

MEMOTECHNIQUES

Still getting to grips with your brand new Memotech? Keith Hook follows up on the manual.



Take a memo

The Memotech is an attractive micro, but as happens so often with new machines, the manual doesn't cover everything. The MTX offers the programmer an excellent variety of display modes for use with text or graphics:

Text mode User definable graphic characters; 40×24 character screen.

Text with graphics User definable graphics, sprites; 32×24 character screen.

Text with 256×192 pixel graphics User definable graphics, sprites, 16 colours for characters.

Using these different modes via Basic creates no problems, as the instruction

manual is very thorough in this respect — although it does omit instructions on how the novice can easily detect sprite collisions. Implementing the assortment of screens using Assembly language is a more daunting task. It is not made any easier by the omission of ROM entry points from the manual, which seems a lack of concern for the more advanced programmer, by a company that took the trouble to incorporate an editor/assembler in the front end of the machine. On the other hand there are numerous pages in the manual devoted to the graphic facilities, though in some places it is so thorough, it is a headache to sort the wood from the trees.

The following explanations and subroutines, using the in-built assembler, should allow the end-user to easily set up different display modes.

Graphic Mode 2 gives the programmer a great deal of scope when designing games programs, and it is this mode that's been chosen for the illustrations. It should, however, be a simple matter to convert the information given, to set up other display modes.

Using this mode, 768 different characters can be designed and displayed at one time. It also provides a choice of defining 2 colours for each byte of the 8 byte character, with the effect that each of the 768 (32×24) screen locations can display an exclusive character.

The following descriptions, use the conventions used by Memotech in their manual ie: the bits of a byte are numbered MSB bit = 0 and LSB bit = 7.

The MTX decides which mode it should operate in by looking at VDP Registers 0 and 1. Mode 2 is entered by setting bit 6 in Register 0 to 1 and resetting bits 3 and 4 in Register 1.

When setting up a screen the MTX uses the following tables which are located in Video RAM: pattern generator table; pattern name table; colour table; sprite attribute table; sprite generator table.

Pattern generator table This is really a sophisticated form of character generator which is normally installed in a ROM chip in other computers, characters so generated are numbered `chr$(0)` to `chr$(255)`. With the MTX you can design all 256 and more.

Pattern name table This table is a copy of the screen in VRAM. If the value of 128 is placed in location 32, it is like saying: `PRINT@32,CHR$(128)`. This is a simple translation but it illustrates the function of the pattern name table.

Colour table Each byte of this table holds the colour information for the corresponding

DATA TRANSFER FORMAT

Write to registers of VDP

Out (02), data

Out (02), register number (bit 7 must be set to 1)

Write to video RAM

Out (02), LSB address

Out (02), MSB address (bit 7 must be 0: bit 6 must be set to 1)

Out (01), data

Read from VRAM

Out (02), LSB address

Out (02), MSB address (bits 7 & 6 must be 0)

In (02), data

Read from DP registers (status register 'read only')

In (02), data

VIDEO RAM LOCATIONS FOR GRAPHIC MODE 2

Pattern generator table
Pattern name table
Colour table
Sprite attribute table
Sprite generator table

Start Location 0000H	END 17FF H
Start location 1800H	END 1AFF H
Start location 2000H	END 37FF H
Start location 1C00 H	(32 x 4 byte entries)
Start location 3800 H	(2048 byte table)

byte of the character in the pattern generator table.

The colour and pattern Generator tables are 6144 bytes long. Each character is 8×8 pixels and 8 pixels are held in each byte. Since there are 768 locations on the screen $8 \times 768 = 6144$ bytes. The pattern name table is 768 bytes long.

As there are only 256 character codes available at any one time, the MTX overcomes this problem by splitting the pattern generator table into 3 sections of 2048 bytes ($8 \text{ bytes} \times 256 \text{ chars}$). Therefore, the 256 character names (0–255) are used 3 times, once in each section of the table.

The pattern name table is similarly divided into 3 sections to match the pattern generator table. The top section of the table corresponds to the top 3rd of the screen. The colour table follows the same format. It is plain that if character code 128 describes a space invader in the top section of the screen, and the invader is moved around the screen, the code 128 must also describe the invader in the screen section it moves into. This is not strictly true, but for the sake of simplicity let's stick to this explanation. See diagram 1.

Once the tables have been set up in VRAM, animation is easily achieved by changing the values in any of the 3 tables. And, as the VDP automatically increments the address register, once the start address has been loaded into the VDP sequential writes to VRAM only need the 1 byte data transfer by Out (01), data.

Setting up VRAM tables The location of each table in VRAM is set by transferring the relevant information to the VDP via the VDP Registers. (see tables 1 and 3). The values chosen set each table to addresses which allow for easy translation between the different tables.

VDP registers The correct procedure for writing information to the VDP Registers

Data first byte

Register number 2nd byte

The VDP then calculates the correct location in VRAM according to the value of the Data byte, eg For Register 2 (name table), the value transferred is multiplied by 1024, and the resulting value is the start address of that table. In our examples Register 2 is located at 6144 (1800h) in VRAM.

For all writes to the VDP Registers, bit 0 must be set to 1 in the Register number byte. This is accomplished by adding 128 (8h) to the value. The sub-routine in listing 1 takes care of VDP writes. First define your values with Data first, register number second, eg

VALUES: db data, Reg no, data, Reg no, etc

Then set up a loop as in listing 1 until all 8 Registers have been fed information.

Write to VRAM Data is written to VRAM by feeding the DESTINATION address to the VDP LSB first MSB second followed by the data byte. Note that the bit 7 must be high (bit 6 set to 1 in the MSB of the address byte). This is done by adding 64 (40h) to this byte. The sub-routine for writing to

MAIN LISTING

```

400D    NOP1 ~~~ THIS PROGRAM WILL DISPLAY CHARACTERS ON YELLOW BACKGROUND
400E    NOP1 TOP 3RD OF SCREEN
400F    NOP1 BLACK CHARACTERS LIGHT RED BACKGROUND MIDDLE 3RD
4010    NOP1 REST OF SCREEN AND BORDER MAGENTA
4011    NOP
4012    NOP
4013    LD SP, (#FH96)1 LOAD STACK POINTER FROM SYSTEM STACK
4017    LD DE, #00001 ZERO ALL REGISTERS
401A    CALL REG
401D    LD B, #081 NUMBER OF REGISTERS
401F    LD HL, REGSET1 MAKE HL POINT TO DATA BUFFER
4022 LP1: LD E, (HL)1 PUT DATA IN DE
4023    INC HL1 NEXT DATA SET
4024    LD D, (HL)
4025    INC HL
4026    CALL REG; SEND TO REGISTER
4029    DJNZ LP1 DO 8 TIMES
402B    LD HL, #3FFF TOTAL LENGTH OF VRAM
402E    LD DE, #00001 ZERO ALL VRAM
4031    CALL VRAM
4034    LD C, #00
4036    CALL LOOP
4039    LD DE, #20001 START OF COLOUR TABLE TOP 3RD OF SCREEN
403C    CALL VRAM1 SEND ADDRESS
403F    LD C, #1A1 I=COLOUR OF CHARACTER 1'S A= COLOUR OF 0'S
4041    LD HL, #02001 HOW MANY BYTES TO FILL
4044    CALL LOOP1 DO IT
4047    LD C, #181 AS ABOVE
4049    LD HL, #0200
404C    CALL LOOP
404F    LD C, #18
4051    LD HL, #0800
4054    CALL LOOP
4057    LD DE, #00001 ADDRESS OF CHARACTER NO 2 TOP 3RD SCREEN
405A    CALL VRAM1 LOAD ADDRESS
405D    LD HL, INU1 MAKE HL POINT TO CHARACTER BYTES
4060    LD B, #081 8 BYTES EACH CHARACTER
4062 AGN1: LD C, (HL)1 PUT DATA INTO C
4063    CALL DATA1 SEND TO VDP
4066    INC HL1 ALIGN TO NEXT BYTE
4067    DJNZ AGN1 DO UNTIL FINISHED
4069    LD DE, #02001 CHARACTER NO 2 MIDDLE 3RD SCREEN
406C    CALL VRAM1 SEND TO REGISTER
406F    LD HL, INU1 AS FOR TOP 3RD
4072    LD B, #08
4074 AGH1: LD C, (HL)
4075    CALL DATA1
4078    INC HL
4079    DJNZ AGH1
407B    LD DE, #18001 START OF TOP 3RD OF SCREEN
407E    CALL VRAM1 SEND ADDRESS
4081    LD C, #011 CHARACTER NUMBER
4083    CALL DATA1 SEND TO VDP
4086    LD C, #011 SEND SAME CHARACTER TO BE DISPLAYED AT SCREEN POS 2
4088    CALL DATA1 DO IT
408B    LD B, #081 SAME AS ABOVE BUT SEND 8 ONE AFTER THE OTHER
408D PL1: LD C, #01
408F    CALL DATA1
4092    DJNZ PL1
4094    LD DE, #19441 THIS IS IN THE MIDDLE 3RD OF SCREEN
4097    CALL VRAM1 SEND ADDRESS
409A    LD C, #011 2ND CHARACTER FROM MIDDLE 3RD OF CHARACTER GEN TABLE
409C    CALL DATA1 DISPLAY
409F    LD C, #011 DISPLAY
40A1    CALL DATA1 SAME CHARACTER NEXT SCREEN POS
40A4 STOP: NOP1 LOOP AT END OF RUN
40A5 JR STOP
40A7 REG: PUSH AF; ##### SEE NOTES FOR THESE SUB ROUTINES
40A8 PUSH BC
40A9 LD A, E
40AA OUT (82), A
40AC LD A, D
40AD ADD A, #80
40AF OUT (82), A
40B1 POP BC
40B2 POP AF
40B3 RET
40B4 VRAM1 PUSH AF
40B5 PUSH BC
40B6 LD A, E
40B7 OUT (82), A
40B9 LD A, D
40BA ADD A, #40
40BC OUT (82), A
40BE POP BC
40BF POP AF
40C0 RET
40C1 DATA1 PUSH AF
40C2 LD A, C
40C3 OUT (81), A
40C5 POP AF
40C6 RET
40C7 LOOP: CALL DATA1
40CA DEC HL
40CB LD A, H
40CC OR L
40CD JR NZ, LOOP
40CF RET
40D0 REGSET1 DB #02, #00, #C2, #01, #06, #02, #FF, #03, #03, #04, #38, #05, #07, #06, #00, #07
40E0 INU1 DB 66, 165, 189, 219, 60, 36, 66, 129
40E3 RET
40E9 RET
40EA RET
40EB RET
40EC RET

```

Symbols:

REG40H7REGSET40D0
LP14822VRAM40B4
LOOP40C7INU40E0
AGN40E2DATA40C1
AGN14074PL40B0
STOP40A4

MEMOTECHNIQUES

◀ 29

VRAM is shown in Listing 2.

Reading from VRAM This is accomplished by writing the address from which you want the information, followed by a 'READ'. See table 1.

Reading from the VDP status register IN (02) data. Bit 2 is the sprite collision flag set whenever pixels from two active sprites overlap. The state of this bit can easily be achieved:

```
IN A,(02)
BIT 6,A (Normal Z80 bit convention)
```

```
JR NZ,COLLIDE
JP Move
```

The function of the other bits is described in the manual.

After filling the pattern generator, the colour table requires setting to the correct values for each of the characters. The first 4 bits (LSB), describe the colour of the 0's in your design, and the MSB's describe the colour of the 1's; in fact, background and text, in the other modes. See Diagram 2.

Completion of these steps allows you to direct the desired format to the screen by loading the pattern name table with the correct value for the character you wish to

display. Don't forget that sequential writes to the pattern name table can be accomplished with only one address transfer ie OUT (02), LSB ADDRESS
OUT (02), MSB ADDRESS
OUT (01), DATA
OUT (01), DATA ... etc

This format holds true for all tables.

Sprites Sprites are introduced to the display by using the sprite generator and sprite attribute tables.

The attribute table requires a 4 byte entry for each sprite. Using all of the 32 sprites available will cause the table to be 128 Bytes long. The format for this table is:
byte 1 Vertical distance from top of screen
byte 2 Horizontal distance from LHS of screen
byte 3 Pointer to sprite pattern
byte 4 Colour of sprite

The sprite generator functions in the same way as the pattern generator table. The maximum length is 2048 bytes, which allows you to define 256 different patterns for the normal size 0 sprite. Using size one (16×16 pixels) reduces the number to 64 (32 bytes each sprite pattern).

Sprites are displayed on the screen by setting the X,Y values in the sprite attribute table. Movement is accomplished by updating the values. Diagram 1 shows the relationship between X,Y and pattern name table addresses. 1 should be deducted from the Y positions shown in actual practice, as $X=0, y=-1$ is the top most lefthand corner of the screen. Positioning of the sprite is from the top lefthand bit of the sprite pattern.

So, changing the pattern or the colour of a sprite is easily accomplished by altering the relevant byte in the attribute table. This leads to very impressive moving displays. If you restrict the majority of movement to 8 pixels each move, calculating the corresponding screen position for checking collisions with other graphics is easily accomplished. Listing 4 takes care of these calculations. Listing 3 calculates the reverse positions.

We have included a fully documented listing that illustrates the practical use of some of these points within a program. The possibilities available to you are only limited by your own imagination.

LISTING 1

```
11 REM -----  
20 REM ON ENTRY HL POINTS TO DATA  
30 REM D = REGISTER NUMBER  
40 REM E = DATA  
50 REM  
60 CODE  
  
409B BUFFER: DB #0C: DATA  
409C DB 041 REGISTER  
409D DB 127: AND SO ON FOR ALL REGISTERS  
409E LD HL,BUFFER: BUFFER HOLDS DATA  
40A1 LD B,#00: NUMBER OF REGISTERS  
40A3 LOOP: LD E,(HL): SET DATA  
40A4 INC HL  
40A5 LD D,(HL): GET REGISTER NUMBER  
40A6 INC HL: ALIGN FOR LOOP  
40A7 CALL SETUP: GO AND TRANSFER IT  
40A8 DJNZ LOOP1: DO IT 8 TIMES  
40AC SETUP: LD A,E: DATA FIRST  
40AD OUT (02),A: SEND IT  
40AF LD A,D: GET REGISTER NUMBER  
40B0 ADD A,#00: MAKE SURE BIT 7 SET  
40B2 OUT (02),A: SEND IT  
40B4 RET: ALL DONE RETURN  
40B5 RET  
40B6 RET  
  
Symbols:  
BUFFER409BSETUP40AC  
LOOP40A3
```

LISTING 3

```
10 REM Convert screen to X,Y CO-ORDINATES  
20 REM -----  
30 REM  
40 REM  
50 REM ON ENTRY HL POINTS TO SCREEN POSITION  
60 REM  
70 REM  
80 CODE  
  
40AE LD DE,#1800: START OF SCREEN  
40B1 OR A: CLEAR CARRY FLAG  
40B2 SBC HL,DE1 SUB FROM SCREEN LOCATION  
40B4 LD DE,#32: NUMBER OF COLUMNS  
40B7 LD B,00: CLEAR ANSWER REG (QUOTIENT)  
40B9 LOOP: OR A: MAKE SURE CARRY FLAG CLEAR  
40BA SBC HL,DE  
40BC JP M,EXIT: EXIT IF MINUS  
40BF INC B: INC ANSWER  
40C0 JR LOOP: GO DO IT UNTIL MINUS  
40C2 EXIT: ADD HL,DE: GET TRUE REMAINDER  
40C3 RET: ON RETURN HL=K POSIT & B=Y POSIT  
40C4 RET  
  
Symbols:  
LOOP40B9EXIT40C2
```

LISTING 4

```
20 REM Convert X,Y to SCREEN POSITION  
30 REM  
40 REM  
50 REM  
60 REM ON ENTRY BC = X POSITION  
70 REM " " HL = Y POSITION  
80 REM  
90 REM  
100 CODE  
  
40B7 PUSH BC: SAVE X POSITION  
40B8 LD B,#00: CLEAR QUOTIENT  
40B9 LOOP: OR A: CLEAR CARRY FLAG  
40B9 SBC HL,DE: DIVIDE BY SUBTRACTION  
40BD JP M,EXIT: EXIT IF MINUS  
40C0 INC B: INC QUOTIENT  
40C1 JR LOOP: DO IT UNTIL MINUS  
40C3 EXIT: ADD HL,HL: MULTIPLY BY 32  
40C4 ADD HL,HL: BY SUCCESSIVE ADDS  
40C5 ADD HL,HL  
40C6 ADD HL,HL  
40C7 ADD HL,HL  
40C8 POP BC: GET X POSITION  
40C9 ADD HL,BC: ADD IT TO ANSWER  
40CA LD DE,#1800: START OF SCREEN LOCATION  
40CD ADD HL,DE: GET SCREEN POSITION  
40CE RET: HL = SCREEN LOCATION  
40CF RET  
40D0 RET  
  
Symbols:  
LOOP40B9EXIT40C3
```

Register set up for above values

Register 0 : Value 02 H	'set to mode 2'
Register 1 : Value C2 H	'8x8 sprite Mag 1'
Register 2 : Value 06 H	
Register 3 : Value FF H	'MSB set to 1'
Register 4 : Value 03 H	'MSB set to 0'
Register 5 : Value 38 H	
Register 6 : Value 07 H	
Register 7 : Value A1 H	(sets Text/Background colours)

NB Setting bit 7 to 1 in REGISTER 3, puts the colour table in the top 8K of VRAM. Set to 0 will put it at the location of the pattern generator in this example — as would setting bit 7 to 1 in the pattern generator register, put it in the location of the colour table in this example. eg Value FF H = location 2000

LISTING 2

```
20 REM -----  
30 REM ON ENTRY DE HOLDS ADDRESS  
40 REM ON ENTRY C HOLDS DATA  
50 REM  
60 REM  
70 CODE  
  
409C LD A,E: GET LSB OF ADDRESS  
409D OUT (02),A: SEND IT  
409F LD A,D: GET MSB OF ADDRESS  
40A0 ADD A,#40: RESET BIT7 SET BIT 6  
40A2 OUT (02),A: SEND IT  
40A4 LD A,C: GET DATA INTO A  
40A5 OUT (01),A: SEND IT  
40A7 RET: RETURN TO CALL OUT  
40A8 RET
```

Next week's issue of PCW will include a handy diagram which will help you understand the Memotech's screen modes better, and will further clarify the points touched on in this article.