CONTENTS

2 .. EDITORIAL
   News, future products, & new software/hardware.

3 .. BUG BYTES
   Problems in Cortex hardware and programming.

4 .. PROGRAMS
   CDOS modification
   3D plane plotter
   Automatic NEW

10 .. USER INFO
   Your requests and information exchange.

11 .. FEATURE
   Adding extra BASIC statements.

13 .. SHORT TIPS
   Programming and hardware tips.

14 .. MACHINE CODE
   Part two: Addressing modes.

16 .. EXTRA FEATURE
   Generating magic squares.

17 .. USER'S ADVERT

We regret that KPH Computaware cannot accept responsibility for the contents of any letters or programs printed in this newsletter.
EDITORIAL

Greetings Cortex owners, and welcome to the sixth issue of the User’s Group Newsletter. In this issue we have six whole pages of programs, another feature by Tim Gray, part 2 of the machine code programming article, and lots of useful information. If you have any items of interest then please send them in. We will try and publish everything that is sent, although certain items may have to be edited to fit in the available space. We are still marketing user written software, and so if you have written any suitable programs then send us a copy along with a full description, and loading/saving instructions. We pay royalties for each copy of your program that we sell.

We regret that we can at present only supply our software on cassette. We are in the process of installing disc drives, and so disc software will eventually be available.

For those of you who are still without discs we are planning to produce a replacement board for the TMS9909. The circuit has been agreed with Neil Quarmby, and he will shortly be completing a compatible version of CDOS. Enquiries about this board are welcome, but we cannot state a definite price at present.

Anybody wishing to purchase CDOS or upgrade early versions of CDOS should contact Neil Quarmby at the following address.

Neil Quarmby
9 Moriston Road
Brickhill
Bedford

NEW SOFTWARE

THE LABYRINTH OF TRAG is the first adventure game for the Cortex. In this text based game you have to explore a series of underground rooms and passageways. Your aim is to stay alive by eating and drinking on the way, whilst looking for keys to open boxes. Your eventual goal is to open the treasure chest, remove its contents, and find your way out again. The main problem is that every four hours the caverns flood, and so you must not be slow.

(Price : £6.00)

Newsletter 6 programs will be available on a tape with the programs to be included in newsletter 7. In this way we hope to reduce the selling price of the tape.

HARDWARE

We have access to most of the chips required by the Cortex expansions, and would be willing to supply them to Cortex users. All enquiries are welcome, and prices of some components are shown below.

<table>
<thead>
<tr>
<th>Chip</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMS 9901</td>
<td>£6.50</td>
</tr>
<tr>
<td>TMS 9902</td>
<td>£6.50</td>
</tr>
<tr>
<td>TMS 9911</td>
<td>£25.00</td>
</tr>
<tr>
<td>TMS 9929</td>
<td>£22.00</td>
</tr>
<tr>
<td>2797 FDC</td>
<td>£36.00</td>
</tr>
<tr>
<td>74LS612</td>
<td>£25.00</td>
</tr>
</tbody>
</table>
BUG BYTES

This section is for ironing out problems which users experience with their Cortices. If you have any problems then we will be glad to include them here. If you think that you know a solution to any of these problems then please let us know, and we will pass it on.

Our problems this time start with one or two disc difficulties.

Syd Champkin of Skirlaugh has recently fitted disk drives to his Cortex, but finds he is unable to fully load the CDOS 1.20 operating system. When operating the "BOOT" command the drive loads track "00" as normal, but when the operating system "core" attempts to load, an error message, "Controller Error" is displayed, and the machine aborts the search. Syd has tried changing R70 and C29 to no avail. Can anyone offer any words of wisdom, and maybe someone local to him could help him by checking his disk on a working system. [S.Champkin., 16 Cawood Crescent, Skirlaugh, North Humberside.]

--------------------------------------------------

Mr.J.Stephens of Northumberland cannot save (or load) to tape when using CDOS 1.11. Upon attempting a load the message "TAPE READ ERROR" results. Any suggestions would be greatly appreciated.

--------------------------------------------------

C.H.Sedwill of Christchurch is having trouble with a timing related fault somewhere around the TMS 414500, which corrupts the RAM/VRAM on a cold start. Again any help will be gratefully received.

--------------------------------------------------

Julian Terry of Rainham would like some help with a programming problem. He has tried to use F020 to store WP registers for call routines, but upon passing more than one parameter the error "ILLEGAL DELIMITER" occurs.

--------------------------------------------------

Finally in this section, a couple of suggestions for improving the quality of the Cortex display.

Mr.O.W.Hulme of Staffordshire suggests that by parting the inner and screen of the coax cable at the aerial socket end the picture can be "pulled" to the right. This would relieve the common problem of left picture shift in GRAPH mode.

--------------------------------------------------

Mr.A.Williams of Sydney, Australia tells us that his display problems were caused by interference between the power supply cables and the disc interface cable. Hence to solve this he merely moved the cables around within the case.
The following programs and routines have been sent in to us by Cortex users. Our theme this time seems to be biased towards disc software. We would, however, like to point out that our selection is obviously limited by the type of software sent in. We welcome all contributions, no matter how short, and will try to include as many as possible in each issue.

Following on from his CDOS modification in issue 5, C.M.Gale has also sent another program, with a comprehensive explanation.

The reason for the development of this program was the occasional overwriting of a disc. Therefore it was decided to investigate the workings of the disc drive handler.

The disc drive handler keeps an account of the sectors in use in the form of a bitmap, which is stored on track 1 of sector 0 of the disc. Each sector on the disc is represented by one bit in a word, i.e. track 0 sector 0 is represented by word 0 bit 0. A set bit represents a sector in use, and a clear bit represents a free sector. The first two bytes represent the sixteen sectors of track 0 which holds the boot file, and these should all be set. The next two bytes represent the sixteen sectors of track 1 which hold the bitmap and the directory, all of which should also be set.

CDOS then fills the disc in a sequential manner, starting from track 2 sector 0. Details of the file are stored in the directory which starts at track 1 sector 1. The first word indicates whether the entry slot is in use, and a zero indicates that the entry slot is free. If the file is a program, BASIC or code, then the first word is set to ASASH for autorun, and SASAH for not autorun. Any other value indicates the record size of a relative file.

The next eight bytes contain the title of the file in ASCII format, followed by the BASIC pointers in the case of a BASIC file, or the beginning and entry point for machine code. The word starting at byte number 16 contains the length of the file. The word starting at byte 32 contains the disc address followed by the number of contiguous blocks from that point. The next seven pairs of words are similar and this allows the file to be split into eight different areas on the disc if necessary.

The program starts by displaying the title, and then asking which drive to use. The drive number is used to index into a list of pointers, which indicate the locations in memory where the disc drive parameters are stored. The parameters contained in memory include the number of blocks per track, the total number of blocks, the number of files, the number of tracks, the number of sides and the number of bytes per sector. It was decided to use these parameters rather than fixed values so that the program will hopefully work on all density drives.

The program then calculates the position of the bitmap and directory and passes this information onto the read/write disc routine. The bitmap and directory are retrieved from the disc and stored from location A000H onwards. A temporary buffer starting from location 9000H is cleared and another bitmap is created using the information from the directory of the disc. A check is built in to make sure any disc address indicated by directory entries are valid.
When the second bitmap has been created, it is then compared with the actual bitmap from the disc, with any discrepancies being listed. If any a discrepancy is found in the bitmap for the bootfile or the directory track, the bitmap on the disc can be set to all "ones" using the disc inspect utility. If a discrepancy is found in a file, then the best course of action is to copy all of the files to a new disc using the "filecopy" utility.

```
7100 START:  LWPI >F020       print title
7104       MSG @7000       print "which drive"
7108 DRIVEND: MSG @7016  get drive number
710C       ANDI R1,>OF00  mask ASCII
7110       MOV R1,R2      put in lower byte
7112       SWPB R2        pointer to drive
7114 SLA R2,1               blocks per track
7116       MOV @>6382(R2),R3  total number of blocks
7118       MOV *R3+,R4      number of files
711C       MOV *R3+,R9      pointer to drive
711E       MOV R9,>70FE     number of sides
7120       INCT R3          number of tracks
7124       MOV *R3,R12     bytes per sector
7126       CLR R8           >calculate disc address
7128       MOV @>6372(R2),R3  of bitmap
712A       MOV @0006(R3),R5  no. of bytes to transfer
712C       MPY R4,R5      actual bitmap buffer
7130       DIV R6,R8      get bitmap & directory
7132       MOV @>6362(R2),R3  check status
7134       MPY R3,R4      print error message
7136       MOV R5,R2      clear buffer
7138       MOV R5,R4      set bits for bootfile
713A       DIV R6,R8      set bits for directory
713C       LI R0,>0000    get file directory
7140       LI R3,>A000  no file?
7142       BLWP @6180    number of blocks
7144       MOV R0,R0     disc address
7146       LDI @6550     is it valid disc address?
7148       LI R1,>9000  print "invalid address"
714A       CLR *R1+     calculate which block
714C       JNE AGAIN  
714E       JNE AGAIN  
7150       JEQ OK1:    
7154 OK1:   B @6550      
7158 AGAIN: CLR *R1+     
715A       CI R1,>A000  
715C       JNE AGAIN  
7160       LI R1,>9000  
7162       SETO *R1+    
7164       SETO *R1    
7166       AI R3,>0080  
7168 NEXTFILE: MOV *R3,*R3  
716A       JEQ NEXT   
716C       MOV @0022(R3),R4  
7170       MOV @0020(R3),R5  
7172       JEQ NEXT   
7174       MOV R5,R6   
7176       SRL R6,4     
7178       C R6,RB     
717C       JLE OK2     
717E       MSG @7028   
7180       JMP BADADD  
7182       JMP BADADD  
7184 OK2:  MOV R6,R2   
7186       SLA R2,1   
7188       ANDI R5,>000F  
718A       CLR R6     
7190
```

6.5
The next program is by J.M.Terry, and is a 3D plane plotter. Although we have already featured a 3D graph program, this is different in that it produces an image with hidden lines omitted, thus adding to the 3D effect. There is also the facility to call a suitable screen dump routine, such as previously featured in the newsletter. The program is written entirely in BASIC, and shows the computing power of the Cortex.
1000 REM ** 3D PLANE PLOTTER **
1010 REM ** BY J.M.TERRY **
1370 REM * Initialisation *
1410 PI=3.1415926536 /DEFINE PI
1420 DIM $FUNC2[255],LY[255] /DIMENSION VARIABLES
1450 REM * Command level *
1480 TEXT
1500 PRINT "3D PLANE.PLOTTER"
1510 PRINT
1520 GOSUB 2090 /GET INPUT DATA
1530 GOSUB 1680 /PLOT THE GRAPH
1540 PRINT @0,(0,23);"Do you want a screen dump?(Y/N)";
1550 INPUT #1:$INV /READ IN ONE CHARACTER
1560 PRINT @0,(0,23);" " /CLEAR S.D.MESAGE
1570 IF $INV="Y" THEN GOTO 1600
1580 IF $INV="N" THEN GOTO 1610
1590 GOTO 1540 /GET A VALID INPUT
1600 GOSUB 1960 /DUMP SCREEN TO PRINTER
1610 TEXT
1620 PRINT "Do you want to plot another function?(Y/N)";
1630 INPUT #1:$INV
1640 IF $INV="Y" THEN GOTO 1490 /RESTART PROGRAM
1650 IF $INV="N" THEN GOTO 1670 /END PROGRAM
1660 GOTO 1620
1670 END
1680 REM * Plot graph routine *
1730 GRAPH
1740 FOR A=0 TO 255 /FILL UPPER Y LIMIT WITH 191 TO ALLOW PLOTTING
1750 UPY[A]=191
1760 NEXT A
1790 FOR Z=5 TO WID+6 /Z-AXIS COUNT FROM NEAR TO FAR
1800 FOR X=5 TO WID+6 /X-AXIS COUNT ACROSS SCREEN
1810 REM LINE 1820=ONLY PRINT IF CORRECT LINE POSITION REACHED AND AND DIRECTION SELECTED
1820 IF (DIR<90)*(MODI(Z-5),DNNJ)=0 OR (DIR>88)*(MODI(X-5),DNNJ)=0
1830 THEN GOTO 1830
1830 VZ=LZ+(Z-5)*(UZ-LZ)/WID; VX=LX+(X-5)*(UX-LX)/WID /VIRT. X,Z
1840 PTX=X+Z*TX /PLOTTING VALUE OF X
1850 Y=FNA[VX,VZ] /GET Y VALUE AT X,Z CORRECTED FOR VERTICAL TILT
1860 GOTO 1870
1870 IF Y<LY OR Y>UY THEN GOTO 1930 /OFF SCREEN POINT NOT PLOTTED
1880 PTY=186-186/(UY-LY)X(Y-LY)-Z*ZV /GET PLOTTING VALUE OF Y
1890 IF PTY>LY[PTX] THEN LY[PTX]=PTY ; GOTO 1910 /IF POINT IS VISIBLE BELOW ANY POINT ALREADY THERE, THEN PLOT IT
1900 IF PTY>UPY[PTX] THEN UPY[PTX]=PTY /SAVE IF NEW LIMIT
1910 PLOT PTX,PTY /PLOT POINT ON SCREEN
1920 IF PTY<UPY[PTX] THEN UPY[PTX]=PTY /SAVE IF NEW LIMIT
1930 NEXT X
1940 NEXT Z
1950 RETURN
1960 REM * Screen dump *
1970 REM
1980 REM Call your screen dump routine here
1990 REM
2020 PRINT "The function : F(X,Z)=";FUN[0];19]
2030 PRINT "X-range is ";LX;" to ";UX
2040 PRINT "Y-range is ";LY;" to ";UY
2050 PRINT "Y-range is ";LY;" to ";UX
2060 PRINT "Vertical tilt is ";TXV
2070 PRINT "Side tilt is ";ZTX
W.D.Eaves sent in the next program and the following explanation of what it does.

Some programs which use BASIC and M/C require that the NEW command is used to load the BASIC at a higher address so that when the M/C is loaded it will not overwrite the BASIC. Examples of this are the games PENGO and FIREBIRD. On the tapes the NEW address is shown as part of the loading instruction. However, if such a program is transferred onto a disk then there is no written instruction and the program can be loaded without first typing NEW xxxx. Obviously the program will not work as the M/C will overwrite the BASIC. Even typing NEW xxxx does not overcome the problem as the BASIC checks a location to see if the M/C has been loaded. However, if the BASIC is at the default address then this check is sometimes fooled by reading part of the BASIC program, and the M/C will not be loaded.

The following example shows how a program can be loaded at any NEW address, and if it is not correct will reload the program at the correct address. The default NEW parameter is unaltered so that when the program is finished the original NEW address will be selected just by typing NEW.

Memory location  ED04 contains the address at which a BASIC program will be loaded (This is the NEW address + 14H). Location ED06 contains the xxxx specified by NEW or the default address. Thus by checking the value of the word at ED04 and modifying if necessary, a program can be loaded at the correct address. (see line 6 in the example). Not altering the word at ED06 leaves the NEW default value intact.

If the word at ED04 is not correct then the program goes to the subroutine shown in the example at line 8000. This routine resets the check location to zero (line 8005), and then reads data to
assemble the M/C shown in the example below the BASIC. If the M/C is assembled at 6090H then the first four data lines in the BASIC are valid in all cases except the data for the BASIC address marked *. The last line contains the program name, in this case PENG0 terminated by 00. (see the routine on p.5 of newsletter III by Tim Gray.) The branch to 0F18 at 6098 initialises the BASIC memory at the new address.

Please note that this routine works only with disk drives as it uses part of the CDOS software, and in any case would be incapable of rewinding a tape! However, the memory check at line 5 and the message at line 8000 can be used with any data storage medium.

EXAMPLE OF AUTOMATIC 'NEW'

4 MOTOR 0
5 IF MWD[0ED04H]<9014* THEN GOTO 8000 /check BASIC start address
6 IF MWD[07B10H] THEN GOTO 30 /check if M/C loaded
8 COLOUR 11,0

8000 ? "<OC>YOU forgot to type 'NEW xxxx'":?"I'll do it automaticall y!"
8005 MWD[07B10H]=0 /set to 0 as M/C not loaded
8050 READ C1,C2: FOR I=C1 TO C2 STEP 2 /assemble M/C
8052 READ C: MWD[I]=C: NEXT I
8053 CALL C1 /reloads program at new address
8054 DATA 6090H, 60AEH
8056 DATA 513, 9014H*, -14335, -4860
8058 DATA 1696, 504, 1217, 514
8060 DATA 24746, 523, 128, 1120, 26012
8062 DATA 20549, 20039, 20224 /program name

MACHINE CODE TO RELOAD PROGRAM

6090 LI R1,>9014* /set BASIC start address
6094 MDV R1,@ED04
6098 BL @>01F8 /execute NEW, default left intact
609C CLR R1
609E LI R2,>60AA /address of program name start
60A2 LI R11,>0080 /load program
60A6 B @>659C
60AA DATA >5045 /ASCII codes of program name, 00 terminated
60AC DATA >4E47
60AE DATA >4F00

Well that's all for this issue. If you have any interesting programs or routines that you would like published, then please send them in. We would ask that you also send a description of the way in which the program works, so as to help other users.

The programs published in this issue will be available on tape. Please see page 2 for details.
USER INFO

John Mackenzie has written recommending the 'COMMTEX' communications package by MARKRO SOFT. It is very flexible and written/structured in such a way that makes it very adaptable by the user. For users of WORTEX, there is a special version of COMMTEX which receives and sends WORTEX pages. John will supply this free to any Wortex user who sends him a disk with Commtex on it. (As proof of purchase of Commtex.)

John also informs us that version 1.5 is now available. To get an updated copy send your original Wortex disk back to him.

For those of you who are new to the user group WORTEX is a complete word processor (disc based) for the Cortex. For details write to;
John Mackenzie, 4 Werston Close, Malvern, Worcs. WR14 3NH

COMMTEX is available from;
P. Roe, 53 Broughton Road, Croft, Leics. LE9 6EB

Phillip Marsden from Leeds wrote in search of some information. He has bought the memory card from MPE, and plans to make a half-megabyte board. He has thoughts about a RAM disc routine to allow faster disk access, and wonders if any other users have already achieved this.

In addition to this, he would like to produce an 80 column screen output, and would like any relevant information on the screen output.

If you can help with either of these requests then we will be glad to pass on information.

 Ladislav Vig of Switzerland wrote to us asking if anyone else is using the MDEX software. He would like to exchange some information.

A.R.C. Badcock wrote to express his praise for CDOS, particularly because it is easily modified, and well supported by its author, Neil Quarby. He also uses the MDEX system, and asks the following questions:

1) Has anyone a utility to read and write to CDOS discs from MDEX, or to transfer files intact (like 'RDCPM' does for the MSDOS)?

2) Has anyone a utility to read and write to cassettes whilst in MDEX, so that I can tape software for safer archiving.

3) Has anyone a fix for the bug in the MDEX BASIC interpreter that prevents the SAVE command from saving compiled code. The interpreter recognises the first 4 letters as a SAVE command which it rejects as source is no longer present. Although the BASIC is a simple one, it would be useful if the compiled feature could be exploited. Perhaps the command table could be patched to rename the command?
FEATURE: ADDING EXTRA BASIC STATEMENTS (By Tim Gray)

BASIC statements are stored in memory in encoded form. When entering BASIC program lines, a check is made to see if the statement entered is included in a table during the normal syntax checking procedure. When the name is found, its position from the start of the table becomes the token in the program. Now when the program is running, this token is used to access a routine start address in another table, and a branch made to the start address of the statements routine.

Having said all that, it's possible to add extra statements by adding extra names to the name lookup table, and start addresses to the start address table.

As some statements have more than three letters there are actually two name tables, one for the first three letters and one for the rest of the name. A list of all the tables is included. Note that this is the list after loading CDO, and includes some changes and extra words used by the file system.

The first table for the name starts at 3020H, and is a 16 bit word table encoded as follows:

<table>
<thead>
<tr>
<th>bit 0</th>
<th>bit 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If the LSB of the word is set to one, then the second name table is used to encode the second part of the name. This second table starts at 3ADAH. The start address for the routines are in a table starting at 3FCCCH. Once your new statement, name and start address is included in these tables, any program can use them.

When the program comes across your new statement it will branch to the routines start address. This is a direct branch so your routine must preserve some of the registers, especially R8 and R15. On completion, your extra routine will have to branch back to a location to continue running the BASIC program. This branch back address is different depending on the type of parameters used. I don’t know all the rules for this part of BASIC, but I have found that some return addresses are 3F2C 3F30 3F36 etc—you will have to experiment.

Also included in the table lists are the tables for functions and some more three letter statements.

STATEMENTS

<table>
<thead>
<tr>
<th>ADR1 WD 1 ADR2 WD 2</th>
<th>TABLE SADR NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>3A2E A3CF 3ADA 001E</td>
<td>3FCC 24FC GOTO</td>
</tr>
<tr>
<td>3A30 9BCF 3ADC 00AA</td>
<td>3FCE 2500 GOSUB</td>
</tr>
<tr>
<td>3A32 9B08 3A0E 000A</td>
<td>3FD0 3FD0 ELSE</td>
</tr>
<tr>
<td>3A34 6964 3AE0 4700</td>
<td>3FD2 3F36 REM</td>
</tr>
<tr>
<td>3A36 93CC 3AE2 FF80</td>
<td>3FDA 2146 FOR</td>
</tr>
<tr>
<td>ADR1 WD 1</td>
<td>ADR2 WD 2</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>3A3B 0000</td>
<td>3AE4 0000</td>
</tr>
<tr>
<td>3A3A 0000</td>
<td>3AE6 0002</td>
</tr>
<tr>
<td>3ACC C15D</td>
<td>3AEB 0028</td>
</tr>
<tr>
<td>3AE5 948B</td>
<td>3AEC 049E</td>
</tr>
<tr>
<td>3AA0 4CA1</td>
<td>3AED 051C</td>
</tr>
<tr>
<td>3A42 6047</td>
<td>3AEE 0018</td>
</tr>
<tr>
<td>3A44 0BD9</td>
<td>3AFF 0008</td>
</tr>
<tr>
<td>3A46 8393</td>
<td>3AFF2 052A</td>
</tr>
<tr>
<td>3A48 0965</td>
<td>3AFF4 0008</td>
</tr>
<tr>
<td>3A4A 9965</td>
<td>3AFF6 93EB</td>
</tr>
<tr>
<td>3A4C A165</td>
<td>3AFF8 744A</td>
</tr>
<tr>
<td>3A4E 7D27</td>
<td>3AFFA 0020</td>
</tr>
<tr>
<td>3A50 4B08</td>
<td>3AFFC 0028</td>
</tr>
<tr>
<td>3A52 6A69</td>
<td>3AFFE 000A</td>
</tr>
<tr>
<td>3A54 B067</td>
<td>3B00 000A</td>
</tr>
<tr>
<td>3A56 9845</td>
<td>3B02 000A</td>
</tr>
<tr>
<td>3A58 1CCB</td>
<td>3B04 2022</td>
</tr>
<tr>
<td>3A5A 28DD</td>
<td>3B06 000E</td>
</tr>
<tr>
<td>3A5C 7065</td>
<td>3B08 6B8C</td>
</tr>
<tr>
<td>3A5E A845</td>
<td>3B0A 0008</td>
</tr>
<tr>
<td>3A60 A38B</td>
<td>3B0C 048A</td>
</tr>
<tr>
<td>3A62 7B21</td>
<td>3B0E 0028</td>
</tr>
<tr>
<td>3A64 83AB</td>
<td>3B10 A3DB</td>
</tr>
<tr>
<td>3A66 63C7</td>
<td>3B12 955E</td>
</tr>
<tr>
<td>3A68 9561</td>
<td>3B14 014E</td>
</tr>
<tr>
<td>3A6A 0CBF</td>
<td>3B16 0220</td>
</tr>
<tr>
<td>3A6C C169</td>
<td>3B18 0228</td>
</tr>
<tr>
<td>3A6E 486F</td>
<td>3B1A 0028</td>
</tr>
<tr>
<td>3A70 0A07</td>
<td>3B1C 0024</td>
</tr>
<tr>
<td>3A72 6D5D</td>
<td>3B1E 9144</td>
</tr>
<tr>
<td>3A74 9A59</td>
<td>3B20 0028</td>
</tr>
<tr>
<td>3A76 7165</td>
<td>3B22 036A</td>
</tr>
<tr>
<td>3A78 9427</td>
<td>3B24 2012</td>
</tr>
<tr>
<td>3A7A 0A27</td>
<td>3B26 0160</td>
</tr>
<tr>
<td>3A7C AC27</td>
<td>3B28 0028</td>
</tr>
<tr>
<td>3A7E 29E7</td>
<td>3B2A 0028</td>
</tr>
<tr>
<td>3A80 7BC5</td>
<td>3B2C 0028</td>
</tr>
<tr>
<td>3A82 0DE7</td>
<td>3B2E 0020</td>
</tr>
<tr>
<td>3A84 7B07</td>
<td>3B30 0102</td>
</tr>
<tr>
<td>3A86 3DBE</td>
<td>3B32 049E</td>
</tr>
<tr>
<td>3A88 0CC7</td>
<td>3B34 016C</td>
</tr>
<tr>
<td>3A8A 21CF</td>
<td>3B36 001C</td>
</tr>
<tr>
<td>3A8C 7B07</td>
<td>3B38 016B</td>
</tr>
<tr>
<td>3A8E A14E</td>
<td>3B3A 0000</td>
</tr>
<tr>
<td>3A90 A560</td>
<td>3B3C 0000</td>
</tr>
<tr>
<td>3A92 0000</td>
<td>3B3E 0030</td>
</tr>
<tr>
<td>3A94 0000</td>
<td>3B40 0040</td>
</tr>
<tr>
<td>3A96 0000</td>
<td>3B42 0040</td>
</tr>
<tr>
<td>3A98 0000</td>
<td>3B44 0040</td>
</tr>
<tr>
<td>3A99 0000</td>
<td>3B46 0040</td>
</tr>
<tr>
<td>3A9C 0000</td>
<td>3B48 0040</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADR1 WD 1</th>
<th>TABLE SADR NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>3A9E 385A</td>
<td>403C 18FA MAG</td>
</tr>
<tr>
<td>3A9A 33EB</td>
<td>403E 31DE TOF</td>
</tr>
<tr>
<td>3AA2 73EB</td>
<td>4040 31D6 TON</td>
</tr>
<tr>
<td>3AA4 83E0</td>
<td>4042 259C POP</td>
</tr>
<tr>
<td>3AA6 6A48</td>
<td>4044 2032 DIM</td>
</tr>
<tr>
<td>3AA8 A15B</td>
<td>4046 2772 LET</td>
</tr>
<tr>
<td>3AAA 0000</td>
<td>4048 2AB8</td>
</tr>
<tr>
<td>3AAC 73C0</td>
<td>404A 29FA ON</td>
</tr>
<tr>
<td>3AAE 3240</td>
<td>404C 25AE IF</td>
</tr>
<tr>
<td>3AB0 314B</td>
<td>404E 200E DEF</td>
</tr>
<tr>
<td>3AB2 895C</td>
<td>4050 01CE NEW</td>
</tr>
<tr>
<td>3AB4 238A</td>
<td>4052 1E3A END</td>
</tr>
<tr>
<td>3AB6 0000</td>
<td>4054 2AB8</td>
</tr>
<tr>
<td>3ABB 0000</td>
<td>4056 6E40</td>
</tr>
<tr>
<td>3ABA A244</td>
<td>4058 4E6A BIT</td>
</tr>
<tr>
<td>3ABC 1486</td>
<td>405A 1F6A CRB</td>
</tr>
<tr>
<td>3ABE 3468</td>
<td>405C 1FBC CRF</td>
</tr>
<tr>
<td>3ACC 695A</td>
<td>405E 290B MEM</td>
</tr>
<tr>
<td>3AC2 25DA</td>
<td>4060 299E MWD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FUNCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3B4A 9882</td>
</tr>
<tr>
<td>3B4C 910C</td>
</tr>
<tr>
<td>3B4E 1CC2</td>
</tr>
<tr>
<td>3B50 7502</td>
</tr>
<tr>
<td>3B52 9BC6</td>
</tr>
<tr>
<td>3B54 860A</td>
</tr>
<tr>
<td>3B56 0CBC</td>
</tr>
<tr>
<td>3B58 A392</td>
</tr>
<tr>
<td>3B5A 3B08</td>
</tr>
<tr>
<td>3B5C 9CB6</td>
</tr>
<tr>
<td>3B5E 7266</td>
</tr>
<tr>
<td>3B60 9466</td>
</tr>
<tr>
<td>3B62 9E66</td>
</tr>
<tr>
<td>3B64 1A68</td>
</tr>
<tr>
<td>3B66 7136</td>
</tr>
<tr>
<td>3B68 A244</td>
</tr>
<tr>
<td>3B6A 1486</td>
</tr>
<tr>
<td>3B6C 3486</td>
</tr>
<tr>
<td>3B6E 695A</td>
</tr>
<tr>
<td>3B70 250A</td>
</tr>
<tr>
<td>3B72 715B</td>
</tr>
<tr>
<td>3B74 404A</td>
</tr>
<tr>
<td>3B76 9BE0</td>
</tr>
<tr>
<td>3B78 63C6</td>
</tr>
<tr>
<td>3B7A 23DA</td>
</tr>
<tr>
<td>3B82 0000</td>
</tr>
</tbody>
</table>
**SHORT TIPS**

The first tip this issue comes from A.R.C. Badcock, and is concerned with the RGB interface circuit. Upon building this board it only gave out black. To solve this he adjusted the biasing of TR8, TR12, and TR16. To achieve the correct colour balances he changed R23, R37 and R48 to 1K0.

Mr. Badcock would also like to warn users not to do the "3.5K FREE RAM" mod, as this is a non-reversible alteration. By installing the memory mapper chip, with no PCB changes (other than removing links) all 4K is accessible under software control.

---

**Pem Holdaway** is one of our newer members from London. He has been going through the older newsletters, and has this suggestion for correcting the lower case data from issue 2 (Andy Kendall's letter).

Line 80 should read:

```
80 DATA 3,-28087,8962,7,4673,1024,3,-28624,9984,16655,4161,8960
```

---

**Bill Eaves** has two tips about CDOS. In issue 3 page 5 there are some suggested modifications, which do not apply to CDOS 1.2. This version already has an auto-load facility which loads a file called AUTOEXEC from the **BOOT** command. The routine which performs the auto-load is situated at 6940H. If users of CDOS 1.2 wish to load filenames of their own choice the hex ASCII codes should be entered at 6938H to 693EH. If a filename of less than 8 characters is used, it should be terminated by 00H.

Bill also informs us that certain programs will not work properly after CDOS has been loaded. The problem is that some programs were written before a way was known to correct the COL function, checked pixels for the latest foreground and background colours. CDOS corrects the fault so such a program checks for the wrong colours! The simplest way (though perhaps not the most elegant) is to set location 1D12H to EE95H which is the value when the Cortex is reset or switched on. Remember to change the the value back to F120H when the program has finished. CDOS 1.2 performs the COL correction at location 69EEH.

---

**Julian Terry** tells us that printing character 10H will stop the cursor from being plotted. Unfortunately there is no way of getting it back without clearing location ED6AH, as reported by Robert in newsletter 2.

---

**John Mackenzie** has a number of points to make about CDOS.

1) The AUTOEXEC program is useful for holding all the little mods and debugs to **BOOT** the Cortex as you require. Here is a little bit for you to add to it.

```
xxxxx BAUD 2,1200 : BASE 00BH
```

---

6.13
This will set up the RS232 port to 1200 Baud, (or amend to suit your printer) and 8 bit. This allows your printer to print all the characters above ASCII 127. Now when you want the printer just type UNIT 2. Retype BAUD 2,1200 to reset to 7 bit.

2) CDOS does not have a system of marking U/S sectors on the disk directory during formatting. A method of doing this comes from the way CDOS saves the files to disk. If during a SAVE to disc you get a persistent disk error (ie a sector is faulty), the system will have already updated the directory. If you rename that file RUBBISH, then when next you save to that disk the bad sector is not used. If you are very clever you can identify the offending sector, and save a very short program over it.

3) With reference to the AUTOLOAD program in issue 5, if you amend the listing as follows then all files will be listed on the screen with no scrolling if the directory is long.

If you call this program AUTO2, then add this last line to your AUTOEXEC program:

```
xxxxxx LOAD 0,"AUTO2"
```

Now change these lines;

```
40 ?:?;" Auto file load from disk 1";?
210 D=1
```

Save this program as AUTO3. Now copy AUTO2 and AUTO3 on to all your disks. Remember to amend line 2030 for each disk.

-----------------------------------------------

**MACHINE CODE PROGRAMMING**

[2] Addressing Modes (by Kevin Holloway)

In part 1 we dealt with moving data between registers, and incrementing/decrementing registers. We will obviously want to access data in the main memory as well, and there are a number of ways of doing this. These are called addressing modes, and the main ones will be discussed in this article.

We have already seen an example of immediate addressing, where a register is loaded directly with data (eg LI R1,1234). We have also seen register addressing, where another register holds the data (eg MOV R1,R2).

The next mode is register indirect addressing. A register holds the address at which the required data is stored. Thus if memory location 7000H contains our data, then we can load it into R2 by;

```
eg1) LI R1,7000
    MOV *R1,R2
```

```
.Load R1 with 7000H
Copy the data stored at the address in R1 into R2
```

The register R2 will now contain a copy of the data stored at location 7000H. The * indicates that the content of R1 is an address at which the required data is stored.
In the above example it would have been simpler to use indirect memory addressing. In this mode the data is loaded directly from memory.

eg2) MOV @>7000,R1  / Copy the data from location 7000H into R1 

The @ sign indicates that indirect addressing is being used.

Do not be worried if there seems to be so many ways of doing the same thing. Once each of the addressing modes is understood, you should be able to see that each one has its own particular use in different types of program.

If we want to use many related data items, say for example, a list of coordinates, then we will probably want to form a table of them. To do this we use indexed addressing. This is best illustrated by another example.

eg3) memory location 7000H ---> ! data0 !

--------
7001H ! data1 !
-------
7002H ! data2 !
-------
7003H ! data3 !
-------
7004H ! data4 !
-------

To access one of the entries we could just calculate the relevant address, but it is easier to use the start address (7000H) as a reference, and a register as an index pointer. Thus to load data3 into register R2 we would do the following:

LI R1,3
MOVB @>7000(R1),R2

This copies the data from address 7000H+R1(=3), [ie 7003H], into R2.

As an extension to the register indirect addressing, we may want to access several data items which are stored sequentially in memory. This may be achieved by using auto-incrementing.

eg4) MOV *R3+,R2

The plus sign following R3 indicates that the contents of the register are to be incremented by two immediately after copying the contents of the address in R3 into R2.

If R3=7000H, and the location 7000H contains the value 1234H, then after executing the above instruction, R2=1234H, and R3=7002H

So far we have only discussed ways of moving data from one place to another (excluding increment/decrement). In the next issue we will move on to look at how we can perform logical operations and simple arithmetic. If there are any points which you would like covered in more detail, then please write and let me know.
On our newsletter program tapes we usually include a short feature BASIC program written by our staff. There are many interesting mathematical problems which can be solved numerically, and therefore are suitable material for programming. It was our intention to market a series of such programs, but it was decided that it would be more useful to print separate articles in the newsletter.

This program calculates and prints out odd magic squares using very simple rules. Any size of square is possible, although the screen size restricts the display to a 9*9 square. The method of generating even magic squares is a little more complex, and so will not be shown here.

For those of you who do not know, a magic square is quite simply a square array of numbers in which every row, column, and long diagonal adds up to the same number. (see fig.1)

8 1 6 fig.1 every row, column and long diagonal
3 5 7 adds up to 15.
4 9 2

The method of producing an odd magic square is quite simple. You start off by filling in the middle element of the top row with a 1. You then proceed to move diagonally upwards to the right filling in successively 2, 3, etc. (NB imagine the square to wrap around itself. ie if you move off the left side then you must rejoin the right side.). If you come to a square which is already filled in then you move down two, left one and continue as before.

10 REM ODD MAGIC SQUARES
15 TEXT
20 INPUT "HOW MANY NUMBERS TO A SIDE (ODD)?";N
30 N=INT(N):IF INT(N/2)=N/2: GOTO 20 ! make sure N is int & odd
40 DIM SQ(N,N)
50 FOR I=1 TO N
60 FOR J=1 TO N
70 SQ(I,J)=0 ! clear all elements of square
80 NEXT J
90 NEXT I
100 I=1+INT(N/2):J=1 ! set i,j to middle of top row
105 SQ(I,J)=1 ! set this element to 1
110 FOR C=2 TO N*N ! rest of elements
130 I=I+1:J=J-1 ! move diagonally upwards and right
140 IF I>N THEN I=I-N ! allow for wrap-around
145 IF I<1 THEN I=I+N
150 IF J<1 THEN J=J+N
155 IF J>N THEN J=J-N
160 IF SQ(I,J)<0 THEN J=J+2:I=I-1:GOTO 140 ! if new position full
then move down two, left one and try again
165 SQ(I,J)=C ! new position empty, so set to count value
170 NEXT C
175 REM PRINT MAGIC SQUARE
180 FOR I=1 TO N
190 FOR J=1 TO N
200 ?((I*3),(J*2));SQ(I,J)
210 NEXT J
220 NEXT I

6.16
Wortex

This is a Word Processor for the Cortex. It runs under CDOS 1.20. The system runs using Twin 40 track single sided single density disk drives. Operation with one drive can be done.

MODES
1. Input text
2. Input page from disk
3. Return input text
4. View disk page
5. Save page to disk
6. Print page/pages
7. Spelling check (requires Speltex)

FUNCTIONS
1. Text input with full character editing
2. Page formatting with:
   a. Auto page number
   b. Center text option
   c. Right justify option
   d. Auto left justification
   e. Left margin control
   f. Right margin control
   g. Auto return
   h. Word wrap
   i. 15 Tab markers
   j. Page length control
   k. Page editing
3. Copy from disk page to memory page
4. Multi page printing

£15.00 plus a 5 1/4 blank disk

Speltex

The spelling checker for Wortex. This runs under CDOS 1.20. The system uses twin 40 track single sided disks with drive 'B' Single Density and drive '1' Double Density. (NOTE only the most recent version CDOS 1.20 supports Double Density).

This is a must for Wortex users. Comes with about 7000 words and the dictionary can go up to around 20000 words.

MODES
1. Check page spelling
2. Edit the Dictionary
3. Return to Wortex
4. Correct errors

FUNCTIONS
1. View the errors
2. Correct the errors
3. Store the error word in the dictionary
4. Add words to Dictionary directly from keyboard
5. Delete words from the Dictionary

£18.00 plus two 5 1/4 DD Disks to

J S Mackenzie
4 Warstan Close
Malvern
WRI 14 3NH

call 06845-65819 evenings

6-17