TATUNG EINSTEIN

RELATIVELY BASIC

SOLO SOFTWARE

Introduction

This book has been written as a guide to understanding some of the lesser known aspects of programming the Tatung Einstein micro computer.

Crystal Research of Torquay have written a worthy basic interpreter for the Einstein and it is the intention of this book to amplify some basic instructions which one may feel require further explanation along with detailing areas of memory which can be altered for various effects.

It should be made clear, however, that this is not a rewrite of the excellent manuals which are supplied with the micro. Owner's manuals cannot supply every detail which may appeal to a wide range of owners, so this book is for those who have put one toe into the water and feel that the time is right to take the plunge and discover some of the little extras which the Einstein and XBAS contain. It commences with explanations of the graphic commands which, even if one has programmed in basic before, are better explained by example programs.

All the programs within this book are direct printouts from working programs and are believed to be correct, if you know otherwise the authors or publisher would be delighted to hear from you. For those readers who prefer not to type in the listings a disc of all the programs is available directly from G. M. Software, details will be found on the final page.

1

Ellipses and arcs

Ellipse is probably the best addition to the group of graphic commands on the Einstein. Try drawing circles on some other micro and see how long it takes.

Ellipse must always be followed by at least 3 parameters:-
The X co-ordinate of the centre
The Y co-ordinate of the centre
and the radius.

Enter this short program:-

```
10 BCOL 1:GCOL 15,0:CLS40:ORIGIN 0,0
20 ELLIPSE 100,100,50
```

and RUN <ENTER>

Notice the circle was drawn in an anti-clockwise direction starting from the 3 o,clock position.

This circle was absolutely round, it wasn't oval shaped.

There are a further 4 optional parameters which can be used with the ELLIPSE command for different results.

The fourth optional parameter defines the ratio between the X and Y axes. The screen is 256 pixels wide - X axis - against only 192 pixels high - Y axis - so if the fourth parameter is omitted the Einstein adjusts the Y axis accordingly by multiplying the given radius by 4/3 to formulate the correct Y radius. In the above program the radius was entered as 50, but it is only the X axis which will have this value. The Y axis will be adjusted by 50*4/3 which equals 66.6.

The resulting shape if the fourth parameter were omitted would be the same had it been entered as 1.333 - which is the sum of 4/3.

Had a value of 1 been entered the circle would appear thus:-

```
20 ELLIPSE 100,100,50,1
RUN <ENTER>
```

Decreasing the value of the fourth parameter will tend to flatten the circle even more.

```
20 ELLIPSE 100,100,0.5
```

However if the fourth parameter is larger than 1.33 the ellipse will grow in height.

```
20 ELLIPSE 100,100,50,1.75
```

Remember it is only the Y radius which is affected, the X radius remains at the given value.

The fifth, and also optional, parameter can have a value between 0 and 5 to denote which type of line is to be used for the ellipse. These are shown on page 70 of the Reference manual. Entering these lines will cause the program to erase the circle after a short delay:-

```
  20 ELLIPSE 100,100,50
  30 FOR A=1 TO 500:NEXT:BEEP
  40 ELLIPSE 100,100,50,,1
and RUN <ENTER>
```

Note the additional comma before the last figure in line 40. This signifies that we wish to add the fifth parameter but omit the fourth. This last parameter -1- caused the circle to be unplotted - erased - rather than drawn.

So much for complete ellipses or circles, but how would one go about drawing a semi-circle or arc?

This is where one would use the sixth and seventh parameters for ELLIPSE. But they must be entered as radians, so if you are acquainted with radians please skip over the next few paragraphs.

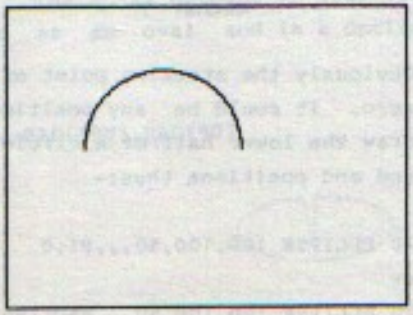
A full circle is made up of 360 degrees, so if one wanted to draw a semi-circle this would equal 180 degrees (360/2). The function which converts degrees to radians is RAD(n) where n represents the number of degrees.

Delete lines 30 and 40 by entering:-

```
DEL 30,40 ‹ENTER›
```

And alter line 20 to:-

```
20 ELLIPSE 100,100,50,,,0,RAD(180)
RUN ‹ENTER›
```



The top half of our original circle has been displayed. The semi-circle was drawn in an anti-clockwise manner starting from the 3 o,clock position round to the 9 o,clock position.

This means that 3 o,clock is equal to zero and 9 o,clock to RAD(180). Enter:-

```
PRINT RAD(180) <ENTER>
```

and 3.14159 has been displayed. Does this figure appear familiar? Enter:-

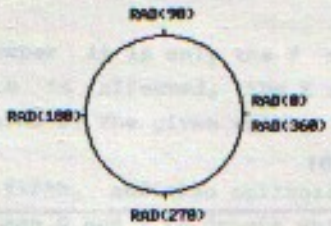
```
PRINT PI <ENTER>
```

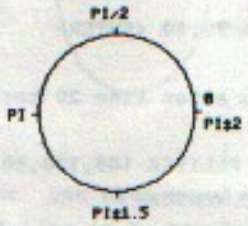
and the same value appears. 3.14159 is the radian value expressed by PI, which is universally used to define circles. Line 20 could just as easily have been entered thus:-

```
20 ELLIPSE 100,100,50,,,0,PI
```

These diagrans show some equivalent positions of a circle using the RAD or PI function. Note that the 3 o,clock position can be entered in 2 forms:

RAD(0) or RAD(360) or simply 0.
Or PI*2 or 0





Obviously the starting point of an arc does not need to be zero, it could be any position on the circumference. To draw the lower half of a circle one could swap the start and end positions thus:-

```
20 ELLIPSE 100,100,50,,,PI,0
```

or

```
20 ELLIPSE 100,100,50,,,RAD(180),0
```

In either line the last parameter - zero - could have been entered as RAD(360) or PI*2 which would still have drawn round to the 3 o,clock position.

To draw lines which connect the start and end positions to the centre - a fan shape - one only needs to enter either, or both, of the final parameters as negative values - preceeded with a - sign. Here are some examples:-

```
20 ELLIPSE 100,100,50,,,RAD(45),-RAD(90)
```

```
20 ELLIPSE 100,100,50,,,-RAD(100),RAD(160)
```

```
20 ELLIPSE 100,100,50,,,-RAD(100),RAD(90)
```

Although the above examples have omitted the 4th and 5th parameters for simplicity, there is nothing to stop one displaying the above shape as an oval and in a dotted line:-

```
20 ELLIPSE 100,100,50,0.5,2,-RAD(100),RAD(90)
```

POLY

This command must be followed by at least 4 parameters:-
The number of sides the polygon is to contain
The X co-ordinate of the centre
The Y co-ordinate of the centre
and the radius.

Apart from the additional first parameter these are similar to those used with ELLIPSE.

Enter this short program:-

```
10 BCOL 1:GCOL 15,0:CLS40:ORIGIN 0,0
20 POLY 5,100,100,50
```

and RUN ‹ENTER›

In fact the POLY command uses the same internal MOS function to display the shape as the ELLIPSE command. If one altered line 20 so that a 30 sided polygon was displayed the Einstein would draw almost the same shape as if one were using the ELLIPSE command, try entering:-

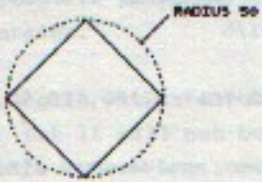
```
20 POLY 30,100,100,50
```

and RUN ‹ENTER›

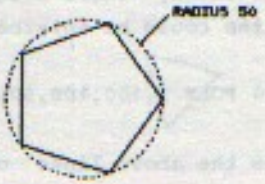
Looks almost the same as a circle, doesn't it? The 5th and 6th parameters are equivalent to the 4th and 5th of ELLIPSE. If the 5th parameter is omitted the ratio between the X and Y radii is adjusted to 4/3. And the 6th can be a value between 0 and 5 denoting which type of line should be used. It will be assumed to be a continuous line if the 6th parameter is omitted.

In the following diagrams the polygons have a dotted circle drawn with the same radius around the shapes showing how

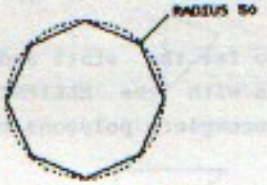
the points touch the radius. Each example only differs in the number of sides.




```
20 POLY 4,100,100,50
```




```
20 POLY 5,100,100,50
```




```
20 POLY 8,100,100,50
```

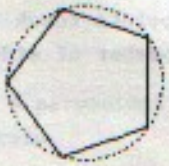
Notice that each polygon started from the 3 o,clock position. One can add 2 final parameters which denote the start and end angle, but as with ELLIPSE, these must be shown in radians.

In the last 8 sided example above one could specify PI/8 as both the start and end angles which would turn the polygon through 22.5 degrees.

PI is equal to 180 degrees therefore 180/8=22.5

```
20 POLY 8,100,100,50,,,PI/8,PI/8
```

The 5 sided polygon overleaf could be altered by changing the angles to PI/5



```
20 POLY 5,100,100,50,,,PI/5,PI/5
```

The angles can also be entered as degrees with the RAD(n) function, where n equals the angle in degrees. The above line could be entered as:-

```
20 POLY 5,100,100,50,,,RAD(36),RAD(36)
```

As the above lines omitted the 5th and 6th parameters - ratio and line type - commas were entered in their place.

So far the start and end angles have been the same. But, as with the ELLIPSE command, they can be altered for incomplete polygons to be displayed.

Enter:-

```
20 POLY 8,100,100,50,,,0,RAD(180)
```

The top half of an 8 sided polygon has been displayed.

This appears straight forward, the shape started at 0 and was drawn round to 180 degrees. But had the number of sides been 5 the resultant shape would be thus:-

```
20 POLY 5,100,100,50,,,0,RAD(180)
```

The polygon stopped drawing after it had reached 180 degrees, when the shape next touched the radius. So some thought must be applied when drawing incomplete polygons as to the angles and amount of sides it contains.

As with ELLIPSE prefixing one of the angles with a minus sign will draw 2 lines to the centre, but it will not be obvious which shape will be the result. Here are some examples:-

```
20 POLY 6,100,100,50,,,RAD(45),-RAD(90)
```

```
20 POLY 8,100,100,50,,,RAD(45),-RAD(90)
```

```
20 POLY 4,100,100,50,,,RAD(45),-RAD(315)
```

```
20 POLY 6,100,100,50,,,RAD(45),-RAD(315)
```

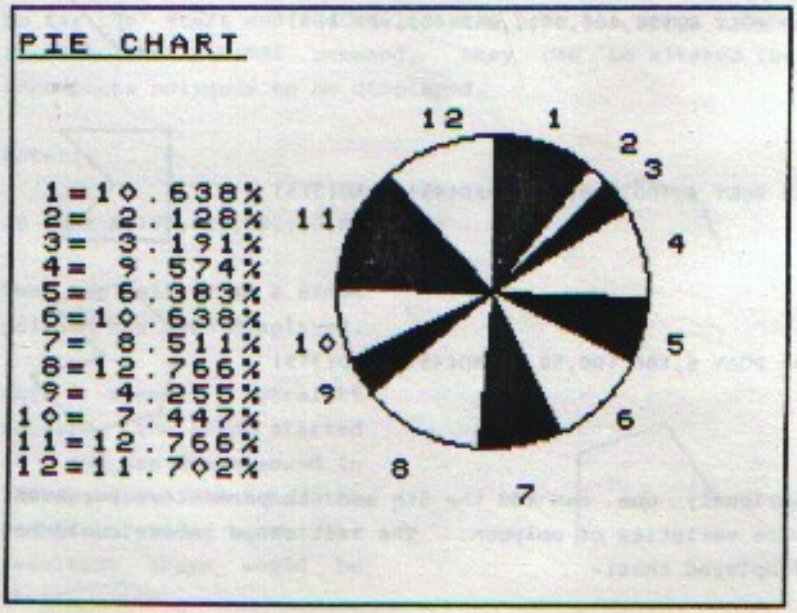
Obviously one can add the 5th and 6th parameters for even more varieties of polygon. The last shape above could be displayed thus:-

```
20 POLY 6,100,100,50,0.5,2,RAD(45),-RAD(315)
```

Pie chart program

The Pie chart is one method of displaying small amounts of data, where the whole circle represents the total and is broken up into sections which show the proportional size of each item in relation to that total.

This program also prints the percentage of each item on the left of the screen with the segment numbers colour matched to the numbers around the chart. The chart below contains 12 segments, but this can easily be altered, although if one increases the number of data items the chart could become overcrowded and lose legibility.



The Pie Chart is comprised of the fan shapes explained in the last chapter. Enter the lines in the order given, and once the program is up and running further lines will be added, so make sure all lines carry the correct line numbers.

Enter NEW <ENTER>

```
30 IOM 4,0: IOM 5,0: ORIGIN 0,0
```

After XBAS has loaded or the RST command has been used all IOM's are set to their initial state of 1. In the case of IOM 4 this signifies that any numeric output will be suffixed with a space and IOM 5 means that a numeric output will be prefixed with a space. As we want to suppress these spaces both IOM 4 and 5 are set to zero which will print the numbers without leading or trailing spaces. However, it should be pointed out that in other programs which deal with negative numbers, they will still print the minus sign before the number, so one may find that tabulated figures do not appear directly beneath each other.

```
40 CLS32: BCOL 1: GCOL 15,0: TCOL 5,0
50 PRINT "PIE CHART"
60 DRAW 0,183 TO 71,183
```

Line 40 sets up the display and line 50 prints the title message at the top left of the screen, because line 40 cleared the screen and positioned the cursor at that position. Line 60 simply draws a line beneath the title message.

```
70 DIM P(12)
80 FOR B=1 TO 12: READ P(B): T=T+P(B): NEXT
90 DATA 10,2,3,9,6,10,8,12,4,7,12,11
```

If your program is to use 10 or less items of data there is no need to DIM P(12), as in line 70, for the Einstein allows up to 10 subscripts without the necessity to DIMension. However it is a good habit to dimension whenever subscripted variables are to be used. The FOR-NEXT loop reads the 12 items of data into variables P(1) to P(12). Subscripted variables start at zero - one could use P(0) as the first subscript - but in this program it is simpler to use 1 to 12. As each item of data is read it is added to variable T which, after the loop has finished, will contain the total of all the values in line 90 - in the above case it will equal 94. It was not necessary to set T to zero before encountering these lines as whenever a program is RUN all variables are set to zero.

```
100 X=160: Y=100
110 S=PI*2.5
```

Line 100 sets up the centre X and Y co-ordinates for the centre point of the chart and line 110 sets the start angle to PI*2.5 We require the first segment of the chart to start from the 12 o,clock position and each further segment be added in a clockwise direction. This is alien to how the Einstein wants to draw them - remember in the ELLIPSE section all drawing was anti-clockwise. So why have we entered PI*2.5 when the 12 o,clock position is PI/2?

When drawing fans - that's arcs with 2 lines drawn to the centre point - we prefix either the start or end position with a minus sign. Travelling anti-clockwise, as we are going to be, if we deducted the start angle from PI/2 as soon as a segment reached the 3 o,clock position our angle would be less than zero. So instead of small segments being drawn one would find that the remainder of the circle was displayed with our segment cut out of it. By adding a complete revolution - PI*2 - to the normal 12 o,clock position of PI/2 this gives us PI*2.5 and the segments

would require a complete circle plus a quarter before our angle reached zero. Once the program is running try altering line 110 to PI/2 and see the result. Incidentally one could use the RAD(n) function instead of PI in line 110 which should be entered as RAD(450) - a complete circle plus a quarter 360+90=450.

```
120 FOR B=1 TO 12
130 E=S
```

Line 120 starts the loop which will display the 12 segments. Line 130 loads variable E - the end angle - to variable S - the start angle. As the Einstein draws the segments in an anti-clockwise manner the start point must be decreased for each new segment while the end point takes on the value of the previous start position. The first time through the loop the first segment will end at the 12 o,clock position - variable E - while the variable S - the start point - is calculated in line 140.

```
140 S=S-(P(B)/T)*RAD(360)
```

Line 140 calculates the size of the segment in relation to the whole circle. Variable P(B), which on the first time round the loop is actually P(1), will contain the value of the first item of data - 10 in line 90. This value is divided by the total of the data values held in variable T - P(B)/T - which in this instance is 10/94.

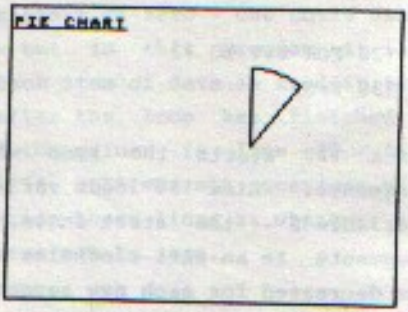
This sum is then multiplied by the number of degrees in a complete circle, but as the angles must be in radians it is converted by RAD(360).

As we are displaying the segments in a clockwise fashion this figure is deducted from variable S which now holds the radian value of the start of the segment.

```
150 ELLIPSE X,Y,50,,,-S,E
```

Line 150 draws the segment. Note that the start angle is prefixed with the minus sign signifying that 2 lines to the centre should be drawn.

Although the program is not complete it should now be capable of displaying the first segment. So RUN the program and compared the results with the diagram.



If it is not the same check the entries so far.

```
210 FMT 2,0
220 PRINT @ 0,B+6; B; "=";
```

Line 210 alters the numeric output format command so that our segment numbers - 1 to 12 - will be aligned correctly when printed on consecutive lines beneath each other. As our numbers are all integers - whole numbers - we do not require any places after the decimal point, so the second parameter is set to 0.

Line 220 prints the value of our loop counter, which also represents the segment number, at column zero, the leftmost column, on lines B+6. This ensures that the first time through the loop printing will begin on line 7, then 8 etc. The value of B is followed by the = sign. Do not forget to end line 220 with a semi-colon.

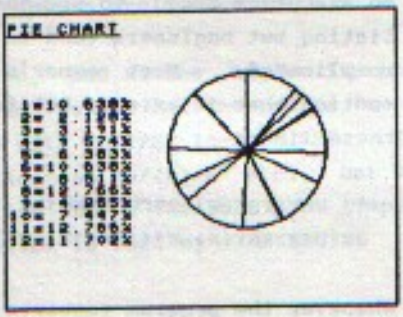
```
230 TCOL 15,0: FMT 2,3
240 PRINT P(B)/T*100;"%"
250 NEXT
```

Line 230 sets the text colour for printing out the percentages to white. FMT 2,3 alters the format command to print with 2 places before, and 3 places after, the decimal point.

Line 240 takes the value of P(B), which on the first time through the loop will equal 10 - the first item in data line 90 - and divides it by the total - variable T - which we know equals 94. This gives a result of 0.106383. Multiplying this figure by 100 will return the percentage value of the data item - 10.6383 - and as the second parameter of FMT was 3 it will be displayed as 10.638 with 3 figures after the decimal point.

Line 240 then prints the % sign and line 250 loops back for the remainder of the items.

Before adding any further lines check on the results so far. It should be similar to this diagram with the PIE CHART message and the number 1 below printed in light blue with the remainder of the display in white.



We now want to fill alternative segments with colour and display each segment number around the circumference of the chart.

To use the FILL command we will need to find a point within a segment which is surrounded by a boundary, or the colour will spill out and possibly flood the screen.

There is a mathematical formula for determining any point on a circle:-

X co-ordinate = Radius*Cosine(Angle) + centre X position
Y co-ordinate = Radius*Sine(Angle) + centre Y position

Now without delving too far into trigonometry, which this book really is not about, suffice to say that the formulae above can be simply adapted to a basic program line without a vast mathematical knowledge, by using the 2 basic functions for cosine and sine - COS and SIN. The Angle should be entered in radians, but as our program already uses them for the ELLIPSE command, this is no problem.

To assist in calculating the correct positions we will use the DEF FN statement. Whenever several program lines contain the same formula it is more efficient, and faster on execution speed, to use the DEFine FuNction early in the listing but beginners tend to avoid it as it appears too complicated. Most owner's reference manuals briefly mention that it exists, but few show working examples. Add these lines:-

```
10 DEF FNX(R)=R*COS(ANG)+X
20 DEF FNY(R)=R*4/3*SIN(ANG)+Y
```

Whenever the program requires the X co-ordinate of a point around the circle all the function requires is that variable ANG has been set to the correct angle, and that the radius is also known. The X latched onto the end of the line is the centre X co-ordinate, which already has a value in line 100.

Line 20 is similar except that the radius is multiplied by 4/3. Remembering that the ELLIPSE statement automatically adjusts the Y radius by this amount, we must adjust our radius of Y similarly.

```
160 ANG=S+(P(B)/(T*2))*RAD(360)
```

This line divides the segment value - in P(B) - by twice the total - T*2. In effect this results in a figure of half the segment's size. By adding this figure to the start position, not deducting as we did in line 140, it will move the angle anti-clockwise by half the size of the last segment. Once again it is converted to radians by RAD(360). And now ANG is set to a central point within the last segment.

```
170 IF B MOD 2=0 THEN 190
180 FILL FNX(45),FNY(45),15
```

As only alternate segments are to be filled in with colour line 170 tests if our loop counter is equal to a multiple of 2, and if it is then the fill line - 180 - is skipped. i.e. only segments with odd numbers are filled-in.

Line 180 uses the FILL command, and as the angle is now known all that is required is to set the radius to a smaller figure than was used with ELLIPSE in line 150. In line 10 we defined the functions as FNX(R)=R* etc., but we do not need to use the function with a variable R. Simply set the radius when the function is used by - FNX(45).

```
190 TCOL B+1,0: FMT 0,0
200 PRINT @ FNX(62)/8,(192-FNY(62))/8; B
```

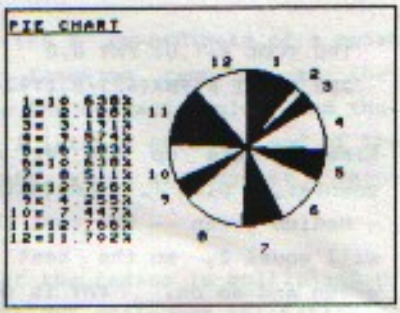
Line 190 sets up the text colour to the value of the loop counter, B plus 1. So initially the text colour will be 2 - medium green - the next time through the loop variable B will equal 2, so the text colour will be 2+1=3 - light green and so on. FMT is set to normal for displaying the segment numbers around the chart. Line 200 cunningly positions these numbers.

Using FNX(62) will return the X co-ordinate of a pixel position outside the circle but lined-up with the centre of the last segment drawn as line 160 has already established the angle. As the ORIGIN was set to 0,0 in line 30 if we now divide this figure by 8 it will return an actual column position for using PRINT @. Think about it, print column 0 starts at pixel position 0, column 1 at 8, column 2 at 16 etc.

Although the figure may run into decimal places - i.e. if FNX(62)=210 and the this figure is then divided by 8, the result would be 26.25 - this would not matter, as the PRINT @ statement only looks at the integer part of a value and would print at column 26.

To calculate which line, from the top, to print at we must first deduct the value in FNY(62) from 191, the top-most pixel position. For although line numbers are measured from the top of the screen the pixel Y co-ordinates are measured from the bottom. If FNY(62)=175 - which is near the top of the screen - and we deduct this from 191 we are left with 16. Divide this figure by 8 and the result is line 2. After calculating the relevant position line 200 prints the segment number in variable B.

The program should now also display the segment numbers down the left of the screen in the same colours as were used for around the chart.



This program can easily be altered so that the data items can be input from the keyboard, but it should be remembered that all the data must be input before the program starts to draw the chart. To achieve this lines 70 to 90 should be deleted by entering:-

```
DEL 70,90 <ENTER>
```

Now add these lines:-

```
41 INPUT "Enter no. of items ";N
42 DIM P(N)
43 FOR B=1 TO N
44 PRINT "Enter data item ";B;: INPUT P(B)
45 T=T+P(B): NEXT: CLS
```

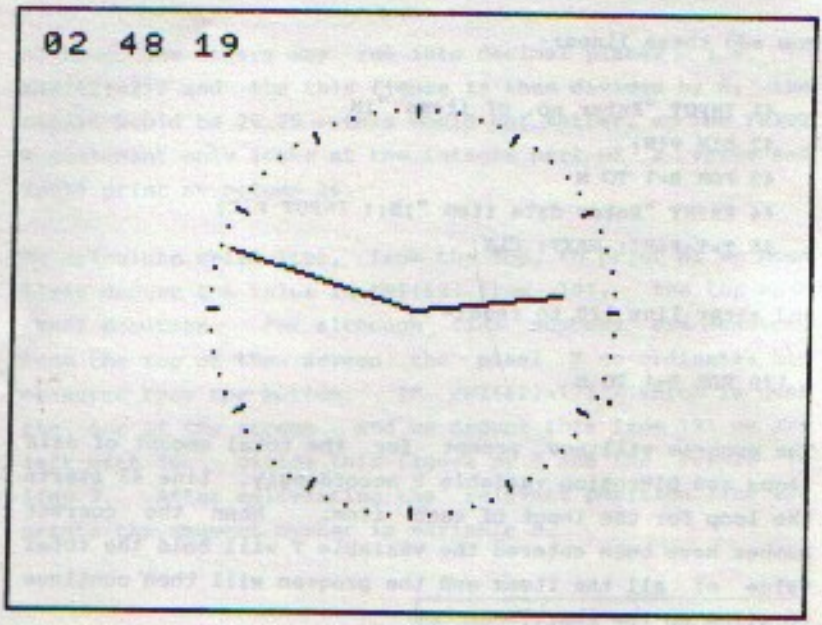
And alter line 120 to read:-

```
120 FOR B=1 TO N
```

The program will now prompt for the total amount of data items and DIMension variable P accordingly. Line 43 starts the loop for the input of each item. When the correct number have been entered the variable T will hold the total value of all the items and the program will then continue to slice up the chart.

Clock program

Although there is an excellent alarm clock program on the demonstration disc, this listing is designed to show how a simple working clock face could be designed.



Firstly we will move the ORIGIN from the bottom left position to the centre of the screen. This means that, unlike the previous example, there will be no need to add the centre position to the end of our formulae. Hence any point on the circumference of the clock face can be decided by:-

```
X = Radius * Cosine(Angle)
Y = Radius * Sine(Angle)
```

Program lines will be explained as they are entered. The functions for plotting the clock face require setting up.

Enter NEW <ENTER>

```
10 ORIGIN 128,96
20 DEF FNX(R)=R*COS(PI/2-RAD(A)*6)
30 DEF FNY(R)=R*4/3*SIN(PI/2-RAD(A)*6)
```

Before using the functions variable A - angle - will need to be set up, but the angle looks different here than in our previous example. As one hour is comprised of 60 minutes we will only require the minute hand to make 60 movements in one complete circle. As one complete circle contains 360 degrees the minute will be multiplied by 6 to decide on the correct angle (60*6=360).

The other difference in the angle section of the argument is PI/2. We want all timings to be made from the 12 o,clock position, which if one refers to the chart in the previous section, is equal to PI/2. We also want the hands to be drawn in a clockwise manner, so the angle - RAD(A)*6 - is deducted from this 12 o,clock position. If the sign was altered from - to + the clock would go backwards, which is fine if you wish to view through a mirror!

```
60 CLS32: MAG1: BCOL 4: TCOL 15,0: GCOL 15,0
70 SHAPE 130,"80 00 00 00 00 00 00 00"
80 SPRITE 0,129,100,15,130
```

Line 60 sets up the screen display and colours, which have been left to white on dark blue, but can obviously be changed once the program has been entered and, dare I say it, debugged and working. Line 70 defines the shape number 130 for the sprite which will rotate around the circumference of the clock face indicating the seconds. Line 80 sets sprite number 0 as being white and allocates

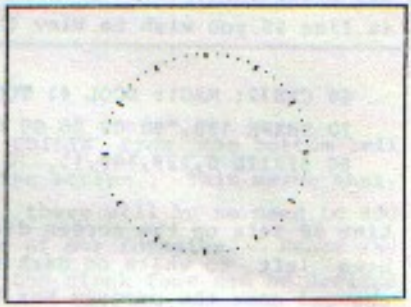
it to shape 130. This line also positions the sprite off screen until the program is ready to use it.

```
  90 FOR A=1 TO 60
 100 IF A MOD 5=0 THEN DRAW FNX(65),FNY(65) TO FNX(62),
FNY(62): GOTO 120
 110 PLOT FNX(65),FNY(65)
 120 NEXT A
```

Line 90 sets up the loop to plot the 60 minute marks around the face. Line 100 checks that the minutes are equal to multiples of 5 with the A MOD 5=0, in which case a line is drawn from radii 65 to 62, otherwise a single pixel is plotted in line 110 for the other minutes around the face.

Notice that here we have not used a variable for the radius. As our DEFined FuNctions were entered as DEF FNX(R)=R* etc., this means that the radius - to the right of the equals sign - will equal the value within the parentheses to the left of the equals sign - FNX(60) would calculate the X co-ordinate to the radius of 60. So to draw a line from radius 65 to radius 62 we simply enter - DRAW FNX(65),FNY(65) TO FNX(62),FNY(62).

At this stage the program can be RUN and checked that the display is similar to that on the right.



If it isn't then re-check the DEF FN lines 20-30 first before checking the remainder.

It is essential that these lines are correct as the main part of the following program relies on them when the minute hand or sprite are moved.

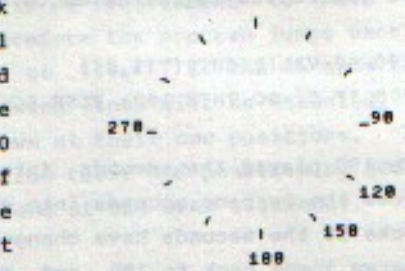
The co-ordinates for the hour hand require defining. As an hour hand does not only point to the exact hour, it gradually moves between each hour mark depending on the number of minutes the time is past the last hour, it is defined slightly differently to the minutes.

```
40 DEF FNXH(R)=R*COS(PI/2-RAD(HR)*30-(RAD(A)/2))
50 DEF FNYH(R)=R*4/3*SIN(PI/2-RAD(HR)*30-(RAD(A)/2))
```

Here the hour angle is defined as RAD(HR)*30 as there are 12 hours in one revolution of the clock, the hour must be multiplied by 30 (360/12=30).

Therefore each hour mark is 30 degrees higher than the last. As with the minutes in lines 20-30 this figure is deducted from PI/2 giving the correct angle past 12 o,clock.

Say the time is 4 o,clock the variable HR will contain 4, so 4*30=120 and as can be seen from the diagram on the right 120 equals 4 o,clock. But if the time is 4.30 variable HR will still contain 4 but we want the hour hand to be pointing to a point mid-way between the 4 and 5 as a normal clock would do.



We are seeking an angle figure of 135 - halfway between 120 and 150. Therefore we divide the minutes - in variable A - by 2 which in the case of 4.30 would be 30/2=15. Remember that we are deducting this angle from PI/2, so -120-15= -135 which is the correct angle for the hour hand past 12 o,clock.

```
130 HR=VAL(LEFT$(TI$,2)) MOD 12
140 MN=VAL(MID$(TI$,3,2))
150 SC=VAL(RIGHT$(TI$,2))
```

These lines take sections of the system variable for time -TI$- which always consists of 6 characters, and places them into the 3 variables. Hours are always held in the first 2 characters in TI$, minutes in the centre 2 and seconds in the rightmost pair.

```
160 A=MN
170 DRAW 0,0 TO FNXH(48),FNYH(48): REM DRAW HOUR HAND
180 DRAW 0,0 TO FNX(62),FNY(62): REM DRAW MIN HAND
```

As explained earlier each time we use the DEFined FuNctions we must give a value for the angle in variable A. As the hour hand is smaller than the minute hand the radius is subsequently smaller - line 170 - but to draw the minute hand the radius is increased to 62 in line 180.

```
190 S2=VAL(RIGHT$(TI$,2))
200 IF S2=SC THEN 190: ELSE SC=S2
```

Line 150 placed the seconds into variable SC. Line 190 places the current seconds into variable S2, while line 200 checks if the seconds have changed. If they haven't the program jumps back to 190 and reads the seconds again. This process recurs until the seconds have changed and then the remainder of line 200 is encountered and SC now equals the new value of the seconds.

```
210 A=SC
220 SPRITE 0,FNX(68),FNY(68)
```

Once again we are going to use one of the functions, only this time it is to position the sprite for pointing to the seconds. The angle - variable A - is set to SC which holds

the current seconds value and our DEFined FuNction will move the sprite on the radius of 68. As we have already defined the shape and colour of sprite 0 in line 80 all that is required to move the sprite is to enter the sprite number and X and Y co-ordinates - line 220.

```
230 PRINT@ 0,0;
240 FOR Z=1 TO 5 STEP 2
250 PRINT MID$(TI$,Z,2);" ";:NEXT
```

Line 230 positions the cursor at the top left corner of the screen. Lines 240-250 disect TI$ into 3 pairs of numbers and prints each with a space following. This is updated each time the seconds change.

```
260 IF SC<>0 THEN GOTO 190
```

If the seconds are not equal to zero - SC equals a value between 1 and 59 - there is no requirement to move the minute and hour hands, therefore the program jumps back to the seconds reading line at 190. If the seconds have reached zero then the following lines will be executed, for the hands need to be re-drawn at their new positions. But first they must be erased from their old position by adding the qualifier - 1 - to the end of the DRAW statement.

```
270 A=MN
280 DRAW 0,0 TO FNXH(48),FNYH(48),1: REM ERASE HOUR HAND
290 DRAW 0,0 TO FNX(62),FNY(62),1: REM ERASE MIN HAND
300 GOTO 130
```

Before jumping back to line 130 the hands must be erased. Remember that although the seconds have reached zero, and the minute section of TI$ must have changed also, but the program has not read the minute section of TI$, only the rightmost 2 figures for the seconds. Therefore variable MN still contains the previous value of minutes and

subsequently both hands can be erased using MN - line 270. Once the program jumps back to line 130 variable MN is set to the new minute and the hands are drawn in lines 170-180. And time continues....

The program can have an input routine for the time added quite easily. Here is a suggested method:-

```
61 PRINT "Enter time as HHMM i.e. 0430"
62 INPUT T$: IF LEN(T$)<>4 THEN 61
63 IF VAL(LEFT$(T$,2))>12 THEN 61
64 IF VAL(RIGHT$(T$,2))>59 THEN 61
65 TI$=T$+"00": CLS
```

This input routine does not allow for the input of seconds, line 65 simply adds "00" to the hours and minutes so TI$ will contain 6 characters.

The system variable TI$ is peculiar as it does not internally check on a valid time. For instance one could input a time as 1175. Adding 75 minutes to 11 o,clock results in a time of 12.15 which is what the hands on the clock would point to, but when the program prints out the value of TI$ it would be displayed as 117500. It will also accept hours greater than 24, entering a time as 2530 would cause the hands to point to 1.30 but TI$ would still get printed as 253000. Therefore a check on both hours and minutes is carried out in lines 63 and 64.

Biorhythm program

It is said that from the day we are born our physical, emotional and intellectual states conform to certain cycles, called Biorhythms.

Each cycle repeats at a rate of 23, 28 and 33 days respectively, meaning that occasionally one will find all three will peak together, or the opposite may happen. Whether you believe in them or not, this program shows off the chart plotting capabilities of the Einstein.

A typical screen display:-

The program is listed first followed by explanatory notes.

```
 10 RST: BCOL 1: GCOL 15,0: TCOL 15,0
 20 CLS40: X=0: T=0: MAG 1
 30 DATA B,I,O,R,H,Y,T,H,M
 40 FOR A= 190 TO 30 STEP-20
 50 READ CH$
 60 X=X+1
 70 SPRITE X,X*22,A,X+1,ASC(CH$)
 80 SPRITE X+9,X*22,220-A,X+1,ASC(CH$)
 90 FOR Y=1 TO 150: NEXT
100 NEXT A
110 FOR Y=1 TO 9
120 SPRITE OFF Y: SPRITE OFF Y+9
130 FOR A=1 TO 100: NEXT
140 NEXT Y
150 T=T+1: IF T=4 THEN 160 ELSE RESTORE: X=0: GOTO 40
160 SPRITE OFF
170 DEF FNY(Y)=INT(72*SIN((-Z+A)*(PI*2)/CD)+.5)+Y
180 D1$="          1 1 1 1 1 2 2 2 2 2 3"
190 D2$="1 3 5 7 9 1 3 5 7 9 1 3 5 7 9 1"
200 DIM MNTH$(12),DAY(12)
210 FOR A=1 TO 12: READ MNTH$(A),DAY(A): NEXT
220 DATA JAN,31,FEB,28,MAR,31,APR,30,MAY,31,JUN,30
230 DATA JUL,31,AUG,31,SEP,30,OCT,31,NOV,30,DEC,31
240 INPUT"Enter Name..";NAME$
250 INPUT"Year Born (i.e.1969)..";YBN
260 IF YBN<1901 THEN PRINT "Sorry cannot do": GOTO 240
270 INPUT"Month     (1 to 12)...";MBN
280 IF MBN<1 OR MBN>12 THEN 270
290 INPUT"Day       (1 to 31)...";DBN
300 INPUT"Year to start.........";YST
310 IF YST<YBN OR YST<1901 THEN PRINT "Sorry cannot do"
: GOTO 300
320 INPUT"Month.................";MST
330 IF MST<1 OR MST>12 THEN 320
340 YR=YBN: MN=MBN: DY=DBN: GOSUB 760
350 IF ER=1 THEN 250
360 GOSUB 800: DE=F: Y1=YBN: M1=MBN: D1=DBN
370 YR=YST: MN=MST: DY=1
```

```
  380 GOSUB 800: H=F
  390 IF MN<3 AND YR MOD 4=0 AND Y1 MOD 4<>0 THEN DE=DE+1
  400 IF MN>2 AND YR MOD 4=0 AND Y1 MOD 4=0 THEN DE=DE-1
  410 IF YR MOD 4<>0 AND Y1 MOD 4=0 THEN DE=DE-1
  420 GCOL 15,0: TCOL 15,0: N=DE-H: IF N>0 THEN GOSUB 790:
 GOTO 250
  430 PH=23*(N/23-INT(N/23)): EM=28*(N/28-INT(N/28))
  440 IT=33*(N/33-INT(N/33))
  450 L=32-LEN(MNTH$(M1)+STR$(D1)+STR$(Y1))
  460 CLS: TCOL 1,15: PRINT "Name: ";NAME$;TAB(L);"BORN: "
;MNTH$(M1);D1;Y1
  470 ORIGIN 0,0: TCOL 13,0: PRINT TAB(16);"High"
  480 DRAW 3,32 TO 207,32 TO 207,176 TO 3,176 TO 3,32
  490 FOR X=9 TO 208 STEP 12
  500 DRAW X,176 TO X,174: DRAW X,32 TO X,34
  510 DRAW X+6,176 TO X+6,32
  520 NEXT X
  530 DRAW 204,104 TO 3,104,2
  540 TCOL 12,0: PRINT @ 0,20;MNTH$(MN);" ";STR$(YR);
  550 NXMN=MN+1: IF NXMN=13 THEN NXMN=1
  560 PRINT TAB(32);MNTH$(NXMN)
  570 PRINT @ 0,21,LEFT$(D1$,DAY(MN))
  580 PRINT LEFT$(D2$,DAY(MN));LEFT$(D2$,(35-DAY(MN)))
  590 TCOL 4,0: PRINT @ 16,20;"Low"
  600 Z=PH: CD=23: COL=9: CYCL$="Phys": GOSUB 670
  610 Z=EM: CD=28: COL=11: CYCL$="Emot": GOSUB 670
  620 Z=IT: CD=33: COL=13: CYCL$="Intel": GOSUB 670
  630 TCOL 1,15: PRINT @ 5,23;"Zero quits..Any other for n
ext";CHR$(11): BEEP
  640 A=INCH: IF A=48 THEN END
  650 MN=NXMN: IF MN=1 THEN YR=YR+1
  660 GOSUB 800: H=F: GOTO 420
  670 ORIGIN 3,0: Y=104: A= -0.5: DOKE &FB96,0,FNY(Y)
  680 GCOL COL,0
  690 FOR A=0 TO 33.5 STEP .5
  700 DRAW TO A*6+3,FNY(Y)
  710 NEXT A
  720 TCOL COL,0: YY=INT(24-(DEEK(&FB98)/8)): XX=35
```

```
730 IF PEEK(PTR(19)+(YY*40+XX))<>32 THEN YY=YY+1:GOTO730
740 PRINT @ XX,YY,CYCL$
750 RETURN
760 ER=0: IF D<DAY(MN) THEN RETURN
770 IF MN<>2 THEN 790
780 IF YR MOD 4=0 AND DY=29 THEN RETURN
790 BEEP: CLS: PRINT "INCORRECT DATE": ER=1: RETURN
800 F=0: IF MN<2 THEN 840
810 FOR A=1 TO MN-1
820 F=F+DAY(A): NEXT A
830 IF YR MOD 4=0 THEN F=F+1: IF MN=2 THEN DAY(2)=29
840 F=F+365*YR+INT(YR/4)+DY: RETURN
```

Lines 10-160 set up the screen and display the word 'BIORHYTHM' along the 2 diagonals of the screen. The word is comprised of sprites for each letter.

Line 170 sets up a function - FNY(Y) - as the equation to find the Y co-ordinate to draw to for each cycle. The X co-ordinate does not require such a function as it increased by the same number of positions equally for each day of the month across the chart.

Lines 180-190 set up string variables D1$ and D2$ with the days of each month for printing below the chart. Make sure the correct spaces are entered into the strings.

Lines 200-230 load up the dimensioned variables MNTH$ and DAY with the names of of the months of the year and number of days in each.

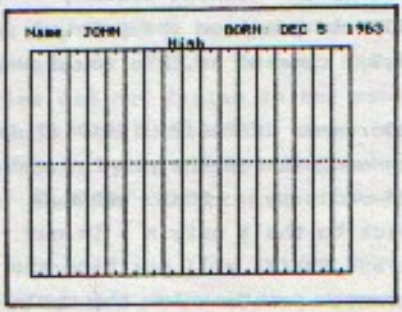
Lines 240-330 check and accept the entry of details from the subject, and trap any subsequent errors of dates etc.

Lines 340-440 calculate the number of days that have elapsed, adding days for leap years.

Lines 450-460 print the subject's details at top of screen.

Line 480 draws the outside of the chart and lines 490-520 draw the vertical day lines across the width of the chart. Line 530 draws the dotted line across the centre, note that DRAW contains a third parameter - 2 - indicating a dotted line.

If one runs the program before entering the remainder of the listing it should present a display similar to the diagram right, although the subject's name and details should be shown as black on a white background.



Lines 540-580 display the current month and days of that month with D1$ and D2$ by using the LEFT$ function and DAY(MN) for the number of days it contains. Depending on the length, the remainder of the line prints the dates of the following month.

Lines 600-620 represent the start of each cycle and load variable CD with the cycle day length and the colour of the line into variable COL. CYCL$ is loaded with the name of each cycle for printing to the right of the chart, where the line ends, before calling the drawing subroutine at line 670.

Lines 630-660 checks for input of zero for ending the program, for if it isn't the following month's chart is displayed.

Lines 670-750 draw each cycle. The X origin is moved by 3 pixels in line 670 to enable the dates to be lined up centrally beneath. Each cycle will be drawn in 68 stages

across the chart. Each time a section is drawn we simply instruct it to DRAW TO X,Y - line 700 - from whatever position the graphics pointer happens to be at. But firstly the graphics pointer must be positioned on the leftmost line of the chart. As there is no basic command for move - without drawing - we must resort to a double POKE to position the pointer. We cannot use the DRAW TO X,Y,1 command as this would unplot any pixels.

Addresses &FB96/7 (64406/7 decimal) contain the current X co-ordinate of the pointer and &FB98/9 (64408/9) hold the Y co-ordinate. DOKE &FB96,0 would position the X pointer back to the X origin - in our case position 3 - and DOKE &FB98,FNY(Y) will position the Y co-ordinate pointer at the correct position for the cycle scale to begin from relative to the centre horizontal dotted line - although one must set Y to 104 first. Although the program uses hexadecimal numbers, one could just as easily have used decimal and entered line 670 thus:-

```
670 ORIGIN 3,0: Y=104: A= -0.5: DOKE 64406,0,FNY(Y)
```

When each cycle has been drawn we want to print the name of that cycle - CYCL$ - adjacent to where the line finished, using the PRINT @ X,Y command. The X co-ordinate - column - will always be the same - 35, but the Y co-ordinate will depend on the final position of the cycle drawn. Line 720 DEEKs the current graphics Y co-ordinate at address &FB98 and divides this figure by 8. It must also subtract this value from 24 to find the correct line number. Why? Because lines are numbered from the top - 0 - to the bottom - 24 - but the graphics pointer operates in reverse with the top pixel line being 191 and the bottom 0.

Hence if DEEK(&FB98) gave a value of 191 this would mean the graphics pointer was at the top of the screen. Dividing this figure by 8 would result in the answer

23.875. Deduct this figure from 24 and the result is 0.125 and as PRINT @ only takes the integer part of a value this would result in PRINT @ 35,0.
Note This only holds true if the origin for the Y co-ordinate is zero - bottom of screen.

We now have our co-ordinates for PRINT @ in variables XX and YY. Before printing the string CYCL$ we must check that one of the previous cycles did not finish in the same position - which is common enough - as this would mean we would be printing over a previous cycle name and erasing it. So a check must be made that no character is already printed at that position.

PTR(19) holds the RAM address of the start of the screen. Each character position on screen is mapped to a block of consecutive memory locations, so if one knows the position it can be PEEKed to discover which ASCII character it contains. Line 730 calculates the screen position we intend to print to by multiplying the line - in YY - by 40 and then adding the column number - in XX. This sum is added to PTR(19) and then PEEKed to check if it contains a space - ASCII code 32 decimal - and if it doesn't the line - in YY - is increased and a further check made until a space is found. Line 740 prints the cycle name held in string variable CYCL$.

The subroutine in lines 760-790 check on valid dates during the entry routine and lines 800-840 calculate the elapsed days up till the current month.

2

Function keys

After loading XBAS the function key buffer is empty, so pressing any of the 8 grey function keys will have no effect. These keys can be very useful when writing or debugging programs. This section shows how to assign various commands to these function keys, and once the general concept has been shown one will obviously be able to use any commands with the keys.

Load XBAS as normal and enter this short program:-

```
10 FOR X=1 TO 10
20 PRINT X
30 NEXT
```

Running this program will simply print the numbers 1 to 10 on consecutive lines. It will only be used to demonstrate the effects that the function keys can produce when assigned various commands. To assign the first grey function key to the command RUN one would enter the following:-

```
KEY 0,"RUN" <ENTER>
```

The keyword RUN was entered as a string, with quotes either side. Pressing the function key marked F0 will now cause the word RUN to be displayed on screen, as if the 3 letters R, U and N had been entered singly from the keyboard, but one will still need to press the red ENTER key for the command to be acted upon. We require to add the ASCII code for ENTER onto the end of the keyword RUN. There are 2 methods of achieving this.

The Reference manual states that one may press both the GRAPH and ENTER keys after the last letter of RUN and before the closing quotation marks. This is fine providing one doesn't press the GRAPH and INS keys by mistake which would send the Einstien into MOS. An alternative, and safer, method would be to add the ASCII code for ENTER to the end of RUN after the closing quotes:-

```
KEY 0,"RUN"+CHR$(13) <ENTER>
```

ASCII codes below 32 decimal are used as control codes, and when the Einstein encounters an ASCII character with a value below 32 it knows that it must perform an action other than simply printing a character to the screen. These control codes are listed on page 206/207 of the Introduction manual. Unfortunately they are only shown in hexadecimal but one will see that code 0C which is the hexadecimal equivalent of 13 is stated as CR - carriage return.

Carriage return is an alternative way of stating Enter, so adding CHR$(13) to RUN will simply print RUN on screen and the Einstein looks to the next character (13) and acts upon it, just as if one had pressed the ENTER key.

Other control codes can be added to these commands for useful results. As an example assume a program was listed on the screen, and one had just edited a line and the cursor was currently positioned to the left of the next program line. Pressing the F0 function key would display RUN, but an error would be caused due to there being other characters on that screen line. Control code 15 hex (21 decimal) causes the remainder of text on the current screen line to be ignored, hence the keyword RUN would be acted upon without error. This could be entered as:-

```
KEY 0,"RUN"+CHR$(21)+CHR$(13)
```

note that the code 13 - carriage return - is always entered last.

Function key 1 could be defined to list a program, and by adding CHR$(13) to the end will list a program fron the first line by only requiring key F1 to be pressed.

```
KEY 1,"LIST"+CHR$(13)
```

The third key could be defined to the LOAD" command. Here we wish the opening quotes to be produced on screen, so that after F2 has been pressed all that will be needed is for the program name to be entered. The only method of printing quotes on screen is by using the CHR$ function with the ASCII code for the quote marks - 34 decimal. For instance one could not enter "LOAD"". This could be entered thus:-

```
KEY 2,"LOAD"+CHR$(34)
```

Pressing key F2 will now display:-

```
LOAD"
```

and the cursor will be flashing next to the quotes, on the same screen line, waiting for one to enter the program name. Here though one will still need to press the ENTER key after the program name.

Although one can clear the screen by entering the CTRL and L keys simultaneously, it is also preferably to achieve a clear screen by one key press. Function key F3 could be used:-

```
KEY 3,"CLS"+CHR$(21)+CHR$(13)
```

As with key F0 we have added the 2 control characters - 21 delete to end of line - and - 13 carriage return. This means that key F3 can be used to perform a CLS even if there is other text on the same screen line.

Function keys can contain more than one instruction provided they are separated with a colon - : - as one would use in a normal program line that contained more than one instruction. If one is using a normal television rather than the Einstein monitor you may have noticed that the 32 column screen mode tends to lose the last character on each line, making editing a big problem. Whereas 40 column display appears correctly. Therefore when editing it is more convenient to list a program in 40 column mode regardless of which mode the program runs in. Key 4 could be assigned thus:-

```
KEY 4,"CLS40:LIST"+CHR$(21)+CHR$(13)
```

which would clear the screen to 40 column mode before listing the current program.

If one is working on a program which experiments with colours one could find the program requires debugging and comes to a halt printing text in the same colour as the background - effectively displaying nothing. Defining a key to restore the colours to sanity would be useful as one could not see what was being entered from the keyboard and probably would not know the position of the cursor, so CHR$(21) should be used too.

```
KEY 5,"BCOL4:TCOL15,0"+CHR$(21)+CHR$(13)
```

One will soon discover which commands are more frequently required when entering or debugging programs, one that is always being used is the DIRectory command. The right hand function key could be used - F7 - for function keys do not need to be assigned in order.

```
KEY 7,"DIR"+CHR$(13)
```

will list the directory of the current disc with only one key press.

Entering KEY LIST <ENTER> should now display the contents of the function keys.

Remember that each key can be used in conjuction with the SHIFT key so producing 16 functions.

Although KEY LIST shows these shifted keys numbered from sF0 to sF7 they would be entered as from F8 to F15.

```
F0: RUN[CR]
F1: LIST[CR]
F2: LOAD"
F3: CLS[CR]
F4: CLS4:LIST[CR]
F5: BCOL4:TCOL15,[CR]
F6:
F7: DIR[CR]
sF0:
sF1:
sF2:
sF3:
sF4:
sF5:
sF6:
sF7:
```

Notice that when CHR$ codes are added to the strings they become part of those strings, and are stored as single characters.

When printing out the KEY LIST the Einstein alters the values of any characters below 32 by adding 128, for if it tried to print some of these control characters it could have adverse effects on the display. The last character in each string, be it a printable or control code character, is altered automatically which will be shown overleaf. So in the case of F0 above the last 2 characters are altered from 21 and 13 which are unprintable to 149 and 141.

The [CR] and [CR] characters are shown in the appendix of this book along with other control code symbols. It should be remembered that the ASCII code area from 128 to 160 can be used for storing sprite patterns, therefore one could find when entering KEY LIST that the [CR] has changed to some other shape, but it will not affect the operation of the function keys.

To clear the function keys simply load them with a space, or even enter the following in direct mode - without a line number - which will clear all the function keys:-

```
FOR A=0 TO 15: KEY A," ": NEXT <ENTER>
```

If memory space in the function key buffer is critical there are some abbreviations which can be made by using some control codes. As an example we defined function key 4 as:-

```
"CLS40:LIST"+CHR$(21)+CHR$(13)
```

which is 12 bytes in total. There are control codes - single bytes - which will clear the screen in the same way that CLS does. There are others which will clear the screen and alter it to 32 or 40 column mode. These are shown in the DOS/MOS Introduction under the CTRL key section. Maybe you didn't realise that these codes are actual ASCII control codes, well they are, and as such can be used within a string of characters.

Pressing the <CTRL> + <N> keys will clear the screen to 40 column mode. So will entering:-

```
PRINT CHR$(14)
```

Another method of remembering the above is that N is the 14th letter in the alphabet.

Armed with this knowledge function key 4 could be reduced by 5 bytes by entering it as :-

```
KEY 4,CHR$(14)+"LIST"+CHR$(21)+CHR$(13)
```

which will only use 7 bytes of the function key buffer.

Obviously using some control codes can have disastrous results! If one does not have a printer connected to the Einstein then do not use CHR$(1) or CHR$(18), which is similar to pressing the <CTRL> + <A> or <CTRL> + <R> keys,

as these codes relate to the printer and the Einstein will drop into a trance and the only way out is to press the reset button which could result in your program being wiped out.

In this chapter we have referred to the function key entries as strings. These strings are similar to the string variables which one uses in programs - combinations of ASCII characters. One could append a control character to a string variable within a program for certain results. CHR$(7) is the control code for a tone similar to that of the BEEP command. If one required a message to sound an audible prompt each time it was displayed one could use CHR$(7). A Typical line could be:-

```
100 A$="What is your Name"+CHR$(7)
```

and whenever A$ was printed on screen the tone would be heard too. One could even prefix the string with a clear screen code so that the screen was always cleared prior to the message being printed:-

```
100 A$=CHR$(12)+"What is your Name"+CHR$(7)
```

The answer is to experiment.

3
Character set

On power up, or after the basic command RST has been used, the character set is loaded in from the ROM area of memory into the Video RAM area. Being in VRAM means the characters are soft - they can be altered.

The complete ASCII set - that's 256 characters - are made up of 8 bytes each, therefore to store the complete set takes 2048 bytes of memory. They are stored from addresses 1800 to 1FFF hex (6144 to 8191 decimal) in VRAM. Characters with a value of 32 or less are all blank as, besides character 32 (space), they are used as control characters and are not printable on screen. The appendix shows how each character is formed and where in VRAM it is located.

To create a new character, or sprite, one may either use the SHAPE command or, as will be seen later, the VPOKE function. The Introduction manual describes how each character is formed by using an 8x8 pixel grid. It must be remembered that all characters will be reset to their original forms after the RST command has been used, this will wipe out any sprites that had been defined. Many commercial programs use RST to reset XBAS to its switch-on condition, and a by product is that the characters are re-initialisation as well.

It is recommended that sprites should be defined in characters 128 - 160 as these are unused. If you read the chapter on Key defining one would have seen that some of the characters in this area are initially set to display the control codes when KEY LIST is entered, but they can still be overwritten. Defining a character is not that

easy, as one should possess some knowledge of how a byte is composed, the Introduction manual should give one an idea on the format. However if the manual is used in conjunction with the next program it should become even clearer. The program, although a bit lengthy, is designed to let you experiment with different shapes for use in one of your own programs. Once you have decided on a final shape you should copy down the shape values which will be displayed for use as characters or sprites in your own program. It will allow one to move the cursor around a displayed grid using the cursor keys. The keys will auto repeat if held down but the cursor will not move out of the specified grid.

Once running the program will ask you to enter '1' for a normal size 8x8 character or '2' for double size 16x16. Afterwhich you should then enter the ASCII number - in decimal - of the character you wish to alter. If you choose an ASCII number which already has a character assigned to it it will be displayed on the grid. Hence entering 141 would display the character as below, which can obviously be altered as it is only used with KEY LIST.

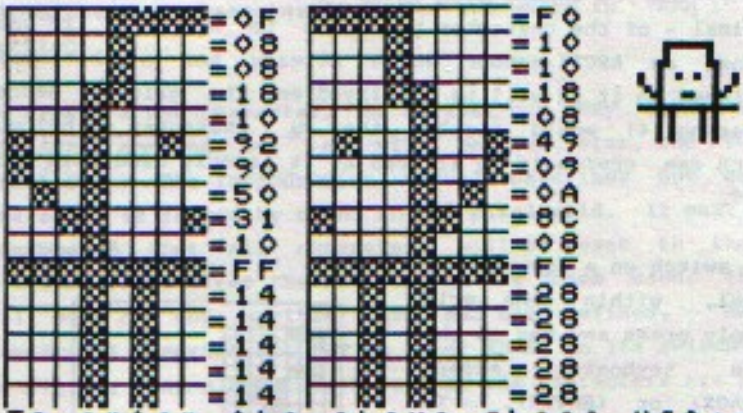
To switch on a square, or pixel, within the grid simply press any key on the main keyboard except ‹SPACE› or ‹ENTER›. To erase one of the squares position the cursor over the square and press the ‹SPACE› key and the pixel will switch off.

Once you are satisfied with the character press the ‹ENTER› key and the shape will be displayed in the top right of the

screen with the correct line which should be entered in one of your own programs shown at the bottom of the screen.

When one has entered a 16x16 character or sprite, which defines 4 characters, the columns are divided on screen and the hex value for each row is shown. One may either use the SHAPE command followed by the 32 items in a string, or enter 4 separate SHAPE command followed by 8 items in the string, the program will prompt you to press any key to show the alternative.

This is how the screen would look after entering this 16x16 shape, with the hex values for each row shown alongside the grid, and the recommended form of entering the shape shown under the grid.



To enter the above shape use:-
SHAPE 128,"0F0808181092905 0"
SHAPE 129,"3110FF1414141414"
SHAPE 130,"F0101018084909 0A"
SHAPE 131,"8C08FF2828282828"
and enter each singly

```
10 SPRITE OFF: IOM 4,0
20 BCOL 4: GCOL 15,0: TCOL 15,0: CLS40
30 PRINT "Enter size of Sprite"
40 PRINT "Press 1 for 8x8...2 for 16x16"
50 SZ=INCH
60 IF SZ<49 OR SZ>50 THEN PRINT: PRINT "ENTER 1 or 2":
BEEP:GOTO 40
70 CLS: SZ=SZ-48: IF SZ=2 THEN MAG 3:ELSE MAG 1
80 PRINT "Enter character number to modify"
90 PRINT "For sprites it is best to use numbers"
100 PRINT "between 128-160."
110 IF SZ =1 THEN 140
120 PRINT "16x16 size MUST be entered"
130 PRINT "as multiples of 4: i.e.128,132,136 etc"
140 INPUT CHAR
150 IF CHAR<0 OR CHAR >255 THEN PRINT "Between 0 and 25
5":BEEP:GOTO 90
160 IF SZ=2 AND CHAR MOD 4<>0 THEN BEEP:GOTO 120
170 X=0
180 D=CHAR*8+6144
190 CLS32
200 FOR E=1 TO SZ
210 FOR A=D TO D+(8*SZ-1)
220 PRINT@ X,POS(2)
230 A$=BIN$(VPEEK(A),8)
240 FOR B=1 TO 8
250 IF MID$(A$,B,1)="0" THEN PRINT CHR$(32);:ELSE PRINT
 CHR$(235);
260 NEXT B
270 PRINT
280 NEXT A
290 X=0:D=A:PRINT@ 0,0;:NEXT E
300 GOSUB 800
310 REM DRAW SINGLE CELL
320 DRAW XC*8,191-(YC*8) TO XC*8+8,191-(YC*8) TO XC*8+8
,183-(YC*8) TO XC*8,183-(YC*8) TO XC*8,191-(YC*8)
330 PRINT@ XN,YN
340 XC=POS(1):YC=POS(2)
```

```
   350 A=INCH
   360 IF A=13 THEN GOTO 440
   370 IF A=4 AND POS(1)<8*SZ-1 THEN XN=XC+1:GOTO 310:ELSE
 IF A=4 GOTO 310
   380 IF A=8 AND POS(1)>0 THEN XN=XC-1:GOTO 310:ELSE IF A
 =8 GOTO 310
   390 IF A=10 AND POS(2)<8*SZ-1 THEN YN=YC+1:GOTO 310:ELS
E IF A=10 GOTO 310
   400 IF A=11 AND POS(2)>0 THEN YN=YC-1:GOTO 310:ELSE IF
A=11 GOTO 310
   410 IF A=32 THEN PRINT " ":GOTO310
   420 PRINT CHR$(235):GOTO 310
   430 REM DEFINE ENTERED SHAPE
   440 IF SZ=1 THEN 520
   450 FOR Y=0 TO 15
   460 LIN$=MID$(SCRN$(Y),9,15)
   470 LIN$="    "+LIN$
   480 PRINT@ 8,Y;LIN$
   490 FOR X=96  TO 152 STEP 8
   500 DRAW X,191-(Y*8) TO X+8,191-(Y*8) TO X+8,183-(Y*8)
TO X,183-(Y*8) TO X,191-(Y*8)
   510 NEXT X,Y
   520 FOR B=0 TO SZ-0.5 STEP 1.5
   530 FOR Y=0 TO 8*SZ-1
   540 LIN$=MID$(SCRN$(Y),B*8+1,B*8+8)
   550 C=0
   560 FOR X=1 TO 8
   570 IF MID$(LIN$,X,1)=" "THEN 590
   580 C=C+2^(8-X):REM THE ^ KEY IS 3RD FROM RIGHT,TOP
ROW
   590 NEXT X
   600 PRINT@ (B+1)*8,Y;"=";HEX$(C,2)
   610 SH$=SH$+HEX$(C,2)
   620 NEXT Y,B
   630 PRINT@ 0,16,"To enter the above shape use:-"
   640 WIDTH 30
   650 PRINT"SHAPE";CHAR;CHR$(44);CHR$(34);SH$;CHR$(34)
   660 SHAPE CHAR,SH$
```

```
670 SPRITE 1,200,180,15,CHAR
680 IF SZ=1 THEN 770
690 PRINT"ALTERNATIVELY..(press any key)"
700 A=INCH
710 WIDTH 32
720 PRINT@ 0,17;
730 FOR Y=0 TO 3
740 PRINT "SHAPE";CHAR+Y;CHR$(44);CHR$(34);MID$(SH$,Y*1
6+1,16);CHR$(34);CHR$(21)
750 NEXT Y
760 PRINT"and enter each singly"
770 WIDTH 0
780 END
790 REM DRAW FULL GRID
800 FOR X=0 TO 64*SZ STEP 8
810 DRAW X,191 TO X,191-64*SZ
820 DRAW 0,191-X TO 64*SZ,191-X
830 NEXT
840 RETURN
```

Alpha characters

We have seen how one can alter each ASCII character by the SHAPE command, but they can also be altered using the VPOKE function which will alter the contents of video RAM memory directly as the POKE function will on RAM. A diagram showing the VRAM map is shown on page 315 of the Reference manual.

The table which holds each character commences at address 1800 hex (6144 dec) in VRAM. To calculate the address where a particular character is stored one would take the character number and multiply by 8 - as each character requires 8 bytes to hold the 8 rows of pixels. This figure is added to 6144 for the start address of the character. To find where the 'A' character is stored we first take the ASCII code of A, which is 65. So 65*8=520. 6144+520=6664, character 'A' is stored from 6664 decimal in VRAM.

The following program alters the A - Z, a - z and 0 to 9 groups to slightly larger, and you may agree, clearer characters by increasing their height. A normal character is usually made up thus:-

With an empty row of pixels top and bottom. Note that the 2 right hand columns are also bare due to the 40 column screen mode only displaying the first 6 columns. So unless one is using 32 column consistently it is unwise to extend the width, but the height? Possibly. One can alter the alpha characters to whatever but, if common sense is to prevail, they should bear some resemblance to their original characters.

```
 10 REM UPPER CASE
 20 FOR X=6648 TO 6871
 30 READ A$: VPOKE X,VAL("&"+A$): NEXT
 40 DATA 38,44,04,08,10,0,10,0:REM ?
 50 DATA 38,44,04,34,54,54,38,0:REM @
 60 DATA 10,28,44,44,7C,44,44,0:REM A
 70 DATA 78,24,24,38,24,24,78,0:REM B
 80 DATA 18,24,40,40,40,24,18,0:REM C
 90 DATA 78,24,24,24,24,24,78,0:REM D
100 DATA 7C,40,40,78,40,40,7C,0:REM E
110 DATA 7C,40,40,78,40,40,40,0:REM F
120 DATA 38,44,40,5C,44,44,38,0:REM G
130 DATA 44,44,44,7C,44,44,44,0:REM H
140 DATA 38,10,10,10,10,10,38,0:REM I
150 DATA 1C,08,08,08,48,48,30,0:REM J
160 DATA 44,48,50,60,50,48,44,0:REM K
170 DATA 40,40,40,40,40,40,7C,0:REM L
180 DATA 44,6C,54,54,44,44,44,0:REM M
190 DATA 44,64,64,54,4C,4C,44,0:REM N
200 DATA 38,44,44,44,44,44,38,0:REM O
210 DATA 78,44,44,78,40,40,40,0:REM P
220 DATA 38,44,44,44,54,48,34,0:REM Q
230 DATA 78,44,44,78,50,48,44,0:REM R
240 DATA 38,44,40,38,04,44,38,0:REM S
250 DATA 7C,10,10,10,10,10,10,0:REM T
260 DATA 44,44,44,44,44,44,38,0:REM U
270 DATA 44,44,44,44,28,28,10,0:REM V
280 DATA 44,44,44,54,54,6C,44,0:REM W
290 DATA 44,44,28,10,28,44,44,0:REM X
300 DATA 44,44,44,38,10,10,10,0:REM Y
310 DATA 7C,04,08,10,20,40,7C,0:REM Z
320 REM LOWER CASE
330 FOR X=6920 TO 7127
340 READ A$: VPOKE X,VAL("&"+A$): NEXT
350 DATA 0,0,38,4,3C,44,3C,0:REM a
360 DATA 40,40,58,64,44,64,58,0:REM b
370 DATA 0,0,38,44,40,44,38,0:REM c
380 DATA 4,4,34,4C,44,4C,34,0:REM d
390 DATA 0,0,38,44,7C,40,38,0:REM e
```

```
400 DATA 8,14,10,7C,10,10,10,0:REM f
410 DATA 0,0,34,4C,4C,34,4,38:REM g
420 DATA 40,40,78,44,44,44,44,0:REM h
430 DATA 10,0,30,10,10,10,38,0:REM i
440 DATA 8,0,18,8,8,8,48,30:REM j
450 DATA 20,20,24,28,30,28,24,0:REM k
460 DATA 30,10,10,10,10,10,38,0:REM l
470 DATA 0,0,68,54,54,54,54,0:REM m
480 DATA 0,0,58,64,44,44,44,0:REM n
490 DATA 0,0,38,44,44,44,38,0:REM o
500 DATA 0,0,58,64,64,58,40,40:REM p
510 DATA 0,0,34,4C,4C,34,4,4:REM q
520 DATA 0,0,58,64,40,40,40,0:REM r
530 DATA 0,0,3C,40,78,4,78,0:REM s
540 DATA 20,20,78,20,20,24,18,0:REM t
550 DATA 0,0,48,48,48,48,34,0:REM u
560 DATA 0,0,44,44,44,28,10,0:REM v
570 DATA 0,0,44,54,54,54,28,0:REM w
580 DATA 0,0,44,28,10,28,44,0:REM x
590 DATA 0,0,44,44,4C,34,4,38:REM y
600 DATA 0,0,7C,8,10,20,7C,0:REM z
610 REM 0 TO 9
620 FOR X=6528 TO 6607
630 READ A$: VPOKE X,VAL("&"+A$): NEXT
640 DATA 38,44,4C,54,64,44,38,0:REM 0
650 DATA 10,30,50,10,10,10,7C,0:REM 1
660 DATA 38,44,4,8,30,40,7C,0:REM 2
670 DATA 38,44,4,18,4,44,38,0:REM 3
680 DATA 8,18,28,48,7C,8,8,0:REM 4
690 DATA 7C,40,40,78,4,44,38,0:REM 5
700 DATA 18,20,40,78,44,44,38,0:REM 6
710 DATA 7C,44,8,10,10,10,10,0:REM 7
720 DATA 38,44,44,38,44,44,38,0:REM 8
730 DATA 38,44,44,3C,4,8,30,0:REM 9
740 REM (  )
750 FOR X=6464 TO 6479
760 READ A$: VPOKE X,VAL("&"+A$): NEXT
770 DATA 8,10,20,20,20,10,8,0:REM (
780 DATA 20,10,8,8,8,10,20,0:REM )
```

Sprite management

The Video display processor is mapped out on page 315 of the 'Reference manual' and it will be seen that an area in the VDP is allocated to the 'sprite attribute table'. Although sprite movement is adequately described in the manuals, the following could be of added interest to those who like POKEing, but in the case of VRAM it would become VPOKEing.

The table starts at 15104 (&3B00) in VRAM and is 128 bytes in length. This table is divided into 32 blocks of 4 bytes each, one block for each sprite. The first block (15104 to 15107) controls sprite 0, the second block (15108 to 15111) controls sprite 1, and so on. If we number the bytes in each block from 0 to 3, the first byte - byte 0 - in each block contains the Y co-ordinate of the top left pixel of the sprite. One should not pay too much attention to the contents of this byte as it does not take into account any altered ORIGIN that may be in force, and also measures the position from the top of the screen and not the bottom. Hence entering the following line:-

```
SPRITE 0,0,8,15,65
```

will display the character 65 - A - in the lower corner of the screen in white. We know that the Y co-ordinate is at position 8, but it is stored in the first byte in the block as 183 (191-8). Enter PRINT VPEEK(15104) <ENTER>

Byte 1 holds the current X co-ordinate in the normal fashion, measured from the left of the screen, but this could also give a false reading if ORIGIN has been moved.

Byte 2 contains the character number, in the above case 65 and byte 3 the colour of the sprite. Now it is perfectly feasible to VPOKE these 2 locations directly. Altering the

character of an existing sprite on screen could be achieved without the need to enter SPRITE 0,x,y, etc., providing one knew the location of the sprite in the attribute table.

Enter this program:-

```
10 MAG 0: SPRITE OFF: CLS
20 FOR A=0 TO 8
30 SPRITE A,0,A*8+8,15,65+A
40 NEXT
```

This simply loads up and display 9 sprites, from 0 to 8, with characters A to I and displays them from the bottom left hand corner upwards. Now to alter the colour or character number of any sprite we could adopt the following:-

Sprite number * 4 + start of table + either 2 or 3 for character or colour.

So to alter the character of sprite 3, which at the moment is character 68 - D - at the fourth from bottom position of the screen, to the letter Z - character 90 - one could calculate thus:-
3*4+15104+2=15118 so entering:-

```
VPOKE 15118,90 <ENTER>
```

will alter the character to Z, one will not need to know the current X or Y co-ordinates of the sprite. To eliminate excess code one could set up either of these calculations as FuNctions by entering early in the program:-

```
5 DEF FNN(S)=S*4+15104+2: REM Character number change
6 DEF FNC(S)=S*4+15104+3: REM Colour change
```

Where S=sprite number.

So to alter the colour of sprite number 7 to black one could enter:-

```
VPOKE FNC(7),1 <ENTER>
```

to alter sprite number 8 to character 90 - Z - enter:-

```
VPOKE FNN(8),90 <ENTER>
```

The advantage of altering colours and characters in this way means that a sprite on the move can be changed without the need to know its present position. One further point regarding the attribute table. When the command SPRITE OFF is encountered all sprites are switched off and each has the first byte in its attribute table - the Y co-ordinate - altered to the value 192. Entering SPRITE OFF 1 would only alter the first byte of sprite 1's attribute block to 192. Normally to turn off several sprites one requires to enter a succession of numbers following SPRITE OFF. However if one VPOKEs the value 208 into any one of the block's first byte all higher numbered sprites will be also turned off. The block for sprite 4 begins at VRAM address 15120 (4*4+15104) and if we enter VPOKE 15120,208 <ENTER> all the sprites from the E character upwards have been cleared from the screen.

There is no method in basic of detecting when 2 sprites have collided - when their pixels overlap. The VDP, which is a customised chip, does record such collisions but it is not simple to grasp how this chip operates, in fact a book could be written on this subject alone. The following routine which is POKEd high up in RAM can be called whenever one wishes to detect a collision. It simply tests one bit of the status register of the VDP (there are 8 registers in total) which is set if sprites overlap. It does not detect which sprites have overlapped, simply that a collision has occurred.

It stores the result in address 57336 and when PEEKed the result must be ANDed with 32 which will test the relevant bit. Here is a simple demonstration, the machine code routine, which is the part one would transpose into your own program, is in lines 20-40.

```
 10 SPRITE OFF
 20 CLEAR 57327
 30 FOR X=57328 TO 57335: READ A: POKE X,A: NEXT
 40 DATA 6,136,219,9,50,248,223,201
 50 FOR X=100 TO 150
 60 SPRITE 0,X,8,15,65
 70 SPRITE 1,250-X,8,1,65
 80 CALL 57328:A=PEEK(57336) AND 32:IF A=32 THEN GOSUB 110
 90 NEXT X
100 END
110 PRINT "OVERLAPPING NOW": RETURN
```

This purely sends 2 sprites towards each other, line 80 does the testing and diverts to line 110 and prints the message only while the sprites are overlapping.

4
Peeks, pokes, do's and don'ts

This chapter highlights some of the lesser known facts concerning the Einstein, and some memory addresses which can be altered in XBAS 4.2 for various effects.

CTRL key pressed in conjunction with various keys will produce results as shown in the 'Reference manual', these give the same results if used with the CHR$ function. Pressing <CTRL>+<G> keys produce a tone similar to the BEEP command as will PRINT CHR$(7), as G is the seventh letter in the alphabet. It must be stressed that if you do not have a printer connected, or it is not switched on, then pressing <CTRL> with either <A> or <R> keys - or entering PRINT CHR$(1) or CHR$(18) - will result in the Einstein completely locking up and unable to continue with a program. If this happens the only solution is to press the reset button or switch off, either way you will lose any program currently in memory.

GCOL requires its second parameter, background, entered with some care if one is to avoid drawing lines with square blobs as background. When the CLS command is encountered the screen will clear and show the background area in the colour of the second parameter of TCOL. Hence if - TCOL 15,1:CLS - is used the screen will be in black (1) with any text in white (15). Any graphics that are subsequently drawn should have their second parameter set to 1 too, irrespective of their foreground colour. Altering the background of TCOL without clearing the screen will upset any lines drawn across that area as GCOL will be using the earlier setting of TCOL for its background. If one is not too sure on this point it could be safer to set BCOL to whatever background colour is required, and set both TCOL

and GCOL second parameters to 0 (transparent) so that both text and graphics are superimposed over the BCOL.

HOLD and CHAIN are powerful commands by today's standards. Many disc based computers allow additional programs to be chained without destroying any existing variables and their contents, but few allow sections of the original program to remain active. As a program is running a lot of free memory contains garbage - areas which contain old values for strings and variables. Every so often the memory is so full of this rubbish that the micro needs to engage in a garbage clearance routine to free some extra space in memory. This normally goes unnoticed by the user. It also occurs when one enters PRINT SIZE.

A problem can arise when using HOLD, for if substantial garbage exists the CHAINed program may not load into memory and, although you may be convinced that there is plenty of free memory for the program, you could be surprised to see the 'Mem Full' error message.

The solution is to force a garbage clearance, before HOLDing part of a program, by entering SIZE. One need not enter PRINT SIZE, just enter a line before any line containing HOLD by using a dummy variable and enter the statement A=SIZE. If sufficient garbage does exist the program may appear to pause, but better pause than crash. However, it should be stressed that this problem only occurred once during testing the database program in the final chapter of this book, and then only after a large file containing 200 string variables had been loaded in from disc. In the program named 'DB' a forced A=SIZE is encountered in the held part of the program - line 200 - each time the program returns to the menu, and occasionally the program pauses, but if your own program is smaller you may find it is not required.

INPUT prompt character - ? - is stored at 9878 (&2696). Entering PRINT PEEK(9878) will display 63 - the ASCII code for ?. To alter to another character enter POKE 9878,new character ASCII code.

IOM 3,0 will disable the SHIFT/BREAK keys from halting a program, but it may still be temporarily paused by pressing the BREAK key on its own. Entering POKE 3536,0,0,0 will completely disable the BREAK key regardless of the setting of IOM 3. To restore to normal enter POKE 3536,205,108,10.

KBD Under differing conditions testing for a key press using KBD, which does not wait for a key to be pressed, can produce spurious results. A typical program line could be - A=KBD: IF A=0 THEN etc - where one is only going to carry out a task if a key is down, when A<>0. However, occasionally when this line is encountered and no key has been pressed, KBD returns a non zero value into variable A. If this occurs simply enter into an earlier program line POKE &013D,0. This will clear any odd value which is lurking in the KBD buffer.

ON ERR is a useful error trapping command where execution of a program will divert to an error routine one has entered into the program, instead of stopping at the error. It should be remembered that once an error has been trapped in this way any further errors will not divert to your routine unless an ON ERR statement is issued again, probably within your error routine. If an error has not occurred and one CHAINs a program the existing line number specified in the previous ON ERR statement will still be valid, and an error in the CHAINed program will still divert to the line number of the error routine, so make sure the CHAINed section contains an error routine with the same line number or appoint another section as the error routine.

RENUM Very useful for opening up the line numbering of a program for the insertion of extra lines. However, never enter just 1 parameter followed by a comma - i.e. RENUM 100, - as this will result in every line having the same number of 100. If you are thinking that if this happens you can correct the listing by entering RENUM again, only this time with 100,10 as the parameters, think on... any GOTOs, GOSUBs will have been altered and will refer to line 100, and will subsequently still refer to line 100 when the corrective measure is entered. The inquisitive reader should make sure that the original program is saved before trying this.

RST is a valid command, although it is not mentioned in my copy of the manual. It must be used with care as it resets some functions to their initial switch-on conditions, these are:-

Clears screen to 40 col.
Resets all characters to originals.
Wipes out any sprites.
TCOL and GCOL to 15,0 (white on transparant)
Resets disc controller and sound generator.
Resets WIDTH, ZONE, SEP, SPEED and NULLS.
Resets all IOMs to 1
Resets ORIGIN to 0,0
Turns cursor on if off.
Resets all PTRs to switch-on state.

RST will not alter the following:-

BCOL - TI$ - Function keys - TEMPO
or the current graphics pointer co-ordinates.

SCRN$ is used for reading the contents of a screen line into a string variable. This may also be used for discovering the contents of 1 character within that line.

One could incorporate the MID$ function. Hence, remembering that screen lines are numbered 0 to 23, the instruction - X$=MID$(SCRN$(0),20,1) - would result in X$ holding the 20th character on the top line. A varying amount of characters could also be found by using the LEFT$ or RIGHT$ function with SCRN$.

PTR 19 can be used as an alternative to the above as it contains the memory address of the start of the 'internal VDU' area. The ASCII values contained at every screen location are copied into this area of RAM - note not VRAM - meaning that one can PEEK into this area to discover screen characters. Hence entering PRINT PTR(19) would result in 59455 being displayed. Subsequently entering PRINT PEEK(59455) will result in the display of the ASCII code for the character situated at top left hand corner position. The second character on the top line would be found at 59456 and so on. To find any character on screen one could simply adopt the following:-
screen line no. * screen mode + position on line + PTR(19)
An example should help, enter:-

```
10 CLS32: PRINT@ 16,4;"B"
```

This will clear the screen to 32 column mode and print 'B' at position 16 on the fifth line from the top of screen - top line is 0 don't forget.

```
20 A=PEEK(4*32+16+PTR(19))
30 PRINT A
```

Running the program will result in the ASCII code for B - 67 - being displayed.

POKEing to this internal VDU copy area will not alter any character on screen, it is purely used for reading the screen area.

WIDTH is used to limit the number of characters output on any line of an output device - i.e. printers. If you use this command to reduce the width then you should alter it back to zero when editing lines on screen, as you may find that long lines are shortened.

Scratchpad locations

After switching on, the machine operating system - MOS - copies into high RAM locations some initial settings for various functions. Some of these settings are altered from commands within basic - such as TCOL GCOL - while others can be altered directly by either the POKE function or by using the (M)odify command within MOS for differing effects. If using MOS to alter the contents one must use hex values. From basic entering PRINT PEEK(address) will display the contents, but it should be remembered that these have been given optimum settings by the designers and should only be altered for curiosity - which most of us have ample shares of - or for some special application that might enhance a program.

64312 (&FB38) contains the current TCOL. Although one would not normally alter this byte directly as this would be carried out by the basic TCOL command, it could be useful to PEEK it to discover the current text colours. If the text colour was currently white on transparent - TCOL 15,0 - the value returned by PRINT PEEK(64312) would be 240 which doesn't tell one too much. This decimal figure is equivalent to F0 hex - check with the hex/dec conversion chart in the appendix. The F, which is equal to 15, is the text colour while zero signifies the background. It might be simpler if one entered PRINT HEX$(PEEK(64312),2) which would display F0.

64313 (&FB39) holds the current settings for GCOL. Same rules apply as above, the first digit of the hex value

returned applies to the foreground while the second is the graphic background colour.

64319 (&FB3F) contains the ASCII value of the cursor character. Normally it contains the value 127 (&7F). Entering POKE 64319,161 would change the cursor to the graphics shape resembling an underline character.

64320 (&FB40) is the ASCII code for the chevron (>) prompt when in MOS.

64321 (&FB41) contains the value 32, which is the blink rate of the cursor. POKEing a lower value will increase the rate while a higher value slows it down. Try POKE 64321,10 to speed it up a little.

64322 (&FB42) holds the value 160 which is the initial delay before a key repeats. Lowering the value will shorten the delay, increasing will lengthen.

64323 (&FB43) controls the repeat rate of a key held down, that is how quickly successive characters are displayed after the initial delay above. It normally contains 16 but a lower value will increase the rate while increasing will slow it down. This could be speeded up within a program before a rapid response from the INCH command was required as this delay also effects INCH - see next.

64326 (&FB46) contains the ASCII value of the last key pressed or held down. This could be used as a faster method of testing for keys held down - for example in games programs. KBD scans the keyboard once and returns the ASCII code of a single key press. If the key is continually held down it will return the value once and thereafter return 0 until another key is pressed. This obviously has advantages where one is asking the user for inputs from the keyboard for if they were slow at releasing

a key it would have no effect. But for faster keyboard checks a PEEK(64326) will return the ASCII value on a continuous basis. Try this:-

```
10 A=KBD: PRINT A
20 GOTO 10
```

and run the program pressing various keys, and hold them down. Press <SHIFT>+<BREAK> to stop the program. A value other than zero is displayed only once when a key is first pressed. Alter line 10 to:-

```
10 PRINT PEEK(64326)
```

and run the program with keys held - spot the difference?

64335 (&FB4F) contains the current screen size - either 32 or 40. It is possible to alter the value of this byte without clearing the screen, hence 32 and 40 column display could be mixed and displayed on the same screen. Try the following:-

```
10 CLS40
20 PRINT "1234567890"
30 POKE 64335,32
40 PRINT "1234567890"
```

There are certainly some adverse side affects, especially when editing, but it could be used in moderation.

64406/7 (&FB96/7) store the current X co-ordinate of the graphics pointer in 2 bytes. POS 1 and 2 return the current text cursor position but it appears there is no system variable for discovering the graphics position. PRINT DEEK(64406) will display the current X co-ordinate, but this must be added to the present ORIGIN position. If ORIGIN is set to 0,0 as it is at switch-on then the above command will give a true position in relation to the viewed screen. One can move the current X co-ordinate to a new position, without using DRAW by DOKE 64406, new X position.

64408/9 (&FB98/9) contain the Y co-ordinate of the graphics pointer and can be used with DEEK and DOKE as overleaf.

64410/1 (&FB9A/B) contain the X origin co-ordinate. These 2 bytes are updated when the basic ORIGIN command is used, however there is no command within basic to discover the current origin. Entering PRINT DEEK(64410) will supply the result.

64412/3 (&FB9C/D) contain the Y origin co-ordinates and can be used as above.

5
Additions to XBAS

This chapter contains two utility routines which can be added to Tatung/Xtal Basic 4.2, the basic interpreter. Developed by Crystal Research the original versions for various machines allowed the user to create extra commands and functions to be added to the interpreter and used as if they were already built into the XTAL basic language.

After further enhancements a superb version specifically written for the Einstein was produced, but it didn't forget its origins, and despite being able to accomplish a host of tasks not thought possible in the early days, it is still capable of being expanded and additional commands added.

As the basic interpreter, XBAS, is stored in RAM it is simple to alter the values contained in any memory location, some might comment too simple as indescriminate changing of values could, and probably would, result in a system crash - programs will not function correctly, if at all. However if this condition does occur no lasting damage will be done it simply means that one will need to re-load XBAS or, at worst, press the reset button on the rear of the machine and start all over.

The methods of altering memory will be demonstrated in these sections. For those readers who have some understanding of machine-code and assembly language mnemonics any alterations made to memory locations will be described along with the respective instructions. Readers who have no knowledge, or aren't bothered with, machine-code need have no worries as the longer additional routines will also be shown as direct basic programs which can be entered by anyone with normal keyboard experience.

Before any alterations are made it is wise to format a new disc for the modified XBAS to be recorded on. The format facility is fully described on page 40 of the DOS/MOS Introduction manual.

XBAS ready message

To gain some practice, and to see some immediate results, of altering memory contents we will alter the 'Ready' message in XBAS. Insert a copy of the master disc and load XBAS in the normal way. The screen will clear and the following should be displayed:-

```
TATUNG/XTAL BASIC 4.2     (C) 1983 1984
Size:  43324
Ready
```

There are 2 methods of changing memory, the first is from MOS which is the machine code monitor section of the Machine Operating System. Enter the command:-

MOS ‹ENTER›

and the following will be screened with the cursor flashing next to the chevron:-

```
MOS 1.2
Ready
›
```

The Ready message is stored in memory starting at location 07A3 hex (1955 decimal). Enter:-

T07A3 07A7 ‹ENTER›

and the screen will display:-

```
07A3 52 65 61 64 F9 Ready
›
```

The hex numbers to the right of 07A3 are the values contained in the 5 bytes 07A3 to 07A7 hex. The values used when in the MOS, or DOS, sections of the operating system are always displayed, or entered, using hexadecimal notation and not decimal.

To the right of the line are the ASCII characters these numbers represent. Check the values with the chart in the appendix of this book. The only character which does not match up with the values is the final one, the letter 'y'. The ASCII code for 'y' is 79 hex (121 decimal) but as the Einstein must be able to separate the end of the message from other instructions within memory, the final character in a string - for that's what it is, a string of characters - has 80 hex (128 dec) added to its value. So adding 80 hex to 79 hex the resultant value becomes F9 hex, in decimal this becomes 128 + 121 = 249, check with the hex/dec conversion table in the appendix.

To make the Einstein even more 'user friendly' one could alter 'Ready' to your own initials, such as 'OK JR' for instance. The 'Ready' message is 5 bytes long so our alteration must not exceed this number. You will no doubt have your own ideas of alteration, but for an example let's try the above initials.

Firstly one should look up the hex values of the ASCII characters to be used in the message. Page 206 of the 'Introduction to the Einstein' shows the hex values. For the above message these are:-

	O	K	sp	J	R
the hex codes are	4F	4B	20	4A	52+80=D2

Note that the code for space (sp) must be used and the final character's value has 80 hex added.

To alter memory contents when in MOS one enters the M command followed by the starting address. In this case one would enter:-

M07A3 <ENTER>
and the display will show:-

```
07A3 52
```

The cursor will be flashing over the 5 of 52. Enter the 5 bytes in succession without spaces and finish off by entering the full stop key and then the 'ENTER' key thus:-

4F4B204AD2. <ENTER>
and the screen should now be displaying:-

```
07A3 4F4B204AD2.
>
```

One should return to XBAS by entering either the 'Y' or 'X' keys followed by the <ENTER> key. One will see the modified message displayed.

If one prefers the message could be enhanced even further by adding a tone to it, similar to that heard when the basic command 'BEEP' is entered. Although one will lose one of the printable characters in the message. The ASCII code for BEEP is 07 and this could be substituted for the final character, but one must add 80 hex to its value making 87 hex, as it is the last character in the message. Return to MOS and alter the final character at 07A7 enter:-

M07A7 <ENTER>
and the display will show:-

```
07A7 D2
```

The cursor will be flashing over the D of D2. Simply overwrite these 2 figures with:-

87. <ENTER>

And don't omit the full stop. On returning to XBAS one should see and hear the altered Ready message.

OK J

The second method of altering memory contents is by using the basic POKE function and this avoids the need to enter MOS. As with the last example one should look up the ASCII codes for the characters required. Unlike MOS one can use either hex or decimal values for the memory location and values, or even a combination of both. Remember in basic hex numbers must be prefixed with the & sign.

If we use the same characters as before one could enter in direct mode, without a line number:-

POKE &07A3,&4F,&4B,&20,&4A,&87 <ENTER>

or alternatively with the values in decimal:-

POKE &07A3,79,75,32,74,135 <ENTER>

or the address could be in decimal too:-

POKE 1955,79,75,32,74,135 <ENTER>

Any of these lines will change the values of 5 consecutive bytes starting from memory location 07A3 hex (1955 dec).

Expanding XBAS 4-2

We have seen a simple example of how memory contents can be altered, but here we are going to expand XBAS in order to add our own commands. First a brief description of XBAS.

XBAS starts loading into memory, as all COM files do, at address 0100 hex. Once loaded it extends up to address 3E00 hex (15872 dec). Programs written in basic are stored from location 3E01 (15873) onwards. PTR 0 is the scratchpad pointer which contains the address of the start of basic programs. Enter in direct mode:-

? PTR(0) <ENTER>
and 15873 should be displayed. Alternatively to display the same address in hex enter:-
? HEX$(PTR(0)) <ENTER>
and the result should be 3E01. The pointers (PTR) are summarised on pages 188/9 of the 'Reference Manual' and although the majority of users will rarely use this command the clues to the expandability of XBAS lie here.

Each basic instruction - reserved word - such as PRINT, RUN, GOTO etc., is stored in a table within XBAS. On entering one of these instructions the interpreter looks up the table to find if it recognises the instruction. If it doesn't then it returns with the 'Syntax Error' message. PTR 2 holds the start address of the standard reserved word table. In addition to this standard word table is an auxiliary reserved word table, the address of which is held in PTR 3. Although this auxiliary table already contains 25 commands which have been added by the developers to cater for the Einstein, it is this table which can be expanded upon to include extra commands. One will see from page 188 of the 'Introduction Manual' that it is also referred to as the 'user' reserved word table.

Coupled with these 2 reserved word tables are the standard address table - PTR 6 - and the auxiliary address table - PTR 8. When a line of a basic program is entered from the keyboard the Einstein converts all the reserved words to unique values - they are tokenised. The reserved word PRINT would be converted and stored as the hex value A2, GOSUB as 90 and so on. This speeds up the running time of a program dramatically. When the program is running the Einstein does not need to check on each reserved word it simply takes the token number and refers to one of the two address tables to discover where in memory it should jump to for that routine.

Although this is a simplified explanation, one should now be realising that in order to add any routines to XBAS not only must we add our new reserved words to the auxiliary word table, but we must add the address in memory of our routine to the auxiliary address table. If all this appears daunting at this stage don't worry as it's all very simple, well almost.

Expanding XBAS means that some of the free user memory will be lost to the interpreter, but not much, as at present the wake up size is 43324 bytes and after the following modifications which extend XBAS by 1Kbyte one will be left with 42300. The auxiliary reserved word and address tables butt up against the standard address table in the interpreter, as can be seen on the chart overleaf, and this is followed by the standard function table (PTR 7), the initialisation routine and a short routine for the DOS command. So to enable the tables to be expanded we must move both of them above this area of memory and split them up making room for additional reserved words and addresses to be added. This also means that the area of memory in which the tables resided will become unused and free to place extra routines.

Before any extra commands can be added we must implement the changes as below in the diagram on the right, and save a modified version of XBAS to disc.

This chart shows the top area of XBAS in memory.

XBAS 4.2			XBAS 4.2 modified		
hex		dec	hex		dec
				New start of basic	
			4201	programs (PTR 0)	16897
			4200	Space for extra	16896
			3FB2	addrs. & routines	16306
			3FB1	Auxiliary addr.	16305
			3F80	table (PTR 8)	16256
			3F7F	Space for extra	16255
			3E71	reserved words	15985
	Programs load		3E70	Aux. Resvd. word	15984
3E01	from here (PTR 0)	15873	3E01	table (PTR 3)	15873
3E00	DOS routine also	15872	3E00	DOS	15872
3DFC	end of XBAS	15868	3DFC	routine	15868
3DFB	Initialisation	15867	3DFB	Initialisation	15867
3DC5	Initial screen	15813	3DC5	Initial screen	15813
3D9D	message	15773	3D9D	message	15773
3D9C	Initialisation	15772	3D9C	Initialisation	15772
3D53		15699	3D53		15699
3D52	Standard Func.	15698	3D52	Standard Func.	15698
3CEF	table (PTR 7)	15599	3CEF	table (PTR 7)	15599
3CEE	Standard addr.	15598	3CEE	Standard addr.	15598
3C67	table (PTR 6)	15463	3C67	table (PTR 6)	15463
3C66	Auxiliary addr.	15462	3C66	This area	15462
3C35	table (PTR 8)	15413		now free for	
3C34	Aux. Resvd. Word	15412		additional	
3BC5	table (PTR 3)	15301	3BC5	routines	15301

It is assumed one has already formatted a disc in readiness for recording. Load XBAS as normal and once loaded take

out the copy master disc and place the newly formatted disc in the current disc drive. Enter:-

```
PTR 0,&4201 <ENTER>
```

This has now raised the starting location of basic programs from 3E01 hex (15873 dec) to 4201 hex (16897). It will still load and run all existing programs. We must now enter MOS to alter the addresses of the auxiliary reserved word and address tables - PTR 3 and 8. Although most PTRs can be changed when under basic control it is a safer method to alter these two from MOS.

To ease legibility the following entries will be shown in 3 columns - DISPLAYED, TO ENTER and COMMENTS. Enter the characters under the TO ENTER column only, then press the <ENTER> key. Characters shown under the DISPLAYED column are what you should see on the next screen line after pressing ENTER. The COMMENTS column contains a description of the entries for reference only and must not be entered.

```
DISPLAYED  TO ENTER          COMMENTS

Ready
           MOS               Move into MOS from basic
MOS 1.2
Ready
>          M2B88             PTR 3 address of Aux word table
2B88 C5    013E              Alter PTR 3 to 3E01
2B8A 04    .                 Full stop - quit Modify mode
>          M2B92             PTR 8 address of Aux addr table
2B92 35    803F              Alter PTR 8 to 3F80
2B94 C8    .                 Quit modify mode
>          C3BC5 3C34 3E01   Copy Aux word table to 3E01
>          C3C35 3C66 3F80   Copy Aux addr table to 3F80
>          F3BC5 3C66 00     Fill old tables with 00
>
```

We have altered both the auxiliary reserved word and auxiliary address tables (PTR 3 and 8) to adresses higher in memory. Note that hex addresses are held in 2 bytes but they are entered in reverse order, so the bright sparks can relax they were not typing errors. We then move the complete tables by using the MOS command C which copies whole sections of memory and places them elsewhere in memory. Hence the auxiliary reserved word table which resides between 3BC5 and 3C34 hex (15301-15412 dec) was copied to start from address 3E01 (15873). We aren't required to enter the finishing address of the new table position as the Einstein calculates this automatically, although for future additions to XBAS we should keep a note of this end location, it now finishes at address 3E70 (15984). The auxiliary address table which started immediately after the word table at 3C35 to 3C66 hex (15413-15462 dec) was copied to start from address 3F80 (16256) and this now ends at 3FB1 (16305).

The F for Fill command simply loaded zeroes into the original area taken up with the tables, as unwanted code at these locations could cause confusion at a later stage.

At this point one could alter the initial sign-on message to show that one is using a modified copy of XBAS. The message is held in memory locations 3D9D to 3DC5 (15773 to 15813). Entering:-
T3D9D 3DC5 <ENTER>
will display the contents of the initial message thus:-

```
3D9D 54 41 54 55 4E 47 2F 58 TATUNG/X
3DA5 74 61 6C 20 42 41 53 49 tal BASI
3DAD 43 20 34 2E 32 20 20 20 C 4.2
3DB5 20 20 28 43 29 20 31 39   (C) 19
3DBD 38 33 20 31 39 38 34 0D 83 1984.
3DC5 8A .
```

Any byte can be altered within these memory locations to another character by using the (M)odify command in MOS, except the last 2 bytes, 0D and 8A. The 0D is the code for carriage return, which moves the cursor back to the beginning of the screen line, and the final character 8A is the code for line feed (0A) which moves the cursor down to the next screen line, with 80 hex added to its value. As with the 'Ready' message the final character in a printable string has 80 hex added to its value signifying the end of message.

One could change the message to include your own version number, as one may make other copies of XBAS with different commands added, it would display the version number each time it loaded.

Here is one suggestion of how it could be changed. If one alters the bytes which display the '(C) 1983 1984' part of the message, that's the bytes between 3DB7 and 3DC3, making 13 bytes in total. As you will not be altering the last byte of the message there will be no need to add 80 hex to the last character, simply look up the hex values of the ASCII characters. Hence to alter the message to include 'Version 1.0' the hex codes for which are:-

V	e	r	s	i	o	n	(sp)	1	.	0
56	65	72	73	69	6F	6E	20	31	2E	30

As there are only 11 characters in the above, the initial message will still print part of the line which contained the year numbers thus:-

TATUNG/Xtal BASIC 4.2 Version 1.084

So adding 2 space characters (20) to the line will erase the extra characters. One should enter from MOS

M3DB7 ‹ENTER›

and the display will show the cursor flashing over the 2 of

the value 28. As with 'Ready' simply type in the values without spaces and finish off with the full stop before pressing the ENTER key:-

```
56657273696F6E20312E302020. <ENTER>
```

Whether you have altered the initial message or not one should now return to basic by entering:-

```
X <ENTER>
```

The screen will now clear and display the initial sign on message. To save the modified basic to disc we must enter DOS by either entering DOS <ENTER> or by pressing the <CTRL>+<BREAK> keys.

Saving files under DOS is different from saving basic programs in that the size of the file needs to be calculated first. Our version of XBAS loads at 0100 up to 4200 hex, therefore the size is 4100 hex (4200 - 0100). This figure should be converted to decimal and then divided by 256 to discover how many blocks it will occupy on the disc. 4100 hex equals 16640 decimal, therefore as 16640/256=65 the block size will be 65. With a newly formatted disc in the drive enter:-

```
SAVE 65 XBAS.COM <ENTER>
```

And after a few seconds the expanded version of XBAS will be saved to a new disc. One need not call it XBAS, it could be saved with just 1 character in the name but one must always add the .COM to whatever it is called.

Now one has an expanded version a XBAS, capable of having additional commands added, there will probably be little need to alter the addresses for the reserved word and address tables as was shown earlier. Simply make any additions and save under DOS as is shown above.

Adding routines

It is difficult to write routines into the basic interpreter without a knowledge of machine code and the way the interpreter operates. These pages show the principles involved and with care one should possess a copy of XBAS with some useful utility aids to programming.

There are 4 stages to adding an extra command to XBAS.

a) Enter the reserved word of the command to the end of the auxiliary reserved word table - PTR 3.

b) Increase the value of the first byte in the table. This byte holds the total number of words in the table.

c) Add the address of the new routine to the end of the auxiliary address table - PTR 8.

d) Enter the machine code routine to start from the address entered above.

There are 2 methods shown for adding the extra commands in this chapter. The first is by using the MOS command Modify. This first example should help clarify most points regarding how the tables are set up, especially for those readers considering writing their own routines. The assembly language mnemonics for each instruction are also shown under the COMMENTS column.

The second is by a basic program in which the values are simply POKEd directly into memory using READ and DATA statements, which some may find easier.

Trace function

The trace function provides the facility of displaying the number of the current line being executed in a basic program. In this routine it is displayed in the top left corner of the screen. This can be useful when trying to track down a bug in one of your own programs or perhaps searching for a line in some commercial software to find how a particular effect is programmed.

Entering TRON before running a program will turn on the trace command while TROFF will turn it off. Although XBAS has a SPEED command for slowing down the output it is either on or off. Included in this routine is the option of pressing keys between 1 and 7, while the program is running, to slow it down when and where required. Pressing the '7' key once - there is no need to keep it pressed - would be the slowest speed. Pressing any other keys of the main keyboard would return the speed to almost normal.

Whilst the TRON command is switched on programs do run slightly slower than normal, but as this command is only used while debugging a program this should not cause any inconvenience for when TROFF is entered the program runs at the normal speed.

Adding words

The first task is to enter our new reserved words -TRON and TROFF - into the auxiliary reserved word table. To check on the correct location to place them we will display that area of memory by using the MOS command Tabulate to see just where the auxiliary word table finishes. In the last section we moved this table to start at address 3E01

(15873) and we calculated that it finished at 3E70 (15984). Load the modified copy of XBAS then enter:-

MOS <ENTER>
We only need the last few bytes of the table, so enter:-

T3E58 3E77 <ENTER>
and the display should be similar to below.

```
3E58 45 50 C2 49 4E 24 28 D2 EPBIN$(R
3E60 53 54 CB 45 59 C1 44 43 STKEYADC
3E68 28 C2 54 4E 28 D0 53 57 (BTN(PSW
3E70 80 FF FF FF FF FF FF FF ........
>
```

Note The bytes above containing FF may be displayed as 00.

Here we can see the last few reserved words: BIN$(RST KEY ADC(BTN(and PSW. Note that the last byte in the table (3E70) is 80 hex. This signifies to the Einstein that it is the end of the table. So our new word, or words in this case, must start from byte number 3E70, and overwrite the old table end value of 80 hex, and we must remember to enter 80 hex after the last character of our new reserved words.

Each word has the top bit of its first character set, slow down I hear from the back. In a similar fashion to how screen messages have 80 hex added to their final character - as with the 'Ready' message - reserved words have 80 hex added to their first character. No this is not done to be arkward, the Einstein must be able to differentiate between words in the table. Take the sixth character in the line commencing with 3E68, it is the letter P of the reserved word PSW. The ASCII code for P is 50 hex but as this is the first character of that word it has 80 hex added, so giving a value of D0 hex.

The codes for the 2 new words are as follows:-

T = 54+80=D4	T = 54+80-D4
R = 52	R = 52
O = 4F	O = 4F
N = 4E	F = 46
	F = 46
+ end of table marker	= 80

Enter:-
M3E70 ‹ENTER›
and the display will show:-

```
3E70 80
```

with the cursor flashing over the 8. Enter:-

D4524F4ED4524F464680.‹ENTER›
If you want to check that the 10 bytes have successfully been entered in the correct place in memory enter:-

T3E68 3E7F ‹ENTER›
and the display should show:-

```
3E68 28 C2 54 4E 28 D0 53 57 (BTN(PSW
3E70 D4 52 4F 4E D4 52 4F 46 TRONTROF
3E78 46 80 FF FF FF FF FF FF F.......
›
```

Note The bytes above containing FF may be displayed as 00.

The end of table marker (80 hex) is now at location 3E79.

Auxiliary word count

The first byte of the auxiliary reserved word table (PTR 3) contains the total number of words in the table. As we have now added 2 more words this byte must have its value

increased. There were 25 words originally in the table, so remembering that it now starts at address 3E01 enter:-

M3E01 <ENTER>
and the display will show:-

3E01 19

19 being the hex equivalent of 25. So with the cursor flashing over the 1 of 19 enter:-

1B. <ENTER>
which increases the word count from 25 to 27 as 1B hex = 27 decimal.

Adding Routine addresses

The next task is to add the 2 addresses into the auxiliary address table (PTR 8) where we will write the code for these respective routines. The table has been moved up to start at location 3F80 hex (16256 dec) and should end at 3FB1 (16305). To check enter:-

T3FA8 3FB7 <ENTER>
and the display will show the last few bytes of the address table (The 6 bytes with FF may be shown as 00):-

```
3FA8 F8 2B 9C 2F 34 2C 4C 2C x+./4,L,
3FB0 FA 35 FF FF FF FF FF FF z5......
>
```

This time we are not looking at words but actual addresses in memory, therefore the ASCII characters to the right of each line are insignificant. The addresses are in pairs, so taking the last 2 at 3FB0 and 3FB1 we get FA and 35. Addresses stored in memory are always in reverse order, so this pair of bytes results in the address 35FA. As it is the last address in the table in can be quite rightly

assumed that it is the address of the routine for the last word in the auxiliary word table, which was the basic reserved word PSW. Working backwards on the line above one will see the pairs of bytes equal the following addresses: 2C4C 2C34 2F9C and 2BFB. These 4 addresses will be the locations of the other words which were displayed at the end of the auxiliary word table: BTN(ADC KEY and RST.

Our routines for TRON and TROFF are going to be placed in the now redundant area of memory where these tables used to sit, remember we filled it with zeroes. The last byte in the address table is 3FB1 (16305), so our 2 addresses must be placed at 3FB2/3 and 3FB4/5 (16306/7 and 16308/9). These are 3BC5 for TRON and 3BCC for TROFF. Enter:-

```
M3FB2 <ENTER>
```

The cursor will be flashing over the first figure, and remembering that each pair must be entered in reverse order, enter:-

```
C53BCC3B. <ENTER>
```

Adding the routines

Before we actually write the routines for TRON and TROFF it is a good idea to check that we have altered the tables correctly so far. Before we started altering the tables had we entered either TRON or TROFF, whilst in basic, the interpreter would have displayed the 'Syntax Error' message as it would not have recognised either word as a valid command.

Before going any further we should run a check to discover if these words are now accepted as valid. If we enter the hex code C9, which is a return instruction in assembly language, to the starting addresses of both routines and

subsequently enter either command, no error message should be displayed, just the 'Ready' message. Go back into basic by entering:-

X ‹ENTER›
Now enter:-
POKE &3BC5,&C9: POKE &3BCC,&C9 ‹ENTER›
and then enter either
TRON ‹ENTER›
or
TROFF ‹ENTER›
If 'Ready' was displayed immediately this means the basic interpreter checked that the word was valid with the auxiliary word table, then got the respective address, from the auxiliary address table, of where in memory it should go to in order to carry out the instruction. It subsequently called on either 3BC5 or 3BCC, depending on if TRON or TROFF had been entered, and once there found it should make an immediate return. If the 'Ready' message has been displayed then all is well and you should continue. If, on the other hand, you got an error message or the Einstein dropped into DOS instead, then it's back to square one, so go back into MOS and tabulate out the areas we have changed, and check them.

The following entries are made under MOS, so if you are in basic enter MOS ‹ENTER›. Once again only enter what is shown under the TO ENTER column. The COMMENTS column will also show the assembly language mnemonics of the entries, so don't worry if you cannot fully understand the COMMENTS column, it is included as an aid to those boffins who have a grasp of machine language. The lines overleaf which start with 3BC5 and 3BCC are shown as containing the value C9. They will only have this value if one POKEd in the line at the top of this page, otherwise they should display as 00.

```
DISPLAYED  TO ENTER    COMMENTS

MOS 1.2
Ready
>          M3BC5
3BC5 C9    3E01        LD A,01         ;TRON. Load A with 1
3BC7 00    32CB3B      LD (3BCB),A     ;set TRON flag
3BCA 00    C9          RET             ;flag set, so return
3BCB 00    00          NOP             ;TRON flag 0=off 1=on
3BCC C9    AF          XOR A           ;TROFF. Zero A
3BCD 00    18F8        JR 3BC7         ;reset TRON flag to off
3BCF 00    ED535101    LD (0151),DE    ;current line number
3BD3 00    3ACB3B      LD A,(3BCB)     ;load A with TRON flag
3BD6 00    B7          OR A            ;test it
3BD7 00    C8          RET Z           ;if 0 then return
3BD8 00    E5          PUSH HL         ;save where we are
3BD9 00    2A3701      LD HL,(0137)    ;current cursor pos.
3BDC 00    E5          PUSH HL         ;save cursor pos.
3BDD 00    110000      LD DE,0000      ;DE=top left position
3BE0 00    CDC622      CALL 22C6       ;place cursor from DE
3BE3 00    CDB802      CALL 02B8       ;print following messg.
3BE6 00    4C696EE5    DB "Line"       ;L,i,n,e+80 hex
3BEA 00    2A5101      LD HL,(0151)    ;HL= current line no.
3BED 00    CDFF02      CALL 02FF       ;print HL
3BF0 00    CDB802      CALL 02B8       ;print following messg.
3BF3 00    202020A0    DB "    "       ;sp,sp,sp,sp+80 hex
3BF7 00    3A3D01      LD A,(013D)     ;A=last key pressed
3BFA 00    FE31        CP 31h          ;was it less than 1?
3BFC 00    3812        JR C,3C10       ;if yes then don't slow
3BFE 00    FE38        CP 38h          ;was it 8 or more?
3C00 00    300E        JR NC,3C10      ;if yes then don't slow
3C02 00    D630        SUB 30h         ;deduct 30hex (48 dec)
3C04 00    87878787    ADD A,A*4       ;key pressed*16
3C08 00    47          LD B,A          ;load delay counter
3C09 00    0E00        LD C,0          ;
3C0B 00    0B          DEC BC          ;count down
3C0C 00    78          LD A,B          ;counter into A
```

```
DISPLAYED  TO ENTER    COMMENTS

3C0D 00    B1          OR C          ;reached zero yet?
3C0E 00    20FB        JR NZ,3C0B    ;not zero, loop back
3C10 00    D1          POP DE        ;DE=orig. cursor pos.
3C11 00    CDC622      CALL 22C6     ;place cursor from DE
3C14 00    E1          POP HL        ;restore where we were
3C15 00    C9          RET           ;ret. to execution loop
3C16 00    .           ;full stop=quit modify
```

There is one final modification which is made to the execution loop of XBAS, for it must be patched to call our routine to check on the condition of the TRON flag. Enter:-

```
M0DEA <ENTER>
```

and the display will show:-

```
      0DEA ED
```

with the cursor over the E of ED enter the following 4 bytes:-

```
CDCF3B00. <ENTER>
```

This calls our routine, and the 4 bytes which this call replaced were entered as the line at address 3BCF.

Saving the additions

Before we entered this routine we loaded in the expanded version of XBAS. As the size has not increased, only bytes within have been altered, we can save this modified version back to disc in the same manner as before. Beware, for if you save it back to the same disc and use the same filename it will overwrite the old file of that name on the disc. If you call it something else i.e. XBASMOD.COM this will result in the disc containing our first expanded version plus this expanded modified version. One could also alter

the initial screen message to a higher version number by referring to the example in the last section.

Press the <CTRL>+<BREAK> keys to enter DOS and enter the save command with a filename of your choice making sure it is no more than 8 characters in length, and end it with .COM:-

```
SAVE 65 XBASMOD.COM <ENTER>
```

Adding TRON with Basic

If the previous section was too much to cope with, here is a basic program which will alter memory for the TRON and TROFF commands to operate. Load in the expanded version of XBAS before entering the program. Afterwhich the program should be RUN which will alter memory. Then save the modified XBAS as shown on the previous page using DOS.

```
 10 T=0
 20 REM TRON/TROFF Routines
 30 FOR X=15301 TO 15381
 40 READ A: POKE X,A: T=T+A: NEXT
 50 IF T<> 8701 THEN PRINT"INCORRECT DATA":STOP
 60 DATA 62,1,50,203,59,201,0,175,24,248,237,83
 70 DATA 81,1,58,203,59,183,200,229,42,55,1,229
 80 DATA 17,0,0,205,198,34,205,184,2,76,105,110
 90 DATA 229,42,81,1,205,255,2,205,184,2,32,32
100 DATA 32,160,58,61,1,254,49,56,18,254,56,48
110 DATA 14,214,48,135,135,135,135,71,14,0,11,120
120 DATA 177,32,251,209,205,198,34,225,201
130 REM Add TRON & TROFF to Aux. word table
140 POKE 15984,212,82,79,78,212,82,79,70,70,128
150 POKE 15873,27:REM Increase word count
160 POKE 16306,197,59,204,59:REM Add addresses to table
170 POKE 3562,205,207,59,0:REM Alter exec loop
```

EDIT command

Although XBAS contains a very versatile LIST command in which various parameters can be added for a host of listing variations not found on most computers, this second utility command will list one line for editing purposes.

When a program has come to an unscheduled halt due an error, and we all get them, if one enters EDIT ‹ENTER› the erroneous line will be listed on screen with the cursor flashing at the end of that line. There is no need to enter the line number as XBAS keeps a record of faulty line numbers. This can only speed up that time consuming task of debugging programs. To list any other line, which had not caused the program to stop, say line 100, one would enter EDIT 100 ‹ENTER› and line 100 would be displayed with the cursor at the end of the line. Once the line has been corrected hitting the ENTER key will store it in memory in the normal way.

This means one will not need to keep entering LIST followed by various parameters when just one line is to be listed. If the program has not stopped due to errors, or is stopped by the SHIFT+BREAK keys then entering EDIT on its own, without a line number, will simply do nothing and abort the edit routine, as there will not be a record of a faulty line. This will also be the case if the program stopped due to a line which contained the STOP command.

After one has entered this routine a grey function key could be assigned with the command.

```
KEY 6,"EDIT"+CHR$(13) ‹ENTER›
```

So that when a program encounters an error one could press just one key - F6 - to list the faulty line.

It is assumed that one is using the expanded version of XBAS and that the previous TRON/TROFF commands have already been added. For those readers who prefer entering the additions via a basic program the equivalent alterations under basic are shown in the alternate program following these MOS entries.

After adding the 2 previous commands the end marker for the auxiliary word table is currently at 3E79 (15593), so we must add our new reserved word EDIT and remember that the first character has 80 hex added to its value. The ASCII hex codes are thus:-

E = 45+80=C5
D = 44
I = 49
T = 54
+ end of table marker = 80

If in basic enter:-
MOS ‹ENTER›
and to prepare for the new word entry enter:-

M3E79 ‹ENTER›
the display will show:-

3E79 80
with the cursor flashing over the 8 enter:-

C544495480. ‹ENTER›
If one wants to check that the bytes have been entered correctly enter:-

T3E68 3E7F ‹ENTER›
which will display that area of memory.

The word count at the start of the auxiliary word table must now be incremented, as we did with TRON/TROFF.

Enter:-

M3E01 ‹ENTER›
and the cursor will be over the 1 of 1B.

```
3E01 1B
```

As we are only adding one word to the table this figure must be changed to 1C, it is increased from 27 to 28 decimal. Enter:-
1C. ‹ENTER›

Now the address of the routine EDIT must be added to the auxiliary address table. Tabulate the last few bytes of the table, enter:-
T3FA8 3FBF ‹ENTER›

```
3FA8 F8 2B 9C 2F 34 2C 4C 2C x+./4,L,
3FB0 FA 35 C5 3B CC 3B FF FF z5E;L;..
3FB8 FF FF FF FF FF FF FF FF ........
,
```

and one can see our 2 previous addresses for TRON and TROFF as the last 4 bytes before the FF codes begin. The bytes which show FF may be displayed as 00, but this is unimportant. Therefore we must place our EDIT address into bytes 3FB6/7 (16310/1).

The EDIT routine is to be placed in memory immediately after the finish of the TRON/TROFF routines, there will be no gaps between the two as this is not neccessary. The last byte, you may remember, was written to address 3C15 (15381) so the start address of EDIT will commence at the next byte, at 3C16. This address must be entered, in reverse order, to the end of the address table above, into 3FB6/7 so enter:-

M3FB6 ‹ENTER›

The cursor will be flashing over the first F of FF - or if your copy shows zeroes in these locations, over the first 0 of 00.
Enter:-

163C. ‹ENTER›

Once again it makes sense at this point to run a quick check in basic to find if the new command EDIT is now acceptable to XBAS. Enter:-

X ‹ENTER›
to go back to basic.

Now alter the first byte of the routine to C9 hex. Enter:-
POKE &3C16,&C9 ‹ENTER›
Now enter:-
EDIT ‹ENTER›
and if the Ready message is displayed immediately we will know that the command is now valid. If one doesn't get this result then check on the entries so far by using the MOS Tabulate command.

Now to enter the code, and only enter the characters shown under the TO ENTER column. If still in basic enter:-

MOS ‹ENTER›

DISPLAYED	TO ENTER	COMMENTS	
MOS 1.2			
Ready			
›	M3C16		
3C16 C9	CDAA02	CALL 02AA	;does line num follow?
3C19 00	2805	JR Z,3C20	;no just edit bad line
3C1B 00	CD3316	CALL 1633	;get line number in DE
3C1E 00	180B	JR 3C2B	;go find line in memory

```
DISPLAYED  TO ENTER    COMMENTS

3C20 00    3AA301      LD A,(01A3)  ;check error flag
3C23 00    FE00        CP 0         ;0=no error
3C25 00    2812        JR Z,3C39    ;if no error then exit
3C27 00    ED5BA401    LD DE,(01A4) ;error line num into DE
3C2B 00    CD3E03      CALL 033E    ;find line in memory
3C2E 00    3009        JR NC,3C39   ;doesn't exist, exit
3C30 00    C5          PUSH BC      ;BC=line addr in mem.
3C31 00    E1          POP HL       ;put addr into HL
3C32 00    23          INC HL       ;increment HL 4 times
3C33 00    23          INC HL       ;to jump over
3C34 00    23          INC HL       ;offset to next line
3C35 00    23          INC HL       ;and line number.
3C36 00    CD010A      CALL 0A01    ;List. DE=no. HL=addr.
3C39 00    C1          POP BC       ;lose return off stack
3C3A 00    C3AB07      JP 07AB      ;back to edit loop
3C3D 00    .                        ;full stop=quit modify
```

There are no further memory alterations for the EDIT routine, unless one wishes to show a version number in the initial message, which has previously been described. To save a version of expanded XBAS including this routine enter DOS then enter:-

SAVE 65 XBASMOD.COM <ENTER>

Or use another file name, provided it ends with .COM as the file type. As we have not increased the actual size of XBAS - it still finishes at 4200 hex as our new routines have been placed within that area of memory - the block size, 65, remains the same, this means that any further alterations shown in this book can be saved into the modified XBAS by the same method.

Adding EDIT with Basic

Load in the expanded version of XBAS that already has the TRON and TROFF routines added, then enter this program. Run the program which will make the alterations to memory and save the modified XBAS under DOS as shown overleaf.

```
 10 T=0
 20 REM EDIT Routine
 30 FOR X=15382 TO 15420
 40 READ A: POKEX,A: T=T+A: NEXT
 50 IF T<> 3433 THEN PRINT"INCORRECT DATA": STOP
 60 DATA 205,170,2,40,5,205,51,22,24,11,58,163
 70 DATA 1,254,0,40,18,237,91,164,1,205,62,3
 80 DATA 48,9,197,225,35,35,35,35,205,1,10,193,195,171,7
 90 REM Add EDIT to Aux. word table
100 POKE 15993,197,68,73,84,128
110 POKE 15873,28: REM Increase word count
120 POKE 16310,22,60: REM Add address to table
```

It is hoped that further additional routines will be published in magazines for the Einstein, however, it must be stated that they could be written to the same areas of memory that have been used here which means that if they are to be used with EDIT and TRON some alterations will be required. Wherever they are placed in memory they will, no doubt, need to alter the tables that we have altered and save a modified basic back to disc. If this happens it is advisable to compare techniques, you now know how the tables are moved and extended along with the auxiliary word count, so try adding the new routines to another section of the interpreter. EDIT is relocatable as all the jumps are relative, one would only need to alter the address table to the new start address. TRON is not so amenable in that the flag at 3BCB is referred to twice - at 3BC7 and 3BD3. Both lines would need to be altered so that they referred to the seventh byte in the routine, and 0DEA would need altering so that it jumped to the 11th byte as it does now.

6
Restoring erased programs

If one has ever erased a program from a disc and then regretted it this section could help.

Each disc contains a track which holds a directory of the files, or programs, held on that disc. The directory for each file is 32 bytes long and contains the file name and type (i.e. .XBAS .COM etc.,) along with the number of blocks on the disc the file is recorded on. When one enters the ERAse command all that happens is the first byte in the directory block for that particular file will be changed from 0 to E5 hex but the actual file, or program, will not be erased from the disc until another file has been saved to that disc. Hence this restoring procedure will only be effective if one has not recorded another program since erasing the one we wish to bring back to life.

Although altering sectors and tracks on a disc is usually beyond the average programmers scope and for most users the need would not arise, except that restoring the directory track could save time. So that one will be acquainted with the principles of restoring a program file one should first experiment with the example shown here.

We will use a copy of the master disc not the original master, so if you haven't done so already now is the time to make a back-up copy of the master disc, which is described on page 40 of the DOS/MOS Introduction manual. It should be remembered that one should always use the back-up master disc and keep the original in a safe place, away from magnetic fields which could damaged the recordings.

Load XBAS from the back-up disc then enter:-

DIR <ENTER>
and the disc directory will be listed. We will erase the basic program called INTRO, but as will be seen from the asterisk alongside its name on the directory it must first be UNLOCKed, which if one had used the LOCK command in the first place there would be no requirement for this section in the book! But, as most programmers are in a hurry, few make full use of the LOCK command. Enter:-
UNLOCK "INTRO" <ENTER>
followed by:-
ERA "INTRO" <ENTER>

If one now lists the directory again, the program INTRO will not be listed as an existing file on the disc, enter:-
DIR <ENTER>
and apparently it no longer exists.

Try entering:-
LOAD "INTRO" <ENTER>
and the 'No File Error' message is screened. We have now reached the stage where although the program INTRO is still recorded on the disc, the directory does not think it is but we can alter one byte to bring it back to life.

To alter tracks on a disc we must enter the Machine Operating System (MOS) by entering:-
MOS <ENTER>

The DOS/MOS Introduction manual describes how disc tracks, and sectors within those tracks, can be read from the disc into memory on page 12. Tracks 0 and 1 contain the Disc Operating System (DOS). Track 2 contains the directory for the disc and this is the track we are interested in. Each track is divided into 10 sectors and each sector contains 512 bytes.

The directory always starts at track 2 sector 0 and can grow in size, depending on the amount of files on the disc, up to the end of sector 3 which is 4 sectors or 2048 bytes.

The display should be showing the cursor flashing next to the chevron shape as below:-

```
MOS 1.2
Ready
>
```

Enter the following:-

R8000 87FF 00 02 <ENTER>
And the disc will be read and its contents stored in memory locations 8000 to 87FF hex (32768 to 34815 dec). This equals 2048 bytes which, as each sector contains 512 bytes, will equal 4 sectors. In the line above the 00 signifies the sector number to begin reading and the 02 is the track number. Although spaces between each parameter have been shown these are optional, one could have entered the above as:-
R800087FF0002 <ENTER>
but the first example with spaces is clearer.

Now we inspect the memory contents which have been copied from the directory track by entering:-

T8000 807F <ENTER>
and the first 128 bytes from 8000 to 807F hex (32768 to 32895 dec) will be displayed in rows of 8 bytes each. On the right of each line is displayed the ASCII characters for the values on the left. This (T)abulation of memory contents is described on page 12 of the DOS/MOS manual. We could have entered a third parameter signifying the amount of bytes to be displayed on each line, but omitting this figure will simply list out 8 bytes for each.

One will see the file names on the right of the display and a typical display of a master disc could appear thus:-

```
8000 00 58 42 41 53 20 20 20 .XBAS
8008 20 C3 4F 4D 00 00 00 7A  COM...z
8010 01 00 02 00 03 00 04 00 ........
8018 05 00 06 00 07 00 08 00 ........
8020 00 42 41 43 4B 55 50 20 .BACKUP
8028 20 C3 4F 4D 00 00 00 0D  COM....
8030 09 00 00 00 00 00 00 00 ........
8038 00 00 00 00 00 00 00 00 ........
8040 00 43 4F 50 59 20 20 20 .COPY
8048 20 C3 4F 4D 00 00 00 28  COM...(
8050 0A 00 0B 00 0C 00 00 00 ........
8058 00 00 00 00 00 00 00 00 ........
8060 00 4C 4F 47 4F 20 20 20 .LOGO
8068 20 C3 4F 4D 00 00 00 60  COM....
8070 0D 00 0E 00 0F 00 10 00 ........
8078 11 00 12 00 13 00 14 00 ........
```

One can see how each file takes up 4 lines of the directory space (4x8= 32 bytes).

We are searching for the file name INTRO, but as it has not yet been found we continue the search by entering:-
T8080 80FF <ENTER>

On the master disc copy used for this example the file INTRO was found at address 80C0, but your version may differ, so to continue searching enter the parameters for the next 128 bytes - T8100 817F <ENTER>
or even further by - T8180 81FF <ENTER>
When it is found it will appear similar to below:-

```
80C0 E5 49 4E 54 52 4F 20 20 eINTRO
80C8 20 58 42 53 00 00 00 19  XBS....
80D0 25 00 26 00 00 00 00 00 %.&.....
80D8 00 00 00 00 00 00 00 00 ........
```

The byte which signifies whether the file exists is the byte before the file name. In the example it is at 80C0 and has the value E5. One can see from the other files listed that they all have this first byte set to zero as they are all active. By using the MOS (M)odify command we can alter this byte quite simply. Enter:-

M80C0 <ENTER>

and the display will show:-

80C0 E5

with the cursor flashing over the E. Overwrite these 2 figures with 2 zeroes by entering:-

00. <ENTER>

And don't forget the full stop after the 00 which quits the modify mode and returns to normal.

To ensure that the byte has actually changed value from E5 to 00 enter the tabulate command again:-

T8080 80FF <ENTER>

Now all that remains is to write back to the disc the same amount of bytes we read in earlier, as although we have altered memory we must still record the alteration back to the disc. Enter:-

W8000 87FF 00 02 <ENTER>

If one finds that after entering the above line the 'Disc: No Drive' error message is screened don't despair as this occurred on our test machine, simply enter the line again only this time omit the spaces between the parameters. If the writing back to disc was successful one will find that after a few seconds the chevron and cursor will be displayed on the next screen line.

To return to basic either enter:-

Y <ENTER>

or to reboot DOS press the <CTRL>+<BREAK> keys, after which

one will need to enter XBAS <ENTER> to load in basic. Once back in basic entering the DIR instruction should display that the program INTRO is now alive and well and listed as an existing file on the disc. Try running it to make sure.

Locked Files

Files which have been locked have the value 80 hex (128 dec) added to the first character of their file type - the top bit is set. Referring back to our tabulated output of the directory one will see that the file types with COM in their name had their first letter - C - listed with a value of C3 hex. The normal ASCII value for the letter C is 43 hex (67 decimal). If one adds 80 hex (128 dec) to this value it becomes C3 hex (195) which denotes that the file is locked.

Most of the files on the master disc are locked, as one will see from any basic files in the directory - such as DEMO. The file type - XBAS - showed the first letter of the file type as 'X' but had a value of D8 hex (216 dec) which is the ASCII code for X - 58 hex (88 dec) - with 80 hex (128 dec) added. One can easily prove this point by referring to the tabulated output of the INTRO file which we unlocked before reading it into memory. Here the file type - XBAS - has the first letter stored with its correct ASCII code of 58 hex (88 dec), which denotes that it is an unlocked file.

System files

System files are those which exist on a disc but are not listed when DIR is entered. Page 35 of the DOS/MOS manual explains how a file, or program, can turned into a systen

7
Sorting data

Micro computers are excellent at playing games, especially the Einstein, but they really come into their own when faced with what we mortals would probably call monotonous tasks which appear to take ages to accomplish, as they can be performed in a fraction of the time on a micro.

There are several algorithms for sorting a series of numbers or names into order, the simplest and best known being the 'bubble' sort method. If you have a program which requires up to 40 items to be sorted then the bubble sort could be all you will need. If, however, you have more than 40 the second example of a sorting routine should be considered, as the bubble sort is painfully slow with higher quantities.

But first enter the following bubble sort program which generates random integers within the range 1 to 100 and sorts them into ascending order using the bubble sort method. You will be prompted to input the amount of numbers to sort, whereupon the program will randomly select and display the numbers on screen. On completion the numbers will be displayed in order and the elapsed time will be shown.

The program lines containing the bubble sort (220-280) are easily identified and therefore can be lifted and placed into your own program, remembering that the amount of numbers is represented by variable 'N' and the numbers themselves are placed in array 'B(N)' which is DIMensioned in line 110. The lines containing REMarks do not need to be entered.

```
100 CLS40:INPUT"TOTAL NUMBERS TO SORT ";N
110 DIM B(N)
120 FOR X = 1 TO N
130 B(X) = INT(RND(100))+1
140 PRINT B(X);
150 NEXT
160 PRINT:PRINT"THESE ARE THE NUMBERS TO SORT"
170 PRINT:PRINT"THE TIME STARTS NOW"
180 TI$="000000"
190 REM
200 REM ******* BUBBLE SORT *******
210 REM
220 FOR T = N TO 1 STEP -1:M=0
230 FOR S = 1 TO T
240 IF B(S)<= M THEN 260
250 M=B(S):L=S
260 NEXT S
270 SWAP B(L),B(T)
280 NEXT T
290 REM
300 REM ******* END OF SORT *******
310 REM
320 A$=TI$
330 FOR X= 1 TO N
340 PRINT B(X);
350 NEXT X
360 PRINT:PRINT"TIME TAKEN TO SORT";N;" NUMBERS WAS"
370 PRINT: PRINTA$
380 END
```

If one runs the program and enters different amounts of data to be sorted it should be clear that a faster, more complicated, routine is required for higher quantities. If one entered a higher figure than 200 one could probably have made a cup of tea whilst waiting for the sort to finish. So clearly if one needs to sort a fairly high number of data items then a faster method must be used.

The following faster sort routine uses the existing lines of the program to generate and display the data but line 190 will divert it to the faster routine and line 540 returns for the sorted printout.

```
190 GOTO 390
390 DIM SL(12),SR(12)
400 S=1: SL(1)=1: SR(1)=N
410 L=SL(S): R=SR(S): S=S-1
420 X=L: J=R: Y=B(INT((L+R)/2))
430 IF B(X)<Y THEN X=X+1:GOTO 430
440 IF Y<B(J) THEN J=J-1:GOTO 440
450 IF X>J THEN 480
460 SWAP B(X),B(J)
470 X=X+1: J=J-1
480 IF X<=J THEN 430
490 IF X>=R THEN 510
500 S=S+1: SL(S)=X: SR(S)=R
510 R=J
520 IF L<R THEN 420
530 IF S<> 0 THEN 410
540 GOTO 320
```

NOTE To alter the sort from low/high to high/low alter the following lines:-

```
430 IF B(X)>Y THEN X=X+1: GOTO 430
440 IF Y>B(J) THEN J=J-1: GOTO 440
```

No. of items	Bubble sort	Faster sort
20	1 sec.	1 sec.
30	3 secs.	2 secs.
40	5	3
50	8	4
60	12	6
80	20	8

cont..

No. of items	Bubble sort	Faster sort
100	31	10
200	2 mins 00	23
300	4 mins 26	36
400	7 mins 51	50
500	12 mins 12	1 min 08

When testing it is quite possible to get slightly different timings as the order of the random numbers will differ, this example was an average for 6 tests for each amount of numbers. There are faster sort routines written in machine code, but these are unfortunately too complicated for inclusion in this book.

To alter the faster sort routine to cater for strings we first need to change the program to generate letters at random. The resulting strings will not be recognisable words simply strings of random letters of varying lengths. Alter or add these lines:-

```
110 DIM B$(N)
125 FOR A=1 TO RND(1)*10
130 B$(X) = B$(X)+CHR$(INT(RND(1)*26)+65): NEXT A
140 PRINT B$(X);" ";
190 GOTO 390
340 PRINT B$(X);" ";
420 X=L: J=R: Y$=B$(INT((L+R)/2))
430 IF B$(X)<Y$ THEN X=X+1: GOTO 430
440 IF Y$<B$(J) THEN J=J-1: GOTO 440
460 SWAP B$(X),B$(J)
```

8
Database

It was suggested that a decent sized program should be included in this book, but what type of program? Fast action games are in plentiful supply and would not necessarily appeal to every reader. A business program might appeal to even less, particularly as each business would need it to be tailor made to be much use. That leaves us with a database, and before some letter writers begin putting pen to paper to complain that the manual contains such a program, I will point out some differences.

The simple mailing lists are purely that, simple, and have been included to show how one can write data to either sequential and random access files on the Einstein, and they do the job! This random access database, however, is novel in that a text window is set up on the top right of the screen enabling one to enter details, not just names and addresses, as one would expect to see them laid out on a record card.

The window is 31 characters wide by 8 lines deep, totalling 248 characters per record, and one can enter characters and move the cursor around within this window. The ENTER key will move down a line but will not move out of the window if on the bottom line, similarly one can't enter text past the bottom right hand corner.

Once the record is visually correct one can exit the entry routine by pressing the <ESC> key. This differs from the usual data base type of programs which normally require input of: Input Name, Input Address etc, and you are stuck with the output format that was written into the program.

Apart from the top line, which is used as a key for sorting and always starts at the leftmost position, the remaining 7 lines can contain formatted text which can begin from any column. Once the text has been confirmed as O.K. any leading spaces on the top, key line, will be deleted and one will see the text being shifted to the leftmost position. The other 7 lines will always be displayed as 'what you see is what you get' which means if you entered line 2 with 3 spaces before any text that is how it will always be displayed, unless you wish to alter it at a later stage. The program can be used for any type of records, apart from an address book - such as club subscriptions, timetables, forward calendar or even a Vet's record book.

After entering and saving the following programs one must run the 'SETUP' program which allows one to enter a heading for each of the 8 lines which are displayed to the left of the window. These could be - Name, Address1, Address2, Phone etc., or Name, Subs due, Date due, Paid on, and so on. If you don't require line headings one can simply press the <ENTER> key when prompted.

A typical display is then screened with any line headings displayed. If one approves of the set up the program then asks for the entry of a file name which, and as one can have several files on disc, must not be the same as another data file. Here one should not enter the .DAT part of the file name as this will be added by the program, just enter a name up to 8 characters, such as ADDRFILE or CALENDAR. The set up program now creates the data file on disc, and records any headings one may have appointed to the 8 lines of the window. This program is only used when creating a new data file, which one could have several of.

Once a new file has been created one will always use the second program which is simply named 'DB' but can obviously be altered. Keeping the name to only 2 characters makes

entry simpler as, after booting up DOS, one could just enter 'XBAS DB' and the <ENTER> key for basic to load in and subsequently load and run the main program. This main program first asks for the name of the data file and, once again, one should just enter the name and not the .DAT part. The data file is read to discover how many entries it contains and the top lines of each entry are loaded into memory and stored, note that it does not read in the whole 248 characters of each record, just the 31 possible characters in the top line of each. Afterwhich the program drops into the menu which also displays the current data file name and total entries:-

```
        THE TOTAL ENTRIES ARE 200

          ON FILE NAMED CALENDAR


                MAIN MENU


          KEY 1 = ADD A RECORD

          KEY 2 = ALTER/CHECK RECORD

          KEY 3 = DELETE A RECORD

          KEY 4 = PRINTOUT RECORDS

          KEY 5 = SORT RECORDS

          KEY 0 = ENDS PROGRAM

Enter 0 to 5
```

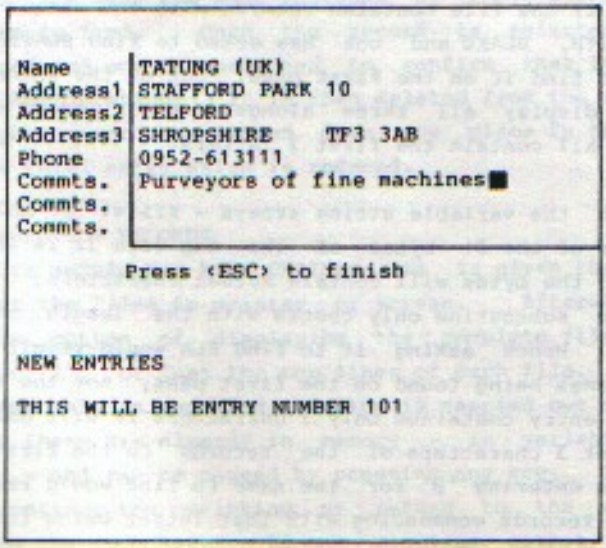
There are 3 further program modules which are called up by the 'DB' program, these are ENTRY, PRINTOUT and SORTING. The ENTRY module is the longest, due to it requiring common routines for the 3 options - Add, Alter and Delete a record. All 3 are numbered 400+ as they are CHAINed to the 'DB' program which uses HOLD 390.

The variable CHN holds the value of the last option selected. Hence after using the 'PRINTOUT' option CHN would contain 4. If, after returning to the menu, one selected option 4 again, the program would not CHAIN 'PRINTOUT', as it would know by testing the value of CHN that it was already CHAINed, and would goto line 400.

The menu options:-

1 ADD A RECORD

When the program is first run one will require this option first, as there will be no records on file. After pressing the '1' key the ENTER module will be CHAINed and control will pass to line 420 - the add a record routine. The screen will display the window and the cursor will be flashing within. Enter the required text and press the <ESC> key when completed. In the diagram below one can also see the names attributed to the 8 lines at the left of the screen.



Remember that any sorting or checking of entries is made with reference to the top line of the window. Therefore if using names it will assist sorting and checking if the surnames are entered first with any initials or firstnames second. Once <ESC> has been entered one is asked if the text is O.K. by entering Y or N. On answering N one is then given the option of altering the text by re-entering the window, or aborting it completely whereupon the program returns to the menu. If accepted the top 4 lines are loaded into string variable A$, the lower 4 lines into B$. The number of records on file is increased - variable EN - and this is added to the first record on the file while A$ and B$ are written to the end of the file.

2 ALTER/CHECK RECORD

To alter a record already on file one is asked to enter the name of the record. This refers to the text shown on the top line. If the program cannot find a record spelt exactly as has been entered it will alter the search by only comparing the first 3 characters entered. As an example if the file contains records with the names BLACK, BLACKSMITH, BLAKE and one has asked to find BLATSFORD it will not find it on the first pass. But on the second pass it will display all three alongside their record number, as they all contain the first 3 letters.

Although the variable string arrays - KY$(x) - hold the contents of the 31 bytes of the top line it is unlikely that all the bytes will contain actual characters. So the checking subroutine only checks with the length of your entry. Hence asking it to find BLA would result in the above names being found on the first pass, not the second. As your entry contained only 3 characters it will only test the first 3 characters of the records on the first pass. Moreover entering 'B' for the name to find would result in all the records commencing with that letter being found on the first pass. This will prove very useful.

If, however, one enters a name such as BOLSOVER, and there are no such names on record, and none which contain BOL as their first 3 letters, a third pass through the records would be made, and all names starting with that letter would be displayed. Although this does not take long - on test with 200 records the 3 passes, with 15 records starting with the letter B took 6 seconds - this was also the time taken to search for entries beginning with Z.

Once similar records have been found one is then asked to enter the record number of the entry to display, which is then loaded in from the disc and displayed in the window for alterations. The program keeps a record of the contents of the file displayed, and once one exits from this mode, by pressing the <ESC> key, it checks if the entry has been altered, and if it has it is recorded back to disc, as there is no point in recording back an unaltered record.

3 DELETE A RECORD

This section uses the same routines as above for entry of the name to find. Once the record is selected it is displayed and one is prompted to confirm that it is the record for deletion. It is then deleted from the disc and all higher records are moved down one place in the file, and the total entry value is reduced.

4 PRINTOUT RECORDS

Once this module has been CHAINed one is given the option to print the files to printer or screen. Afterwhich one has the option of displaying the complete files, by entering 'A', or just the top lines of each file. If the top line only is required printing is carried out straight away as these are already in memory - in variable array KY$(x) - and can be paused by pressing any key. One can then continue the printing or return to the menu by pressing the zero key. If one requires output of the

complete files then these are printed as they are loaded in from disc, but this option takes a little longer. Once again the printout can be halted as above but here one also has the option of moving backwards through the file.

5 SORT RECORDS

When there are a number of records on file it will be a good idea to sort them into order. The files are sorted by whatever is entered on the top lines. Whether your records show names or numbers on the top line they will be sorted by their ASCII codes. Hence B is larger than A but A is larger than the number 2. Records which show lower case - small - letters will be listed as higher than those using upper case - capital - letters. Refer to the ASCII code chart if you aren't sure on this point.

The sorting routine is fairly quick, even sorting 200 names isn't too slow. The routine is similar to that shown in the last section except that the strings aren't SWAPped, only the subscripts. This cuts down the time taken due to the strings not being moved about in memory. The slowest part of the sort module is when each string is read from one file and placed in the ordered position in a temporary file. 200 entries will take nearly 2 minutes, so don't use it too often. Once recording is completed one will have 2 data files on the same disc so the routine erases the old file and renames the temporary file, which has been recorded in the sorted order, to that of the original before returning to the menu.

In its present form the program will not accept commas when inputting a record, but one can alter this by using the SEP command. The program has been tested with 200 records, but if one wishes to extend this number, without any guarantees, one should increase the DIM variables in line 40 of the program 'DB'. These are KY$(x),SL(x),SR(x) and

TP(x). Variable TF is dimensioned to 100 and is only used when searching for entries. It may only need increasing if more than 100 entries are similar to each other, which is doubtful.

No error traps have been included for experience shows that these can lead the unwary to confusion when they haven't designed the program themselves, but once the program is up and running correctly, add them by all means. If one finds the program doesn't behave and gives erroneous results then check each program with the listings, I know it can be boring but one would be surprised of the number of times one can look at a line and miss an error, so let someone else read it too.

If one has appended the TRON routine shown in the basic additions chapter this can be used, but should be avoided during record entries as the routine reads the contents of the top screen line which will corrupt the records. If TRON was not entered before the program started one can stop the program, by entering ‹SHIFT› + ‹BREAK› enter 'TRON' ‹ENTER› and continue the program from where it stopped by entering 'CONT' ‹ENTER›. Another thought is to add lines with the STOP command so one can tell where a program has reached. They can soon be deleted if the error hasn't occurred before that line is encountered, or one can enter 'CONT' to continue providing lines have not been altered while the program was STOPped. If you find a bug in one of the module programs - a program other than 'DB' - and correct it, remember that 'DB' is still latched onto the front end - up to line 390 - so delete these lower lines before saving the module back to disc. And finally if one has added additional lines, for different colours etc., do not renumber the programs until they are working as this can make back tracking almost impossible.

Have a nice da..ta base.

```
  5 REM ::::::: SETUP PROGRAM
 10 ORIGIN 0,0:TCOL 15,0
 20 CLS40:BCOL 8: GCOL 15,0
 30 PRINT"SET UP PROGRAM"
 40 PRINT:PRINT"THERE ARE 8 LINES TO EACH RECORD"
 50 PRINT:PRINT"ANY OR ALL THE LINES MAY HAVE A NAME "
 60 PRINT"UP TO 8 CHARACTERS LONG TO USE AS A"
 70 PRINT"REFERENCE WHICH WILL BE DISPLAYED EACH"
 80 PRINT"TIME THE FILE IS USED"
 90 PRINT:PRINT
100 PRINT"YOU WILL BE PROMPTED FOR THE 8 NAMES"
110 PRINT"IF NO NAME IS TO USED"
120 PRINT"PRESS THE <ENTER> KEY"
130 PRINT
140 FOR A=0 TO 7
150 PRINT"NAME FOR LINE";A+1 ;: INPUT" ";N$(A)
160 IF LEN(N$(A))>8 THEN PRINT "ONLY 8 CHARACTERS":BEEP1
0:GOTO 150
170 IF LEN (N$(A))<8THEN N$(A)=N$(A)+" ":GOTO 170
180 NEXT
190 BCOL 4:CLS
200 DRAW 53,191 TO 53,127 TO 256,127
210 FOR A=0 TO 7
220 PRINT N$(A)
230 NEXT
240 PRINT@ 0,9;"THIS IS HOW THE DISPLAY WILL APPEAR"
250 PRINT"ARE THE NAMES O.K.  (Y/N)
260 A$=INCH$: IF A$="Y" OR A$= "y" THEN 280
270 GOTO 10
280 PRINT:PRINT"WHAT IS THE NAME OF THE DATA FILE TO"
290 PRINT"BE CALLED?  8 CHARACTERS IS MAXIMUM"
300 INPUT FILE$
310 L=LEN(FILE$):IF L>8 THEN 280ELSE 320
320 FOR A=0 TO 7
330 NAME$=NAME$+N$(A):NEXT
340 NAME$="000"+NAME$
350 CREATE FILE$+".DAT",FD$,127
360 PRINT# FD$,0;NAME$
370 CLOSE
```


JAVE "SETUP"

```
  10 REM ::::::: DB PROGRAM..Once a file has been set up
always run this program first"
  20 ORIGIN 0,0: BCOL 8: TCOL 15,0: CLS
  30 PRINT"DATABASE LOADING MODULE"
  40 DIM KY$(200),TF(100),SL(200),SR(200),TP(200)
  50 PRINT: PRINT "ENTER NAME OF DATA FILE"
  60 INPUT FILE$: FILE$=FILE$+".DAT"
  70 OPEN FILE$,FD$,127
  80 INPUT# FD$,0;NAME$
  90 EN=VAL(LEFT$(NAME$,3))
 100 FOR A=0 TO 7
 110 N$(A)=MID$(NAME$,A*8+4,8)
 120 NEXT A
 130 IF EN =0 THEN 180
 140 FOR A=1 TO EN
 150 INPUT # FD$,A*2-1;A$
 160 KY$(A)=LEFT$(A$,31)
 170 NEXT A
 180 CLOSE
 190 IOM 5,0: FMT3,0
 200 A=SIZE: BCOL 4: TCOL 15,0: CLS
 210 PRINT TAB(7);"THE TOTAL ENTRIES ARE ";EN
 220 PRINT:PRINT TAB(10);"ON FILE NAMED ";LEFT$(FILE$,LEN(
FILE$)-4)
 230 PRINT@ 13,6;"MAIN MENU"
 240 PRINT@ 13,7;MUL$(CHR$(230),9)
 250 PRINT@ 8,10;"KEY 1 = ADD A RECORD"
 260 PRINT@ 8,12;"KEY 2 = ALTER/CHECK RECORD"
 270 PRINT@ 8,14;"KEY 3 = DELETE A RECORD"
 280 PRINT@ 8,16;"KEY 4 = PRINTOUT RECORDS"
 290 PRINT@ 8,18;"KEY 5 = SORT RECORDS"
 300 PRINT@ 8,20;"KEY 0 = ENDS PROGRAM"
 310 PRINT@ 0,22;"Enter 0 to 5 ";
 320 C=INCH: IF C<48 OR C>53 THEN BEEP:GOTO 310
 330 C=C-48:IF C=0 THEN END: ELSE IF CHN=0 THEN GOTO 350
 340 IF CHN<4 AND C<4 OR CHN=C THEN GOTO 400
 350 CHN=C: HOLD 400
 360 ON C GOTO 370,370,370,380,390
```

```
370 CHAIN "ENTRY": REM USED BY OPTION 1,2 AND 3
380 CHAIN "PRINTOUT"
390 CHAIN "SORTING"
400 STOP: REM IF IT GETS HERE WE ARE IN TROUBLE
```

End of program named DB

The following program must be saved as "ENTRY"

```
400 ON C GOTO 420,510,640
410 REM ************* ADD RECORD
420 BCOL 2: TCOL 15,0: GCOL 15,0
430 HEAD$="NEW ENTRIES"
440 EN=EN+1: X=EN
450 CLS40: PRINT@ 0,13; HEAD$
460 PRINT@ 0,15;"THIS WILL BE ENTRY NUMBER ";X
470 GOSUB 1310
480 IF Q=1 THEN EN=EN-1: GOTO 200:REM ENTRY ABORTED
490 GOSUB 1690: GOSUB 1760: GOTO 200
500 REM ********** ALTER/CHECK RECORD
510 HEAD$= "CHECK/ALTER SECTION"
520 BCOL 8: GOSUB 880
530 IF L$="0" THEN 200
540 IF X=0 THEN 200
550 TA$=A$: TB$=B$: REM CHECK IF ALTERED
560 GOSUB 1310: REM DISPLAY FILE
570 IF Q=1 THEN 520
580 GOSUB 1690
590 IF A$=TA$ AND B$=TB$ THEN GOTO 520
600 KY$(X)=LEFT$(A$,31)
610 OPEN FILE$,FD$,127
```

```
620 GOSUB 1800: GOTO 520
630 REM ********** DELETE RECORD
640 HEAD$= "DELETIONS"
650 BCOL 3: TCOL 1,0: GOSUB 880
660 IF L$="0" THEN 200
670 IF X=0 THEN 200
680 PRINT@ 8,9;"DELETE THIS ENTRY?  (Y/N)"
690 A$=INCH$: IF A$="Y" OR A$="y" THEN 710
700 GOTO 650
710 PRINT@ 8,11;" ARE YOU SURE  (Y/N)": BEEP
720 A$=INCH$: IF A$="Y" OR A$="y" THEN 740
730 GOTO 650
740 EN=EN-1: OPEN FILE$,FD$,127
750 IF X>EN THEN 810
760 FOR A=X TO EN
770 INPUT# FD$,A*2+1;A$,B$
780 PRINT# FD$,A*2-1;A$,B$
790 KY$(A)=KY$(A+1)
800 NEXT A
810 NAME$=RIGHT$(NAME$,LEN(NAME$)-3)
820 NAME$=RIGHT$("00"+STR$(EN),3)+NAME$
830 PRINT@ FD$,0;NAME$
840 CLOSE
850 GOTO 650
860 REM
870 REM ************** GET RECORD
880 CLS
890 PRINT HEAD$
900 GOSUB 1020: REM FIND IT
910 IF ASC(L$)=48 THEN RETURN
920 PRINT:PRINT "Enter record number and <ENTER> key"
930 PRINT "<0> and <ENTER> returns to menu "
940 INPUT X
950 IF X=0 THEN RETURN
960 IF X>EN THEN PRINT "RECORD";X;" DOES NOT EXIST":
GOTO 890
970 CLS:PRINT@ 0,15;HEAD$
```

```
 980 GOSUB 1850: REM LOAD FILE X+DISPLAY
 990 RETURN
1000 REM
1010 REM ******** FIND AND DISPLAY NAMES
1020 PRINT: PRINT"Enter NAME of entry and the <ENTER> KEY"
1030 PRINT: PRINT"OR  <0> and the <ENTER> key will quit"
1040 INPUT L$
1050 IF ASC(L$)=48 THEN RETURN
1060 GOSUB 1160
1070 IF FIND=0 THEN 1020
1080 CLS:PRINT HEAD$:PRINT
1090 PRINT "These are the entries found"
1100 FOR A= 1 TO FIND
1110 P=TF(A)
1120 PRINT "Rec."; P;TAB(9);KY$(P)
1130 NEXT A: RETURN
1140 REM
1150 REM******** CHECK NAME ENTERED WITH RECORDS
1160 FIND=0
1170 NL=LEN(L$)
1180 FOR A=1 TO EN
1190 IF LEFT$(L$,NL)=LEFT$(KY$(A),NL) THEN FIND =FIND+1:
TF(FIND)=A
1200 NEXT A: IF FIND <>0 THEN RETURN
1210 IF NL=3 THEN GOTO 1260
1220 IF NL=1 THEN GOTO 1280
1230 PRINT:PRINT"CANNOT FIND EXACT MATCH.."
1240 PRINT"NOW TRYING SIMILAR ENTRIES"
1250 NL=3: GOTO 1180
1260 PRINT:PRINT "NOT THERE.  NOW SEARCHING FOR SAME"
1270 PRINT "STARTING LETTER":BEEP: NL=1: GOTO 1180
1280 PRINT:PRINT "THERE ARE NO ENTRIES THAT BEGIN WITH "
1290 PRINT"THAT LETTER....":BEEP5: RETURN
1300 REM ********** DISPLAY FILE
1310 Q=0:PRINT@ 0,0;
1320 FOR A=0 TO 7:PRINTN$(A):NEXT
1330 DRAW 52,191 TO 52,127 TO 256,127
```

```
 1340 TCOL 4,15
 1350 PRINT@ 8 ,9;"Press <ESC> to finish              ":TCOL
15,0
 1360 XP=9:YP=0
 1370 IF YP>7THEN YP=7
 1380 IF YP<0 THEN YP=0
 1390 IF XP>39THEN XP=9 :YP =YP+1:GOTO 1370
 1400 IF XP=39 AND YP=7 THEN BEEP
 1410 IF XP<9 THEN XP =39: YP=YP-1:GOTO 1370
 1420 PRINT@ XP,YP;
 1430 A=INCH
 1440 IF A=13 THEN YP=YP+1: XP=9: GOTO 1370
 1450 IF A=4 THEN XP=XP+1: GOTO 1370
 1460 IF A=8 THEN XP=XP-1: GOTO 1370
 1470 IF A=10 THEN YP=YP+1: GOTO 1370
 1480 IF A=11 THEN YP=YP-1: GOTO1370
 1490 IF A=25 AND XP=9 THEN PRINT" ";CHR$(25):XP=39:YP=YP-
1:GOTO 1370
 1500 IF A=25 AND XP>9 THEN PRINT CHR$(25): XP=XP-1: GOTO
1370
 1510 IF A=27 THEN 1570
 1520 IF XP=39 AND YP=7 THEN BEEP:PRINTCHR$(A): GOTO 1370
 1530 IF A<>26 THEN 1560
 1540 IF A=26 AND ASC(MID$(SCRN$(YP),39,1))>32 THEN BEEP:
GOTO 1370
 1550 PRINT CHR$(A): GOTO 1370
 1560 PRINTCHR$(A):XP=XP+1:GOTO 1370
 1570 PRINT@  8,9;"IS THE ABOVE O.K.   (Y/N)       ": BEEP
 1580 A$=INCH$:IF A$="Y"OR A$="y"THEN 1610
 1590 PRINT@ 8,9;"<Y> TO RE-ALTER....<N> TO QUIT  "
 1600 A$=INCH$:IF A$="Y" OR A$="y" THEN 1340:ELSE Q=1: RET
URN
 1610 L1$=MID$(SCRN$(0),10,30)
 1620 FOR B=1 TO 30
 1630 IF LEFT$(L1$,1) <>" "THEN B=30: RETURN
 1640 IF LEFT$(L1$,1)=" " THEN L1$=RIGHT$(L1$,31)+" ": PRI
NT@ 9,0;L1$
```

```
 1650 NEXT
 1660 PRINT@0,22;"THERE MUST BE AN ENTRY ON THE TOP LINE":
BEEP20
 1670 PRINT@0,22;MUL$(" ",39):GOTO 1340
 1680 REM ********** LOAD UP STRING
 1690 A$="":B$="":KY$(X)=RIGHT$(SCRN$(0),31)
 1700 FOR A=0 TO 3
 1710 A$=A$+RIGHT$(SCRN$(A),31)
 1720 B$=B$+RIGHT$(SCRN$(A+4),31)
 1730 NEXT
 1740 RETURN
 1750 REM ********** SAVE RECORD X AND INCREASE COUNT
 1760 OPEN FILE$,FD$,127
 1770 NAME$=RIGHT$(NAME$,LEN(NAME$)-3)
 1780 NAME$=RIGHT$("00"+STR$(EN),3)+NAME$
 1790 PRINT# FD$,0;NAME$
 1800 PRINT# FD$,X*2-1 ;A$,B$
 1810 CLOSE
 1820 A$="": B$="": RETURN
 1830 REM END OF SAVER
 1840 REM ********** LOAD IN RECORD X
 1850 PRINT@ 0,0;
 1860 OPEN FILE$,FD$,127
 1870 INPUT# FD$,X*2-1;A$,B$
 1880 FOR B=1 TO 94  STEP 31
 1890 PRINT TAB(10);MID$(A$,B,31);CHR$(11):NEXT B
 1900 FOR B=1 TO 94  STEP 31
 1910 PRINT TAB(10);MID$(B$,B,31);CHR$(11):NEXT B
 1920 CLOSE
 1930 RETURN
```

End of program named

The following program must be saved as "PRINTOUT"

```
  400 CLS:HEAD$="PRINTOUT":PRINT HEAD$
  410 PRINT:PRINT "PRESS <SPACE> AT ANY TIME TO PAUSE"
- 420 TCOL 4,15: PRINT:PRINT"PRESS <P> FOR OUTPUT TO PRINT
ER"
- 430 PRINT:PRINT"ANY OTHER FOR OUTPUT TO SCREEN"
- 440 TCOL 15,0: P=INCH: PRINT: PRINT
- 450 IF P=80 THEN PRINT HEAD$;" TO PRINTER"
  460 PRINT"PRESS KEY <A> FOR COMPLETE FILE"
  470 PRINT:PRINT"PRESS ANY OTHER FOR FIRST LINE OF FILE"
  480 A=INCH: IF A<>65 THEN 730
  490 CLS:PRINT HEAD$
  500 POKE&013D,0
- 510 IF P=80 THEN PRINT#1
  520 OPEN FILE$,FD$,127
  530 FOR A=1 TO EN
  540 PRINT:PRINT MUL$("-",40);:PRINT
  550 INPUT# FD$,A*2-1;A$,B$
  560 PRINT"REC.";A;
  570 FOR B= 1 TO 94 STEP 31
  580 PRINT TAB(9);MID$(A$,B,31): NEXT B
  590 FOR B= 1 TO 94 STEP 31
  600 PRINT TAB(9);MID$(B$,B,31): NEXT B
  610 INPUT#0: B=KBD: IF B=0 THEN 690
  620 TCOL4,15: PRINT#0;"<0>=MENU..<1>=GO BACK..ANY OTHER=
CONT ";: TCOL 15,0
  630 B=INCH: PRINTCHR$(11);MUL$(" ",80);CHR$(11);CHR$(11)
;
- 640 IF P=80 THEN PRINT#1
  650 IF B=48 THEN CLOSE: GOTO 860
  660 IF B<>49 THEN 690
  670 A=A-2: IF A<1 THEN A=1: GOTO 540
  680 GOTO 540
  690 NEXT A: CLOSE
  700 TCOL 4,15: PRINT#0;"<1>PRINT AGAIN..ANY OTHER FOR ME
NU ";: BEEP: TCOL 15,0
```

```
  710 B=INCH: IF B=49 THEN 490
  720 GOTO 860
  730 CLS:PRINT:PRINT HEAD$
  740 POKE&013D,0
 -750 IF P=80 THEN PRINT#1
  760 FOR A=1 TO EN
  770 PRINT"REC.";A;TAB(9);KY$(A)
  780 S$=KBD$: IF ASC(S$)=0   THEN 830
  790 TCOL 4,15:PRINT#0;"<0>=MENU...ANY OTHER=CONT ";:BEEP:
TCOL 15,0
  800 B=INCH: IF B=48 THEN 860
  810 PRINT CHR$(11);MUL$(" ",39);CHR$(11)
 -820 IF P=80 THEN PRINT#1
  830 NEXT A
  840 TCOL 4,15: PRINT#0;"<1>=PRINT AGAIN..ANY OTHER FOR ME
NU ";:
  850 TCOL 15,0: B=INCH: IF B=49 THEN 730
  860 GOTO 200
```

End of program named PRINTOUT

The following program must be saved as "SORTING"

```
  400 HEAD$="SORTING"
  410 CLS:PRINT HEAD$
  420 FOR A=1 TO EN: TP(A)=A: NEXT
  430 S=1: SL(1)=1: SR(1)=EN
  440 L=SL(S): R=SR(S): S=S-1
  450 X=L: J=R: X$=KY$(TP(INT((L+R)/2)))
  460 IF KY$(TP(X))<X$ THEN X=X+1:GOTO 460
  470 IF X$<KY$(TP(J)) THEN J=J-1:GOTO 470
  480 IF X>J THEN 510
```

```
490 SWAP TP(J),TP(X)
500 X=X+1: J=J-1
510 IF X<=J THEN 460
520 IF X>=R THEN 540
530 S=S+1: SL(S)=X: SR(S)=R
540 R=J
550 IF L<R THEN 450
560 IF S<> 0 THEN 440
570 BEEP: A=SIZE: PRINT"ENTRIES ARE NOW SORTED"
580 PRINT:PRINT"WRITING UPDATED FILE TO DISC"
590 TEMP$="TEMP.DAT"
600 CREATE TEMP$,TP$,127
610 PRINT# TP$,0;NAME$
620 OPEN FILE$,FD$,127
630 FOR A=1 TO EN
640 INPUT# FD$,TP(A)*2-1;A$,B$
650 PRINT# TP$,A*2-1;A$,B$
660 KY$(A)=LEFT$(A$,31)
670 NEXT A
680 CLOSE
690 ERA FILE$
700 REN TEMP$ TO FILE$
710 GOTO 200
```

End of program named SORTING

You should now have 5 programs saved

1. SETUP name can differ
2. DB name can differ
3. ENTRY name cannot differ
4. PRINTOUT name cannot differ
5. SORTING name cannot differ

Appendix

Basic tokens and addresses of XBAS 4·2

The following list is of all the reserved words and their respective token values along with the hex routine addresses in the interpreter. Some tokens do not show routine addresses as they are handled within a parent routine - such as SPC is handled through the PRINT routine. The first section is for words in the standard reserved word and address tables. The starting addresses for both are stored in PTR 2 and PTR 6.

Res. word.	Token	Addr.	Res. word	Token	Addr.
SPC	6F	--	DATA	87	0E86
STEP	70	--	DEF	88	0C85
TAB	71	--	DEL	89	1098
TO	72	--	DIM	8A	18AE
THEN	73	--	DOKE	8B	0448
+	74	--	DRIVE	8C	2A23
-	75	--	ELSE	8D	0E86
^	76	--	END	8E	0A72
*	77	--	FOR	8F	0D76
/	78	--	GOSUB	90	0B8D
MOD	79	--	GOTO	91	0B9F
AND	7A	--	HOLD	92	10D2
OR	7B	--	IF	93	0E2E
XOR	7C	--	INPUT	94	0E99
>	7D	--	LET	95	0C32
=	7E	--	LIST	96	0957
<	7F	--	LOAD	97	2846
AUTO	80	102B	MGE	98	10FA
CHAIN	81	0ADE	MOS	99	2BD3
CLEAR	82	0B11	NEW	9A	0A96
CLOSE	83	298F	NEXT	9B	0D0E
CLS	84	2C61	OFF	9C	069E
CONT	85	0E6D	ON	9D	0BE6
CREATE	86	2987	OPEN	9E	2979

Res. word.	Token	Addr.	Res. word	Token	Addr.
OUT	9F	0479	ABS	C4	1D60
POKE	A0	0438	ASC	C5	148E
POP	A1	0BBD	ATN	C6	1E8A
PRINT	A2	0FA6	CHR$	C7	14A5
READ	A3	0EBE	COS	C8	1EC4
REM	A4	0E86	DEEK	C9	04B4
RENUM	A5	1170	EVAL	CA	160B
UNPLOT	A6	2CFA	EXP	CB	1E4A
RESTORE	A7	0E52	HEX$	CC	158B
RETURN	A8	0BD1	INP	CD	055F
RUN	A9	0AEC	INT	CE	1DC0
SAVE	AA	2897	LEN	CF	147F
PLOT	AB	2CFC	LN	D0	1B93
STOP	AC	0A65	LOG	D1	1B87
SWAP	AD	1A66	PEEK	D2	04BD
VERIFY	AE	28D7	POINT	D3	062D
WAIT	AF	047F	POS	D4	04A2
FMT	B0	066D	RND	D5	361F
APPEND	B1	364A	SCRN$	D6	156D
DIR	B2	3678	SGN	D7	1D6C
ERA	B3	3740	SIN	D8	1ECA
LOCK	B4	361F	SQR	D9	1DFD
REN	B5	3703	STR$	DA	153B
UNLOCK	B6	3638	TAN	DB	1EB0
MUSIC	B7	3145	VAL	DC	1523
CALL	B8	045F	LEFT$	DD	14BA
IOM	B9	0627	MID$	DE	14F2
NULL	BA	041D	RIGHT$	DF	14E9
PTR	BB	059A	ERR	E0	0576
SEP	BC	05F8	ERL	E1	04D7
SPEED	BD	0429	EOF	E2	06B9
WIDTH	BE	0604	FN	E3	0CA0
ZONE	BF	0611	INCH	E4	04E9
TI$	C0	30BE	KBD	E5	052A
TEMPO	C1	30EA	MUL$	E6	15D1
VOICE	C2	3104	NOT	E7	1820
PSG	C3	3089	PI	E8	0499
			SIZE	E9	04C4

The auxiliary word and address tables can be found by PTR 3 and PTR 8. These words are tokenised into 2 bytes, the first of which is always FF hex.

Res. word.	Token	Addr.	Res. word	Token	Addr.
DRAW	FF 80	2D14	FILL	FF 8D	2E2B
TCOL	FF 81	2CAB	VPOKE	FF 8E	348D
GCOL	FF 82	2CB0	VPEEK	FF CF	34BA
BCOL	FF 83	2C9C	VDOKE	FF 90	349F
SPRITE	FF 84	2EBB	VDEEK	FF D1	34C7
MAG	FF 85	2EAC	BEEP	FF 92	3075
SHAPE	FF 86	2E68	BIN$	FF 93	349F
ORIGIN	FF 87	2CEE	RST	FF 94	2BF8
ELLIPSE	FF 88	2D94	KEY	FF 95	2F9C
DOS	FF 89	3DFC	ADC	FF 96	2C34
RAD	FF 8A	34DA	BTN	FF 97	2C4C
DEG	FF 8B	34EC	PSW	FF 98	35FA
POLY	FF 8C	2D7C			

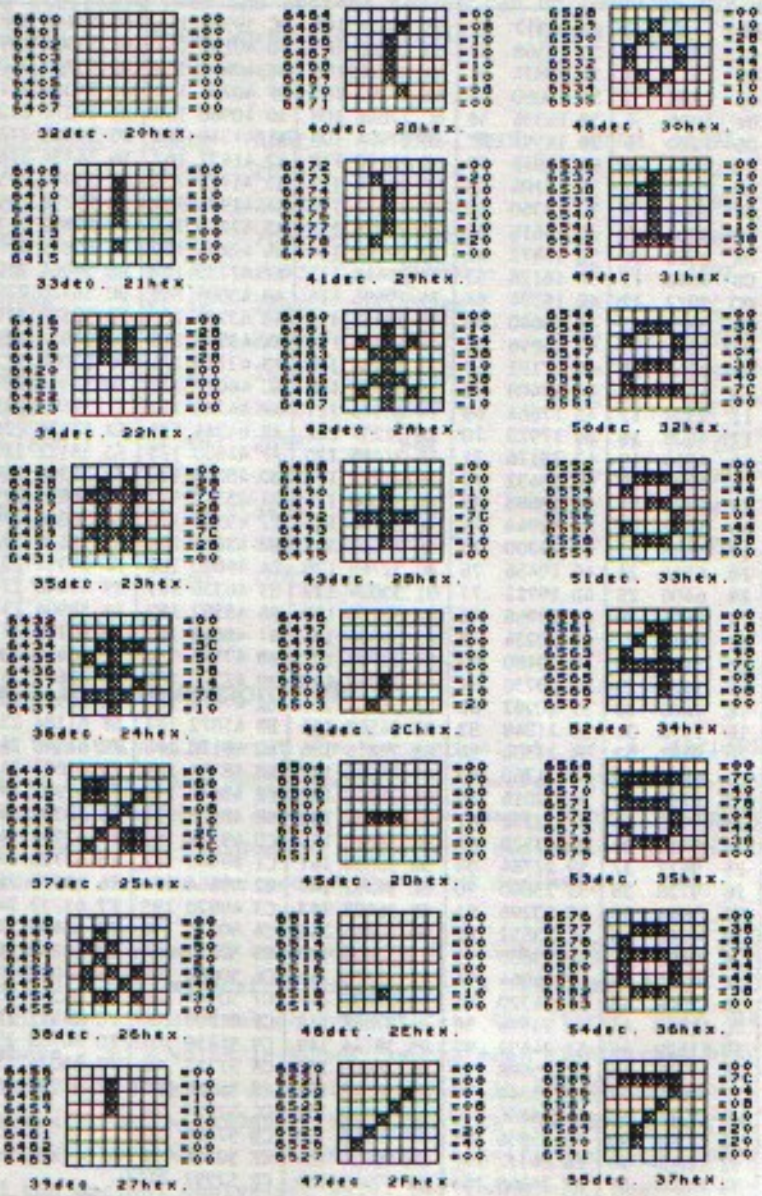
Hex/Dec conversion table

The Einstein will print the hex equivalent of a decimal number by entering ?HEX$(number), however this may not always be convenient so the following page shows a table for converting hexadecimal numbers to decimal. The first column is hex and the second and third decimal.

For example take the value of PTR 6, which is 3C67 hex, the start of the standard address table in XBAS 4.2. To calculate the decimal equivalent take the first pair of numbers - 3C - which is known as the most significant byte (MSB), and look up its decimal value in one of the centre columns. 3C = 15360. Now look up the second pair of numbers - 67 - known as the least significant byte (LSB), and the decimal equivalent will be found in one of the righthand columns. 67 = 103 so add the 2 results which is 15360 + 103 = 15463.

Hex	Dec MSB	Dec LSB	Hex	Dec MSB	Dec LSB	Hex	Dec MSB	Dec LSB	Hex	Dec MSB	Dec LSB	Hex	Dec MSB	Dec LSB
00	0	0	34	13312	52	68	26624	104	9C	39936	156	D0	53248	208
01	256	1	35	13568	53	69	26880	105	9D	40192	157	D1	53504	209
02	512	2	36	13824	54	6A	27136	106	9E	40448	158	D2	53760	210
03	768	3	37	14080	55	6B	27392	107	9F	40704	159	D3	54016	211
04	1024	4	38	14336	56	6C	27648	108	A0	40960	160	D4	54272	212
05	1280	5	39	14592	57	6D	27904	109	A1	41216	161	D5	54528	213
06	1536	6	3A	14848	58	6E	28160	110	A2	41472	162	D6	54784	214
07	1792	7	3B	15104	59	6F	28416	111	A3	41728	163	D7	55040	215
08	2048	8	3C	15360	60	70	28672	112	A4	41984	164	D8	55296	216
09	2304	9	3D	15616	61	71	28928	113	A5	42240	165	D9	55552	217
0A	2560	10	3E	15872	62	72	29184	114	A6	42496	166	DA	55808	218
0B	2816	11	3F	16128	63	73	29440	115	A7	42752	167	DB	56064	219
0C	3072	12	40	16384	64	74	29696	116	A8	43008	168	DC	56320	220
0D	3328	13	41	16640	65	75	29952	117	A9	43264	169	DD	56576	221
0E	3584	14	42	16896	66	76	30208	118	AA	43520	170	DE	56832	222
0F	3840	15	43	17152	67	77	30464	119	AB	43776	171	DF	57088	223
10	4096	16	44	17408	68	78	30720	120	AC	44032	172	E0	57344	224
11	4352	17	45	17664	69	79	30976	121	AD	44288	173	E1	57600	225
12	4608	18	46	17920	70	7A	31232	122	AE	44544	174	E2	57856	226
13	4864	19	47	18176	71	7B	31488	123	AF	44800	175	E3	58112	227
14	5120	20	48	18432	72	7C	31744	124	B0	45056	176	E4	58368	228
15	5376	21	49	18688	73	7D	32000	125	B1	45312	177	E5	58624	229
16	5632	22	4A	18944	74	7E	32256	126	B2	45568	178	E6	58880	230
17	5888	23	4B	19200	75	7F	32512	127	B3	45824	179	E7	59136	231
18	6144	24	4C	19456	76	80	32768	128	B4	46080	180	E8	59392	232
19	6400	25	4D	19712	77	81	33024	129	B5	46336	181	E9	59648	233
1A	6656	26	4E	19968	78	82	33280	130	B6	46592	182	EA	59904	234
1B	6912	27	4F	20224	79	83	33536	131	B7	46848	183	EB	60160	235
1C	7168	28	50	20480	80	84	33792	132	B8	47104	184	EC	60416	236
1D	7424	29	51	20736	81	85	34048	133	B9	47360	185	ED	60672	237
1E	7680	30	52	20992	82	86	34304	134	BA	47616	186	EE	60928	238
1F	7936	31	53	21248	83	87	34560	135	BB	47872	187	EF	61184	239
20	8192	32	54	21504	84	88	34816	136	BC	48128	188	F0	61440	240
21	8448	33	55	21760	85	89	35072	137	BD	48384	189	F1	61696	241
22	8704	34	56	22016	86	8A	35328	138	BE	48640	190	F2	61952	242
23	8960	35	57	22272	87	8B	35584	139	BF	48896	191	F3	62208	243
24	9216	36	58	22528	88	8C	35840	140	C0	49152	192	F4	62464	244
25	9472	37	59	22784	89	8D	36096	141	C1	49408	193	F5	62720	245
26	9728	38	5A	23040	90	8E	36352	142	C2	49664	194	F6	62976	246
27	9984	39	5B	23296	91	8F	36608	143	C3	49920	195	F7	63232	247
28	10240	40	5C	23552	92	90	36864	144	C4	50176	196	F8	63488	248
29	10496	41	5D	23808	93	91	37120	145	C5	50432	197	F9	63744	249
2A	10752	42	5E	24064	94	92	37376	146	C6	50688	198	FA	64000	250
2B	11008	43	5F	24320	95	93	37632	147	C7	50944	199	FB	64256	251
2C	11264	44	60	24576	96	94	37888	148	C8	51200	200	FC	64512	252
2D	11520	45	61	24832	97	95	38144	149	C9	51456	201	FD	64768	253
2E	11776	46	62	25088	98	96	38400	150	CA	51712	202	FE	65024	254
2F	12032	47	63	25344	99	97	38656	151	CB	51968	203	FF	65280	255
30	12288	48	64	25600	100	98	38912	152	CC	52224	204			
31	12544	49	65	25856	101	99	39168	153	CD	52480	205			
32	12800	50	66	26112	102	9A	39424	154	CE	52736	206			
33	13056	51	67	26368	103	9B	39680	155	CF	52992	207			

ASCII codes


32dec. 20hex.
40dec. 28hex.
48dec. 30hex.
33dec. 21hex.
41dec. 29hex.
49dec. 31hex.
34dec. 22hex.
42dec. 2Ahex.
50dec. 32hex.
35dec. 23hex.
43dec. 2Bhex.
51dec. 33hex.
36dec. 24hex.
44dec. 2Chex.
52dec. 34hex.
37dec. 25hex.
45dec. 2Dhex.
53dec. 35hex.
38dec. 26hex.
46dec. 2Ehex.
54dec. 36hex.
39dec. 27hex.
47dec. 2Fhex.
55dec. 37hex.

56dec. 38hex.

64dec. 40hex.

72dec. 48hex.

57dec. 39hex.

65dec. 41hex.

73dec. 49hex.

58dec. 3Ahex.

66dec. 42hex.

74dec. 4Ahex.

59dec. 3Bhex.

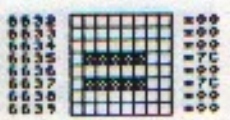
67dec. 43hex.

75dec. 4Bhex.

60dec. 3Chex.

68dec. 44hex.

76dec. 4Chex.

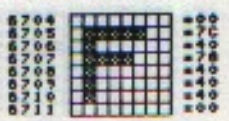


61dec. 3Dhex.

69dec. 45hex.

77dec. 4Dhex.

62dec. 3Ehex.



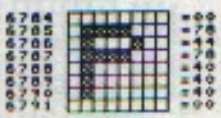
70dec. 46hex.

78dec. 4Ehex.

63dec. 3Fhex.

71dec. 47hex.

79dec. 4Fhex.



80dec. 50hex.

88dec. 58hex.

96dec. 60hex.

81dec. 51hex.

89dec. 59hex.

97dec. 61hex.

82dec. 52hex.

90dec. 5Ahex.

98dec. 62hex.

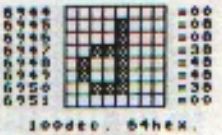
83dec. 53hex.

91dec. 5Bhex.

99dec. 63hex.

84dec. 54hex.

92dec. 5Chex.



100dec. 64hex.

85dec. 55hex.

93dec. 5Dhex.

101dec. 65hex.

86dec. 56hex.

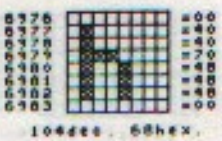
94dec. 5Ehex.

102dec. 66hex.

87dec. 57hex.

95dec. 5Fhex.

103dec. 67hex.



104dec. 68hex.

112dec. 70hex.

120dec. 78hex.

105dec. 69hex.

113dec. 71hex.

121dec. 79hex.

106dec. 6Ahex.

114dec. 72hex.

122dec. 7Ahex.

107dec. 6Bhex.

115dec. 73hex.

123dec. 7Bhex.

108dec. 6Chex.

116dec. 74hex.

124dec. 7Chex.

109dec. 6Dhex.

117dec. 75hex.

125dec. 7Dhex.

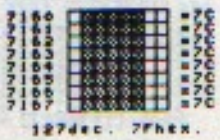
110dec. 6Ehex.

118dec. 76hex.

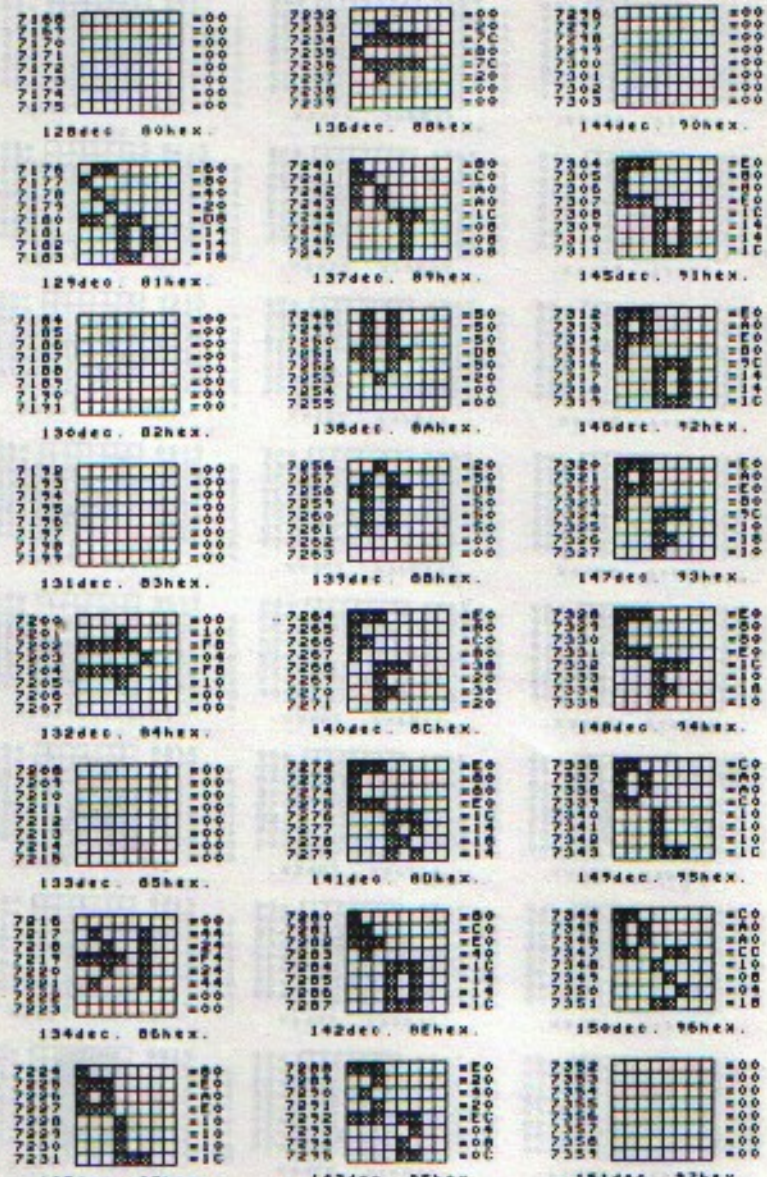
126dec. 7Ehex.

111dec. 6Fhex.

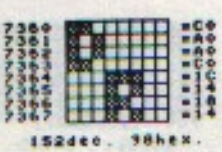
119dec. 77hex.



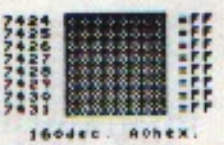
127dec. 7Fhex.



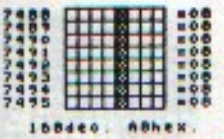
128dec. 80hex.
129dec. 81hex.
130dec. 82hex.
131dec. 83hex.
132dec. 84hex.
133dec. 85hex.
134dec. 86hex.
135dec. 87hex.
136dec. 88hex.
137dec. 89hex.
138dec. 8Ahex.
139dec. 8Bhex.
140dec. 8Chex.
141dec. 8Dhex.
142dec. 8Ehex.
143dec. 8Fhex.
144dec. 90hex.
145dec. 91hex.
146dec. 92hex.
147dec. 93hex.
148dec. 94hex.
149dec. 95hex.
150dec. 96hex.
151dec. 97hex.



152dec. 98hex.

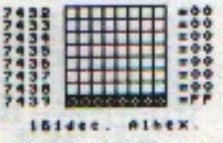


160dec. A0hex.



168dec. A8hex.

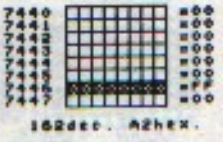
153dec. 99hex.



161dec. A1hex.

169dec. A9hex.

154dec. 9Ahex.



162dec. A2hex.

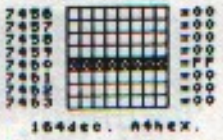
170dec. AAhex.

155dec. 9Bhex.

163dec. A3hex.

171dec. ABhex.

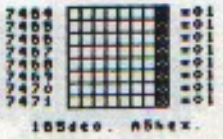
156dec. 9Chex.



164dec. A4hex.

172dec. AChex.

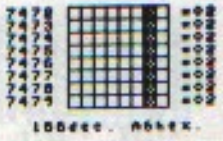
157dec. 9Dhex.



165dec. A5hex.

173dec. ADhex.

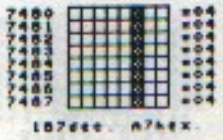
158dec. 9Ehex.



166dec. A6hex.

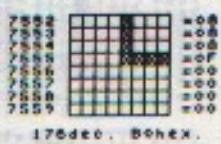
174dec. AEhex.

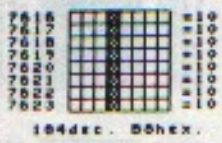
159dec. 9Fhex.

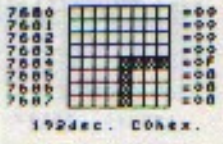


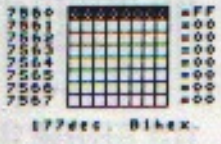
167dec. A7hex.

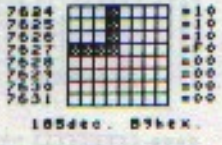
175dec. AFhex.


176dec. B0hex.

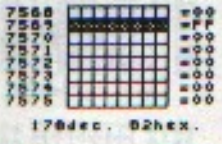

184dec. B8hex.


192dec. C0hex.


177dec. B1hex.

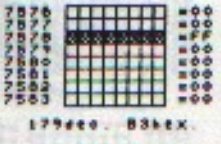

185dec. B9hex.

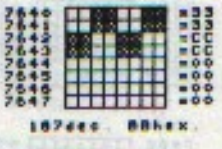

193dec. C1hex.


178dec. B2hex.

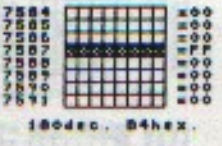

186dec. BAhex.


194dec. C2hex.

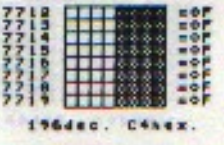

179dec. B3hex.


187dec. BBhex.

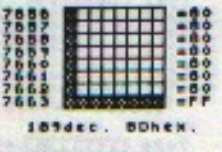

195dec. C3hex.

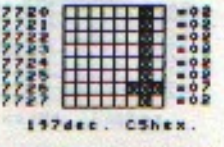

180dec. B4hex.

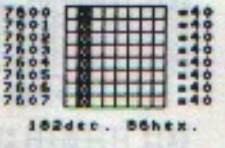

188dec. BChex.


196dec. C4hex.

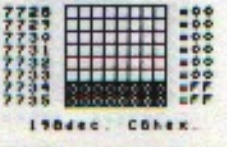

181dec. B5hex.

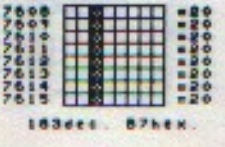

189dec. BDhex.


197dec. C5hex.

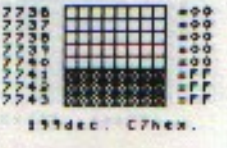

182dec. B6hex.

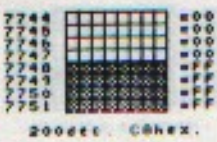

190dec. BEhex.


198dec. C6hex.


183dec. B7hex.


191dec. BFhex.

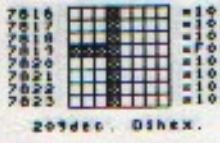

199dec. C7hex.


200dec. C8hex.

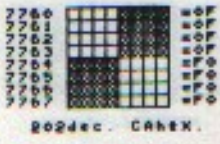

208dec. D0hex.


216dec. D8hex.


201dec. C9hex.

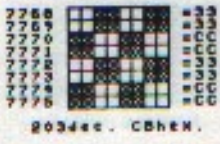

209dec. D1hex.


217dec. D9hex.

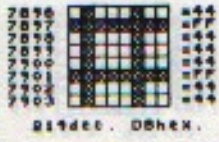

202dec. CAhex.

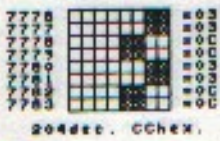

210dec. D2hex.

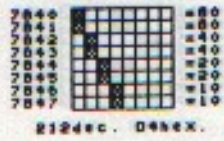

218dec. DAhex.


203dec. CBhex.


211dec. D3hex.


219dec. DBhex.

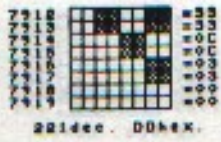

204dec. CChex.


212dec. D4hex.


220dec. DChex.

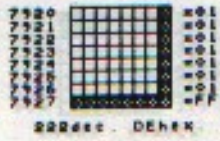

205dec. CDhex.


213dec. D5hex.


221dec. DDhex.


206dec. CEhex.

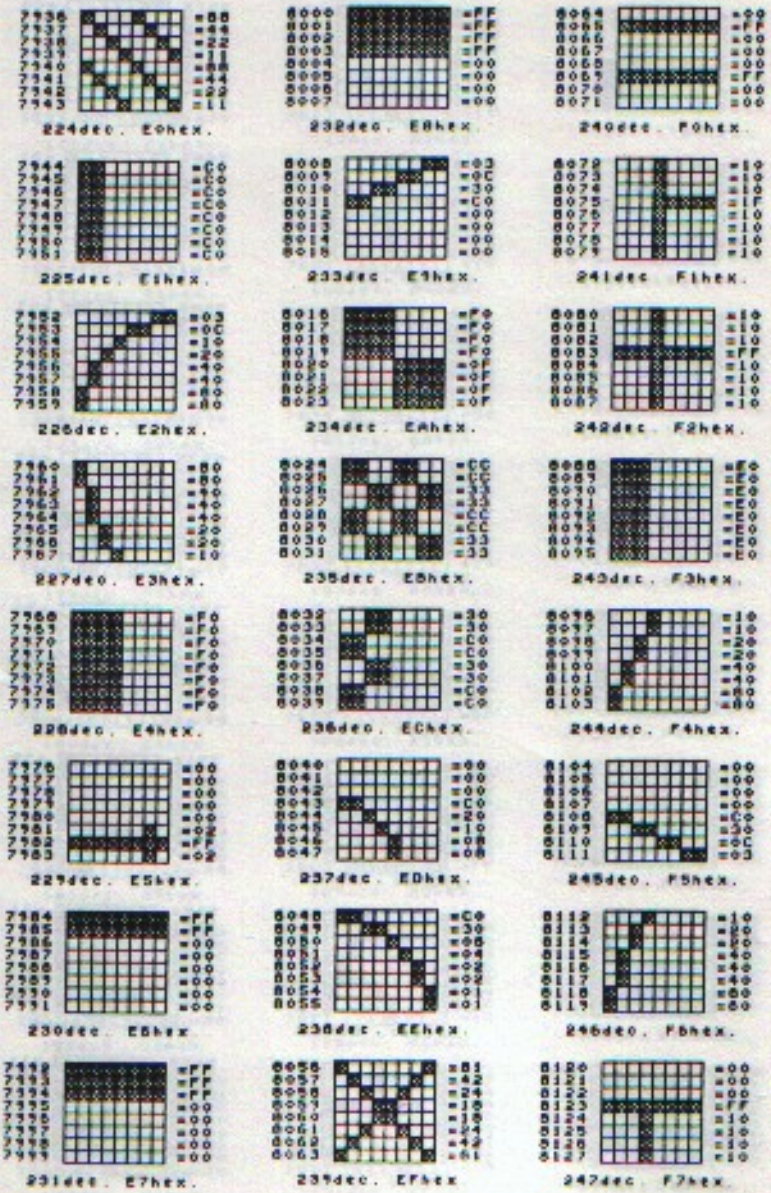

214dec. D6hex.


222dec. DEhex.


207dec. CFhex.


215dec. D7hex.


223dec. DFhex.


224dec. E0hex.
232dec. E8hex.
240dec. F0hex.
225dec. E1hex.
233dec. E9hex.
241dec. F1hex.
226dec. E2hex.
234dec. EAhex.
242dec. F2hex.
227dec. E3hex.
235dec. EBhex.
243dec. F3hex.
228dec. E4hex.
236dec. EChex.
244dec. F4hex.
229dec. E5hex.
237dec. EDhex.
245dec. F5hex.
230dec. E6hex.
238dec. EEhex.
246dec. F6hex.
231dec. E7hex.
239dec. EFhex.
247dec. F7hex.


248dec. F8hex.
253dec. FDhex.
249dec. F9hex.
254dec. FEhex.
250dec. FAhex.
255dec. FFhex.
251dec. FBhex.
252dec. FChex.

Index

Adding addresses 81
Adding routines 77
Adding words 78
Additions to XBAS 4.2 65
Alpha characters 48
ASCII codes 126
Auxiliary word count 80

Basic token addresses 122
Biorhythm program 27
Bubble sort 101

CALL 55
Character set 42
CLEAR 55
Clock program 20
CTRL 56
Copy MOS 74

Database program 104
DEF FN 16
DIM 12
DRAW 24

EDIT 87
ELLIPSE 1
Expanding XBAS 70

Fast sort 102
Fill MOS 74
FILL 17
FMT 14
Function keys 34
Function key buffer 39

GCOL 56

Hex/dec conversion 124
HOLD 57

IOM 11

KBD 58

Locked files 98

MID$ 25
Modify MOS 68

ORIGIN 20

PI 4
Piechart program 10
PLOT 22
POLY 6
PRINT @ 14
PTR 0 70
PTR 2 70
PTR 3 70
PTR 6 71
PTR 8 71
PTR 19 60

RAD 3
Ready message 66
RENUM 93
Restoring erased progs 15
RST 59

Saving additions 85
SCRN$ 59
SHAPE 44
Sorting data 100
Sprite attribute table 52
System files 98

Tabulate MOS 74
TI$ 26
TRON/TROFF 78

VAL 24
VPOKE 48

WIDTH 61