CORTEX USER GROUP NEWSLETTER (NOVEMBER 1987)

Issue Number 14

CONTENTS

1. Index
2. Editorial
3. Programming the V.D.P.
17. Programme (3D Bar graph)

REMEMBER TO SEND IN YOUR ARTICLES FOR THE NEXT NEWSLETTER
Editorial

It's 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 dimensional bar graph programme written by Tim Gray. It generates block bar graphs that look solid.

3D BAR GRAPH


REMEMBER TO SEND IN YOUR ARTICLES FOR THE NEXT NEWSLETTER
Manipulating the VDP with Machine Code

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:

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
<th>Port</th>
<th>R or W</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7</td>
<td>$F121</td>
<td>Write</td>
<td></td>
</tr>
</tbody>
</table>

Write to VDP register

Byte 1 Data  \(D_6, D_5, D_4, D_3, D_2, D_1, D_0\)  \$F121  Write
Byte 2 Reg. select  \(1 0 0 0 0 R_A, R_{R1}\)  \$F121  Write

Read from Status Reg.

Byte 1 Read data  \(D_6, D_5, D_4, D_3, D_2, D_1, D_0\)  \$F121  Read

Write to VRAM

Byte 1 Address set up  \(A_9, A_8, A_7, A_6, A_5, A_4, A_3\)  \$F121  Write
Byte 2  \(0 1 A_6, A_5, A_4, A_3, A_2\)  \$F121  Write
Byte 3 Data write  \(D_6, D_5, D_4, D_3, D_2, D_1, D_0\)  \$F120  Write

Read from VRAM

Byte 1 Address set up  \(A_9, A_8, A_7, A_6, A_5, A_4\)  \$F121  Write
Byte 2  \(0 0 A_6, A_5, A_4, A_3, A_2\)  \$F121  Write
Byte 3 Data read  \(D_6, D_5, D_4, D_3, D_2, D_1, D_0\)  \$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, MOV B is used ('Move Byte'). This moves the leftmost i.e. most significant, byte of a register. Similarly, MOV B @\$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 \$3FFF. In a register containing a VRAM address, the lower byte will hold \(A_6\) to \(A_3\), and the upper byte \(A_2\) to \(A_0\), like this:

\[
\begin{array}{c}
\text{MSB} & \text{LSB} \\
0 0 & 0 0
\end{array}
\]

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

\[ \text{e.g. LI R1, address ORI R1, \$4000 or AI R1, \$4000 etc} \]

The 16K VRAM is divided up this way:

**GRAPH MODE**

\[ \begin{array}{|c|}
\hline
\$0000 & \text{Pattern table.} \\
\hline
\$1800 & \text{Name table.} \\
\hline
\$2000 & \text{Sprite attribute table.} \\
\hline
\$3800 & \text{Colour table.} \\
\hline
\$4000 & \text{Sprite pattern table.} \\
\hline
\end{array} \]

As Graph mode is the most useful for games, the rest of this 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 $01800$ to $01880$, arranged with four bytes per sprite:

<table>
<thead>
<tr>
<th>MSB</th>
<th>0 1 2 3 4 5 6 7</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>vertical position</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>horizontal position</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ECB [0 0 0] colour</td>
<td></td>
</tr>
</tbody>
</table>

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 $\mu$s 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 proceed 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:

RL = read byte (in LSB of register)
RO = 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 RO if the shift count is zero (as above). If RO is zero, then R2 will be shifted right sixteen times. Other values give the following effects:

RO  Least significant byte of R2
0
1
2
3
4
5
6
7

MSB  LSB  ■=bit set

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

COC R2,R1 (Compare Ones Corresponding), see 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:
RO= 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
     MOV R0,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
     MOV R0,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.
     MOV R10,›F121  Move; top byte.
     MOV R10,R10  Delay
     SWPB R10
     MOV R10,›F121  Delay.
     MOV *R10,*R10  Return from subroutine.
     RT

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 address 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
MOV R10, @F121
MOV R10, R10
SWPB R10
MOV 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
MOV R6, *R2
```

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

The eight plot commands mean that only an eighth 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 ¥020
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
MOV R2,R6
LI R7,Y0003
SLA R2,1
S R2,R7
C R5,R6
JHE END
BL @PLOT
MOV R7,R7
JEQ D=0
JGT D>0
D<0
AI R7,Y0006
MOV R5,R2
SLA R2,2
A R2,R7
INC R5
JMP LOOP
Loop
D>0
AI R7,Y000A
MOV R5,R2
S R6,R2
SLA R2,2
A R2,R7
DEC R6
JMP INCX
Jump back and inc. X
END
C R5,R6
JEQ PLOTIT
RTWP
OTHERWISE end
PLOTIT
BL @PLOT
Plot 8 points
RTWP
Then end
PLOT
AGAIN
LI R9,2 Loop counter
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: GOTO 30
50 D=(2*DY)-DX: I1=2*DY: I2=2*(DY-DX)
60 IF X1<X2 THEN X=I2: Y: Y2: XE=X1: YE=Y1
70 ELSE X=X1: Y=Y1: XE=X2: YE=Y2
80 IF YE=0 THEN DR=-1
90 IF F THEN PLOT Y, X
100 ELSE PLOT X, Y
110 IF XE=YE 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
CLR R6

DYDX
MOV R2,R8
ABS R8
MOV R3,R9
S R1,R9
ABS R9
C R8,R9
JHE NOSWAP
MOV R0,R10
MOV R1,R0
MOV R2,R1
MOV R3,R2
MOV R10,R3
INC R6
JMP DYDX
C R0,R2
JLE NOMOVE
MOV R0,R10
MOV R2,R0
MOV R10,R2
MOV R1,R10
MOV R3,R1
MOV R10,R3
NOMOVE
C R3,R1
JHE HIGHER
LI R7,FFFF
HIGH
SLA R9,1
MOV R9,R10
S R8,R10
MOV R10,R3
S R8,R3
MOV R5,$F026
MOV R4,$F024
MOV R0,$F020
MOV R1,$F022
MOV R6,R6
JEQ NOREVERSE
MOV R0,$F022
MOV R1,$F020
NOREVERSE
BLWP @POINT PLOT
C R0,R2
JL NOEND
RTWP
NOEND
INC R0
MOV R10,R10
JGT ADD12
JEQ ADD12
A R9,R10
JMP PLOT LOOP
A R3,R10
A R7,R1
JMP PLOT 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                    RO=CRF[0]
SWPB R0                      Swap data to LSByte
ANDI R0,#00FF                AND to clear rubbish
```

RO=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:

```assembly
C     RO,@LAST VALUE
JEQ   DATA VALID
MOV   R1,@LAST VALUE
RTWP
DATA VALID
CI    RO,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:

```
<table>
<thead>
<tr>
<th>Offset</th>
<th>char</th>
<th>char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>x+1</td>
</tr>
<tr>
<td>32</td>
<td>x+2</td>
<td>x+3</td>
</tr>
</tbody>
</table>
```

The offset table looks like this:

OFFSET   DATA > 0001
DATA > 2021

and can be printed using:

```assembly
LI     R0,start screen location
LI     R1,first character number
CLR    R2
CLR    R3
LOOP
  MOV   @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 "CIRCLE",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
[]JU 6200 6406
6200 026A ORI R10,Y4000
6204 06CA SWPB R10
6206 D80A MOV B R10, @YF121
620A C28A MOV R10,R10
620C 06CA SWPB R10
620E D80A MOV B R10, @YF121
6212 C69A MOV *R10,*R10
6214 045B RT
6216 F020 SOC B @Y61C,R0
621A 0000 DATA @Y0000
621C 0300 LIMI @Y0000
6220 C201 MOV R1,R8
6222 0248 ANDI R8, @FFF8
6226 0A58 SLA R8,5
6228 0241 ANDI R1, @Y0007
622C A201 A R1,R8
622E C100 MOV R0,R4
6230 0244 ANDI R4, @FFF8
6234 A204 A R4,R8
6236 0240 ANDI R0, @Y0007
623A C288 MOV R8,R10
623C 06A0 BL @Y6204
6240 0580 INC R0
6242 D160 MOV B @YF120,R5
6246 06C5 SWPB R5
6248 A050 SLA R3,0
624A 0245 ANDI R5, @YFEFF
624E C0C3 MOV R3,R3
6250 1602 JNE @Y6256
6252 0225 AI R5, @Y0100
6256 0905 SRL R5,0
6258 06C5 SWPB R5
625A 06A0 BL @Y6200
625E D805 MOV B R5, @YF120
6262 04C5 CLR R5
6264 0228 AI R8, @Y2000
6268 C288 MOV R8,R10
626A 06A0 BL @Y6204
626E D160 MOV B @YF120,R5
6272 06C5 SWPB R5
6274 0245 ANDI R5, @Y000F
6278 0A42 SLA R2,4
627A A142 A R2,R5
627C 06C5 SWPB R5
627E 06A0 BL @Y6200
6282 D805 MOV B R5, @YF120
6286 0380 RTWP
6288 0209 LI R9, @Y0002
628C C804 MOV R4, @YF024
6290 C803 MOV R3, @YF026
6294 C800 MOV R0, @YF020
6298 C801 MOV R1, @YF022
629C A805 A R5, @YF020
62A0 A806 A R6, @YF022
62A4 0420 BLWP @Y6216
62A8 C804 MOV R4, @YF024
62AC C800 MOV R0, @YF020
62B0 C801 MOV R1, @YF022
62B4 A805 A R5, @YF020
62BB 6B06 S R6, @YF022
62BC 0420 BLWP @Y6216
62CC 6B05 S R5, @YF020
62DD A806 A R6, @YF022
62DE 0420 BLWP @Y6216
62DF 6B04 S R5, @YF020
62E8 6B06 S R6, @YF022
62EA 0420 BLWP @Y6216
62EB 6B04 S R5, @YF020
62EC 0420 BLWP @Y6216
62EC 0420 BLWP @Y6216
62FC 0300 LI R1 @Y0000
6300 04C5 CLR R5
6302 C182 MOV R2, R6
6304 0207 LI R7, @Y0003
6308 A12 SLA R2, 4
630A 61C2 S R2, R7
63CC 8185 C R5, R6
630E 1414 JNE @Y6338
6310 06A0 BL @Y6288
6314 C1C7 MOV R7, R7
6316 1308 JEQ @Y6328
6318 1507 JGT @Y6328
631A 0227 AI R7, @Y0006
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Operand 1</th>
<th>Operand 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>631E</td>
<td>C085</td>
<td>MOV</td>
<td>R5, R2</td>
</tr>
<tr>
<td>6320</td>
<td>0A22</td>
<td>SLA</td>
<td>R2, R2</td>
</tr>
<tr>
<td>6322</td>
<td>A1C2</td>
<td>A</td>
<td>R2, R7</td>
</tr>
<tr>
<td>6324</td>
<td>0585</td>
<td>INC</td>
<td>R5</td>
</tr>
<tr>
<td>6326</td>
<td>10F2</td>
<td>JMP</td>
<td>$630C</td>
</tr>
<tr>
<td>6328</td>
<td>0227</td>
<td>AJ</td>
<td>R7, $000A</td>
</tr>
<tr>
<td>632C</td>
<td>C085</td>
<td>MOV</td>
<td>R5, R2</td>
</tr>
<tr>
<td>632E</td>
<td>6086</td>
<td>S</td>
<td>R6, R2</td>
</tr>
<tr>
<td>6330</td>
<td>0A22</td>
<td>SLA</td>
<td>R2, R2</td>
</tr>
<tr>
<td>6332</td>
<td>A1C2</td>
<td>A</td>
<td>R2, R7</td>
</tr>
<tr>
<td>6334</td>
<td>0606</td>
<td>DEC</td>
<td>R6</td>
</tr>
<tr>
<td>6336</td>
<td>10F6</td>
<td>JMP</td>
<td>$6324</td>
</tr>
<tr>
<td>6338</td>
<td>8185</td>
<td>C</td>
<td>R5, R6</td>
</tr>
<tr>
<td>633A</td>
<td>1301</td>
<td>JEQ</td>
<td>$633E</td>
</tr>
<tr>
<td>633C</td>
<td>0380</td>
<td>RTWP</td>
<td></td>
</tr>
<tr>
<td>633E</td>
<td>06A0</td>
<td>BL</td>
<td>@#$6288</td>
</tr>
<tr>
<td>6342</td>
<td>0380</td>
<td>RTWP</td>
<td></td>
</tr>
<tr>
<td>6344</td>
<td>F040</td>
<td>SOCB</td>
<td>R0, R1</td>
</tr>
<tr>
<td>6346</td>
<td>6300</td>
<td>S</td>
<td>R0, R12</td>
</tr>
<tr>
<td>6348</td>
<td>04C3</td>
<td>CLR</td>
<td>R3</td>
</tr>
<tr>
<td>634A</td>
<td>0200</td>
<td>LI</td>
<td>R0, $0080</td>
</tr>
<tr>
<td>634E</td>
<td>C800</td>
<td>MOV</td>
<td>R0, @F040</td>
</tr>
<tr>
<td>6352</td>
<td>0200</td>
<td>LI</td>
<td>R0, $0060</td>
</tr>
<tr>
<td>6356</td>
<td>C800</td>
<td>MOV</td>
<td>R0, @F042</td>
</tr>
<tr>
<td>635A</td>
<td>C803</td>
<td>MOV</td>
<td>R3, @F044</td>
</tr>
<tr>
<td>635E</td>
<td>04E0</td>
<td>CLR</td>
<td>@F046</td>
</tr>
<tr>
<td>6362</td>
<td>C003</td>
<td>MOV</td>
<td>R3, R0</td>
</tr>
<tr>
<td>6364</td>
<td>0240</td>
<td>ANDI</td>
<td>R0, $000F</td>
</tr>
<tr>
<td>6368</td>
<td>C800</td>
<td>MOV</td>
<td>R0, @F048</td>
</tr>
<tr>
<td>636C</td>
<td>0420</td>
<td>BLWP</td>
<td>@#$6344</td>
</tr>
<tr>
<td>6370</td>
<td>0583</td>
<td>INC</td>
<td>R3</td>
</tr>
<tr>
<td>6372</td>
<td>0283</td>
<td>CI</td>
<td>R3, $005F</td>
</tr>
<tr>
<td>6376</td>
<td>16E9</td>
<td>JNE</td>
<td>$634A</td>
</tr>
<tr>
<td>6378</td>
<td>0380</td>
<td>RTWP</td>
<td></td>
</tr>
<tr>
<td>637A</td>
<td>0000</td>
<td>DATA</td>
<td>$0000</td>
</tr>
<tr>
<td>637C</td>
<td>0000</td>
<td>DATA</td>
<td>$0000</td>
</tr>
<tr>
<td>637E</td>
<td>0000</td>
<td>DATA</td>
<td>$0000</td>
</tr>
<tr>
<td>6380</td>
<td>0300</td>
<td>LIMI</td>
<td>$0000</td>
</tr>
<tr>
<td>6384</td>
<td>0207</td>
<td>LI</td>
<td>R7, $0001</td>
</tr>
<tr>
<td>6388</td>
<td>04C6</td>
<td>CLR</td>
<td>R6</td>
</tr>
<tr>
<td>638A</td>
<td>C202</td>
<td>MOV</td>
<td>R2, R8</td>
</tr>
<tr>
<td>638C</td>
<td>6200</td>
<td>S</td>
<td>R0, R8</td>
</tr>
<tr>
<td>638E</td>
<td>0748</td>
<td>ABS</td>
<td>R8</td>
</tr>
<tr>
<td>6390</td>
<td>C243</td>
<td>MOV</td>
<td>R3, R9</td>
</tr>
<tr>
<td>6392</td>
<td>6241</td>
<td>S</td>
<td>R1, R9</td>
</tr>
<tr>
<td>6394</td>
<td>0749</td>
<td>ABS</td>
<td>R9</td>
</tr>
<tr>
<td>6396</td>
<td>8248</td>
<td>C</td>
<td>R8, R9</td>
</tr>
<tr>
<td>6398</td>
<td>1408</td>
<td>JHE</td>
<td>$63AA</td>
</tr>
<tr>
<td>639A</td>
<td>C280</td>
<td>MOV</td>
<td>R0, R10</td>
</tr>
<tr>
<td>639C</td>
<td>C001</td>
<td>MOV</td>
<td>R1, R0</td>
</tr>
<tr>
<td>639E</td>
<td>C04A</td>
<td>MOV</td>
<td>R10, R1</td>
</tr>
<tr>
<td>63A0</td>
<td>C282</td>
<td>MOV</td>
<td>R2, R10</td>
</tr>
<tr>
<td>63A2</td>
<td>C083</td>
<td>MOV</td>
<td>R3, R2</td>
</tr>
<tr>
<td>63A4</td>
<td>COCA</td>
<td>MOV</td>
<td>R10, R3</td>
</tr>
<tr>
<td>63A6</td>
<td>0586</td>
<td>INC</td>
<td>R6</td>
</tr>
<tr>
<td>63A8</td>
<td>10F0</td>
<td>JMP</td>
<td>$638A</td>
</tr>
<tr>
<td>63AA</td>
<td>8080</td>
<td>C</td>
<td>R0, R2</td>
</tr>
<tr>
<td>63AC</td>
<td>1206</td>
<td>JLE</td>
<td>$63BA</td>
</tr>
<tr>
<td>63AE</td>
<td>C280</td>
<td>MOV</td>
<td>R0, R10</td>
</tr>
<tr>
<td>63B0</td>
<td>C002</td>
<td>MOV</td>
<td>R2, R0</td>
</tr>
<tr>
<td>63B2</td>
<td>C08A</td>
<td>MOV</td>
<td>R10, R2</td>
</tr>
<tr>
<td>63B4</td>
<td>C281</td>
<td>MOV</td>
<td>R1, R10</td>
</tr>
<tr>
<td>63B6</td>
<td>C043</td>
<td>MOV</td>
<td>R3, R1</td>
</tr>
<tr>
<td>63B8</td>
<td>COCA</td>
<td>MOV</td>
<td>R10, R3</td>
</tr>
<tr>
<td>63BA</td>
<td>8063</td>
<td>C</td>
<td>R3, R1</td>
</tr>
<tr>
<td>63BC</td>
<td>1402</td>
<td>JHE</td>
<td>$63C2</td>
</tr>
<tr>
<td>63BE</td>
<td>0207</td>
<td>LI</td>
<td>R7, $FFFF</td>
</tr>
<tr>
<td>63C2</td>
<td>0A19</td>
<td>SLA</td>
<td>R9, R1</td>
</tr>
<tr>
<td>63C4</td>
<td>C289</td>
<td>MOV</td>
<td>R9, R10</td>
</tr>
<tr>
<td>63C6</td>
<td>6288</td>
<td>S</td>
<td>R8, R10</td>
</tr>
<tr>
<td>63C8</td>
<td>COCA</td>
<td>MOV</td>
<td>R10, R3</td>
</tr>
<tr>
<td>63CA</td>
<td>6068</td>
<td>S</td>
<td>R8, R3</td>
</tr>
<tr>
<td>63CC</td>
<td>C805</td>
<td>MOV</td>
<td>R5, @F026</td>
</tr>
<tr>
<td>63D0</td>
<td>C804</td>
<td>MOV</td>
<td>R4, @F024</td>
</tr>
<tr>
<td>63D4</td>
<td>C800</td>
<td>MOV</td>
<td>R0, @F020</td>
</tr>
<tr>
<td>63D8</td>
<td>C801</td>
<td>MOV</td>
<td>R1, @F022</td>
</tr>
<tr>
<td>63DC</td>
<td>C186</td>
<td>MOV</td>
<td>R6, R6</td>
</tr>
<tr>
<td>63DE</td>
<td>1304</td>
<td>JEQ</td>
<td>$63E8</td>
</tr>
<tr>
<td>63E0</td>
<td>C800</td>
<td>MOV</td>
<td>R0, @F022</td>
</tr>
<tr>
<td>63E4</td>
<td>C801</td>
<td>MOV</td>
<td>R1, @F020</td>
</tr>
<tr>
<td>63E8</td>
<td>0420</td>
<td>BLWP</td>
<td>@#$6216</td>
</tr>
<tr>
<td>63EC</td>
<td>8080</td>
<td>C</td>
<td>R0, R2</td>
</tr>
<tr>
<td>63EE</td>
<td>1A03</td>
<td>JL</td>
<td>$63F6</td>
</tr>
<tr>
<td>63F0</td>
<td>0300</td>
<td>LIMI</td>
<td>$000F</td>
</tr>
<tr>
<td>63F4</td>
<td>0380</td>
<td>RTWP</td>
<td></td>
</tr>
<tr>
<td>63F6</td>
<td>0580</td>
<td>INC</td>
<td>R0</td>
</tr>
<tr>
<td>63FA</td>
<td>C28A</td>
<td>MOV</td>
<td>R10, R10</td>
</tr>
<tr>
<td>63FC</td>
<td>1503</td>
<td>JCT</td>
<td>$6402</td>
</tr>
<tr>
<td>63FE</td>
<td>1302</td>
<td>JEQ</td>
<td>$6402</td>
</tr>
<tr>
<td>6400</td>
<td>10E7</td>
<td>JMP</td>
<td>$63D0</td>
</tr>
<tr>
<td>6402</td>
<td>A283</td>
<td>A</td>
<td>R3, R10</td>
</tr>
<tr>
<td>6404</td>
<td>A047</td>
<td>A</td>
<td>R7, R1</td>
</tr>
<tr>
<td>6406</td>
<td>10E4</td>
<td>JMP</td>
<td>$63D0</td>
</tr>
</tbody>
</table>
THREE DIMENSIONAL 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
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

14.17
CORTEX USERS CLUB SALE

$=POUNDS  PLEASE NOTE SOME ITEMS HAVE INCREASED IN PRICE DUE

TO IC AND COMPONENT PRICE INCREASE

<table>
<thead>
<tr>
<th>Component</th>
<th>Bare Board</th>
<th>Kit</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB INTERFACE</td>
<td>$8.00</td>
<td>$25.00</td>
</tr>
<tr>
<td>CENTRONICS INTERFACE</td>
<td>$7.00</td>
<td>$15.00</td>
</tr>
<tr>
<td>E BUS - ALL IC'S</td>
<td></td>
<td>$30.00</td>
</tr>
</tbody>
</table>

SEMI CONDUCTORS

<table>
<thead>
<tr>
<th>Component</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TMS9902</td>
<td>$2.00</td>
</tr>
<tr>
<td>74LS612</td>
<td>$25.00</td>
</tr>
<tr>
<td>(3 AVAILABLE)</td>
<td></td>
</tr>
</tbody>
</table>

ON OFFER

<table>
<thead>
<tr>
<th>Component</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>74LS611/74LS611</td>
<td>$10.00</td>
</tr>
<tr>
<td>(NEED PULL UP RESISTORS)</td>
<td></td>
</tr>
</tbody>
</table>

E BUS EXPANSION

<table>
<thead>
<tr>
<th>Component</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E BUS (4K RAM, 8K EPROM SCKT, 16 IN/OUT LINES)</td>
<td>$15.00</td>
</tr>
<tr>
<td>NOTE-THOSE CARDS ARE EX EQUIPMENT TESTED AND WORKING</td>
<td></td>
</tr>
<tr>
<td>E BUS (8*8K EPROM SCKT CARD BUILT NO EPROMS FITTED)</td>
<td>$30.00</td>
</tr>
</tbody>
</table>

NEW

<table>
<thead>
<tr>
<th>Component</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E BUS 512K DRAM 'THRO' PLATED BARE BOARD</td>
<td>$40.00</td>
</tr>
<tr>
<td>USES TMS4500 AND TMS4464 OR EQUIVALENT + 6 COMMON SUPPORT CHIPS</td>
<td></td>
</tr>
</tbody>
</table>

CORTEX EXPANSION

<table>
<thead>
<tr>
<th>Component</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTERNAL VIDIO INTERFACE BARE BOARD</td>
<td>$15.00</td>
</tr>
<tr>
<td>KIT</td>
<td>$80.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK CONTROLLER (WD 2797+BOARD) CORTEX I</td>
<td>$55.00</td>
</tr>
<tr>
<td>DISK CONTROLLER (WD 2797+BOARD) CORTEX II</td>
<td>$60.00</td>
</tr>
</tbody>
</table>

CORTEX SOFTWARE

<table>
<thead>
<tr>
<th>Component</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK OPERATING SYSTEM CDOS 1.20 AND 2.00</td>
<td>$45.00</td>
</tr>
<tr>
<td>CDOS 2.00 FOR 2797 SYSTEM CDOS 2.0 FOR 2797 SYSTEM ALL FORMATS</td>
<td></td>
</tr>
<tr>
<td>NEW FORMAT 3&quot; 40T SINGLE SIDED</td>
<td></td>
</tr>
</tbody>
</table>

MEMBERS SOFTWARE

<table>
<thead>
<tr>
<th>Component</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WORTEX-WORD PROCESSING</td>
<td>$15.00</td>
</tr>
<tr>
<td>INCLUDES SPELLING CHECKER</td>
<td></td>
</tr>
<tr>
<td>SEND TO J S MACKENZIE (NEW ADDRESS)</td>
<td></td>
</tr>
<tr>
<td>20 WEST ROAD</td>
<td></td>
</tr>
<tr>
<td>BARTON STACY</td>
<td></td>
</tr>
<tr>
<td>WINCHESTER</td>
<td></td>
</tr>
<tr>
<td>HANTS SO21 3SB</td>
<td></td>
</tr>
<tr>
<td>(INCLUDE TWO 5&quot; DD DISKS)</td>
<td></td>
</tr>
<tr>
<td>DRAWTECH-GRAPHICS DRAWING PACKAGE</td>
<td>$20.00</td>
</tr>
<tr>
<td>SEND TO T GRAY C/O CORTEX USERS GROUP ADDRESS</td>
<td></td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Game</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL GAMES</td>
<td>$2.50 EACH</td>
</tr>
<tr>
<td>BURGLAR</td>
<td>MUNCHER</td>
</tr>
<tr>
<td>FROGGER</td>
<td>G DESIGN</td>
</tr>
<tr>
<td>INVADERS &amp; ASTEROIDS</td>
<td>WALL</td>
</tr>
<tr>
<td>HUNCHBACK</td>
<td>ARCHIE</td>
</tr>
<tr>
<td>MAZE</td>
<td>NIGHT ATTACK</td>
</tr>
<tr>
<td>OLYMPICS NEW</td>
<td>CORTELLO</td>
</tr>
</tbody>
</table>

CORRECTION

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