CORTEX USERS GROUP

T Gray,1 Larkspur Drive,Featherstone,Wolverhampton,West Midland WV10 7TN.
E Serwa,93 Long Knowle Lane,Wednesfield,Wolverhampton,West Midland WV11 1JG.
Tel No: T Gray 0902 729078, E. Serwa 0902 732659

14

CORTEX USER GROUP NEWSLETTER (NOVEMBER 1987)

Issue Number 14

CONTENTS

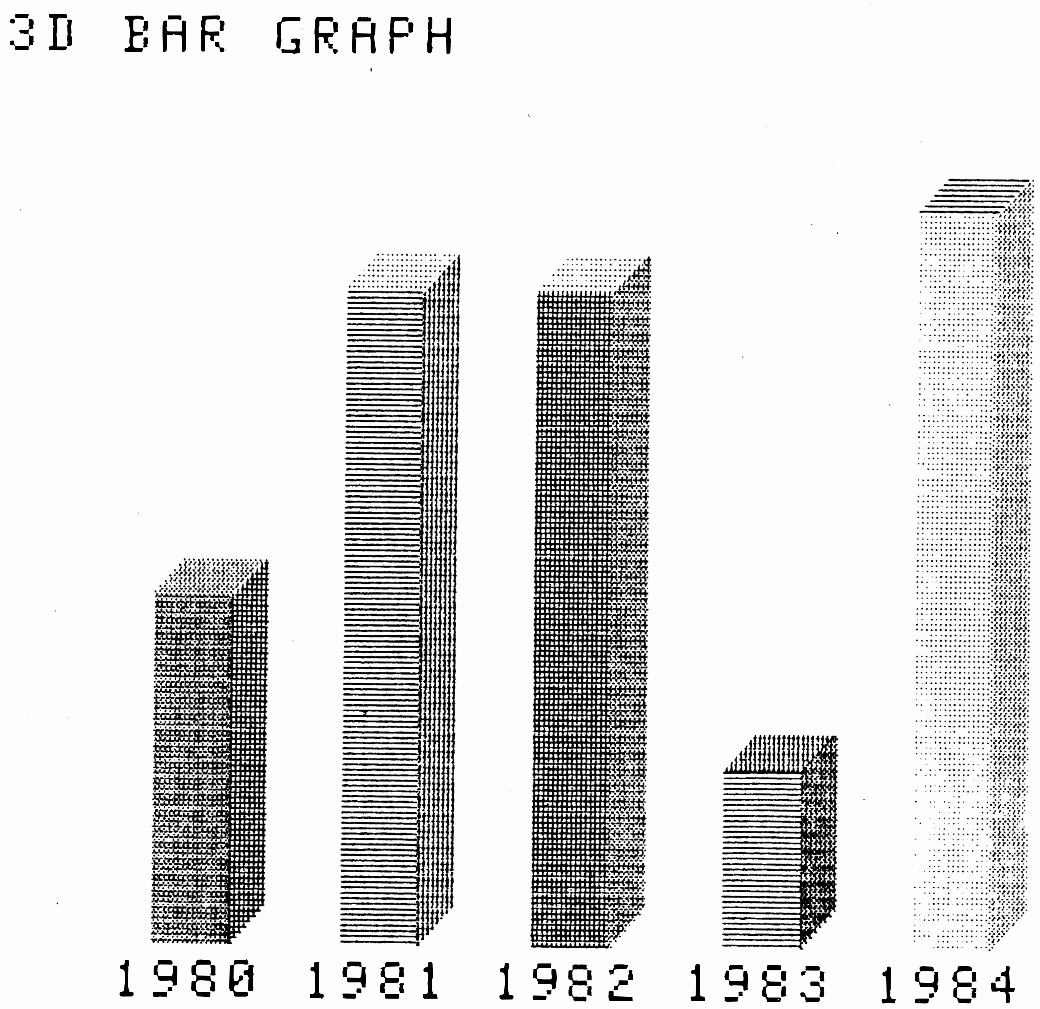
REMEMBER TO SEND IN YOUR ARTICLES FOR THE NEXT NEWSLETTER

Editorial

Its not very often that we receive an article so comprehensive that it takes up most of the user group newsletter but this one written by Mark Rudnicki explains so much about programming the V.D.P. in machine code that we thought it best to print it all in one issue. The routines used also may help to explain the mystery of machine code programming to some of you who have not had much experience in this field. Some of the routines are shown as a Basic programme first and then in machine code after. This is a technique used a lot by ourselves as most of the debugging can be done on the basic programme before converting it to machine code.

Mark as also sent in some games programmes for the newsletter and these will be included in the next issue.

The other article in this issue is a three dimentional bar graph programme written by Tim Gray. It generates block bar graphs that look solid.



REMEMBER TO SEND IN YOUR ARTICLES FOR THE NEXT NEWSLETTER

 Mark Rudnicki.

1: The Video Display Processor.

The Cortex boasts a large amount of user memory since the large amount of RAM necessary for the implementation of high resolution graphics has been effectively removed from the memory map and put onto the other side of a two byte port. This leads to some advantages and some major disadvantages:

+ Frees 16K of RAM for programming

but
- All access to VRAM is via two 8 bit ports, causing programming complications.
- Multiple instructions needed to alter the VRAM contents, leading to reduced speed.

The VDP port lies at >F120 and >F121. There are four ways of accessing the VDP and VRAM:

	MSB 0 1 2 3 4 5 6 7 LSB	Port	R or W
Write to VDP register			
Byte 1 Data	$D_0\ D_1\ D_2\ D_3\ D_4\ D_5\ D_6\ D_7$	>F121	Write
Byte 2 Reg. select	$1\ 0\ 0\ 0\ 0\ R_{S0}\ R_{S1}\ R_{S2}$	>F121	Write
Read from Status Reg.			
Byte 1 Read data	$D_0\ D_1\ D_2\ D_3\ D_4\ D_5\ D_6\ D_7$	>F121	Read
Write to VRAM			
Byte 1 Address set up	$A_6\ A_7\ A_8\ A_9\ A_{10}\ A_{11}\ A_{12}\ A_{13}$	>F121	Write
Byte 2	$0\ 1\ A_0\ A_1\ A_2\ A_3\ A_4\ A_5$	>F121	Write
Byte 3 Data write	$D_0\ D_1\ D_2\ D_3\ D_4\ D_5\ D_6\ D_7$	>F120	Write
Read from VRAM			
Byte 1 Address set up	$A_6\ A_7\ A_8\ A_9\ A_{10}\ A_{11}\ A_{12}\ A_{13}$	>F121	Write
Byte 2	$0\ 0\ A_0\ A_1\ A_2\ A_3\ A_4\ A_5$	>F121	Write
Byte 3 Data read	$D_0\ D_1\ D_2\ D_3\ D_4\ D_5\ D_6\ D_7$	>F120	Read

Data.

In all cases, the data to be written or read is in byte form which means that a little care is needed when transferring data to or from the VRAM. To move data from a workspace register, MOVB is used ('Move Byte'). This moves the leftmost i.e. most significant, byte of a register. Similarly, MOVB @>F120,R1 will read data from the VDP and move it to the uppermost byte of Register 1.

Address.

This is a 14 bit value to give the full 16384 byte (16K) coverage, from >0000 to >3
lower byte will hold A_6 to A_{13}, and the upper byte A_0 to A_5, like this:

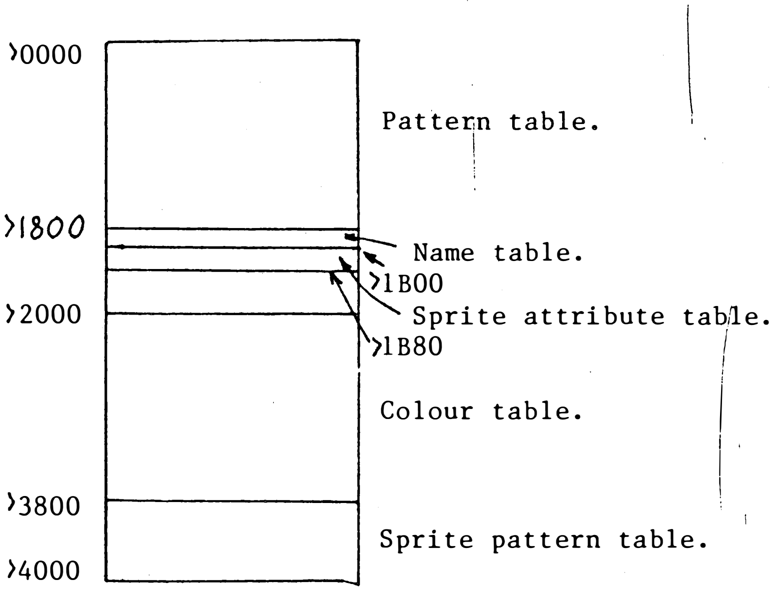
MSB [$0\ 0\ A_0\ A_1\ A_2\ A_3\ A_4\ A_5$] [$A_6\ A_7\ A_8\ A_9\ A_{10}\ A_{11}\ A_{12}\ A_{13}$] LSB

To read from the VRAM, bits 0 and 1 must be clear, but to write, bit l must be set. The latter can be done either by ORing with >4000 or by Adding >4000.

```
e.g.    LI    R1,address        LI   R1,address
        ORI   R1,>4000     or   AI   R1,>4000
        etc
```

The 16K VRAM is divided up this way:

GRAPH MODE



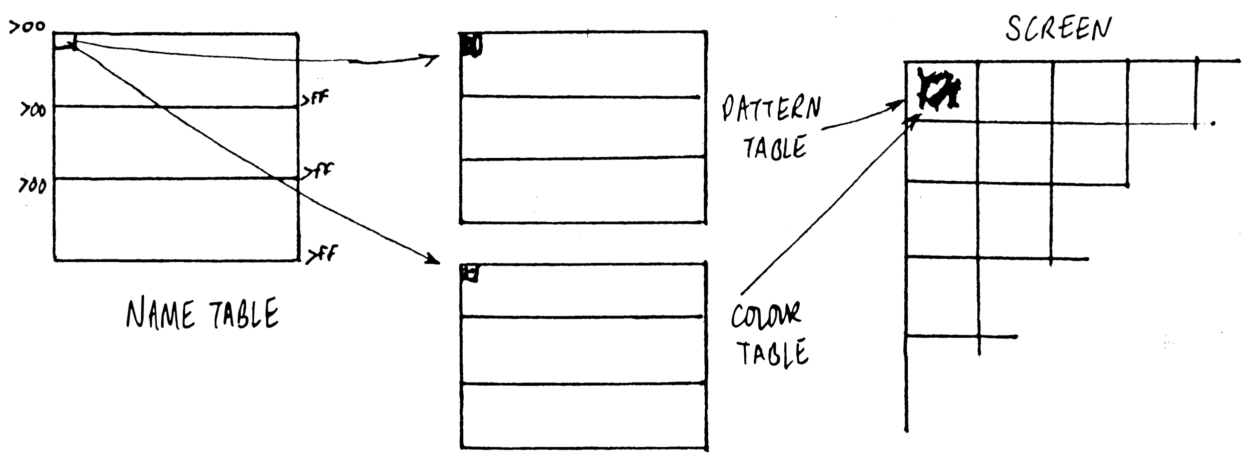
As Graph mode is the most useful for games, the rest of article will concentrate on this.

The Pattern table, the Colour table and the Name table.

The pattern table is 6K long divided into three 2K segments- each segment corresponds to a block of 256 character codes for a block of 256 screen locations.

Each 2K block is divided into 256 8 byte blocks. In this way, every pixel on the screen can be controlled achieving the 256*192 resolution. The Colour table has a similar arrangement with 8 colour bytes per screen location i.e. one colour code for each row of an eight row screen character.

The VDP knows which pattern to display by checking the Name table which indicates which pattern is to be used for each screen location. In the Cortex, the name table is arranged so that successive name tables. Hence, it is set up with the numbers 0 thru' 255 three times.



The consequences of this mode of operation are as follows:

+ Each screen location has a unique pattern/ colour combination so that each screen pixel can be individually controlled.
+ This allows for high resolution line graphics to be displayed i.e. for graphs etc.

but - To create a 'character' requires 16 accesses to VRAM: 8 colour bytes and 8 pattern bytes, which is slow.

Alternative use of the VDP.

The other way to use the graphics mode is to make each entry in the Name table point to a preset character in the Pattern and Colour tables, as with TEXT mode. This leads to:

- Lower resolution- Screen data must be moved around in character sized chunks.
- Individual lines can no longer be drawn.
+ Much faster- only a single byte has to be written to VRAM to place a character on the screen.
+ SGET, or its equivalent, now takes on some meaning, as in text mode, rather than moving 8 meaningless bytes around from one place to another.

These are some pros and cons for both methods, but certainly the second is easier to use and faster.

The Sprite Table.

This table is 128 bytes long, running from >1B00 to >1B80, arranged with four bytes per sprite:

MSB	0	1	2	3	4	5	6	7 LSB
0	vertical position							
1	horizontal position							
2	Name							
3	ECB	0	0	0	colour			

SPRITE No.,X,Y,Shape,Colour

The early clock bit, if set, shifts the sprite 32 pixels to the left, to allow the sprite to bleed in from the left edge of the display.

The Sprite Pattern table stores 256 8-byte blocks of data which make up the characters as defined by the 'SHAPE' command.

Machine code considerations for the TMS 9928/9.

The CPU reads or writes to the VRAM via a 14 bit auto- incrementing address register- this means that once an initial address has been set up subsequent locations can be accessed without setting up a new address every time. The VDP requires 8 μs to fetch a VRAM byte following a data transfer, so this delay must be taken into consideration when programming. This delay can be performed using a meaningless MOV *R1,*R1 instruction.

If long routines which alter the VRAM contents are called from Basic, then it is wise to preceed them with a LIMI >0000 instruction (Load Interrupt Mask Immediate) to disable the processor interrupts, and to end with a LIMI >000F. This might be needed to prevent the system mucking about with the VRAM in the course of the user routine. Note, the LIMI >0000 instruction stops the software clock.

Using the Cortex Graphics mode.

Individual points can be accessed using the formula:

Point= X,Y

VRAM byte = 256*INT(Y/8)+8*INT(X/8)+MOD(Y,8)

The relevant bit number is MOD(X,8)   0=MSB, 7=LSB

To see if this bit is set, try the following:

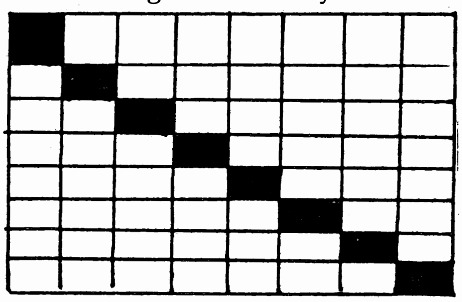
```
R1= read byte (in LSB of register)
R0= bit number

LI   R2,>0080
SRL  R2,0
COC  R2,R1
JNE  bit not set
bit set...
```

SRL (Shift Right Logical) takes a shift count from R0 if the shift count is zero (as above). If R0 is zero, then R2 will be shifted right sixteen times. Other values give te following effects:

R0   Least significant byte of R2

0
1
2
3
4
5
6
7



The bit set in R2 points to the bit to be tested in R1

MSB   LSB   ■=bit set

COC R2,R1 (Compare Ones Corresponding), sees whether the bits set in R2 are also set in R1; if so, then the equal flag ST2 is set. The JNE (Jump Not Equal) operates if the relevant screen bit was not set. Using this, the status of a screen bit can be tested and acted upon.

Colour of a pixel.

The colour of a pixel can be found as follows:

Colour data =>2000 + Screen byte address

If the pixel is set then the foreground colour should be returned, otherwise the background colour should be given.

Firstly, the screen byte must be calculated:

```
                 R0= X Coord.  R1= Y Coord.

TRUE COLOUR      MOV  R1,R10             R10=Y
                 ANDI R10,>FFF8          R10=8*INT(Y/8)
                 SLA  R10,5              R10=32*R10
                 ANDI R1,>0007           R1=MOD[Y,8]
                 A    R1,R10             R10=R10+R1
                 MOV  R0,R4              R4=X
                 ANDI R4,>FFF8           R4=8*INT(X/8)
                 A    R4,R10             R10=Screen byte address
                 ANDI R0,>0007           R0=MOD(X,8)
                 BL   @>READ ADDRESS     Set up address to read VRAM
                 CLR  R5                 R5=0
                 MOVB @>F120,R5          Move screen data into R5
                 SWPB R5                 Swap it into the lower byte.
                 AI   R10,>2000          Add to access colour table.
                 BL   @>READ ADDRESS     Set up address to read VRAM
                 CLR  R6                 R6=0
                 MOVB @>F120,R6          Move colour data to R6
                 SWPB R6                 Swap it into lower byte.
                 LI   R7,>0080           Test bit start.
                 SRL  R7,0               Shift R0 times right.
                 COC  R7,R5              See if bit set
                 JNE  BIT NOT SET        No.
                 SRL  R6,4               Yes- select foreground colour.
BIT NOT SET      ANDI R6,>000F           Isolate colour code.

                  rest of program....
```

To set up the VRAM address, the following subroutine is needed. It takes the VRAM address held in R10 and sets up the VDP for a VRAM data read.

```
READ ADDRESS     SWPB R10                Least sig. byte first.
                 MOVB R10,@>F121         Move top byte.
                 MOV  R10,R10            Delay
                 SWPB R10
                 MOVB R10,@>F121
                 MOV  *R10,*R10          Delay.
                 RT                      Return from subroutine.
```

The BL (Branch and Link) instruction behaves like a GOSUB- its return address is stored in R11, but unlike a Basic GOSUB, it cannot be nested. Any attempt to do so will simply overwrite the previous return address. If nesting of subroutines is required, then the BLWP (Branch and Load Workspace Pointer) command must be used. The operand must contain the address of two words- the first will be the start address of a new workspace (32 bytes), and the second the adress of the subroutine. >F020 and >F040 are two convenient locations for workspace registers as they are in fast on-chip RAM.

To set up the VDP for a data write, the following code is needed:

```
WRITE ADDRESS ORI  R10,>4000             Set bit 1
              JMP  READ ADDRESS
```

This sets bit 1 of the address word, which tells the VDP to expect a data write. The read subroutine can then be called to transfer the address.

The to routines can be condensed as follows:

```
WRITE ADDRESS ORI  R10,>4000
READ ADDRESS  SWPB R10
              MOVB R10,@>F121
              MOV  R10,R10
              SWPB R10
              MOVB R10,@>F121
              MOV  *R10,*R10
              RT
```

The entry point is chosen depending upon whether a VRAM read or write is required.

Returning values to Basic.

If values need to be returned to Basic, then use must be made of the Basic ADR function, which gives the position of the variable in memory.

e.g. for the 'True colour of a pixel' routine, this can be done as follows:

```
A=0: CALL "TRUE COLOUR",Address,X,Y,ADR(A)
```

Where A is any variable, and X and Y are the pixel coords.

ADR(A) will be stored in R2 when the routine is called. R6 contains the true pixel colour, and can be stored in the variable with the addition of this code:

```
INCT R2
INC  R2              R2=R2+3
MOV  R6,*R2          Store R6 in variable.
```

R2 has to be incremented three times so that it points to the correct word to be altered (see Cortex instruction manual, page 2-12).

Setting and resetting pixels.

pixel operations are necessary for line and circle drawing routines, and for building up characters. Whilst Basic caters for the line drawing, the routine is not accessible from machine code yet, until more information about the Basic is released.

R0= X Coord.
R1= Y Coord
R2= Colour
R3= 0 for set, 1 for reset

```
e.g. CALL "PLOT",Address,X,Y,Colour,Plot?
```

```
PLOT          MOV  R1,R8
              ANDI R8,>FFF8
              SLA  R8,5
              ANDI R1,>0007
              A    R1,R8
              MOV  R0,R4
              ANDI R4,>FFF8
              A    R4,R8
              ANDI R0,>0007
              MOV  R8,R10             R8=Screen byte address.
              BL   @>READ ADDRESS
              INC  R0
              MOVB @>F120,R5          Read current screen byte.
              SWPB R5
              SLA  R5,0
              ANDI R5,>FFEF           Shift it and reset target bit.
              MOV  R3,R3              Test R3 for zero.
              JNE  BIT NOT SET        Branch if zero
              AI   R5,>0100           Otherwise set bit.
BIT NOT SET   SRL  R5,0               Shift back
              SWPB R5
              BL   @>WRITE ADDRESS
              MOVB R5,@>F120          Write screen byte.
              CLR  R5
              AI   R8,>2000
              MOV  R8,R10
              BL   @>READ ADDRESS     Set up colour table address.
              MOVB @>F120,R5          Read current colour.
              SWPB R5
              ANDI R5,>000F           Isolate current background.
              SLA  R2,4
              A    R2,R5              Add new foreground.
              SWPB R5
              BL   @>WRITE ADDRESS
              MOVB R5,@>F120          Write new colour byte.
              RTWP                    Return from subroutine.
```

Line and circle plotting.

For fast line and circle algorithms, integer routines have been developed e.g. Bresenham, in 'Interactive Computer Graphics' by Foley and Van Dam. This is important since floating point routines are inherently slow.

Bresenham's Circles.

The best way to describe this routine is to present it in Basic first, to show its simplicity.

```
10  X=0: Y=R: D=3-2*R: A=128: B=96
20  IF X>=Y THEN GOTO 80
30  GOSUB 100
40  IF D<0 THEN D=D+4*X+6
50   ELSE D=D+4*(X-Y)+10:Y=Y-1
60  X=X+1
70  GOTO 20
80  IF X=Y THEN GOSUB 100
```

```
90  END
100 PLOT A+X,B+Y:PLOT A+X,B-Y:PLOT A-X,B+Y:PLOT A-X,B-Y
110 PLOT A+Y,B+X:PLOT A+Y,B-X:PLOT A-Y,B+X:PLOT A-Y,B-X
120 RETURN
```

The eight plot commands mean that only an eigth of the circle needs to be computed- the rest is derived through symmetry. However, in machine code, the coding is fairly long and tedious. Use can be made of the previously defined PLOT subroutine, to create this new command:

```
CALL "CIRCLE",Address,X,Y,Radius,Plot?,Colour
                      ~~~            ↑
                     Centre     Plot or unplot.
```

The point plot subroutine needs to the BLWP'd, so 2 additional words are needed:

```
POINT PLOT     DATA >F020
               DATA >Start address of PLOT
```

>F020 will be the new workspace when the PLOT routine is called, and is in fast on-chip memory.

```
>F020= X Coord of point
>F022= Y Coord of point
>F024= Colour
>F026= Plot or unplot
```

```
CIRCLE        CLR  R5            R5=X
              MOV  R2,R6         R6=Y
              LI   R7,>0003      R7=3
              SLA  R2,1          R2=2*R
              S    R2,R7         R7= D=3-2*R
LOOP          C    R5,R6         Is X>=Y?
              JHE  END           Yes, then goto end bit.
              BL   @PLOT         Plot 8 points
              MOV  R7,R7         Set flags for D
              JEQ  D>=0          Jump if D equals zero
              JGT  D>=0          Jump if D > zero
D<0           AI   R7,>0006      D=D+6
              MOV  R5,R2         R2=X
              SLA  R2,2          R2=X*4
              A    R2,R7         D=D+X*4
INCX          INC  R5            X=X+1
              JMP  LOOP          Loop
D>=0          AI   R7,>000A      D=D+10
              MOV  R5,R2         R2=X
              S    R6,R2         R2=X-Y
              SLA  R2,2          R2=4*(X-Y)
              A    R2,R7         D=D+4*(X-Y)
              DEC  R6            Y=Y-1
              JMP  INCX          Jump back and inc. X
END           C    R5,R6         Compare X and Y
              JEQ  PLOTIT        X=Y? If yes, then jump
              RTWP               Otherwise end
PLOTIT        BL   @PLOT         Plot 8 points
              RTWP               Then end
```

```
PLOT        LI   R9,2          Loop counter
AGAIN       MOV  R4,@>F024     Store colour
            MOV  R3,@>F026     Store plot?
            MOV  R0,@>F020
            MOV  R1,@>F022
            A    R5,@>F020          PLOT A+X,B+Y
            A    R6,@>F022     and  PLOT A+Y,B+X
            BLWP @POINT PLOT
            MOV  R4,@>F024
            MOV  R0,@>F020
            MOV  R1,@>F022
            A    R5,@>F020          PLOT A+X,B-Y
            S    R6,@>F022     and  PLOT A+Y,B-X
            BLWP @POINT PLOT
            MOV  R4,@>F024
            MOV  R0,@>F020
            MOV  R1,@>F022
            S    R5,@>F020          PLOT A-X,B+Y
            A    R6,@>F022     and  PLOT A-Y,B+X
            BLWP @POINT PLOT
            MOV  R4,@>F024
            MOV  R0,@>F020
            MOV  R1,@>F022
            S    R5,@>F020          PLOT A-X,B-Y
            S    R6,@>F022     and  PLOT A-Y,B-Y
            BLWP @POINT PLOT
            MOV  R5,R8         Reverse X and Y
            MOV  R6,R5
            MOV  R8,R6
            DEC  R9            End of loop?
            JNE  AGAIN         Not yet
            RT                 Now it is!
```

There are probably better ways of doing this- I'll leave this one to you!

Bresenham's line algorithm.

Again, in Basic, this goes as follows:

```
10  INPUT X1,Y1,X2,Y2
20  F=0: DR=1
30  DX=ABS(X2-X1): DY=ABS(Y2-Y1)
40  IF DY>DX THEN A=X1:X1=Y1:Y1=A:A=X2:X2=Y2:Y2=A:F=1:GOTO30
50  D=(2*DY)-DX:I1=2*DY:I2=2*(DY-DX)
60  IF X1>X2 THEN X=X2:Y=Y2:XE=X1:YE=Y1
70   ELSE X=X1:Y=Y1:XE=X2:YE=Y2
80  IF YE<=Y THEN DR=-1
90  IF F THEN PLOT Y,X
100  ELSE PLOT X,Y
110 IF X>=XE THEN END
120 X=X+1
130 IF D<0 THEN D=D+I1
140  ELSE Y=Y+DR:D=D+I2
150 GOTO 90
```

The call for this is:

```
CALL "PLOT LINE",Address,X1,Y1,X2,Y2,Colour,Plot?
```

And the machine code:

```
PLOT LINE    LI    R7,>0001        DR=1
             CLR   R6              F=0
DYDX         MOV   R2,R8           R8=X2
             ABS   R8              R8= ABS(X2-X1) =DX
             MOV   R3,R9           R9=Y2
             S     R1,R9           R9=Y2-Y1
             ABS   R9              R9= ABS(Y2-Y1) =DY
             C     R8,R9           DX>DY?
             JHE   NOSWAP          No swap if DX =DY
             MOV   R0,R10
             MOV   R1,R0           Swap X1,Y1
             MOV   R10,R1
             MOV   R2,R10
             MOV   R3,R2           Swap X2,Y2
             MOV   R10,R3
             INC   R6              F=1
             JMP   DYDX            Recalculate DX,DY
NOSWAP       C     R0,R2           Compare X1 and X2
             JLE   NOMOVE          Jump if X1 =X2
             MOV   R0,R10
             MOV   R2,R0           Otherwise swap X1 and X2
             MOV   R10,R2
             MOV   R1,R10
             MOV   R3,R1           and swap Y1 and Y2
             MOV   R10,R3
NOMOVE       C     R3,R1           Compare YE and Y
             JHE   HIGHER          Jump if higher or equal
             LI    R7,>FFFF        Else DR=-1
HIGHER       SLA   R9,1            D9=2*DY = I1
             MOV   R9,R10          R10 (D) = D9
             S     R8,R10          R10 = 2*DY-DX
             MOV   R10,R3          R3=2*DY-DX
             S     R8,R3           R3=2*(DY-DX) =I2
             MOV   R5,@>F026       Store Plot?
PLOT LOOP    MOV   R4,@>F024       Store colour
             MOV   R0,@>F020       Store X
             MOV   R1,@>F022       Store Y
             MOV   R6,R6           Check for F
             JEQ   NOREVERSE       Jump if zero
             MOV   R0,@>F022       Otherwise reverse X
             MOV   R1,@>F020       and Y
NOREVERSE    BLWP  @POINT PLOT     Then plot point
             C     R0,R2           Compare X and XE
             JL    NOEND           Jump if lower
             RTWP                  Else end
NOEND        INC   R0              X=X+1
             MOV   R10,R10         Check D
             JGT   ADDI2           Jump if D>0
             JEQ   ADDI2           Jump if D=0
             A     R9,R10          Otherwise D=D+I1
             JMP   PLOT LOOP       Loop
             A     R3,R10          D=D+I2
             A     R7,R1           Y=Y+DR
             JMP   PLOT LOOP       Loop
```

The routine follows almost the same fromat as the Basic program- note that the actual program loop is short, keeping up the speed.

The use of these routines allows simple vector graphics type displays to be built up, especially from machine code where the speed difference becomes more noticable (the CALLs are slowed by Basic checking the passed parameters).

Redefining the Graphics mode.

The other way to use to graphics mode is to store predefined character/ colour combinations in the pattern and colour tables, and to use the Name table to select which character appears on the screen. Since the Pattern and Colour tables are divided into three groups, each character must be defined three times, once in each section of the tables. Once accomplished, displays of very colourful characters exploiting the full resolution of the mode can be built up.

All the routines have been presented in the Cortex Users Group newsletter, nos. 2 and 3. Please write to the Users Group if you require back numbers.

Use of the routines.

Once redefined, screen data can be thrown around fairly easily e.g. Burglar, Invaders. The effects in Burglar are created by redefining the characters which make up the ladders etc. so that they all appear to move, wherever they are placed.

For more adventurous use of machine code, two more standard routines are needed. These are for key pickup, and for printing and erasing gaming characters.

Keyboard pickup

The 2536 keyboard controller sends back either the ASCII code of the key being pressed, or random data if there is no key down. Hence, any keyboard routine will have to compare, after a short delay, the current keyboard data with its previous value to see if the value remains constant- if yes, then the data is reliable and can be acted upon. This suitable delay could be the program loop, if short enough.

Keyboard data can be read using the following:

```
CLR  R12           BASE 0
STCR R0,0          R0=CRF[0]
SWPB R0            Swap data to LSByte
ANDI R0,>00FF      AND to clear rubbish

R0=ASCII code of key/ random data
```

A spare word can be used to hold the 'LAST DATA' i.e. the previous value read from the keyboard chip. The present value can be checked against this, and if they are equal, then the key is valid. Otherwise, the new value is stored in 'LAST DATA' and the routine left.

The routine may continue:

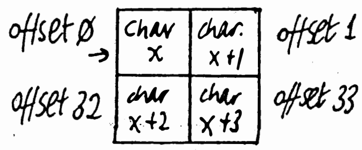
```
              C    R0,@LAST VALUE
              JEQ  DATA VALID
              MOV  R1,@LAST VALUE
              RTWP
DATA VALID    CI   R0,KEYCODE1
              JEQ  ROUTINE 1
              CI   R0,KEYCODE2
              JEQ  ROUTINE 2

               etc.
```

Printing and clearing characters.

Often it is necessary to print player or other characters which are made up of more than one block. This can be done using an offset table and a character code table. However, because all the characters have to be user defined, they can be arranged successively.

e.g. for a 2 by 2 character:



The offset table looks like this:

```
OFFSET        DATA >0001
              DATA >2021
```

and can be printed using:

```
              LI   R0,start screen location
              LI   R1,first character number
              CLR  R2
LOOP          CLR  R3
              MOVB @OFFSET(R2),R3
              SWPB R3
              A    R0,R3
              MOV  R3,@>F020
              MOV  R1,@>F022
              BLWP @PUT CHAR
              INC  R1
              INC  R2
              CI   R2,4
              JNE  LOOP
              RT(WP)
```

To clear the character, blanks (ASCII 32), can be moved to >F022 during a similar routine. The fifth instruction above is an example of indexed addressing- R2 is added to 'OFFSET' to create the address for the data to be moved.

Full listing of line and circle plots.

Commands are: CALL "POINT PLOT",6220H,X,Y,Colour,Plot?
CALL "CIRLE",6300H,X,Y,Radius,Plot?,Colour
CALL "DEMO",6248H
CALL "LINE PLOT",6380H,X1,Y1,X2,Y2,Colour,Plot?

```
MON
Monitor Rev. 1.1  1982
[]U 6200 6406
6200 026A ORI   R10,>4000
6204 06CA SWPB  R10
6206 D80A MOVB  R10,@>F121
620A C28A MOV   R10,R10
620C 06CA SWPB  R10
620E D80A MOVB  R10,@>F121
6212 C69A MOV   *R10,*R10
6214 045B RT
6216 F020 SOCB  @>621C,R0
621A 0000 DATA  >0000
621C 0300 LIMI  >0000
6220 C201 MOV   R1,R8
6222 0248 ANDI  R8,>FFF8
6226 0A58 SLA   R8,5
6228 0241 ANDI  R1,>0007
622C A201 A     R1,R8
622E C100 MOV   R0,R4
6230 0244 ANDI  R4,>FFF8
6234 A204 A     R4,R8
6236 0240 ANDI  R0,>0007
623A C288 MOV   R8,R10
623C 06A0 BL    @>6204
6240 0580 INC   R0
6242 D160 MOVB  @>F120,R5
6246 06C5 SWPB  R5
6248 0A05 SLA   R5,0
624A 0245 ANDI  R5,>FEFF
624E C0C3 MOV   R3,R3
6250 1602 JNE   >6256
6252 0225 AI    R5,>0100
6256 0905 SRL   R5,0
6258 06C5 SWPB  R5
625A 06A0 BL    @>6200
625E D805 MOVB  R5,@>F120
6262 04C5 CLR   R5
6264 0228 AI    R8,>2000
6268 C288 MOV   R8,R10
626A 06A0 BL    @>6204
626E D160 MOVB  @>F120,R5
6272 06C5 SWPB  R5
6274 0245 ANDI  R5,>000F
6278 0A42 SLA   R2,4
627A A142 A     R2,R5
627C 06C5 SWPB  R5
627E 06A0 BL    @>6200
6282 D805 MOVB  R5,@>F120
6286 0380 RTWP
6288 0209 LI    R9,>0002
628C C804 MOV   R4,@>F024
6290 C803 MOV   R3,@>F026
6294 C800 MOV   R0,@>F020
6298 C801 MOV   R1,@>F022
629C A805 A     R5,@>F020
62A0 A806 A     R6,@>F022
62A4 0420 BLWP  @>6216
62A8 C804 MOV   R4,@>F024
62AC C800 MOV   R0,@>F020
62B0 C801 MOV   R1,@>F022
62B4 A805 A     R5,@>F020
62B8 6806 S     R6,@>F022
62BC 0420 BLWP  @>6216
62C0 C804 MOV   R4,@>F024
62C4 C800 MOV   R0,@>F020
62C8 C801 MOV   R1,@>F022
62CC 6805 S     R5,@>F020
62D0 A806 A     R6,@>F022
62D4 0420 BLWP  @>6216
62D8 C804 MOV   R4,@>F024
62DC C800 MOV   R0,@>F020
62E0 C801 MOV   R1,@>F022
62E4 6805 S     R5,@>F020
62E8 6806 S     R6,@>F022
62EC 0420 BLWP  @>6216
62F0 C205 MOV   R5,R8
62F2 C146 MOV   R6,R5
62F4 C188 MOV   R8,R6
62F6 0609 DEC   R9
62F8 16C9 JNE   >628C
62FA 045B RT
62FC 0300 LIMI  >0000
6300 04C5 CLR   R5
6302 C182 MOV   R2,R6
6304 0207 LI    R7,>0003
6308 0A12 SLA   R2,1
630A 61C2 S     R2,R7
630C 8185 C     R5,R6
630E 1414 JHE   >6338
6310 06A0 BL    @>6288
6314 C1C7 MOV   R7,R7
6316 1308 JEQ   >6328
6318 1507 JGT   >6328
631A 0227 AI    R7,>0006
```

```
631E C085 MOV  R5,R2
6320 0A22 SLA  R2,2
6322 A1C2 A    R2,R7
6324 0585 INC  R5
6326 10F2 JMP  >630C
6328 0227 AI   R7,>000A
632C C085 MOV  R5,R2
632E 6086 S    R6,R2
6330 0A22 SLA  R2,2
6332 A1C2 A    R2,R7
6334 0606 DEC  R6
6336 10F6 JMP  >6324
6338 8185 C    R5,R6
633A 1301 JEQ  >633E
633C 0380 RTWP
633E 06A0 BL   @>6288
6342 0380 RTWP
6344 F040 SOCB R0,R1
6346 6300 S    R0,R12
6348 04C3 CLR  R3
634A 0200 LI   R0,>0080
634E C800 MOV  R0,@>F040
6352 0200 LI   R0,>0060
6356 C800 MOV  R0,@>F042
635A C803 MOV  R3,@>F044
635E 04E0 CLR  @>F046
6362 C003 MOV  R3,R0
6364 0240 ANDI R0,>000F
6368 C800 MOV  R0,@>F048
636C 0420 BLWP @>6344
6370 0583 INC  R3
6372 0283 CI   R3,>005F
6376 16E9 JNE  >634A
6378 0380 RTWP
637A 0000 DATA >0000
637C 0000 DATA >0000
637E 0000 DATA >0000
6380 0300 LIMI >0000
6384 0207 LI   R7,>0001
6388 04C6 CLR  R6
638A C202 MOV  R2,R8
638C 6200 S    R0,R8
638E 0748 ABS  R8
6390 C243 MOV  R3,R9
6392 6241 S    R1,R9
6394 0749 ABS  R9
6396 8248 C    R8,R9
6398 1408 JHE  >63AA
639A C280 MOV  R0,R10
639C C001 MOV  R1,R0
639E C04A MOV  R10,R1
63A0 C282 MOV  R2,R10
63A2 C083 MOV  R3,R2
63A4 C0CA MOV  R10,R3
63A6 0586 INC  R6
63A8 10F0 JMP  >638A
63AA 8080 C    R0,R2
63AC 1206 JLE  >63BA
63AE C280 MOV  R0,R10
63B0 C002 MOV  R2,R0
63B2 C08A MOV  R10,R2
63B4 C281 MOV  R1,R10
63B6 C043 MOV  R3,R1
63B8 C0CA MOV  R10,R3
63BA 8043 C    R3,R1
63BC 1402 JHE  >63C2
63BE 0207 LI   R7,>FFFF
63C2 0A19 SLA  R9,1
63C4 C289 MOV  R9,R10
63C6 6288 S    R8,R10
63C8 C0CA MOV  R10,R3
63CA 60C8 S    R8,R3
63CC C805 MOV  R5,@>F026
63D0 C804 MOV  R4,@>F024
63D4 C800 MOV  R0,@>F020
63D8 C801 MOV  R1,@>F022
63DC C186 MOV  R6,R6
63DE 1304 JEQ  >63E8
63E0 C800 MOV  R0,@>F022
63E4 C801 MOV  R1,@>F020
63E8 0420 BLWP @>6216
63EC 8080 C    R0,R2
63EE 1A03 JL   >63F6
63F0 0300 LIMI >000F
63F4 0380 RTWP
63F6 0580 INC  R0
63F8 C28A MOV  R10,R10
63FA 1503 JGT  >6402
63FC 1302 JEQ  >6402
63FE A289 A    R9,R10
6400 10E7 JMP  >63D0
6402 A283 A    R3,R10
6404 A047 A    R7,R1
6406 10E4 JMP  >63D0
```

THREE DIMENTIONAL BAR GRAPH PROGRAMME Tim Gray.

This programme could be used as a subroutine of a larger programme for displaying data in 3D form. It generates block bar graphs that look solid.

```
 10  REM *** 3D BAR GRAPH DEMO PROGRAMME ***
 20  REM ***          TIM GRAY          ***
 30  REM
 40  COLOUR 15,1: GRAPH
 50  REM
 60  REM ** B= Baseline
 70  REM ** H = Hight up to 100
 80  REM ** BLK = Block Number
 90  REM ** C1 C2 C3 = Front,Side,Top Colours
 100  REM *** Set random data for block ***
 110  B=180
 120  BLK=1: H=RND*150: C1=5: C2=4: C3=7: $A="1980"
 130  GOSUB 260
 140  BLK=2: H=RND*150: C1=9: C2=8: C3=11: $A="1981"
 150  GOSUB 260
 160  BLK=3: H=RND*150: C1=3: C2=2: C3=14: $A="1982"
 170  GOSUB 260
 180  BLK=4: H=RND*150: C1=9: C2=6: C3=13: $A="1983"
 190  GOSUB 260
 200  BLK=5: H=RND*150: C1=11: C2=10: C3=9: $A="1984"
 210  GOSUB 260
 220  COLOUR 15,0: PRINT @(1,1);"PRESS ANY KEY": GOSUB 450
 230  REM
 240  REM *** Draw the block ***
 250  REM
 260  COLOUR 15,0: PRINT @(BLK*5-1,23);$A
 270  COLOUR C1,C2: D=BLK*40+16
 280  FOR F=B TO B-6 STEP -1
 290   COLOUR C1,C2: PLOT BLK*40,F TO BLK*40+15,F
 300   COLOUR C2,0: PLOT BLK*40+16,F TO D,F
 310   D=D+1: NEXT F
 320  FOR F=B-7 TO B-H-7 STEP -1
 330   COLOUR C1,C2: PLOT BLK*40,F TO BLK*40+15,F
 340   COLOUR C2,C2: PLOT BLK*40+16,F
 350  NEXT F
 360  C=BLK*40: D=C+16
 370  FOR T=B-7-H TO B-13-H STEP -1
 380   COLOUR C3,0: PLOT C,T TO BLK*40+15,T
 390   C=C+1
 400   COLOUR C3,C2: PLOT BLK*40+16,T TO D,T
 410   D=D+1
 420  NEXT T
 430  RETURN
 440  REM *** Loop for another go ***
 450  LET K=KEY[0]
 460  IF K<>0 THEN PRINT "<0C>": WAIT 100: GOTO 60
 470    ELSE GOTO 450
```

CORTEX USERS CLUB SALE

$=POUNDS PLEASE NOTE SOME ITEMS HAVE INCREASED IN PRICE DUE TO IC AND COMPONENT PRICE INCREASE

RGB INTERFACE BARE BOARD $8.00 KIT $25.00
CENTRONICS INTERFACE BARE BOARD $7.00 KIT $15.00
E BUS -ALL IC'S KIT $30.00

SEMICONDUCTORS

TMS9902 $2.00
74LS612 (3 AVALABLE) $25.00

ON OFFER

74LS611/74LS611 (NEED PULL UP RESISTORS) $10.00

E BUS EPANSION

E BUS (4K RAM,8K EPROM SCKT,16 IN/OUT LINES) $15.00
NOTE-THESE CARDS ARE EX EQUIPMENT TESTED AND WORKING
E BUS (8*8K EPROM SCKT CARD BUILT NO EPROMS FITTED) $30.00

NEW E BUS 512K DRAM 'THRO' PLATED BARE BOARD $40.00
USES TMS4500 AND TMS4464 OR EQUIVELENT + 6 COMMON SUPPORT CHIPS

CORTEX EXPANSION

EXTERNAL VIDIO INTERFACE BARE BOARD $15.00 KIT $80.00
DISK CONTROLER (WD 2797+BOARD) CORTEX I $55.00
DISK CONTROLER (WD 2797+BOARD) CORTEX II $60.00

CORTEX SOFTWARE

DISK OPERATING SYSTEM CDOS 1.20 AND 2.00 $45.00
CDOS 2.00 FOR 2797 SYSTEM CDOS 2.0 FOR 2797 SYSTEM ALL FORMATS
NEW FORMAT 3" 40T SINGLE SIDED

MEMBERS SOFTWARE

WORTEX-WORD PROCESSING $15.00
INCLUDES SPELLING CHECKER
SEND TO J S MACKENZIE (NEW ADDRESS)
20 WEST ROAD
BARTON STACY
WINCHESTER
HANTS S021 3SB
(INCLUDE TWO 5"" DD DISKS)
DRAWTECH-GRAPHICS DRAWING PACKAGE $20.00
SEND TO T GRAY C/O CORTEX USERS GROUP ADDRESS

ASSEMBLER CDOS COMPATIBLE FOR DISK SYSTEM WRITEN BY C J YOUNG
FOR THE USER GROUP (WE RECOMMEND)
AVAILABLE FROM CORTEX USERS GROUP $15.00
(ALL FORMATS ON DISK)

CORTEX USERS GROUP SOFTWARE MOST NOW ON DISK (ALL FORMATS)

ALL GAMES $2.50 EACH

BURGLAR	MUNCHER	THE ZOO	FIRE BIRD
FROGGER	G DESIGN	GOLF	RESCUE
INVADERS&ASTEROIDS	WALL	MICROPED	PENGO
HUNCHBACK	ARCHIE	SPACE-BUG	MOONBASEII
MAZE	NIGHT ATTACK	MAZE 3D	LABYRINTH OF TAG
OLIMPICS NEW	CORTELLO	PENGO	CENTIPEDE

CORRECTION

ASSEMBLER FROM R M LEE IS NOW $14.00 HIS NEW ADDRESS IS
8 RENOWN ROAD
LORDSWOOD
CHATHAM
KENT ME5 8SG