* CORTEX USER GROUP *

NEWSLETTER ISSUE NO. 6   May 1986

CONTENTS

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

---

kph computaware

63 Highlands Road, Andover, Hants. SP10 2PZ

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 compatabile 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

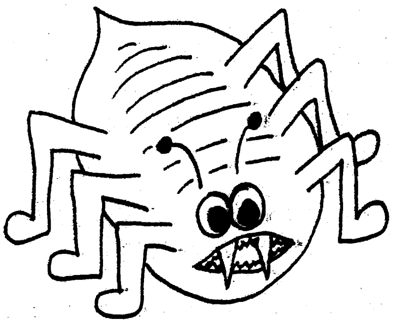
THE LABYRINTH OF TRAG is the first adventure game for the Cortex. In this text based game you have to explore a series of underground rooms and passageways. Your aim is to stay alive by eating and drinking on the way, whilst looking for keys to open boxes. Your eventual goal is to open the treasure chest, 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.

TMS 9901	@	£6.50
TMS 9902	@	£6.50
TMS 9911	@	£25.00
TMS 9929	@	£22.00
2797 FDC	@	£36.00
74LS612	@	£25.00



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.N.Sedwell of Christchurch is having trouble with a timing related fault somewhere around the TMS 4500, 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.

PROGRAMS

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 thens 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 that 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 A5A5H for autorun, and 5A5AH 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 discdrive 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
7104            MSG    @>7000            print title
7108 DRIVENO:   MSG    @>7016            print "which drive"
710C            EKO    R1                get drive number
710E            ANDI   R1,>0F00          mask ASCII
7112            MOV    R1,R2
7114            SWPB   R2                put in lower byte
7116            SLA    R2,1
7118            MOV    @>6382(R2),R3     pointer to drive
711C            MOV    *R3+,R4           blocks per track
711E            MOV    *R3+,R9           total number of blocks
7120            MOV    R9,@>70FE
7124            INCT   R3
7126            MOV    *R3,R12           number of files
7128            CLR    R8
712A            MOV    @>6372(R2),R3     pointer to drive
712E            MOV    @>0006(R3),R5     number of sides
7132            MPY    R4,R5
7134            DIV    R6,R8             number of tracks
7136            MOV    @>6362(R2),R3     bytes per sector
713A            MPY    R3,R4             )calculate disc address
713C            MOV    R5,R2             )of bitmap
713E            MOV    R5,R4             no. of bytes to transfer
7140            LI     R0,>0000
7144            LI     R3,>A000          actual bitmap buffer
7148            BLWP   @>6180            get bitmap & directory
714C            MOVB   R0,R0             check status
714E            JEQ    OK1
7150            B      @>6550            print error message
7154 OK1:       LI     R1,>9000
7158 AGAIN:     CLR    *R1+              clear buffer
715A            CI     R1,>A000
715E            JNE    AGAIN
7160            LI     R1,>9000
7164            SETO   *R1+              set bits for bootfile
7166            SETO   *R1               set bits for directory
7188            AI     R3,>0080
718C NEXTFILE:  MOV    *R3,*R3           get file directory
718E            JEQ    NEXT              no file?
7170            MOV    @>0022(R3),R4     number of blocks
7174            MOV    @>0020(R3),R5     disc address
7178            JEQ    NEXT
717A            MOV    R5,R6
717C            SRL    R6,4
717E            C      R6,R8             is it valid disc address?
7180            JLE    OK2
7182            MSG    @>7028            print "invalid address"
7186            JMP    BADADD
7188 OK2:       MOV    R6,R2             calculate which block
718A            SLA    R2,1
718C            ANDI   R5,>000F
7190            CLR    R6
```

```
7192            AI    R6,>8000
7196            MOV   R5,R5
7198            JEQ   SETBIT
719A MORESEC:   SRL   R6,1
719C            DEC   R5
719E            JNE   MORESEC
71A0            JMP   SETBIT
71A2 MOREBLOK:  SRL   R6,1
71A4            JNC   SETBIT
71A6            INCT  R2
71A8            LI    R6,>8000
71AC SETBIT:    A     R6,@>9000(R2)    set bit in map
71B0            DEC   R4
71B2            JNE   MOREBLOK         any more blocks?
71B4 NEXT:      AI    R3,>0040         next file entry
71B8            DEC   R12              any more files?
71BA            JNE   NEXTFILE
71BC            LI    R1,>A000
71C0            LI    R2,>9000
71C4            MOV   @>70FE,R3
71C8            SRL   R3,4
71CA            C     *R1+,*R2+        compare actual to
71CC            JEQ   OK3              calculated bitmap
71CE            MSG   @>7044           bootfile error
71D2 OK3:       C     *R1+,*R2+
71D4            JEQ   OK4
71D6            MSG   @>7066           directory error
71DA OK4:       DECT  R3
71DC NEXTBLOK:  C     *R1+,*R2+
71DE            JEQ   OK5
71E0            MSG   @>7088           file error
71E4 OK5:       DEC   R3
71E6            JNE   NEXTBLOK
71E8 ANOTHER:   MSG   @>70A6           ask if another disc
71EC            EKO   R1
71EE            CI    R1,>5900         yes?
71F2            JEQ   DRIVENO
71F4            CI    R1,>4E00
71F8            JNE   ANOTHER
71FA            B     @>0080           back to monitor
71FE            DATA  0
7200 BADADD:    MOV   R3,R6            print file name
7202            LI    R0,>0007         with bad address
7206            INCT  R6
7208 NEXTCHAR:  MOVB  *R6+,R7
720A            WRIT  R7
720C            DEC   R0
720E            JNE   NEXTCHAR
7210            JMP   NEXT
```

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 ommited, 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 $FUN[22],UPY[255],LY[255]   /DIMENSION VARIABLES
1450 REM * Command level *
1480 TEXT
1500 PRINT "        3D PLANE.PLOTTER"
1510 PRINT
1520 GOSUB 2090     /GET INPUT DATA
1530 GOSUB 1680     /PLOT THE GRAPH
1540 PRINT @(0,23);"Do you want a screen dump?(Y/N)";
1550 INPUT #1;$INP     /READ IN ONE CHARACTER
1560 PRINT @(0,23);"                                 "  /CLEAR S.D.MESSAGE
1570 IF $INP="Y" THEN GOTO 1600
1580 IF $INP="N" THEN GOTO 1610
1590 GOTO 1540   /GET A VALID INPUT
1600 GOSUB 1960  /DUMP SCREEN TO PRINTER
1610 TEXT
1620 PRINT "Do you want to plot another function?(Y/N)";
1630 INPUT #1;$INP
1640 IF $INP="Y" THEN GOTO 1490   /RESTART PROGRAM
1650 IF $INP="N" THEN GOTO 1670   /END PROGRAM
1660 GOTO 1620
1670 END
1680 REM * Plot graph routine *
1730 GRAPH
1740 FOR A=0 TO 255  /FILL UPPER Y LIMIT WITH 191 TO ALLOW PLOTTING
1750  UPY[A]=191
1760 NEXT A
1790 FOR Z=5 TO WID+6   /Z-AXIS COUNT FROM NEAR TO FAR
1800  FOR X=5 TO WID+6  /X-AXIS COUNT ACROSS SCREEN
1810   REM LINE 1820=ONLY PRINT IF CORRECT LINE POSITION REACHED AND
           AND DIRECTION SELECTED
1820   IF (DIR<>90)*(MOD[(Z-5),DNN]=0) OR (DIR<>88*(MOD[(X-5),DNN]=0
       ) THEN GOTO 1830
1830   VZ=LZ+(Z-5)*(UZ-LZ)/WID: VX=LX+(X-5)*(UX-LX)/WID /VIRT. X,Z
1840   PTX=X+Z*ZTX  /PLOTTING VALUE OF X
1850   Y=FNA[VX,VZ] /GET Y VALUE AT X,Z CORRECTED FOR VERTICAL TILT
1860   GOTO 1870
1870   IF Y<LY OR Y>UY THEN GOTO 1930 /OFF SCREEN POINT NOT PLOTTED
1880   PTY=186-186/(UY-LY)*(Y-LY)-Z*ZXV   /GET PLOTTING VALUE OF Y
1890   IF PTY>LY[PTX] THEN LY[PTX]=PTY : GOTO 1910    /IF POINT IS
        VISIBLE BELOW ANY POINT ALREADY THERE, THEN PLOT IT
1900   IF PTY>UPY[PTX]: GOTO 1930  /IF POINT HIDDEN THEN DON'T PLOT
1910   PLOT PTX,PTY   /PLOT POINT ON SCREEN
1920   IF PTY<UPY[PTX] THEN UPY[PTX]=PTY  /SAVE IF NEW LIMIT
1930  NEXT X
1940 NEXT Z
1950 RETURN
1960 REM * Screen dump *
1970 REM
1980 REM Call your screen dump routine here
1990 REM
2020 PRINT "The function : F(X,Z)=";$FUN[0;19]
2030 PRINT "X-range is ";LX;" to ";UX
2040 PRINT "Z-range is ";LZ;" to ";UZ
2050 PRINT "Y-range is ";LY;" to ";UY
2060 PRINT "Vertical tilt is ";ZXV
2070 PRINT "Side tilt is ";ZTX
```

```
2080 RETURN
2090 REM * Data input routine *
2140 INPUT "Please give a function of Y in terms of X and Z<OA><OA>
      <OD>";$FUN[0]     /INPUT FUNCTION
2150 REM * Get max and min axes values *
2160 PRINT "<OA><OA>Please give the value of"
2170 INPUT " lower X :";LX;" upper X :";UX    /GET X COORD RANGE
2180 INPUT " lower Z :";LZ;" upper Z :";UZ    /GET Z COORD RANGE
2190 INPUT " lower Y :";LY;" upper Y :";UY    /GET Y COORD RANGE
2200 INPUT "<OA><OA>Please give side tilt 0 to 1:";ZTX
2210 INPUT "<OA><OA>Please give vertical tilt 0 to 1:";ZXV
2220 PRINT "<OA><OA>Do you want lines in both X and Z directions?"
2230 INPUT #1;"If yes then enter Y, else enter X or Z";$DIR
2240 DIR=ASC[$DIR]
2250 IF DIR<>88 AND DIR<>89 AND DIR<>90 THEN GOTO 2220
2260 INPUT "<OA>How many lines do you want in each direction (1 to 3
      0?)";DEN
2270 $FUN[0;i]=/"2290 DEF FNA[X,Z]= "    /CREATE LINE STRING
2280 ENTER $FUN[0]    /ENTER LINE INTO PROGRAM
2290 REM * This line is replaced by the ENTER command *
2300 WID=245/(1+ZTX)    /CALCULATE X PLOTTING DISTANCE NEEDED
2310 DNN=INT[(WID)/(DEN-1)]  /CALCULATE SPACE BETWEEN PLOTTED LINES
2320 WID=DNN*DEN-DNN  /ADJUST WIDTH SO THAT ALL LINES ARE PLOTTED
2330 RETURN
```

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 PENGO 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]<>09014* THEN GOTO 8000  /check BASIC start address
6 IF MWD[07810H] THEN GOTO 30   /check if M/C loaded
8 COLOUR 11,0



8000 ? "<0C>You forgot to type 'NEW xxxx'":?"I'll do it automaticall
     y!"
8005 MWD[07810H]=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 MOV 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 Quarmby. 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 SAVEX 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 CDOS, 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:-

```
 bit 0                bit 15
     00000 00000 00000 0
       :     :     :   if set then this name has more than 3 letters
       :     :     :
       :     :     --- ascii code for 1st letter
       :     --------- ascii code for 2nd letter
       --------------- ascii code for 3rd letter
```

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 3FCCH. 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

ADR1	WD 1	ADR2	WD 2	TABLE	SADR	NAME
3A2E	A3CF	3ADA	001E	3FCC	24FC	GOTO
3A30	9BCF	3ADC	00AA	3FCE	2500	GOSUB
3A32	9B0B	3ADE	000A	3FD0	3FD0	ELSE
3A34	6964	3AE0	4700	3FD2	3F36	REM
3A36	93CC	3AE2	FF80	3FDA	2146	FOR

STATEMENTS

ADR1	WD 1	ADR2	WD 2	TABLE	SADR	NAME
3A38	0000	3AE4	0000	3FD6	2772	
3A3A	A049	3AE6	0002	3FD8	3F36	DATA
3A3C	C15D	3AE8	0028	3FDA	2240	NEXT
3A3E	948B	3AEA	049E	3FDC	1EA4	ERROR
3A40	4CA1	3AEC	051C	3FDE	2AB8	PRINT
3A42	6047	3AEE	0018	3FE0	1F1E	CALL
3A44	0BD9	3AF0	0008	3FE2	6580	LOAD
3A46	8393	3AF2	052A	3FE4	262E	INPUT
3A48	0965	3AF4	0008	3FE6	2CFA	READ
3A4A	9965	3AF6	93E8	3FE8	2D36	RESTOR
3A4C	A165	3AF8	74AA	3FEA	256C	RETURN
3A4E	7D27	3AFA	0020	3FEC	1E20	STOP
3A50	4BAB	3AFC	0028	3FEE	31E4	UNIT
3A52	6A69	3AFE	000A	3FF0	313C	TIME
3A54	B067	3B00	000A	3FF2	6790	SAVE
3A56	9845	3B02	000A	3FF4	1F86	BASE
3A58	1CCB	3B04	2C02	3FF6	2138	ESCAPE
3A5A	2BDD	3B06	00E6	3FF8	213E	NOESC
3A5C	7065	3B08	6BC8	3FFA	2C9E	RANDOM
3A5E	A845	3B0A	0008	3FFC	0164	BAUD
3A60	A38B	3B0C	048A	3FFE	20F6	ENTER
3A62	7B21	3B0E	0028	4000	5142	PLOT
3A64	83AB	3B10	A3D8	4002	513E	UNPLOT
3A66	63C7	3B12	955E	4004	1AD8	COLOUR
3A68	9561	3B14	014E	4006	2BFC	PURGE
3A6A	0C8F	3B16	0220	4008	1A84	GRAPH
3A6C	C169	3B18	0028	400A	1A7E	TEXT
3A6E	486F	3B1A	0028	400C	1B58	WAIT
3A70	0A07	3B1C	0024	400E	1A88	CHAR
3A72	6D5D	3B1E	9144	4010	29CC	NUMBER
3A74	9A59	3B20	0028	4012	3C0A	LIST
3A76	7165	3B22	036A	4014	2D78	RENUM
3A78	9427	3B24	2D12	4016	1B9E	SPRITE
3A7A	0A27	3B26	0160	4018	1B6A	SHAPE
3A7C	AC27	3B28	0028	401A	19FC	SPUT
3A7E	29E7	3B2A	0028	401C	1992	SGET
3A80	7BC5	3B2C	0028	401E	321A	BOOT
3A82	0DE7	3B2E	0020	4020	3348	SWAP
3A84	7B07	3B30	0102	4022	173A	CLOAD
3A86	A3DB	3B32	049E	4024	186E	MOTOR
3A88	0CC7	3B34	016C	4026	6540	CSAVE
3A8A	2C1F	3B36	001C	4028	6A00	OPEN
3A8C	7B07	3B38	0166	402A	6900	CLOSE
3A8E	A14E	3B3A	0000	402C	6AE0	GET
3A90	A560	3B3C	0000	402E	6AE4	PUT
3A92	0000	3B3E	0000	4030	0000	
3A94	0000	3B40	0000	4032	0000	
3A96	0000	3B42	0000	4034	0000	
3A98	0000	3B44	0000	4036	0000	
3A9A	0000	3B46	0000	4038	0000	
3A9C	0000	3B48	0000	403A	0000	

3 LETTER STATEMENTS

ADR1	WD 1	TABLE	SADR	NAME
3A9E	385A	403C	1BFA	MAG
3AA0	33E8	403E	31DE	TOF
3AA2	73E8	4040	31D6	TON
3AA4	83E0	4042	259C	POP
3AA6	6A48	4044	2032	DIM
3AA8	A158	4046	2772	LET
3AAA	0000	4048	2AB8	
3AAC	73C0	404A	29FA	ON
3AAE	3240	404C	25AE	IF
3AB0	3148	404E	200E	DEF
3AB2	B95C	4050	01CE	NEW
3AB4	238A	4052	1E3A	END
3AB6	0000	4054	2AB8	
3AB8	0000	4056	6E40	
3ABA	A244	4058	4E6A	BIT
3ABC	1486	405A	1FA6	CRB
3ABE	3486	405C	1F8C	CRF
3AC0	695A	405E	2908	MEM
3AC2	25DA	4060	299E	MWD

FUNCTIONS

3B4A	9882	48CA	2466	ABS
3B4C	910C	48CC	29C8	ADR
3B4E	1CC2	48CE	2936	ASC
3B50	7502	48D0	18B4	ATN
3B52	9BC6	48D2	5346	COS
3B54	860A	48D4	4A28	EXP
3B56	0C8C	48D6	2A9A	FRA
3B58	A392	48D8	2440	INT
3B5A	3BD8	48DA	5056	LOG
3B5C	C956	48DC	2482	KEY
3B5E	7266	48DE	535A	SIN
3B60	9466	48E0	53E6	SQR
3B62	9E66	48E2	241C	SYS
3B64	1A68	48E4	3128	TIC
3B66	7136	48E6	3038	SGN
3B68	A244	48E8	4E9E	BIT
3B6A	1486	48EA	1FEE	CRB
3B62	3486	48EC	1FC4	CRF
3B6E	695A	48EE	2924	MEM
3B70	25DA	48F0	29B8	MWD
3B72	7158	48F2	1F7C	LEN
3B74	40DA	48F4	1F6E	MCH
3B76	9BE0	48F6	1F4A	POS
3B78	63C6	48F8	1B1A	COL
3B7A	23DA	48FA	24C8	MOD
3B72	33CA	48FC	6918	EOF
3B7E	0000	48FE	0000	
3B80	0000	4900	0000	
3B82	0000	4902	0000	

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.

Prem 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 there 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.

```
    xxxx  BAUD 2,1200 : BASE 080H
    yyyy  CRB[14]=1 : CRF[8]=023H
```

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 formating. 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:

```
    xxxxx 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       /Load R1 with 7000H
        MOV *R1,R2        /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.

EXTRA FEATURE :MAGIC SQUARES

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
```

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 formating 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 51/4 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 '0' 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 dirrect from keyboard
5. Delete words from the Dictionary

£ 10.00 plus two 51/4 DD Disks to

J S Mackenzie
4 Werstan Close
Malvern
WR14 3NH

call 06845-65619 evenings