CORTEX USERS GROUP

93, Long Knowle Lane, Wednesfield, Wolverhampton, West Midlands, WV11 1JG

Tel No: T Gray 0902 729078, E. Serwa 0902 732659

19

CORTEX USER GROUP NEWSLETTER (OCT 1988)

Issue Number 19

CONTENTS

Editorial

We have had a few complaints that the newsletter is very late this time. It must be remembered that this is a users newsletter and as such is reliant on articles sent in by users. If we don't get anything sent in we can't print it. We usually suffer from a lack of enthusiasm during the summer months but have been holding this newsletter back in case anything came through. This issue will use up all the information in our file so if you have any programmes or other interesting articles to send please get them into us so thet we can produce the next issue without to much delay.

The 9938 VDP

The most interesting thing on the Cortex scene at the moment is the possibillity of adding a new updated graphics chip. The Yamahar 9938 is directly upwardly compatible with the 9929 as fitted in the Cortex so it is possible to fit it inside and retain all the existing fetures as well as adding more.

Maplin sell a kit that includes a 9938 VDP and printed circuit board as part of a framestore project they have. When made up the PCB can be fitted inside the cortex and wired to a 40 pin header plug in place of the original VDP. The new maplin PCB provides output as R.G.B. and sync feeds for use with an analogue R.G.B. monitor.

The new VDP works as a direct replacement for the 9929 so the cortex will function without any software update. But as well as the graphics and 40 column text modes you are used to it also provides other modes of operation including :-

TEXT MODE 2	80 column text mode
GRAPHICS MODE 3	same as graphics 2 but with sprite mode 2
GRAPHICS MODE 4	pixel mapped 256 X 212 with 16 colours
GRAPHICS MODE 5	pixel mapped 512 X 212 with 4 colours
GRAPHICS MODE 6	pixel mapped 512 X 212 with 16 colours
GRAPHICS MODE 7	pixel mapped 256 X 212 with 256 colours

Al the new graphics modes can use sprite mode 2 which allows more sprites and better colour control of them. Plus all the colours in the new VDP are programable via a colour pallette.

Chris Young has written a set of software drivers that allow the use of all the new graphics modes with the new VDP either fitted in place of the old one or fitted seperately on the E.bus.

Ted Serva has written the following guide to fitting the Maplin board inside the Cortex.

DIRECT REPLACEMENT OF TMS9928/29 FOR V9938

Hardware required

1 * V9938 KIT FROM MAPLIN

1 * 64 WAY A&C DIN 41612 CONNECTOR

1 * 40 PIN IC HEADER/IC SCKT

1 * 14 PIN IP HEADER/IC SCKT

RIBBON CABLE 5"

The following table is the connections required between the TMS9928/29 and the V9938

	40 PIN HEADER/IC SCKT PIN No		DIN 41612 A&C CONNECTOR PIN No
	GRND	12	32a&c
MODE	A15	13	11c
	CSW	14	19c
	CSR	15	19a
	INT	16	23a
LSB	D0	17	30c
	D1	18	30a
	D2	19	29c
	D3	20	29a
	D4	21	28c
	D5	22	28a
	D6	23	27c
	D7	24	27a
RESET		34	21c
+SUPPLY		33	1a&c

(1)Make up a cable with connections as above using a 40 pin header and DIN 41612 connector

(2)Connect pin 4 and 6 on 14 pin header

(3)Connect pin 8 and 9 on 14 pin header

(4)Replace IC1 on the V9938 board with the header that has just been made

(5)Replace TMS9928/29 with 40 pin header and V9938 board

NOTE The V9938 requires A14 this is taken from the top track leading to pin 2 of the TMS9909 sckt and connected to pin 11a on the DIN41612 connector

A set of programs to initialise and use all the extra built in facilaties of the new VDP(V9938) is available on disc.Please state if the VDP is used on the E BUS or as a replacement for the TMS9928/29 Disc is priced at TEN POUNDS all inclusive from

C J YOUNG ,107 RINGWOOD ,GREAT HOLLAVNDS ,BRACKNALL ,BERKS

```
1 REM Memory Mapper Setup utility  A.R.C.Badcock      QBASIC
{
 This utility allows the Memory Mapper
 registers to be loaded manually by the
 user. Each mapper register holds a
 value to which 4kbyte memory block in
 the bottom 64k of the memory map will
 be vectored. This only occurs after a
 CKON opcode switches the mapper on and
 ceases after a CKOF opcode switches the
 mapper off.
 This function allows any 4k page to be
 switched into the lower address area.
 This utility has certain safe-guards for
 the inexperienced. The user is prevented
 from setting the register controlling
 00000-00FFF, and the registers from
 0C000-0EFFF as there is a danger of fouling
 the operating system. Also the lowest page
 that can be vectored is 0F000-0FFFF to avoid
 a tangle up in lower memory.
 However, the experienced programmer can
 always get around this in a program if
 needs be. This utility provides a direct
 method of setting the mapper up when
 experimenting.
}
5 PRINT:PRINT"Memory Mapper Setup Utility v1.0"
  PRINT"================================="
  PRINT:PRINT"By A.R.C.Badcock (c) 1988 ":PRINT

   MAPLOC% = 0F100      {Mapper base address}

10 errorflag% = 0     {clear error flag}
   GOSUB 100         {Input register value}

15 IF errorflag% = 1 THEN 30
20 GOSUB 200         {Input register data }

30 INPUT"More ? ";reply$
   IF reply$ = "Y" OR reply$ ="y" THEN 10
   PRINT:PRINT"Mapper status":PRINT
   PRINT"Register   :   Mapper value":PRINT

40 FOR OFFSET% = 0 TO 30 STEP 2
    MAPREG% = MAPLOC% + OFFSET%
    REGVAL% = PEEK(MAPREG%)
    PRINT MAPREG%#;"      =      ";REGVAL%#
   NEXT OFFSET%

50 PRINT
   PRINT" - Done - "
   PRINT
   STOP
100 REM - Select Register
    INPUT"Which mapper register to set (1 - 11 allowed) ?
";MAPREG%
    IF MAPREG% < 1 OR MAPREG% >11 THEN GOSUB 300 {Check valid
register}
    IF errorflag% = 1 THEN RETURN
```

```
    OFFSET% = MAPREG% * 2          {allow for word increments}
    REGVAL% = PEEK(MAPLOC%+OFFSET%)
    PRINT "Current value of register is ";REGVAL%#
    RETURN

200 REM Set register value

    INPUT"Set mapper register value (15 - 255 allowed)
";MAPVALUE%
    IF MAPVALUE% < 15
       MAPVALUE% = MAPREG%   {set default value}
       PRINT "ERROR - value not allowed, default loaded "
    ENDIF
       PRINT
       POKE(MAPLOC%+OFFSET%),MAPVALUE%     {load value in
register}
    RETURN

300 REM Error routine

    PRINT"This register not allowed - affects operating system
area."
    errorflag% = 1
    RETURN
999 END {End of program}
```

```
. MAPPER SWITCH for MDEX by A.R.C.Badcock     MDEX ASSEMBLER
.
. This utility switches the mapper on.
.
      IDT  "MAP-ON"
      COPY "1/JSYS$"
.
       RORG
.
START  CKON    .issue mapper on command
.
       JSYS FINISH
FINISH BYTE EXIT$,0
       DATA 00000
.
 END    START
```

```
. MAPPER SWITCH by A.R.C.Badcock            MDEX ASSEMBLER
.
. This utility switches the mapper off.
.
      IDT  "MAP-OFF"
      COPY "1/JSYS$"
.
       RORG
.
START  CKOF    .issue mapper off command
.
       JSYS FINISH
FINISH BYTE EXIT$,0
       DATA 00000
.
 END    START
```

```
{ COLOUR UTILITY FOR CORTEX SCREEN
  Copyright A.R.C.BADCOCK @1988    }

{
   This utility allows the Cortex screen
   colours - foreground & background -
   to be set by the user.
}

5 PRINT
  PRINT"CORTEX Screen colour setup utility v1.0"
  PRINT"======================================="
  PRINT
  PRINT"By A.R.C.Badcock    - (c) 1988"
  PRINT

 10 PRINT:PRINT"SET SCREEN COLOURS"
    PRINT
    INPUT"FOREGROUND = ";FG%
    INPUT"BACKGROUND = ";BG%
    COLOURCODE% = FG%*16+BG%    {Calculate code value to write to
VDP}
    REGIDENT% = 087             {Code to actuate colour load in
VDP}
    POKE(0F121),COLOURCODE%     {Write colours to VDP command
register}
    POKE(0F121),REGIDENT%       {Write actuate code to VDP
command register}
    PRINT:PRINT"OK":PRINT:PRINT
   STOP
   END {of program}
```

INSIDE CORTEX BASIC

This article is intended to show how BASIC programmes are stored in the CORTEX memory with descriptions of the various tables that are employed. The article will not cover programming in BASIC but will try to describe HOW the BASIC works. Some users who obviously have an in depth knowledge of the subject, judging from past newsletters will have to bear with me. However I hope the articles will be of use to the user group. I have been investigating the BASIC in order to find a method of getting rid of so called Phantom Variables. The aim of these articles is to combine my findings with ideas that have been presented in previous programmes and articles in the newsletters. I will try not to waste too much space by repeating articles already published in the newsletters but will refer to some from time to time.

I will include some programmes to demonstrate the items covered and to enable people to have a look for themselves. The first programmes will be short and in BASIC to allow people to get the feel of what to look for before using the monitor and machine code routines.

The subjects I intend to cover are:

1. Introduction - Direct & Programme Modes.
2. Basic Pointers and how to find the tables.
3. Keywords Functions and Tokens.
4. Varables, Data, Numbers & Strings.
5. Summing Up

1. Introduction.

So! We have all written programmes in BASIC, but how much thought has been given as to what is happening. You type in your programme, edit the lines until it works and then run it. Type LIST and you can see your programme. Simple isn't it?

Actually the text you type and list is not what actually runs or what is stored in the computer's memory. In fact there is always a programme running even when the computer is apparently idle. Even when users machine codes are not running, the Operating System is keeping things going, displaying screens to your TV/Monitor scanning the keyboard waiting for you to type etc. When the RETURN key is pressed the operating system branches to the EDITOR to process the data you have typed in.

If the data starts with a line number and the syntax is correct then the line is encoded and the BASIC tables are updated accordingly. No execution of commands is carried out (except for a few exceptions I shall mention later). If ther is NO line number and the syntax is correct then the editor branches to the routines to perform the commands that have been typed. I refer to this mode of operation as Direct Mode. The branches to the routines and encoding of BASIC use the tables shown in Newsletter 6 pages 11&12.

When RUN is typed the editor passes control to another routine the INTERPETER. This progamme scans through the basic tables and uses the tokens to execute the BASIC commands! Although the Interpreter and Editor both access the same routines (eg 'PRINT' does the same thing whether typed directly or within a programme), the way in which these routines are invoked are different. More will be said of this in section 3(Keywords,functions and tokens.). The action of the interpreter upon the tables is what I refer to as Programme Mode.

When LIST is typed, control is passed to a machine code routine which decodes the data stored in the BASIC tables back into lines of text which are then displayed. The EDIT key uses the same routine as LIST to decode a BASIC line and then passes control to the

Editor. The Editor and LIST both use a buffer located at EB04H to hold the text data. The monitor does not use this buffer. If you type MON and then examine memory starting at EB04 you will see the Ascii Codes for M O N terminated by a zero byte, This buffer is immediately above the variable storage memory.

As always there are exceptions to the rule. Remember what I said about the editor detecting line numbers, well try this!

Type 'NEW' to clear the current programme
Enter the line

10 SIZE

Interesting ?
Now type LIST. Nothing there! More about this in section 3!

2. Basic Pointers and Tables.

A BASIC programme in the Cortex is held in four tables. The first three of these are stored to tape or disk when the SAVE command is used. Because BASIC programmes can vary widely in size and in the numbers of variables used a system of pointers is used which hold the starting locations of each table. This system of using pointers is prefered to a fixed area of memory for each table for the following reasons.

1. Memory use is more flexible, allowing space not taken up by programme lines to be used by variable tables and vice-versa.
2. With a fixed adressing method a short programme would take up as much memory as a long programme. The pointer system leaves memory available for use by machine code routines. Also only the area used by a BASIC programme needs to be stored and can be stored as a consecutive block of memory, leading to short LOADING times for smaller programmes.
3. A Disadvantage is that the table contents have to be moved when new lines are typed in or old ones deleted.

However, the advantages of 1 & 2 far outway the disadvantage of 3!

The four BASIC tables are as follows:-
1. Encoded form of the BASIC, each entry being one basic line.
2. Line number table, containg line numbers and offsets to table 1.
3. Variable Name Table, containing encoded forms of the variable names used in the programme.
4. Variable address table, containing the addresses of assigned variables.

The pointers to the tables are at the following locations, shown in hex.format.
ED04 - Start of the Basic Code table.
EFBA - Start of the Line No. table.
EFBC - Start of the Variable Name table.
EFBE - Start of the Variable Address table.
EFC0 - Start of available memory above BASIC.

The NEW command if used with a parameter alters the address held at ED04 (and also at ED06). A value of 14H is added to the parameter so NEW 6000H puts the start of the Code table at 6014H.

From now on I will use the @ symbol to mean the word held at an address.
eg the address @EFBA is the start address of the Line Table.
This nomenclature is similar to that used by assembly language.

THE BASIC CODE TABLE - @ED04 to @EFBA -2
Each byte in this table is a token for either a keyword, variable, function, delimiter,

operator or part of a number or string. Basic lines are stored in sequence WITHOUT being preceded by a line number and each line is terminated by a zero byte. Tokens are effectively pointers to start addresses of commands, functions and variable tables, meaning that BASIC lines can be interpreted very quickly. Each entry is of variable length depending on the length of the original basic line.

THE LINE NUMBER TABLE - @EFBA to @EFBC -2
This Table contains line numbers in REVERSE ORDER at every second word.The word following a line number is the offset to the start of that line in the CODE TABLE. ie code for a line will start at @ED04 + OFFSET. The table is used to locate the code when GOTO and GOSUB statements are encountered. The table is also used when decoding lines for listing or editing.

THE VARIABLE NAME TABLE - @EFBC to @EFBE -2
Contains encoded names of the variables. It is used in decoding lines for display and editing. It is also used to examine or alter the value of a variable when a programme is halted. The first three words of this table are empty and the fourth contains 11D2 which is the encoded name for RND which is treated like a variable that can be read from but not written to!

THE VARIABLE ADDRESS TABLE - @EFBE to @EFC0 -2
Once a variable has been assigned either from the keyboard or from within a programme it is given an address which is put into this table. The position in this table is equivalent to the position of its name in the previous table. The first four words of this table are zero (RND does not use an address held in this table). Unassigned variables also have a zero entry. Dimensioned variables only have the address for index (0) stored in the table, higher index addresses being calculated by the operating system.

The variable addresses are themselves pointers to the data stored in a varable. Variables are created as they are defined and progressively move DOWNWARDS through memory. The variables occupy memory upto but not including location EB04. The lower limit set for variable storage is the location @EFC0. The start address of the last variable assigned is stored in the pointer EFC2.

In the next article I shall look at how commands, variables and data are stored in the Code Table and how the variable names are encoded. I will show how tables are updated and accessed in both Direct and Programme modes.

To round off I have presented some Basic programmes for users to experiment with.

Pointer	@Pointer	DESCRIPTION
ED04	7114	Start of Basic
EFBA	7418	Start of Line Table
EFBC	746A	Start of Var. Names
EFBE	7486	Start of Var. Addresses
EFC0	74A2	Memory Above Basic.
EFC2	EADC	Last Variable Address.

SEE FOR YOURSELF

The following listings will allow you to look at the contents of the BASIC CODE TABLE. Essentially they are BASIC programmes which look at themselves! Please use the same line numbering as I have, the reason will become obvious soon.

The first programme 'LIST1' shows the encoded basic in the form of single bytes. This sort of programme is useful to look for tokens of keywords and variables.

The second programme 'LIST2' shows the CODE TABLE as Ascii characters. This form is useful for looking at strings and REM statements.

The above two codes show the lines listed. It would be more useful to show only code upto line 9000. This will enable users to add lines below 9000 and show the encoded Basic without having to show the routine which displays the data. To understand how to do this it would be useful to look at the Line Number Table. The third programme LINES shows how Line Numbers are stored in the table. Note that the lines are stored in reverse order.

With the knowledge of how to access line numbers it is possible to modify LIST1 or LIST2. The final programme LIST3 will only display encoded Basic upto line 9000. The programme looks for the value 9000 in the line number table so now it is obvious why I wished the same line numbering to be used. It is also possible to modify LIST2 in the same way.

I have not included outputs from the programmes as it is more useful to let poeple run the programmes themselves. The programmes are very short and do not involve much typing!

LIST1

```
9000  REM CODE TO SHOW BASIC ENCODING
9020  P1=MWD[0ED04H]: P2=MWD[0EFBAH]-2  ! START & FINISH OF BASIC STORAGE
9030  ? "<C>START OF BASIC ";£,P1: ? "  END OF BASIC ";£,P2
9500  REM *** OUTPUT RESULTS ***
9510  FOR I=P1 TO P2: Z=MEM[I]
9540   ? £;Z;"  ";
9550  NEXT I
```

LIST2

```
9000  REM CODE TO SHOW BASIC ENCODING
9020  P1=MWD[0ED04H]: P2=MWD[0EFBAH]-2  ! START & FINISH OF BASIC STORAGE
9030  ? "<C>START OF BASIC ";£,P1: ? "  END OF BASIC ";£,P2
9500  REM *** OUTPUT RESULTS ***
9510  FOR I=P1 TO P2: Z=MEM[I]
9520   $C=" ": IF Z LAND 128: $C="_"
9530   X=Z LAND 127: IF X<32: X=46  !Non Printable Characters Show up as a Dot
9535   $C=$C+%X
9540   ? $C;
9550  NEXT I
```

LINES

```
100  ? "<C>LINE NUMBER TABLE DISPLAY": ? "--------------------------": ? : ?
900  ? "LINE NUMBER    OFFSET (HEX)    (DECIMAL)"
910  ? "-----------    ------------    ---------": ?
1000  ST=MWD[0EFBAH]: FI=MWD[0EFBCH]-2
1010  FOR I=ST TO FI STEP 4
1020   ? MWD[I],,£,MWD[I+2],,MWD[I+2]
1030  NEXT I
3000  REM  THIS LINE NUMBER IS THE FIRST DISPLAYED
```

LIST3

```
9000  REM CODE TO SHOW BASIC ENCODING
9020  P1=MWD[0ED04H]: P2=MWD[0EFBAH]-2  ! START & FINISH OF BASIC STORAGE
9030  ? "<C>START OF BASIC ";£,P1: ? "  END OF BASIC ";£,P2
9040  LNO=9000: P3=MWD[0EFBCH]-2
9050  FOR I=P3 TO P2+2 STEP -4
9060   IF MWD[I]=LNO: GOTO 9100
9070  NEXT I
9080  ? "NO LINE 9000 IN THIS PROGRAMME!"
9090  END
9100  P2=P1+MWD[I+2]-2  ! RESETS P2 TO END OF LINE BEFORE 9000
9500  REM *** OUTPUT RESULTS ***
9510  FOR I=P1 TO P2: Z=MEM[I]
9540   ? £;Z;"  ";
9550  NEXT I
```

MSAVE & MLOAD COMMANDS

Have you ever wanted to merge programmes or get rid of 'phantom variables' (See Newsletter 10) without having to Source List programmes to tape? I have written two commands MSAVE and MLOAD which save source listings in high memory.

In case anyone does not know what is meant by a source listing, it is a way of storing a programme as character data in the form of ASCII codes. Normally a programme is stored to disk or tape as a memory dump of the highly encoded BASIC and line number and variable tables. The advantage of a source saving is that on retreival any BASIC programme in memory is NOT wiped out hence allowing merging of routines. Also no variable attributes are stored. Retreiving a source listing is like 'retyping' in all the lines only much faster and with less effort. The disadvantage with source listings is that much more memory or tape/disk storage is taken up.

Programme Description

The machine code programme UTILS (short for utilities) consists of routines for three commands FIND,MSAVE,MLOAD. The MSAVE command uses the same type of routines as the FIND command in newsletter 3 which I have modified slightly to save on memory when combined with MSAVE. The enclosed listing with added comments shows how the routines work. The choice of buffer locations is entirely up to the user, I tend to use 70A0H as a general buffer for CAT,FIND,Return vector storage etc. The buffer for MSAVE should be large in order to store sensible sized chunks of BASIC source lines, which is why I chose F200H to FE00H (the last BASIC line stored will exceed this value). The only reason I have not extended the buffer to FFxx is that I have a problem with random numbers occuring in locations FEFA to FEFE. Has anyone any ideas about this? Users who have an EBUS extended memory which I do not, will obviously be able to use quite large storage buffers!

The commands are patched into the Command tables at 3A20,3A22 and 3A24H. The commands cannot be used from within a BASIC programme but only in direct mode. Only the first 3 characters of the commands need to be typed eg MSA will dump the current BASIC programme to memory. This section of the programm intrudes into the beginning of CDOS. This does not provide any problems as this part of CDOS appears only to be used once at BOOT time!

The first two words in the storage buffer are used as pointers which MSAVE & MLOAD use when transfering data between the store and the BASIC environment.

There are branches to two operating sytem routines. The routine at 3C80H is the routine that converts a programme line to ASCII format for display, used by LIST and the line editor. The routine at 3404H converts lines in ASCII format into a programme line, whilst performing syntax checks. This routine is used by ENTER and the line editor.

It will be noticed that there are two ways to display basic lines. One is using the MID CODE 0002, the other is by using MSG @>EB04. The MID CODE looks for a line number at the start of the buffer at EB04 and if found the appropriate BASIC line printed. The MSG simply prints the whole contents of the buffer at EB04. The latter is used in the MLOAD programme as the line in the BASIC table may not exist or be different to that being recalled from high memory. For example if MLOADing a programme after a NEW command has been used, then the MID would only display line numbers whereas the MSG will display the full line being retrieved.

USING THE COMMANDS

The FIND is used exactly the same as that in newsletter III.

eg. FIND textstring

The space is important and the word FIND must be at the beginning of a line.

MSAVE and MLOAD use no parameters and can be typed anywhere on a line. The two uses are MERGING codes or SOURCE LOADING to get rid of phantom variables.

MERGING. MSAVE the routine to be merged. If this is part of another code then delete unwanted lines and renumber as appropriate. An ** END OF BUFFER WARNING ** is given if the code is larger than the buffer. In this case use the Monitor to dump the buffer to disk as a non executable m/c programme. Delete the programme lines that have been saved and repeat the process. The last MSAVE does not need to be transfered to disk! Load the main programme as normal. Use MLOAD to merge the new code. MLOAD will replace existing line numbers - be careful with your numbering. If the buffer is stored on disk LOAD 'filename' - You do not need to use the monitor and then MLOAD. The code required will now be merged.

DELETING PHANTOM VARIABLES. Firstly MSAVE the basic programme. Most programmes will have to be done in several stages saving the buffer to disk using the monitor. When all the code has been saved type NEW. MLOAD the code back into BASIC, retreiving files from disk if necessary. On completion you will have a copy of the original code but with vacant variable space.

MSAVE will display the basic lines as they are stored. If the end of buffer is reached the last line stored will be that above the error message. MLOAD displays the basic lines as they are retreived.

To save the buffered ascii code onto disk use the monitor D command starting at F200H. The data finishes at the pointer stored in F202H, however if in doubt storing memory upto FE50H should contain all the data required.

UTILS

Name	ADDR	CODE	INSTR	INSTRUCTIONS	COMMENTS
FIND	6010	0200	LI	R0,>EB09	Editor Buffer +5
	6014	0201	LI	R1,>70A0	Storage buffer
	6018	DC70	MOVB	*R0+,*R1+	Store byte
	601A	16FE	JNE	>6018	Next byte
	601C	0002	DATA	>0002	MID OPCODE - empty line
	601E	06A0	BL	@>60AE	Start Routine
	6022	06A0	BL	@>60C8	Decode Routine
	6026	0202	LI	R2,>EB04	Buffer - decoded line
	602A	0201	LI	R1,>70A0	Storage buffer
	602E	D494	MOVB	*R2,*R2	Check end of line
	6030	13F8	JEQ	>6022	Jump if at end
	6032	9C91	CB	*R1,*R2+	match?
	6034	16FC	JNE	>602E	No - try again
	6036	0581	INC	R1	
	6038	D451	MOVB	*R1,*R1	Test of Search string
	603A	1303	JEQ	>6042	End found
	603C	9C91	CB	*R1,*R2+	match?
	603E	16F5	JNE	>602A	Try again for match
	6040	10FA	JMP	>6036	Next char in search string
	6042	0002	DATA	>0002	Print empty line
	6044	000A	DATA	>000A	Print BASIC line
	6046	10ED	JMP	>6022	Next BASIC line
MSAVE	6048	0203	LI	R3,>F204	Start of Storage Space
	604C	C803	MOV	R3,@>F200	Pointer to Start
	6050	C803	MOV	R3,@>F202	Pointer to End
	6054	06A0	BL	@>60AE	Start Routine
	6058	06A0	BL	@>60C8	Decode Routine
	605C	60E0	MOV	@>F202,R3	End Pointer in R3
	6060	0201	LI	R1,>EB04	Buffer - decoded line
	6064	DCF1	MOVB	*R1+,*R3+	Put character into store
	6066	16FE	JNE	>6064	Next character
	6068	0002	DATA	>0002	MID - print empty line
	606A	000A	DATA	>000A	MID - print BASIC line
	606C	C803	MOV	R3,@>F202	Update pointer
	6070	0283	CI	R3,>FE00	Check Storage Space
	6074	11F1	JLT	>6058	OK
	6076	0002	DATA	>0002	Print empty line
	6078	0FA0	MSG	@>60EA	BUFFER FULL MESSAGE
EXIT	607C	0002	DATA	>0002	Print empty line
	607E	0460	B	@>021C	Back to BASIC
MLOAD	6082	0201	LI	R1,>EB04	Editor buffer
	6086	C0E0	MOV	@>F200,R3	Start pointer in R3
	608A	DC73	MOVB	*R3+,*R1+	Move char to KB buffer
	608C	16FE	JNE	>608A	Next character
	608E	C803	MOV	R3,@>F200	Update pointer
	6092	06A0	BL	@>3404	Enter BASIC line
	6096	0002	DATA	>0002	Print empty line
	6098	0FA0	MSG	@>EB04	Display BASIC line
	609C	8820	C	@>F200,@>F202	End of Stored Data?
	60A2	11EF	JLT	>6082	No - Get Next Line
	60A4	0201	LI	R1,>F204	Reset Start pointer

```
         60A8  C801  MOV   R1,@>F200          "     "     "
         60AC  10E7  JMP   >607C             Exit

START    60AE  04C1  CLR   R1                R1 Holds 1st Line No
         60B0  0706  SETO  R6
         60B2  C220  MOV   @>EFBC,R8         Start of Variable Table
         60B6  0648  DECT  R8                End of Line No Table
         60B8  8808  C     R8,@>EFBA         Start of Line No Table
         60BC  12DF  JLE   >607C             Exit
         60BE  0228  AI    R8,>FFFC          R8=R8-4
         60C2  8601  C     R1,*R8            Compare table to start line
         60C4  15F9  JGT   >60B8             Repeat until Line No found
         60C6  045B  RT                      Return

DECODE   60C8  C808  MOV   R11,@>60E8        Store Return Vector
         60CC  8808  C     R8,@>EFBA         Check end of table
         60D0  1AD5  JL    >607C             If YES then Exit
         60D2  C078  MOV   *R8+,R1           Get Offset in BASIC Table
         60D4  8181  C     R1,R6             Is it Valid
         60D6  1BD2  JH    >607C             No then Exit
         60D8  06A0  BL    @>3C80            Decode BASIC to ASCII
         60DC  0228  AI    R8,>FFFA          R8=R8-6
         60E0  C2E0  MOV   @60E8,R11         Get Return Vector
         60E4  045B  RT                      Return
         60E6  1000  NOP

MESSAGE  60E8  605C  DATA  >605C
         60EA  2A2A  DATA  >2A2A             End of Buffer
         60EC  2045  DATA  >2045             Message
         60EE  6E64  DATA  >6E64
         60F0  206F  DATA  >206F
         60F2  6620  DATA  >6620
         60F4  4275  DATA  >4275
         60F6  6666  DATA  >6666
         60F8  6572  DATA  >6572
         60FA  202A  DATA  >202A
         60FC  2A20  DATA  >2A20
         60FE  0000  DATA  >0000

NAMES    6100  724C  DATA  >724C             Name 'FIND'
         6102  0CDA  DATA  >0CDA             Name 'MSAVE'
         6104  7B1A  DATA  >7B1A             Name 'MLOAD'
SADDRESS 6106  6010  DATA  >6010             St. address FIND
         6108  6048  DATA  >6048             St. address MLOAD
         610A  6082  DATA  >6082             St. address MSAVE

SETUP    610C  0201  LI    R1,>3A20          Entry adress for Autorun
         6110  0202  LI    R2,>3ACC          IS 610CH
         6114  0203  LI    R3,>6100
         6118  0204  LI    R4,>6106
         611C  CC73  MOV   *R3+,*R1+         Patch Names into Table
         611E  CCB4  MOV   *R4+,*R2+         Patch addr. into Table
         6120  0281  CI    R1,>3A28          All done ?
         6124  1AFB  JL    >611C             No - Next
EXIT2    6126  0460  B     @>021C            Back to BASIC
         612A  1000  NOP

SAVING   ----> ----> -----> D IDT="UTILS"    6010,612B,610C Autorun=Y
```

Cortex User Group Sale Items

Hardware

R.G.B. interface P.C.B	£8.00
Centronics P.C.B	£7.00
E.Bus 512K DRAM P.C.B plated through hole	£40.00
Externel Video interface P.C.B	£15.00
Disk controller WD2797 + P.C.B Cortex I	£55.00
Disk controller WD2797 + P.C.B Cortex II	£60.00
E.Bus interface complete Kit	£30.00
E.Bus 8 X 8K EPROM socket card built but no EPROMS	£30.00
E.Bus 4K RAM 8K EPROM socket 16 I/O lines ex equipment	£15.00
TMS9902 UART IC's	£2.00
74LS612 Mapper IC's	£25.00
74LS611 or 74LS613 Mapper IC's (req pull up R's)	£10.00

Other IC's in stock please write in for quote

Software all disk formats please specify when ordering

CDOS basic disk system 1.20 for TMS9909	£45.00
CDOS basic disk system 2.00 for WD2797	£45.00
Wortex word processor + spelling check by J.Makenzie	£15.00
Drawtech graphics drawing package by Tim Gray	£20.00
Menue generator by A.R.C.Badcock	£10.00
Two pass assembler by R.M.Lee	£14.00
Two pass asembler by C.J.Young	£15.00
Cdos utilites disk - copy charge only	£2.00

Cdos programes and games all £2.50 each :-

ARCHIE	ASTEROID	BREAKOUT	BURGLAR	CATERPIL	C-PEDE
CANYON	COTELLO	FIREBIRD	FROGGER	GDESIGN	GOLF
HUNCHBACK	INVADERS	MAZE	MAZE-3D	MBASE	MICROPED
MIS-COM	MUNCHER	NIBBLERS	N-ATTACK	OLYMPICS	P.BOAT
PENGO	PONTOON	RESCUE	S-ATTACK	SPACE-BU	THE-ZOO
TRAG	VADERS	WALL	X/O		

MDEX Software all formats please specify

MDEX CORE with debug monitor text editor and basic	£10.00
MDEX ASM & LINK assembler and linker	£10.00
MDEX SYSGEN system generation kit	£10.00
MDEX WORD word processor	£10.00
MDEX P.D.S. all the above in one package	£30.00
MDEX S.P.L. system programming language	£10.00
MDEX META compiler generator	£10.00
MDEX QBASIC basic compiler	£15.00
MDEX PASCAL sequential pascal	£10.00
MDEX WINDOW full screen editor	£15.00
MDEX SPELL spelling checker	£10.00
MDEX utilities copy charge only	£2.00