THE SOURCE

by

KEITH HOOK

Produced and printed by ORION SOFTWARE 1987

THE SOURCE

by

KEITH HOOK

Produced and printed by ORION SOFTWARE 1987

This book is dedicated To MARLYN & GEOFF BOYD for their undying belief in a computer that is worthy of consideration by any computer buff. Without their dedication the MTX would have died. Instead, the phoenix arose from the ashes in the guise of the MCL SERIES TWO.

Further dedications are directed at the many pioneer users who suffered greatly in the 'early days' but have remained loyal to Black Beauty. My personal thanks go to these people who over the years have given me the faith to continue.

Keith Hook
Higher Reedley
1987

ACKNOWLEDGEMENTS

The author wishes to thank MCL and the many people who have submitted material to him over the past years and for their assistance with ideas and material.

Contents

CHAPTER TWO

The Memotech's screen handling can initially seem difficult to get to grips with - superficially it doesn't seem to have direct memory mapping of the video display, and the manual doesn't explain how you can write to and read from the screen using POKE & PEEK. The confusion is basically caused by the way the display operations are managed. The Memotech uses the Texas TMS9129A Video Display Processor [VDP] to handle all data relating to the display, while other micros tend to use the cpu for this operation.

So, although the presence of the VDP is confusing, it is actually an advantage, giving you 16k of video ram on top of the normal ram, and giving you added flexibility once you get to grips with it.

Normally the screen is memory mapped in ram. For instance, the Colour Genie computer is memory mapped at 4400H to 47FFH (17408 - 18431) for the low-resolution screen. Fast writes or reads from/to the screen can be accomplished by Peek (address) or Poke,address value.

At first sight it seems that writing to the screen using Pokes or reading from the screen using Peeks is not possible on the Mtx - the instruction manual certainly doesn't mention the subject. However, memory mapping of the screen via Vram is directly comparable with the system described above for the Colour Genie, except that it is managed by the Vdp and not the Z80 Cpu.

Mtx Basic sets the start of the Text Screen see [diagram 4] at 1C00H (7168) in Vram. This address corresponds to the first position on the screen, top, left-hand corner.

Writing data to Vram involves sending the destination address to the Vdp via Port 2. Once the address has been set up data can be transferred to Vram through Port 1. But bear in mind the following :

The Vdp contains an auto incrementing logic, which means that once the address has been set up, sequential writes to the screen need only involve sending data. For example:

Write three blank spaces one after the other.

```
OUT (02),ADDRESS
OUT (01),32
OUT (01),32
OUT (01),32
[32 = ASCII CODE FOR A BLANK SPACE   CHR$(32)]
```

All addresses must be sent to the Vdp Least Significant Byte firs, followed

CHAPTER THREE

USING THE ASSEMBLER

The immediate advantage of the Mtx assembler over most other machines is that it is very easy to use and has a simple instruction set that can be learned with ease. It does not have to be loaded into memory and it is called from Basic as an **inline** assembler. All code, entered at the keyboard, is stored in memory as machine executable **object code**. The readable **source** file is generated by using the LIST command, and at this time the Mtx disassembles the object code by inserting labels, text etc., which are stored in tables above the object code - this is one reason why a LISTing becomes slower as the program grows in size.

At this point it is important to realise the following:

A] SINCE CODE IS STORED IN A BASIC LINE THE ACTUAL LOCATION OF THE CODE WILL CHANGE IF THE BASIC PART OF THE PROGRAM (BELOW THE ASSEMBLER LINE) IS MODIFIED, OR LINES ARE ADDED.

B] A PROGRAM LISTING THAT USES TWO SEPARATE CODE LINES ...ASSEM 20 : ASSEM 200 WILL NOT MATCH UP WITH THE ORIGINAL ABSOLUTE ADDRESSES [THOSE SHOWN IN THE LISTING] IF COMMENT LINES ARE OMITTED. THIS IS NOT A PROBLEM AS LONG AS LABELS HAVE BEEN USED. THE CODE IS THEN RE-LOCATABLE AND THE REMARKS STATEMENTS WILL NOT AFFECT THE ACTUAL OPERATION OF THE CODE.

```
10 CODE

8007         LD HL,BUFFER
800A         LD B,08
800C LOOP:   LD A,(HL)
800D         LD (HL),A
800E         INC HL
800F         DJNZ LOOP
8011 BUFFER: DB 30,40,50,60,70,80
8017         RET
8018         RET

Symbols:
BUFFER  8011     LOOP     800C
```

```
1 REM THIS LINE HAS BEEN ADDED LATER
10 CODE

802C         LD HL,BUFFER       ;<= Notice how address has changed
802F         LD B,08
8031 LOOP:   LD A,(HL)
8032         LD (HL),A
8033         INC HL
8034         DJNZ LOOP
8036 BUFFER: DB 30,40,50,60,70,80
803C         RET
803D         RET

Symbols:
BUFFER  8036     LOOP     8031
```

CHAPTER FOUR

USING THE FRONT PANEL

Once an assembly program has been written, some means of testing the code is desirable. One wrong byte can send a machine code program on a journey to nowhere. It is, therefore, an advantage if some means of **single-stepping** through a program is available - in this way we can examine registers and detect when the code does not do what is expected. At this point, the Front Panel comes into its own.

The Front Panel is a debugging aid which will allow the assembly programmer to look at or single step through an assembled program. Most assembly programs move data between registers, store data in specific memory locations, or carry out some form of test on the **flag** register. Program failure often occurs due to one of these operations performing in an unexpected manner.

To use the Front Panel you must first become familiar with the type of operations that can be performed from the keyboard.

KEY	OPERATION
RETURN	MOVES MEMORY CURSOR FORWARD
DOWN ARROW	MOVES MEMORY CURSOR DOWN
UP ARROW	MOVES MEMORY CURSOR UP
-[MINUS]	MOVES MEMORY CURSOR BACKWARD
.[FULL-STOP]	MOVES REGISTER CURSOR

Typing L will list from the current memory location and typing L #4000 will list from #4000.

Typing D will display from the current **Memory Block Cursor** while typing D #4000 will display from #4000.

There are two cursors and both use the same symbol ">". One is the **Memory** cursor and one the **Register** cursor.

Typing I toggles the memory display between hexidecimal notation and Ascii.

How to use the Front Panel is best explained by example, so before we continue type in the following listing - it doesn't do very much but it will serve as a demonstration program. First enter the assembler by typing ASSEM 10 <RET>. In answer to the **Assemble>** prompt press <RET>. Your display should now look exactly like this :-

```
8007  RET  [Mtx 500]   or  4007  RET  [Mtx 512+]
```

CHAPTER FIVE

THE VIDEO DISPLAY

All display operations are managed by the Texas TMS9129 Video Display Processor (VDP). The chip is not a secret weapon developed by Texas Instruments in order to fill up the psychiatric wards with budding programmers. It is a sophisticated piece of electronic wizardry which allows complex screen displays to be utilised. However, as with all powerful electronics, the chip requires what appears, at first sight, a complicated set of instructions. The VDP manages an area of ram, which is seperate and extra to normal ram called Video Ram (VRAM).

The VDP communicates with VRAM via Ports 1 and 2.

Port 2 is used for address transfers
Port 1 is used for date transfers

All addresses throughout VRAM are 14-bit. Address transfers require a two-byte transfer with 2 bits unused.

The VDP has five available display modes.

a) TEXT
b) MULTICOLOUR MODE
c) GRAPHIC MODE 1
d) GRAPHIC MODE 2
e) MODIFIED GRAPHIC MODE 1

Only TEXT and GRAPHIC MODE 2 are available directly from MTX BASIC but the other modes can be accessed by creating your own VDP setups.

TEXT MODE provides a screen which is 40 columns wide by 24 rows deep. Two colours are available in this mode.

GRAPHIC MODE 2 offers a display of 32 columns by 24 rows deep. Sixteen colours are available and plotted displays are also allowed.

CHAPTER SIX

THE DISPLAY MODES

TEXT MODE

This is the same as the normal MTX Basic text mode.

The VDP is initialised to text mode when the **mode bits** M1 = 1 : M2 = 0 : M3 = 0

(See previous chapter - VDP REGISTERS)

Text mode provides the following features:-

SCREEN

24 rows of 40 columns (960 character positions).

Up to 256 unique characters can be defined at any one time. The pixel size of text characters should be six wide by eight deep. These character patterns can be dynamically changed by transferring patterns from character libraries held in unused portions of Vram or Z80 ram.

Two colours are available: one for text colour and one for the backdrop. The colours can be chosen from a palette of fifteen hues including transparent.

MTX Basic uses the following set-up for text mode:-

FUNCTION	VRAM START ADDRESS	VRAM END ADDRESS
TEXT PATTERN	6144 (#1800)	7167 (#1BFF)
SCREEN	7168 (#1C00)	7191 (#1C17)

Because, in Basic, the text mode has been designed as an integral part of

CHAPTER SEVEN

ZILOG COUNTER TIMER CIRCUIT

The Zilog Counter Timer Circuit (CTC for short) handles all interrupts on the MTX including VDP interrupts.

Its features are:-

4 independently programmable Counter/Timer channels

Standard Z80 daisy-chain interrupt structure provides full vectored, prioritised interrupts.

The CTC can generate MODE 2 interrupts from any of its four independently programmable channels. It can act as a TIMER or COUNTER working with the Z80 clock or an external trigger.

CTC operations are controlled by addressing four MTX ports - one for each channel.

PORT	CHANNEL	FUNCTION
08	0	VDP INTERRUPT
09	1	4 MHZ SYSTEM CLOCK/13
#0A	2	4 MHZ SYSTEM CLOCK/13
#0B	3	CASSETTE EDGE INPUT

TIME CONSTANT

When the counter/timer channel is programmed, the time constant register receives and stores the value which can be in the range 1 - 256 (0=256). The constant is then loaded into the down-counter when the counter channel is initialised and subsequently whenever the count reaches zero.

CHAPTER EIGHT

SPRITES

Sprites are very important in animated game displays and have all manner of uses in graphic displays.

A sprite is a special animation pattern which can be moved, one pixel at a time, in a horizontal, vertical or diagonal direction, and is motivated in a way that is totally independent of the background pattern.

The sprite can be coloured in any one of 15 colours plus transparent. Multicoloured sprites can be designed by overlaying two or more sprites in different colours. Care must be taken to ensure that the overlay pattern is limited to four sprites or the fifth sprite syndrome may have unpredicted results on the display.

Sprites can assume any one of several sizes and magnification. They can be 'bled' into the display from any direction.

With the exception of sprite co-ordinates, pattern shapes, colour, all other maintenance such as background replace is carried out under hardware control.

Two sections of Vram are responsible for management - SPRITE ATTRIBUTE TABLE and SPRITE GENERATOR TABLE.

SPRITE ATTRIBUTE TABLE

The attribute table is responsible for the control of sprites. Thirty two sprites are available on the MTX. Each sprite has four bytes of dedicated information which means this table is 128 bytes long.

The start address of the table is determined by the contents of VDP Register 5 which locates the table on an 128 byte boundary.

Each sprite has an hardware priority index assigned to it by the VDP. Sprite 0 has a higher priority than Sprite 1 and the higher the sprite number the lower its priority. The sprite with the higher priority will appear to pass in front of the sprite with a lesser priority.

Sprite 0 is assigned attribute bytes 0,1,2,3 and has the highest priority. Sprite 31 has the lowest priority of all sprites and is assigned attribute bytes 124,125,126,127.

CHAPTER NINE

SCREEN RESTART ROUTINES RST10

For those of you who are just a little faint-hearted RST10 calls provide a reasonable solution to using machine code for screen displays.

An RST instruction is a unique one byte command that allows a call to any one of eight addresses in low memory. Because it is a one byte instruction speed of execution is assured.

RST10 is used by the MTX rom for at least 90 per cent of rom graphic routines available under Basic. The rom has been designed so that machine code programmers can take advantage of all the routines and once mastered RST10 instructions are very easy to use.

The function of the RST10 call is to send Ascii or **control** codes to the screen or printer depending on which bit is set in system variable IOPR For the sake of simplicity we shall assume that all writing will take place to the screen.

RST10 commands can operate in four different modes:-

SEND A NUMBER OF CHARACTERS TO SCREEN
SEND ONE BYTE TO SCREEN
CLEAR AND SELECT VIRTUAL SCREEN
OUTPUT CONTENTS OF BC REGISTER PAIR TO SCREEN

The format for this type of call is as follows:-

```
RST 10
DB <DATA TO SEND>
```

The fact that data is placed in the path of program flow may seem confusing at first sight but these commands are really easy to use. Just try this...

CHAPTER TEN

KEYBOARD SCANNING

It is important to note that the MTX Keyboard scan is active when the output is low (0) and not high as is the norm with most computer Keyboard scans.

The left hand joystick is mapped onto the cursor keypad with the fire button replacing the **home** key.

FIRE	=	HOME KEY
JOY LEFT	=	CURSOR LEFT
JOY RIGHT	=	CURSOR RIGHT
JOY UP	=	CURSOR UP
JOY DOWN	=	CURSOR DOWN

This form of mapping is very useful as it dispenses with the need to read joystick ports. Reading joystick data is a simple matter of scanning the keyboard. Unfortunately, this does have the disadvantage that multiple movements have to be managed in software. Later, we shall write a routine to detect multiple keypresses from the joystick.

The **sense lines** and **read lines** on the MTX keyboard are tied into **port** 5.

SENSE BYTE	=	OUTPUT TO PORT 5
READ BYTE	=	INPUT FROM PORT 5

OUT (5), n latches data to the **eight** drive lines of the 8 x 10 keyboard matrix.

IN (5) will read the eight **least significant** bits from a ten bit read byte.

IN (6) Reads the **two most significant** bits of the read byte.

The most difficult part of writing a keyboard scan routine is selecting the correct values to output on the sense line and then knowing which lines to look at when testing for a value being returned. To aid understanding, a little explanation may be in order.

The MTX keyboard is divided into eight sections each managing ten keys.

CHAPTER ELEVEN

SOUND

All sound processing on the MTX is managed by the Texas SN76489A sound generator which is capable of producing a wide variety of complex sounds under software control.

In order to perform sound synthesis while allowing the processor to continue its other tasks, the chip can continue to produce sound after the initial parameters have been sent to the control registers. Realistic sound production often involves more than one effect and this is satisfied by three independently controllable tone channels and one pink noise channel.

The principal element of the chip is the array of eight write only registers which are responsible for directing the data to the relevant blocks within the sound chip.

The chip is i/o mapped on port 6. All data transfers are required to be passed through this port.

```
LD      A,#FE
OUT     (06),A
```

This data is then strobed into the sound device by performing a dummy read on port 3.

```
IN      A,(03)
```

FORMULA FOR WRITING TO P.S.G.

a) SEND DATA TO PORT 6
b) STROBE DATA TO CHIP ON PORT 3

```
LD      A,#FE
OUT     (06),A
IN      A,(03)
```

Successive strobes across port 3 require at least 32 computer clock cycles between each read.

APPENDIX A

SYSTEM VARIABLES

SYSTEM VARIABLES

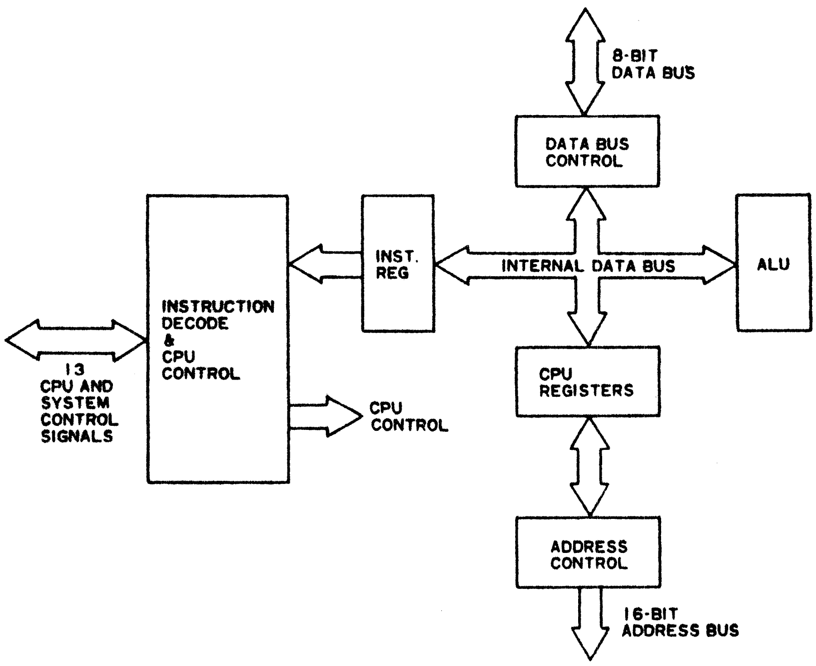
FA52	CTRBADR	DS 40	Control buffer for sound
FA7A	LSTPG	DS 1	This contains the number of 32K RAM pages present -1
FA7B	VARNAM	DS 2	This contains the address of the bottom of the variable table
FA7D	VALBOT	DS 2	This contains £FF. VALBOT plus 1 is the address of the bottom of the variable value name table
FA7F	CALCBOT	DS 2	This contains the address of the bottom of the calculator stack
FA81	CALCST	DS 2	Stack Pointer - this contains the address of the top of the calculator stack +1. ie. the next available free byte
FA83	KBDBUF	DS 2	This contains the address of the Keyboard Buffer.
FA85	USYNT	DS 4	

This contains the syntax bytes which are used to tell the computer what to expect when the BASIC command USER is met. These bytes may be defined by the operator, as listed below. They are examined from the top of the four byte block to the bottom, and the last one must contain a RET instruction.

APPENDIX B

Z.80 OPERATION CODES.

ARCHITECTURE



Z-80 CPU BLOCK DIAGRAM

APPENDIX C

AUTO LOAD FOR LARGE PROGRAMS

Creating Basic auto-run versions of programs that extend into, or beyond page one of ram is not obvious. However, the solution is simple.

Type in the program listed below and save it to tape. The program will only work, and is only necessary, with computers with over 64K of ram.

To use the program carry out the following steps:

a) Load this program into memory and insert the correct program name into line 20 .. this will be the name of the large program you wish to auto-run.

b) Now save this version to a new tape by typing Goto 20.

c) Once the program has been saved stop the tape **but do not rewind it.** Now remove the tape and re-set the computer.

d) Now load in the program you wish to have as an auto-run version. Now save this program onto the tape with the previously saved auto program. **You must save this program as you would a normal auto-run program.**

```
10  Your program
20   ditto
100  Save "This program"

GOTO 100  <RET>
```

The tape should now contain the auto-run program and the program you wish to have auot-run. This program will now auto-run whenever it is loaded into the computer.

It doesn't matter what the program is called. The listing will load it and auto-run it regardless.

APPENDIX D

The following program is a demonstration of how to set up Vram and load an independent character set. The program also shows how to provide simple animation.

```
 5 CODE

 8007         LD SP,(£FA96)      ;     Make sure Stack Pointer loaded from system
 800B         NOP          ;##########################################################
 800C         NOP          ; SET UP VDP REGISTERS AND LOAD ASCII CHARACTERS INTO VRAM
 800D         NOP          ;##########################################################
 800E         LD B,08    ;         Number of VDP Write Registers.
 8010         LD HL,REGSET       ;       Make HL point to VDP Register Data.
 8013 LF1:    RST 8        ;      This command does => LD E(HL):INC HL:LD D,(HL):INC HL.
 8014         CALL REG   ;      Call Subroutine to send Register No & Data.
 8017         DJNZ LF1   ;      DJNZ loops until B Register = 0 in this case 8 times.
 8019 INIT:   LD HL,256  ;      HL points to first Ascii character in top third of
 801C         NOP        ;       Generator table which starts at 0000Hex ** Ascii 32 is first
 801D         NOP        ;      printable character [Space] (8*32 = location 256)
 801E         CALL ASCMVE        ;       Call subroutine to fill all 728 bytes of
 8021         NOP        ;      Character generator [ 91 characters * 8 = 728 bytes]
 8022         LD HL,2304 ;       2nd Third of Generator table
 8025         CALL ASCMVE
 8028         LD HL,4352 ;      Bottom third of generator table.
 802B         CALL ASCMVE
 802E         LD HL,8448 ;      Point to equivalent colour table location for
 8031         NOP        ;      Top third of graphics generator.Colour table is 8192 bytes
 8032         NOP        ;      higher in vram so add 8192 to character position = Col pos.
 8033         CALL COLSET        ;       go send relevant information.
 8036         LD HL,10496        ;        2nd third colour table.
 8039         CALL COLSET
 803C         LD HL,12544        ;       bottom third colour table.
 803F         CALL COLSET
 8042         NOP          ;      Everything is now set up for G2 screen in a character mapped
 8043         NOP          ;      format..We can now write to the screen bby sending the
 8044         NOP          ;      Ascii character through port 01....LD A,"X" : OUT(01),A
 8045         NOP          ;       or....LD A,88 [Ascii no X]:OUT(01),A
 8046         NOP          ;##########################################################
 8047         NOP          ;    LD This will print a message on screen
 8048         NOP          ;##########################################################
 8049         JP START
 804C         NOP          ;####################################################
 804D         NOP          ;        DATA TO SET UP VDP REGISTERS SAME ADDRESSES AS VS 4
 804E         NOP          ;####################################################
```

APPENDIX E

THE SUBROUTINES

```
;
;Random number routine.  Call this routine then load parameters into PARA which is
;a two byte word.  Result is return in VAL  BC must be preserved until
;result has been obtained. Procedure:  LD A,R:CALL
;RND: LD HL,PARAMS: LD A,HIGH
;VALUE: CALL PARA : LD A,(VAL) = RANDOM NUMBER
;
RND:
        PUSH    AF
        PUSH    BC
        PUSH    DE
        PUSH    HL
        LD      A,R
        LD      (SEED3),A
        LD      DE,(SEED)
        LD      HL,(SEED2)
        LD      B,07
RND10:
        CALL    SHIFT
        DJNZ    RND10
        LD      B,03
RND20:
        CALL    SUB
        DJNZ    RND20
        LD      (SEED),DE
        LD      (SEED2),HL
        LD      A,7FH
        AND     D
        LD      (VAL),A         ;TEMP STORE FOR RANDOM NUMBER BEFORE
        POP     HL              ;CALLING PARAMS
        POP     DE
        POP     BC
        POP     AF
        RET
SHIFT:
        ADD     HL,HL
        EX      DE,HL
        ADC     HL,HL
        EX      DE,HL
        RET
```