The mysteries of the Memotech's screen handling features are dispelled by Keith Hook's expert explanations.

Screen studies of an

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 screen, and the manual doesn't explain how you can write to and read from the screen using POKE and PEEK. The confusion is basically caused by the way the display operations are managed. The Memotech uses the Texas TM 9929A Video Display Processor (VDP) to handle all data relating to the screen, while other micros tend to use the cpu for this.

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 is memory mapped at 4400Hex to 47FFHex for the low-resolution screen. Fast writes or reads from/to screen can be accomplished by PEEK (*address*) or POKE *address, value* (as shown in Diagrams 1 and 2).

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 (Diagram 3) at 1C00 Hex (7168 decimal) 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
```

● All addresses must be sent to the VDP LSB first, followed by MSB.

● The value of each address is contained in 14 bits. Bits 6 and 7 of the MSB inform the VDP which type of operation it has to perform, *eg* write to registers, write to VRAM, read from VRAM, read Status

DIAGRAM 1

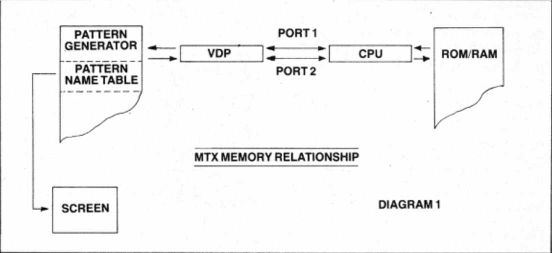


DIAGRAM 2

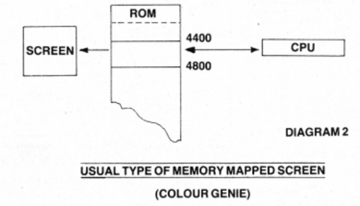
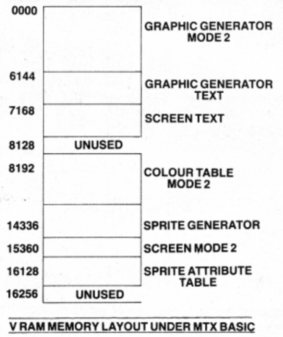


DIAGRAM 3



MTX

Register. When writing to VRAM, bit 7 must be 0 and bit 6 must be 1. This is taken care of with line 130 in Listing 1. The subroutines to POKE and PEEK VRAM differ slightly in their make-up; when reading from VRAM both bits 7 and 6 must be zero.

POKEing and PEEKing VS 4 — the high-resolution screen — is a very complicated business. In fact, PEEKing in the normal sense of the word is almost impossible, as in the high-resolution mode no values are contained in the pattern generator, and each value is loaded into the pattern name table and then the relevant colour byte is set. We will therefore restrict our routines to the text screen.

Sprite collision

It is a surprising fact that the MTX does not contain any Basic command dedicated to the detection of sprite collisions.

Sprite collisions can be detected by examining the value of bit 5 in the Status (Read Only) Register. Whenever the '1' bits of two sprites coincide on the screen the VDP sets bit 5 in the Status Register to 1, otherwise bit 5 is zero.

```
Status    Bit: 7   6   5   4   3210
Register       F   5s  C   —   5th sprite no.
```

However, there is one drawback to this method of checking sprite collisions: the VDP will detect collisions between sprites which are not on the visual plane, and will detect them between those that are not even active, if their X,Y coordinates coincide. To overcome this problem you have to place a value of 208 in the Y coordinate of the sprite number directly after the last sprite you wish to include. The VDP will then terminate its checking when this value is encountered. This procedure can be accomplished by ADJSPR 3, *sprite number*, 208.

Listing 2 gives details of the subroutine that will check bit 5 and return to the main program with the value of 0 = no collision, 1 = sprite collision. The Status Register is read by INP (02).

Joysticks

The left-hand joystick is mapped to the cursor control keys, and the functions of the joystick are identical to those of the cursor keys, *eg* cursor left = joystick left.

Detecting multiple key presses with the INKEY$ function involves calling a subroutine at least twice. A better way to detect keyboard movement is to by-pass the INKEY$ function and carry out a strobe of the computer's sense-lines.

On the MTX each key has a unique value that is output on port 5 — this is termed the sense-byte. If the key is then depressed, a value of 127 will be returned when port 5 is read.

You can test this for yourself with:

```
10 LET A = PEEK (64894); System LASTDR
20 PRINT A;
30 GOTO 10
```

The screen should fill up with the value of 127. If you press any key, say the Home key, the value of 223 will be printed on the screen — this is the value that is sent out via port 5 to test if the Home key has been depressed.

Listing 2 will return the following values:

223	Fire Button
247	Joy left
239	Joy right
251	Joy up
191	Joy down

Listing 3 is a subroutine that allows you to test for a multiple key press. As stated in the listing, you will have to build a routine around this that will allow you to take the appropriate action for either a single key press or a multiple key press. PCN

LISTING 1

```
;        SUBROUTINE TO SET UP VRAM ADDRESS
;        THIS IS ON THE ASSUMPTION THAT VARIABLE 'AD' ALWAYS = START ADD
;        VARIABLE 'LOC' = ACTUAL SCREEN ADDRESS
;
;
100 LET AD = 1024 * 7              ; AD = START OF TEXT SCREEN
110 LET AD = AD + LOC              ; AD = ACTUAL SCREEN LOCATION
120 LET LSB = AD-(INT(AD/256)*256)
130 LET MSB = AD/256 + 64          ; MAKE SURE BIT 7=0 & BIT 6=1
140 OUT (02),LSB
150 OUT (02),MSB
160 RETURN
;
;
;
;
; SUBROUTINE TO SEND DATA TO ADDRESS SET UP WITH ABOVE ROUTINE.
; VARIABLE 'DTA' = VALUE OF DATA TO WRITE TO SCREEN.
;
200 OUT (01),DTA
210 RETURN
;
;
;
; SUBROUTINE TO READ A VALUE FROM TEXT SCREEN.
; ON EXIT FROM ROUTINE VARIABLE 'VRD' = VALUE ON SCREEN.
;
300 LET AD = 1024 * 7 : LET AD = AD + LOC
310 LET LSB = AD-(INT(AD/256)*256)
320 LET MSB = AD/256
330 OUT (02),LSB : OUT (02),MSB
340 LET VRD = INP(01)
350 RETURN
```

LISTING 2

```
; SUBROUTINE TO READ VDP STATUS REGISTER FOR SPRITE COLLISION.
; VARIABLE 'COL' WILL BE = 1 IF COLLISION DETECTED ELSE = 0
;
;
400 LET SCOL = INP(02)
410 LET COL = MOD(INT(SCOL/32),2)
420 RETURN
```

LISTING 3

```
; SUBROUTINE TO READ SENSE LINES FOR MULTIPLE MOVEMENT OF JOYSTICK
; E.g. FIRE BUTTON AND MOVE LEFT
;
; AFTER CALLING THIS ROUTINE SOME PROGRAM WILL BE NEEDED TO TAKE
; THE APPROPRIATE ACTION....EG
; 20 GOSUB 500
; 30 ON SRD GOSUB  50,60,70,80,90,100  etc , etc
; VALUES RETURNED ARE:- 1     JOYSTICK LEFT
;                       2     JOYSTICK RIGHT
;                       3     JOYSTICK UP
;                       4     JOYSTICK DOWN
;                       5     FIRE BUTTON PRESSED
; FIRE BUTTON AND JOYSTICK LEFT RETURNS A VALUE OF 6 AND SO ON...
;
;
500 LET SRD = 0 : LET SB = 223: OUT (05),SB
510 LET RB = INP(05)
520 IF RB = 127 THEN SRD = 5
530 LET SB = 247 : OUT (05),SB
540 LET RB = INP(05)
550 IF RB = 127 THEN SRD = SRD + 1 : RETURN
560 LET SB = 239: OUT (05),SB
570 LET RB = INP(05)
580 IF RB = 127 THEN SRD = SRD + 2 : RETURN
590 LET SB = 251 : OUT (05),SB
600 LET RB = INP(05)
610 IF RB = 127 THEN SRD = SRD + 3 : RETURN
620 LET SB = 191 : OUT (05),SB
630 LET RB = INP(05)
640 IF RB = 127 THEN SRD = SRD + 4
650 RETURN
```