 | 0 | 0 | 0 | 0 | PATTERN GENERATOR | |
| 5 | 0 | SPRITE ATTRIBUTE TABLE BASE ADDRESS | | | | | |
| 6 | 0 | 0 | 0 | 0 | 0 | SPRITE PATTERN | |
| 7 | TEXT ON COLOUR | | | | TEXT OFF/BACKDROP | | |
| STATUS | F | 5SP | C | FIFTH SPRITE NUMBER | | | |

REGISTER 0: Only the LSB and bit 1 are used;
Bit 0 - external VDP control, set to 0.
Bit 1 - mode select bit 3, used with register 1 bits 3 @ 4 to set the VDP operating mode.

REGISTER 1:
Bit 0 - 0 selects no sprite magnification, 1 selects *2.
Bit 1 - 0 selects sprite size 8*8, 1 selects 16*16.
Bit 2 - not used.
Bit 3 - mode select bit 2.
Bit 4 - mode select bit 1.
Bit 5 - not used.
Bit 6 - 0 to blank screen, 1 normal.
Bit 7 - set to 1 on Einstein for 16k video RAM.

REGISTER 2:
Only the 4 LSB bits are used, the upper 4 being set to 0. The start address of the name table = contents of register 2*400 Hex.

REGISTER 3:
The contents of register 3 form the upper 8 bits of the fourteen bit pattern colour table base address. The pattern colour table base address = contents of register 3 * 40 Hex. In graphics mode II the colour table may only be located at 0000 or 2000 Hex.

REGISTER 4:
Only the 3 LSB are used, all other bits set to 0. The 3 LSB define the base address of the pattern generator table. The base address = contents of register 4*800 Hex. In graphics mode II this may only be located at 0000 or 2000 Hex.

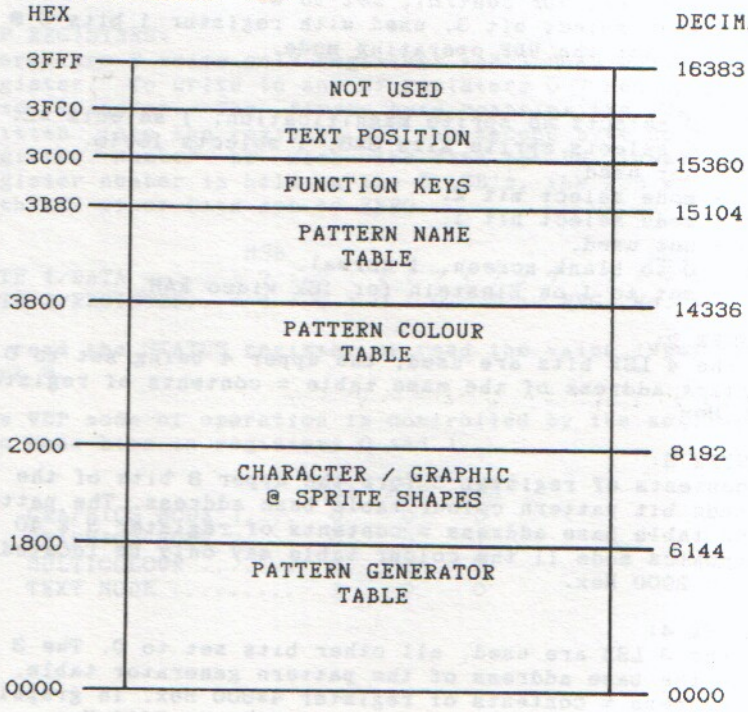
REGISTER 5:
Bit 7 - set to 0
The remaining bits define the base address of the sprite attribute table, address = contents of register 5*80 Hex.

REGISTER 6:
Only the 3 LSB are used all others set to 0. Forms the base address of the sprite pattern generator. Address = contents of register * 800 Hex.

REGISTER 7:
The upper 4 bits contain the colour code for the ON pixells in text mode, the lower four bits the colour code for the OFF pixells in text mode and the backdrop colour in all modes.

REGISTER 8: (STATUS REGISTER)
The 5 LSB contain the number of the 5tf sprite, the VDP can only display a maximum of 4 sprites correctly on one line.
Bit 5 - Is set to 1 if two or more sprites overlap.
Bit 6 - Set to 1 when 5 or more sprites occur on one line.
Bit 7 - Interrupt flag, set to 1 at the end of a raster.

DEFAULT MEMORY MAP:



There now follows a routine to save and load a screen using a machine code program. If you want to use this program from Basic instructions are on the next pages.

```
ORG 0100H
LOAD 4000H
SCRATCH:EQU 0D000H
LD HL,0B000H        ;START LOCATION RAM
LD (SCRATCH),HL     ;POINTER TO RAM
LD A,00             ;LSB OF ADDRESS
OUT(9),A            ;SEND TO VDP
LD A,00             ;MSB OF ADDRESS
AND 3FH             ;SET CONTROL BITS
OUT(9),A            ;SEND TO VDP
LD B,24             ;24 ROWS ON SCREEN
LOOP1:PUSH BC       ;ROW NUMBER
LD B,0              ;COUNT 256 PIXELLS
LOOP2:IN A,(8)      ;READ FROM VDP
LD HL,(SCRATCH)     ;UPDATE
LD (HL),A           ;THE
INC HL              ;RAM
LD (SCRATCH),HL     ;POINTER
DJNZ LOOP2          ;256 PIXELLS DONE?
POP BC              ;GET NEXT ROW
DJNZ LOOP1          ;DO 256 PIXELLS
RET                 ;FINISHED SAVE
LD DE,0B000H        ;START OF LOAD
LD HL,0             ;FROM RAM TO VRAM
LD B,24             ;24 ROWS
LOOP3:PUSH BC       ;SAVE ROW COUNT
LD B,0              ;256 PIXELLS
LOOP4:PUSH HL       ;SAVE VRAM ADDRESS
PUSH DE             ;SAVE RAM ADDRESS
LD A,L              ;LSB VRAM ADDRESS
OUT (9),A           ;SEND TO VDP
LD A,H              ;MSB VRAM ADDRESS
OR 40H              ;SET CONTROL BITS
OUT (9),A           ;SEND TO VDP
POP DE              ;GET RAM ADDRESS
LD A,(DE)           ;DATA FROM  RAM
OUT(8),A            ;SEND TO VDP
POP HL              ;UPDATE
INC HL              ;VRAM POINTER
INC DE              ;RAM POINTER
DJNZ LOOP4          ;256 PIXELLS?
POP BC              ;NEXT ROW
DJNZ LOOP3          ;NEXT 256 PIXELLS
RET                 ;FINISH LOAD
NOP
END
```

The first part of this program will make a copy of the Pattern Table, VRAM locations 0000-17FF Hex, into main RAM locations B000-C7FF Hex. The second part of the program will copy main RAM back into VRAM. To use the screen load requires a call to location 0125 Hex and the screen information must be in main RAM from location B000 Hex.

If you do not have an assembler or prefer to enter the code from MOS this is what you need to do;
Enter DOS with a CTRL-BREAK, then type MOS<E>.

```
1 .. M0100<E>
```

You will see location 0100H and it's present contents being displayed on the screen. What you need to now do is enter the following code, I usually enter lines of 8 bytes followed by ENTER. After you have finished entering the code exit the modify by using .<E> (full stop then enter). You should now check that everything is correct by tabulating the memory you have just altered,

```
2 .. T 0100 0180<E>
```

You should see the same Hex dump as is printed below, if it is not the same then go back and modify the differing locations. When you have completed entering the code the next step is to save it as a file so it can be used from XBAS. The type of file we need is a .OBJect file, so enter DOS with a CTRL-BREAK and type

```
3 .. SAVE 1 SCREEN.OBJ<E>
```

We now have a machine code file that can be LOADed into a basic program and CALLed. If you check the DIRectory you will see the file SCREEN.OBJ now present, the 1 after the SAVE command tells the DOS how much code to save in 256 byte blocks. Hex dump of the screen save/load program;

```
0100 21 00 B0 22 00 D0 3E 00
0108 D3 09 3E 00 E6 3F D3 09
0110 06 18 C5 06 00 DB 08 2A
0118 00 D0 77 23 22 00 D0 10
0120 F4 C1 10 EE C9 11 00 B0
0128 21 00 00 06 18 C5 06 00
0130 E5 D5 7D D3 09 7C F6 40
0138 D3 09 D1 1A D3 08 E1 23
0140 13 10 ED C1 10 E7 C9 00
```

To use the program from XBAS we have to be careful where we load the SCREEN.OBJ file. The program is using memory locations B000Hex to C7FFHex to save the VRAM, XBAS uses memory upto approximately 4000Hex so we can use memory somewhere between the two to locate our .OBJ file. The example uses A000Hex which leaves plenty of room for any basic programing. All we need to do is to CLEAR &A000 and CALL &A000 to run the first part of the program, the screen save. To run the second part, the screen load we need to CALL &A025. Having saved a screen we can then save the RAM as another .OBJ file for use later on.

SCREEN SAVE:

```
10 CLS
20 CLEAR &A000
30 LOAD"SCREEN.OBJ"
40 ELLIPSE 100,100,60
50 CALL &A000
60 SAVE "DEMO.OBJ",&B000,&C7FF
```

SCREEN LOAD:

```
10 CLS
20 CLEAR &A000
30 LOAD "SCREEN.OBJ"
40 CLEAR &B000
50 LOAD "DEMO.OBJ"
60 CALL &A025
```

Line 30 in the save program is where your artistic impressions would go. This principle could be used to flash between two or more screens saved in RAM or a form of windowing could be designed. It will not work with colour as it stands but a very similar routine for the colour table could be written.

SPRITE ATTRIBUTES:
The locations 3B00 Hex to 3B7F Hex hold information on the 32 available sprites. 4 bytes of memory are used for each sprite, the first two bytes determine the sprites position on the screen, the third byte is a pointer to the character shape from the pattern table, the last byte is used to define the sprites colour. If more than 4 sprites are displayed on the same horizontal line the 4 highest sprites will be displayed normally the 5th and subsequent sprites are transparent on that line. The 5th sprite flag is set in the status register as is the number of the 5th sprite.

FUNCTION KEYS:
The data for the function keys is held in VRAM locations 3B80 Hex to 3BFF Hex. Each key setting is seperatef from the next one by adding 80 Hex to the last character in the string. If less than all 128 bytes are used a 00 defines the end of the settings.

UNUSED VRAM:
The very top of VRAM, locations 3FC0 Hex to 3FFF Hex are not used. These locations are not reset unless the machine is powered off and on, a reset using the button at the rear of the machine does not affect these locations.

## CHARACTER SET

Here is a complete new character set. You can use it in your own programs, however it is not compatible with 40 columns. Once the character is loaded try CLS40 and you will see what we mean.

To return to the normal character set, just use the RST command.

```
   10 CLS32
   20 FOR F=32 TO 122
   30 READ A$
   40 SHAPE F,A$
   50 NEXTF
   60 PRINT@13,6;"O.K.";@3,8;"CHARACTER SET NOW LOADED "
   70 PRINT:PRINT:FOR F=32TO122:PRINTCHR$(F);:NEXT
 1000 DATA "0000000000000000"
 1001 DATA "3838383838003800"
 1003 DATA "6C6C244800000000"
 1004 DATA "44FE444444FE4400"
 1005 DATA "107C407C047C1000"
 1006 DATA "E2A4E8102E4A8E00"
 1007 DATA "60908040A090E800"
 1008 DATA "3838083000000000"
 1009 DATA "1830606060301800"
 1010 DATA "30180C0C0C183000"
 1011 DATA "00187E3C7E180000"
 1012 DATA "0010107C18180000"
 1013 DATA "00000000000C0C18"
 1014 DATA "0000007E7E000000"
 1015 DATA "0000000000181800"
 1016 DATA "0000020408102000"
 1017 DATA "7C444C5464647C00"
 1018 DATA "7010101010107C00"
 1019 DATA "7E02027E60607E00"
 1020 DATA "3E02023E06067E00"
 1021 DATA "404444447E0C0C00"
 1022 DATA "7C40407C0C0C7C00"
 1023 DATA "7C44407C64647C00"
 1024 DATA "7E42040818181800"
 1025 DATA "7C44447C64647C00"
 1026 DATA "7C44447C0C4C7C00"
 1027 DATA "0038380038380000"
 1028 DATA "0038380038380830"
 1029 DATA "0C1830603018C000"
 1030 DATA "007C7C007C7C0000"
 1031 DATA "30180C060C183000"
 1032 DATA "7E42021E10001000"
 1033 DATA "FE82BAAABE80FE00"
 1034 DATA "FC84FEC2C2C2C200"
 1035 DATA "FC84FEC2C2C2FE00"
 1036 DATA "FE8280C0C0C2FE00"
 1037 DATA "FC8282C2C2C2FC00"
 1038 DATA "FE80FCC0C0C0FE00"
 1039 DATA "FE80FCC0C0C0C000"
```

```
1040 DATA "FE8280DEC2C2FE00"
1041 DATA "8282FEC2C2C2C200"
1042 DATA "1010181818181800"
1043 DATA "0202060606867C00"
1044 DATA "828488F8C4C2C200"
1045 DATA "8080C0C0C0C0FE00"
1046 DATA "FC92D2D2D2D2D200"
1047 DATA "E292D2D2D2D2CE00"
1048 DATA "FE82C2C2C2C2FE00"
1049 DATA "FE8282FEC0C0C000"
1050 DATA "FE82C2C2CAC4FA00"
1051 DATA "FE8282FEC4C2C200"
1052 DATA "FE8060FE0606FE00"
1053 DATA "FE1018181818180"
1054 DATA "8282C2C2C2C2FE00"
1055 DATA "8282C2C2442810000"
1056 DATA "9292D2D2D2D2FE00"
1057 DATA "8244281028448200"
1058 DATA "828282FE10181800"
1059 DATA "FE0408102060FE00"
1060 DATA "7E7E8060607E7E00"
1061 DATA "0000402010080400"
1062 DATA "7E7E0606067E7E00"
1063 DATA "183C664200000000"
1064 DATA "00000000000000FF"
1065 DATA "7E42F8606060FE00"
1066 DATA "00007E027E427E00"
1067 DATA "40407E4242427E00"
1068 DATA "00007E4040407E00"
1069 DATA "02027E4242427E00"
1070 DATA "00007E427E407E00"
1071 DATA "3E20782020202000"
1072 DATA "00007E42427E027E"
1073 DATA "40407E4242424200"
1074 DATA "1800181818181800"
1075 DATA "0200020202027E"
1076 DATA "4040424478444200"
1077 DATA "1818181818181800"
1078 DATA "00007C5454544400"
1079 DATA "00007E4242424200"
1080 DATA "00007E4242427E00"
1081 DATA "00007E42427E4040"
1082 DATA "00007E42427E0202"
1083 DATA "00007E4240404000"
1084 DATA "00007E407E027E00"
1085 DATA "20207E2020203E00"
1086 DATA "0000424242427E00"
1087 DATA "0000444444281000"
1088 DATA "0000445454547C00"
1089 DATA "0000442810284400"
1090 DATA "00004242427E027E"
1091 DATA "00007C0810207C00"
```

CHARDES.

## CHARACTER DESIGNER

This program allows you to design your own character set from A to z, (ASCII 65 to 122)and save it as an OBJECT file. Then using the short routine following you can reload your character sets for use with other programs.

```
 10 CLEAR&B000:FORF=6664TO7392:A=VPEEK(F):POKE F+&95F8,A
    :NEXT F
 20 MAGO:RST:BCOL1:  TCOL15,10:CLS32:GOSUB550
 30 BEEP
 40 SPRITE1,X,Y,1,159
 50 A=KBD
 60 IF A=80THENX=X+8:IF X>136THEN X=80
 70 IF A=79THENX=X-8:IF X<80THEN X=136
 80 IF A=81THENY=Y+8:IF Y>144THEN Y=88
 90 IF A=65THENY=Y-8:IF Y<88THEN Y=144
100 IF A=32THENTCOL15,4:PRINT@X/8,24-Y/8;CHR$(158):
    TCOL15,10
110 IF A=70 THENPRINT@X/8,24-Y/8;CHR$(160)
120 IF A=42 THEN GOSUB 550:REM CLEAR
130 IF A=43 THEN GOSUB 160
140 IF A=63 THEN GOSUB 480
150 GOTO40
160 REM STORE DATA TO CHARACTER
170 BEEP:FORF=1TO200:NEXT :TCOL1,4:  PRINT@0,21;"ENTER
    CHARACTER TO CHANGE";
180 POKE64326,0:CH=KBD:IF CH>64ANDCH<122THEN190:ELSE 180
190 PRINT@0,0;"S T O R E";@1,2;CH;@5,2;CHR$(CH):PRINT@
    0,21;"R TO REDO C TO CONTINUE  ";
200 POKE64326,0:Q=INCH:IF Q<>82 AND Q<>67 THEN BEEP:
    GOTO200
210 IF Q=82 THEN BEEP:TCOL15,10:PRINT@0,21;SPC(31):FOR
    F=0TO3:PRINT@0,F;SPC(9):NEXT  :RETURN
220 FOR F=0TO7
230 A$(F)=SCRN$(F+6)
240 A$(F)=MID$(A$(F),11,8)
250 NEXTF
260 PRINT@0,21;"PLEASE WAIT-STORING CHARACTER "
270 FOR G=0TO7
280 TT=0
290 FOR F=0TO7
300 N=ASC(MID$(A$(G),F+1,1))
310 IF N=160 AND F=0 THEN TT=TT+128
320 IF N=160 AND F=1 THEN TT=TT+64
330 IF N=160 AND F=2 THEN TT=TT+32
340 IF N=160 AND F=3 THEN TT=TT+16
350 IF N=160 AND F=4 THEN TT=TT+8
360 IF N=160 AND F=5 THEN TT=TT+4
370 IF N=160 AND F=6 THEN TT=TT+2
380 IF N=160 AND F=7 THEN TT=TT+1
390 NEXTF
400 CHV(G)=TT
410 NEXTG
420 N=0
430 MEM=&B000+((CH-65)*8)
```

```
440 FOR F=MEM TOMEM+7
450 POKE F,CHV(N):N=N+1
460 NEXT F
470 Q=82:GOTO210
480 REM SAVE REDEFINED CHARACTER SET
490 PRINT@0,21;"SAVE CHARACTER SET YES OR NO  ";:A$=INCH$
500 IF A$="Y"THEN 510:ELSE Q=82:GOTO210
510 PRINT@0,21;"ENTER FILE NAME (8 CHARS MAX) ":INPUTF$
520 F$=F$+".OBJ"
530 SAVE F$,&B000,&B000+728
540 Q=82:PRINT@0,22;SPC(20): GOTO210
550 REM SETUP
560 SHAPE159,"FFFFC3C3C3C3FFFF"
570 SHAPE158,"FF818181818181FF"
580 X=80:Y=144:TCOL15,4
590 FOR F=10TO17
600 FOR G=6TO13
610 PRINT@F,G;CHR$(158)
620 NEXTG,F
630 PRINT@20,0;" M E N U "
640 PRINT@19,2;"LEFT  ... O"
650 PRINT@19,4;"RIGHT ... P"
660 PRINT@19,6;"UP    ... Q"
670 PRINT@19,8;"DOWN  ... A"
680 PRINT@19,10;"FILL  ... F"
690 PRINT@19,12;"UNFILL..SPC"
700 PRINT@19,14;"CLEAR ... *"
710 PRINT@19,16;"STORE ... +"
720 PRINT@19,18;"SAVE  ... ?"
730 TCOL15,10:  RETURN
```

This is the loader program for the object file produced by the character set designer. Remember if the characters don't come out properly you can return to the normal character set with the RST command.

```
 10 RST:TCOL15,4:BCOL1:CLS32
 20 CLEAR &B000:FORF=&B000TO&B000+728:POKEF,0:NEXTF
 30 TCOL4,15:PRINT@6,1;"CHARACTER SET LOADER":TCOL15,4
 40 PRINT@0,3;:INPUT"ENTER FILENAME ";F$
 50 F$=F$+".OBJ"
 60 BEEP:PRINT@0,3;SPC(32);@0,3;"Loading ";F$
 70 LOAD F$
 80 BEEP:PRINT@0,3;SPC(32);@0,3;"Pokeing character set
    into VRAM"
 90 N=0
100 FOR F=&B000 TO &B000+728
110 A=PEEK(F)
120 VPOKE6664+N,A:N=N+1
130 NEXTF
140 BEEP:FOR F=2TO22:PRINT@0,F;SPC(32):NEXT F
150 PRINT@0,3
160 FOR F=65 TO122:PRINTCHR$(F);:NEXTF
170 END
```

There now follow two routines to enable the dumping of pictures from the screen to the printer. The first explanation uses the program from Einstein User Vol 1,4. Our thanks to Tatung for allowing its publication. The second program is a modification of the first which gives a double sized dump, this was originally written by Dave Salvage.

To dump drawings or pictures from the screen to the printer requires the use of a printer capable of printing in a pin addressable mode. If we briefly look at the way a dot matrix printer works we can see what this means. The print head comprises of a number of 'pins' these pins are pressed against an inked ribbon and onto the paper creating a dot for each pin. This is the same principle as the screen pixells, a matrix of dots is used to form a character and the human eye merges the dots together. The closer the dots are printed the clearer the character, this principle is used for Near Letter Quality printers where two passes of the print head are made with a very slight movement of the paper between the two passes. The second pass of the print head fills in the spaces between the dots and a clearer image is created. A printer which can address each pin seperately is capable of graphic dumps. As there is no routine within the MOS to do this we need a small piece of code which converts the pixells on the screen to pins on the printer.

The source listing is included after the Hex dump if you wish to use an assembler, as the code is fairly short it is as quick to use the MOS as follows;

```
1 .. MOS<E>
2 .. M 0100<E> Now enter the following code:

     0100  DB20E61CFE10C02A<E>
     0108  9AFBE52A9CFBE521<E>
     0110  BF00229CFB210000<E>
     0118  229AFBE5DDE1E5FD<E>
     0120  E1DDE506082171A0<E>
     0128  7ECF9F2310FAFDE5<E>
     0130  0E01C5CFC7C12801<E>
     0138  37CB11FD2B30F379<E>
     0140  CF9FFDE1DD23DDE5<E>
     0148  F1B728E2DDE101F8<E>
     0150  FFFD093E0ACF9FFD<E>
     0158  E5E101BF000938C1<E>
     0160  3E1BCF9F3E40CF9F<E>
     0168  E1229CFBE1229AFB<E>
     0170  C90D1B41081B4B00<E>
     0178  01.<E>

3 .. CTRL-BREAK<E>
4 .. SAVE 1 GDUMP1.OBJ<E>
```

We now have a machine code file that can be used from Basic. The values in locations 0126 and 0127 define where in memory the file should be loaded, as it is written this is &A000. To use the dump from Basic the following lines need to be added to your program;

```
CLEAR &A000
LOAD "GDUMP1.OBJ"
CALL &A000
```

SOURCE LISTING FOR GDUMP1:

```
ORG 0A000H
LOAD 0A000H
ORGX:EQU 0FB9AH
ORGY:EQU 0FB9CH
AUXREG:EQU 20H
IN A,(AUXREG)
AND 1CH
CP 10H
RET NZ
LD HL,(ORGX)
PUSH HL
LD HL,(ORGY)
PUSH HL
LD HL,191
LD (ORGY),HL
LD HL,0
LD (ORGX),HL
PUSH HL
POP IX
PUSH HL
POP IY
GDUMPO:PUSH IX
LD B,9
LD HL,PARAMS
GDUMP1:LD A,(HL)
RST 8
DEFB 9FH
INC HL
DJNZ GDUMP1
GDUMP2:PUSH IY
LD C,1
GDUMP3:PUSH BC
RST 8
DEFB 0C7H
POP BC
JR Z,GDUMP4
SCF
GDUMP4:RL C
DEC IY
JR NC,GDUMP3
LD A,C
RST 8
DEFB 9FH
POP IY
INC IX
PUSH IX
POP AF
OR A
JR Z,GDUMP2
POP IX
LD BC,0FFF8H
ADD IY,BC
LD A,0AH
RST 8
DEFB 9FH
PUSH IY
POP HL
LD BC,191
ADD HL,BC
JR C,GDUMPO
LD A,1BH
RST 8
DEFB 9FH
LD A,40H
RST 8
DEFB 9FH
POP HL
LD (ORGY),HL
POP HL
LD (ORGX),HL
RET
PARAMS:DEFB 0DH,1BH,41H,08H,1BH
DEFB 2AH,05H,00H,01H
END
```

The first U.K.E.U.G. newsletter featured an article by Chris Giles to add the dump program to the DOS tracks of the disc. Once done, the dump program is automatically loaded into memory as the DOS is loaded. All that is needed to access the program is a call to the location containing the code. To add the dump to DOS 1.31 we need to read into memory the DOS tracks, add the dump and write back the 'new' DOS. If you have already entered the dump as above and saved it as an object file then this is what is required;

1 .. CTRL-BREAK<E>: To enter DOS.
2 .. LOAD GDUMP1.OBJ<E>: The dump program is now in memory from location 0100 Hex onwards.
3 .. MOS<E>: To enter MOS.
4 .. R 8000 9800<E>: Read DOS tracks into memory locations 8000-9800 Hex.
5 .. C 0100 0178 8170<E>. Copy dump program from 0100 to memory containing DOS at 8170 Hex.
6 .. M 8196<E>: Modify pointer to printer parameters.
7 .. E1E2.<E> : Enter new values.
8 .. W 8000 9800<E>: Write modified DOS to disc.

If we now do a CTRL-BREAK with this disc in drive 0 the modified DOS will be loaded into memory.To use the dump a call to location E270 Hex is required, you can check the program from MOS by typing .. G E270<E>. From Basic you will need to add the following lines to a program;

```
CLEAR &E26F
CALL &E270
```

If this disc is used during a format operation then the modified DOS will be transfered to any other discs. If you want to transfer this DOS to another disc then insert disc: into drive 0 and;

1 .. MOS<E>
2 .. R 4000 6800<E>
3 .. Insert disc to be modified.
4 .. W 4000 6800<E>

Remember that using the W command from MOS writes directly to the disc so it is possible to get it wrong, make a backup of any important disc first, just in case. If you have a different type of printer to an Epsom or Tatung TP80/100, the parameters that set the printer into dot graphic mode may need changed, these start at location 0171 Hex in the above listing.

The next program creates a double size graphic dump, i.e. 2x2 pixells on the printer for each pixell on the screen. It is designed to run at location A000 Hex, so from Basic you will need to:

```
CLEAR &A000
LOAD "GDUMP2.OBJ"
CALL &A000
```

SOURCE LISTING FOR GDUMP2:

```
ORG 0A000H
LOAD 0A000H
ORGX:EQU 0FB9AH
ORGY:EQU 0FB9CH
AUXREG:EQU 20H
IN A,(AUXREG)
AND 1CH
CP 10H
RET NZ
LD HL,(ORGX)
PUSH HL
LD HL,(ORGY)
PUSH HL
LD HL,191
LD (ORGY),HL
LD HL,0
LD (ORGX),HL
PUSH HL
POP IX
PUSH HL
POP IY
GDUMP0:PUSH IX
LD B,9
LD HL,PARAMS
GDUMP1:LD A,(HL)
RST 8
DEFB 9FH
INC HL
DJNZ GDUMP1
GDUMP2:PUSH IY
LD C,1
GDUMP5:LD B,2
GDUMP3:PUSH BC
RST 8
DEFB 0C7H
POP BC
JR Z,GDUMP4
SCF
GDUMP4:RL C
DJNZ GDUMP3
DEC IY
JR NC,GDUMP5
LD A,C
RST 8
DEFB 9FH
RST 8
DEFB 9FH
POP IY
INC IX
PUSH IX
POP AF
OR A
JR Z,GDUMP2
POP IX
LD BC,0FFFCH
ADD IY,BC
LD A,0AH
RST 8
DEFB 9FH
PUSH IY
POP HL
LD BC,191
ADD HL,BC
JR C,GDUMP0
LD A,1BH
RST 8
DEFB 9FH
LD A,40H
RST 8
DEFB 9FH
POP HL
LD (ORGY),HL
POP HL
LD (ORGX),HL
RET
PARAMS:DEFB 0DH,1BH,41H,08H,1BH
DEFB 2AH,05H,00H,02H
END
```

GDUMP2 HEX DUMP:

```
0100  DB20E61CFE10C02A<E>
0108  9AFBE52A9CFBE521<E>
0110  BF00229CFB210000<E>
0118  229AFBE5DDE1E5FD<E>
0120  E1DDE506072177A0<E>
0128  7ECF9F2310FAFDE5<E>
0130  0E010602C5CFC7C1<E>
0138  280137CB1110F5FD<E>
0140  2B30EF79CF9FCF9F<E>
0148  FDE1DD23DDE5F1B7<E>
0150  28DCDDE101FCFFFD<E>
0158  093E0ACF9FFDE5E1<E>
0160  01BF000938BB3E1B<E>
0168  CF9F3E40CF9FE122<E>
0170  9CFBE1229AFBC90D<E>
0178  1B53393939390002.<E>
```

To enter the code from MOS;

```
1 .. MOS<E>
2 .. M 0100<E>
3 .. Enter the above code finishing with a full stop.
4 .. CTRL-BREAK to enter DOS.
5 .. SAVE 1 GDUMP2.OBJ
```

**ERRATUM:**

**Location 0124 should be;**
**09 not 07.**
**Locations 0178 onwards should read;**
**0178 1B41081B2A050002.<E>**

**The above code is for Epsom printers.**

**For Tatung TP80/100:**
**Location 0124 should be 08.**
**Locations 0178 onwards should read;**
**0178 1B41081B4B0002.<E>**

The SCRATCHPAD area of memory is used by the operating system to define certain parameters. It is initialised on power up by the Machine Operating System, (MOS), and is used by whatever language is in operation. The scratchpad area of memory starts from FB00Hex. Listed below are some of the more useful scratchpad locations, for a complete list and explanation refer to the Crystal Research book 'Albert Revealed'.

| | |
|---|---|
| FB00 | INTERRUPT VECTOR FOR CTC 0 |
| FB02 | INTERRUPT VECTOR FOR CTC 1 |
| FB04 | INTERRUPT VECTOR FOR CTC 3 |
| FB06 | INTERRUPT VECTOR FOR CTC 4, REAL TIME CLOCK. |
| FB08 | INTERRUPT VECTOR FOR KEYBOARD |
| FB0A | INTERRUPT VECTOR FOR A/D CONVERTOR. |
| FB0C | INTERRUPT VECTOR FOR FIRE BUTTON. |
| FB0E | INTERRUPT VECTOR FOR USER USE. |
| FB10 | INTERRUPT VECTOR FOR PRINTER. |
| FB12 | INTERRUPT VECTOR FOR PIO B. |
| FB14-FB16 | ROUTINE TO READ MEMORY FROM ROM. |
| FB17 | ROUTINE FOR THE END OF A 'GO' COMMAND FROM MOS. |
| FB1A-FB20 | ROUTINE TO BLOCK COPY, USED BY 'C' COMMAND FROM MOS, (UP). |
| FB21-FB27 | ROUTINE TO BLOCK COPY, USED BY 'C' COMMAND FROM MOS, (DOWN). |
| FB28-FB2F | ROUTINE FOR EXTERNAL CALL INTO MOS. |
| FB30 | BREAK VECTOR. |
| FB32 | COLD START VECTOR |
| FB34 | WARM START VECTOR. |
| FB36 | VDP MODE. |
| FB38 | TEXT COLOUR. MS NIBBLE-FOREGROUND, LS NIBBLE-BACKGROUND. |
| FB39 | GRAPHICS COLOUR, AS ABOVE. |
| FB3A | MOS FUNCTION TABLE VECTOR |
| FB3C | MOS INITIALISATION VECTOR |
| FB3E | ALPHA LOCK TOGGLE, UPPER CASE=136 LOWER CASE=8 |
| FB3F | CURSOR CHARACTER CODE. |
| FB40 | PROMPT CHARACTER CODE. |
| FB41 | CURSOR BLINK RATE. |
| FB42 | KEY REPEAT DELAY. |
| FB43 | KEY REPEAT SPEED. |
| FB44 | SECTOR SIZE IN 256 BYTE BLOCKS. |
| FB46 | CODE OF LAST KEY PRESSED. |
| FB4A | SCREEN COLUMN. (X) |
| FB4B | SCREEN ROW. (Y) |
| FB4F | LINE LENGTH. I.E. 32,40 OR 80. |
| FB50 | CURRENT DRIVE NUMBER. |
| FB51 | CURRENT TRACK NUMBER. |
| FB52 | CURRENT SECTOR NUMBER. |
| FB53 | BUFFER FOR DISC IN/OUT. |
| FB55 | FLAG TO INDICATE READ OR WRITE. |
| FB56 | ERROR STATUS VALUE. |

| | |
|---|---|
| FB57 | NUMBER OF TRIES ON READING DISC BEFORE SIGNALING AN ERROR. |
| FB58 | TRACK COUNT 0. |
| FB59 | TRACK COUNT 1. |
| FB5A | TRACK COUNT 2. |
| FB5B | TRACK COUNT 3. |
| FB5C | STORE FOR VALUE AT BREAKPOINT. |
| FB5D | LOCATION AT BREAKPOINT. |
| FB5F | VDP STATUS. |
| FB60-FB78 | THIS AREA IS USED TO STORE THE REGISTERS WHEN EXECUTING A PROGRAM. THE ORDER OF STORAGE IS: I,IX,IY,SP,AF',BC',DE',HL',AF,BC,DE,HL,PC. |
| FB79 | TEMPORARY STORE FOR HL. |
| FB7B | TEMPORARY STORE FOR SP. |
| FB7D-FB8B | DISC INFORMATION. |
| FB8C-FB91 | TIME STORAGE AREA. |
| FB92 | FLAG FOR INTERRUPTS. |
| FB93 | POINTER FOR FUNCTION KEY IN VIDEO RAM. |
| FB94 | POINTER TO FUNCTION KEY AREA IN VIDEO RAM. |
| FB96 | DESTINATION ADDRESS OF 'DRAW' TO X. |
| FB98 | DESTINATION ADDRESS OF 'DRAW' TO Y. |
| FB9A | X ORIGIN. |
| FB9C | Y ORIGIN. |
| FB9E | POLYGON CENTRE CO-ORDINATE X. |
| FBA0 | POLYGON CENTRE CO-ORDINATE Y. |
| FBA2 | ELLIPSE RADIUS X. |
| FBA4 | ELLIPSE RADIUS Y. |
| FBA6 | NUMBER OF SIDES OF POLYGON. |
| FBA8 | DOTON. |
| FBA9 | DOTOFF. |
| FBAA | DOTON2. |
| FBAB | DOTOFF2. |
| FBAC | STORE FOR DOT COUNT FOR DOTON/DOTOFF. |
| FBAD | FILL MODE, BACKGROUND=0 FOREGROUND=1. |
| FBAE | FILL STACK POINTER. |
| FBB0 | STEPPING RATE FOR DISC. 00-6mS, 01-12mS, 02-20mS, 03-30mS. |
| FBB1 | SIDE FLAG. SINGLE=0 DOUBLE=1 |
| FBB8-FBDF | INPUT BUFFER. |
| FBE0-FBFF | 32 BYTE BUFFER FOR COPYING TO AND FROM VDP. |

When writing BASIC programs it is sometimes useful to Peek or Poke some of the above Scratchpad areas in order to achieve results otherwise not available or for a reduction in the amount of code used. For instance FB3E contains the code for UPPER or lower case characters. Pokeing this location with 136 will enable UPPER case and pokeing it with 8 gives lower case characters. This does not however affect the Alpha lock light, this can be toggled with the XBAS command INP(&22).