

MSX2 TECHNICAL HANDBOOK

Edited by: ASCII Systems Division  
Published by: ASCII Corporation - JAPAN  
First edition: March 1987

Text file typed by: Nestor Soriano (Konami Man) - SPAIN  
March 1997

Changes from the original:

- In Figure 4.72, last "10000H" is corrected to "1FFFFH".
- In Table 4.6, in TEOR line, "else DC+..." is corrected to "else DC=..."
- In Figure 4.76, in R#45 figure, DIX and DIY bits have been placed correctly (they were inverted in the original).
- In Figure 4.79, in R#42 and R#43 explanation, "NY -> of dots..." has been changed to "NY -> number of dots..."
- In List 4.9, in the line with the comment "YMMM command", 11010000 bitfield has been corrected to 11100000.
- In Figure 4.84, "\*" mark removed from the explanation of NX.
- In Figure 4.85, in R#45 explanation, "select source memory" text has been corrected to "select destination memory".
- In List 4.13, labels beginning with "LMMC" have been corrected to "LMCM".
- In List 4.15, in the line with the comment "NY", the "OUT (C),H" instruction has been corrected to "OUT (C),L".
- In section 6.5.9, the explanation of usage of the LINE command were mixed with other text. It has been corrected.
- In Figure 4.94, a line explaining the meaning of R#44 has been added.
- In Figure 4.97, BX9 bit has been suppressed in S#9 figure.
- In Figure 4.99, a line explaining the meaning of R#44 has been added.
- In Table 4.7, "CLR L" has been corrected to "CMR L".

-----

CHAPTER 4 - VDP AND DISPLAY SCREEN (Part 6)

6. VDP COMMAND USAGE

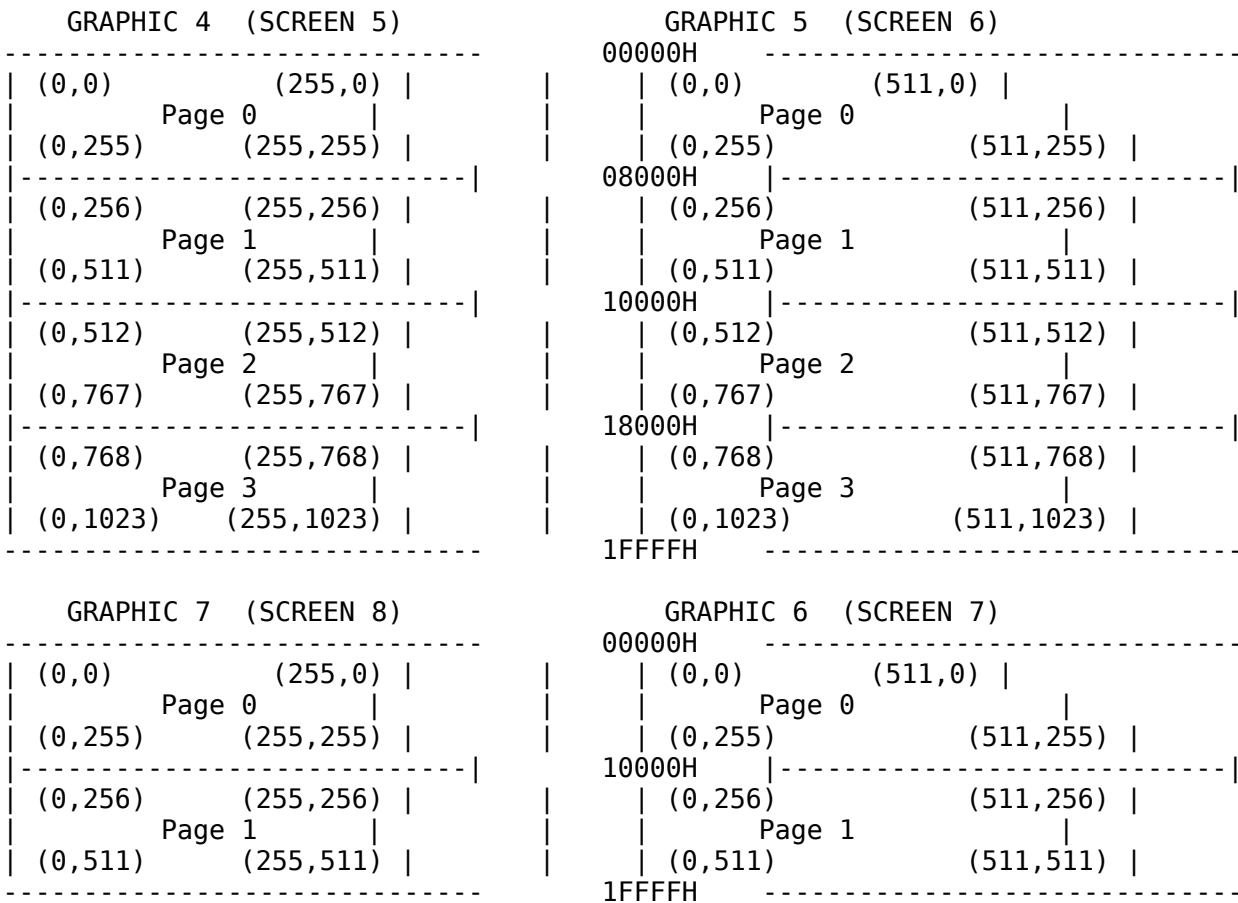
MSX-VIDEO can execute basic graphic operations, which are called VDP commands. These are done by accessing special hardware and are available in the GRAPHIC 4 to GRAPHIC 7 modes. These graphic commands have been made easy to implement, requiring only that the necessary parameters be set in the

proper registers before invoking them. This section describes these VDP commands.

### 6.1 Coordinate System of VDP Commands

When VDP commands are executed, the location of the source and destination points are represented as (X, Y) coordinates as shown in Figure 4.72. When commands are executed, there is no page division and the entire 128K bytes VRAM is placed in a large coordinate system.

Figure 4.72 Coordinate system of VRAM



### 6.2 VDP Commands

There are 12 types of VDP commands which can be executed by MSX-VIDEO. These are shown in Table 4.5.

Table 4.5 List of VDP commands

Command name	Destination	Source	Units	Mnemonic	R#46 (4 hi ord)
High speed	VRAM	CPU	bytes	HMMC	1 1 1 1
	VRAM		VRAM	YMMM	1 1 1 0

move	VRAM	VRAM	bytes	HMMM	1 1	0 1
	VRAM	VDP	bytes	HMMV	1 1	0 0
Logical	VRAM	CPU	dots	LMMC	1 0	1 1
move	CPU		VRAM	dots	LMCM	1 0
	VRAM	VRAM	VRAM	dots	LMMM	1 0
	VRAM	VDP	dots	LMMV	1 0	0 0
Line	VRAM	VDP	dots	LINE	0 1	1 1
Search	VRAM	VDP	dots	SRCH	0 1	1 0
Pset	VRAM	VDP	dots	PSET	0 1	0 1
Point	VDP	VRAM	dots	POINT	0 1	0 0
Reserved	----	----	-----	----	0 0	1 1
	----	----	-----	----	0 0	1 0
	----	----	-----	----	0 0	0 1
Stop	----	----	-----	----	0 0	0 0

\* When data is written in R#46 (Command register), MSX-VIDEO begins to execute the command after setting 1 to bit 0 (CE/Command Execute) of the status register S#2. Necessary parameters should be set in register R#32 to R#45 before the command is executed.

\* When the execution of the command ends, CE becomes 0.

\* To stop the execution of the command, execute STOP command.

\* Actions of the commands are guaranteed only in the bitmap modes (GRAPHIC 4 to GRAPHIC 7).

### 6.3 Logical Operations

When commands are executed, various logical operations can be done between data in VRAM and the specified data. Each operation will be done according to the rules listed in Table 4.6.

In the table, SC represents the source color and DC represents the destination colour. IMP, AND, OR, EOR and NOT write the result of each operation to the destination. In operations whose names are preceded by "T", dots which correspond with SC=0 are not the objects of the operations and remains as DC. Using these operations enables only colour portions of two figures to be overlapped, so they are especially effective for animations.

List 4.7 shows an example of these operations.

Table 4.6 List of logical operations

Logical name	L03	L02	L01	L00

IMP	DC=SC	0 0 0 0
AND	DC=SCxDC	0 0 0 1
OR	DC=SC+DC	0 0 1 0
EOR	$\overline{DC}=SCxDC+SCx\overline{DC}$	0 0 1 1
NOT	$\overline{DC}=SC$	0 1 0 0
----		0 1 0 1
----		0 1 1 0
----		0 1 1 1
-----+-----+-----		
TIMP	if SC=0 then DC=DC else DC=SC	1 0 0 0
TAND	if SC=0 then DC=DC else DC=SCxDC	1 0 0 1
TOR	if SC=0 then DC=DC else DC=SC+DC	1 0 1 0
TEOR	if SC=0 then $DC=\overline{DC}$ else $DC=SCx\overline{DC}+SCxDC$	1 0 1 1
TNOT	if SC=0 then $DC=\overline{DC}$ else DC=SC	1 1 0 0
----		1 1 0 1
----		1 1 1 0
----		1 1 1 1
-----		

\* SC = Source colour code  
 \* DC = Destination colour code  
 \* EOR = Exclusive OR

#### List 4.7 Example of the logical operation with T

```

1000 '*****
1010 ' List 4.7 logical operation with T
1020 '*****
1030 '
1040 SCREEN8 : COLOR 15,0,0 : CLS
1050 DIM A%(3587)
1060 '
1070 LINE (50,50)-(60,100),48,8 : PAINT (51,51),156,48
1080 CIRCLE (55,30),30,255 : PAINT (55,30),240,255
1090 COPY(20,0)-(90,100) TO A%
1100 CLS
1110 '
1120 R=RND(-TIME)
1130 FOR Y=0 TO 100 STEP 3

```

```

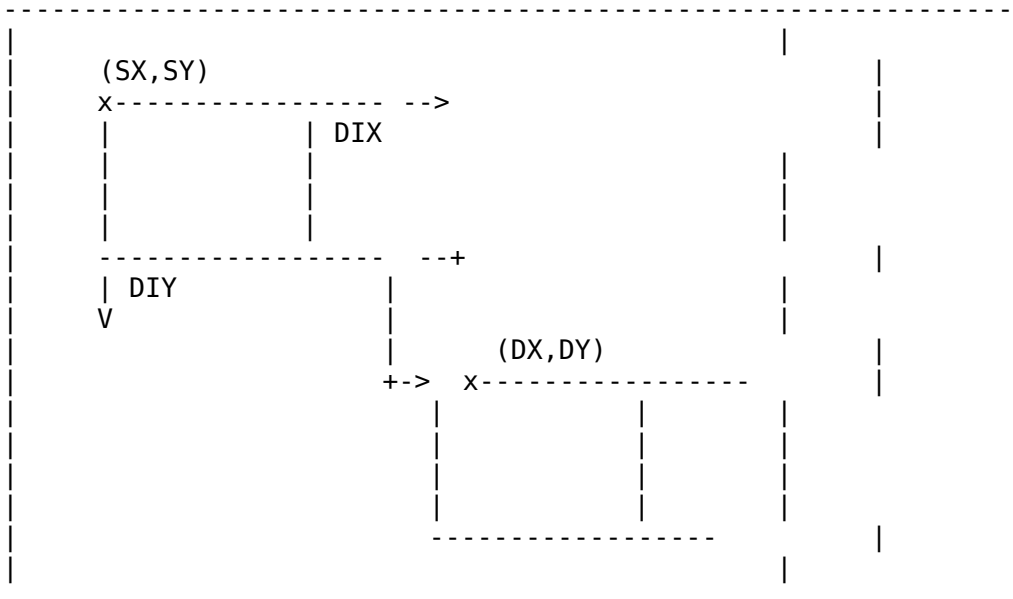
1140 X=INT(RND(1)*186)
1150 COPY A% TO (X,Y),,TPSET
1160 NEXT
1170 '
1180 GOTO 1180

```

## 6.4 Area Specification

AREA-MOVE commands are for transferring screen data inside areas surrounded by a rectangle. The area to be transferred is specified by one vertex and the length of each side of the rectangle as shown in Figure 4.73. SX and SY represent the basic point of the rectangle to be transferred and NX and NY represent the length of each side in dots. The two bits, DIX and DIY, are for the direction of transferring data (the meaning of DIX and DIY depends on the type of command). The point where the area is to be transferred is specified in DX and DY.

Figure 4.73 Area specification



## 6.5 Use of Each Command

Commands are classified into three types, high-speed transfer commands, logical transfer commands, and drawing commands. This section describes the commands and their use.

### 6.5.1 HMMC (CPU -> VRAM high-speed transfer)

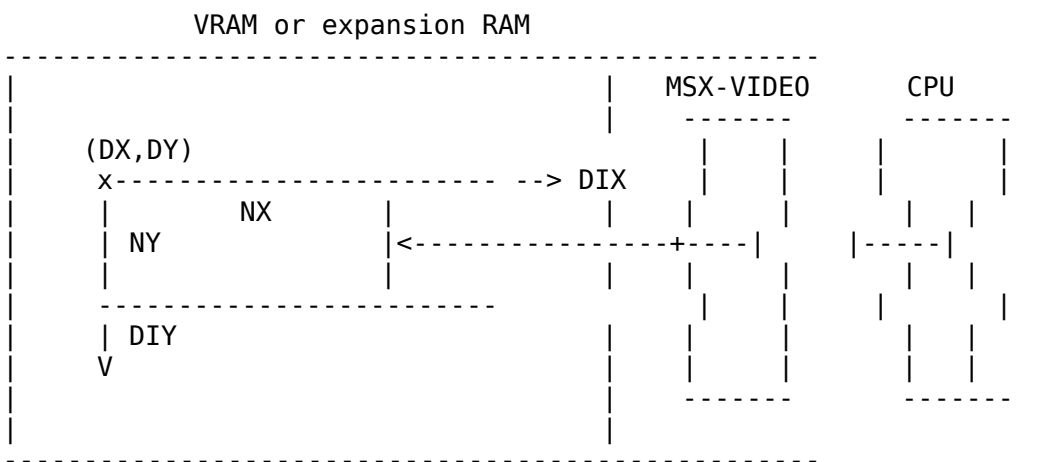
Data is transferred into the specified area of VRAM from the CPU (see Figure 4.74). Logical operations cannot be specified. Data is transferred in bytes in high-speed transfer commands such as HMMC. Note that the low order bit of the X-coordinate is not referred to in GRAPHIC 4, or 6 modes. The two low

order bits are not referred to in GRAPHIC 5 mode (see Figure 4.75).

Set the parameters as shown in Figure 4.76 to the appropriate registers. At this point, write only the first byte of data to be transferred from the CPU in R#44. Writing the command code F0H in R#46 causes the command to be executed, and UMSX-VIDEO receives data from R#44 and writes it to VRAM, then waits for data from the CPU.

The CPU writes data after the second byte in R#44. Note that data should be transferred after MSX-VIDEO can receive data (in the case that TR bit is "1"), referring to TR bit of S#2. When the CE bit of S#2 is "0", this means that all data has been transferred (see figure 4.77). List 4.8 shows an example of using HMMC.

Figure 4.74 Action of HMMC command



MXD: select the destination memory 0 = VRAM, 1 = expansion RAM

NX: number of dots to be transferred in X direction (0 to 511)\*

NY: number of dots to be transferred in Y direction (0 to 1023)

DIX: direction of NX from the origin 0 = right, 1 = left

DIY: direction of NY from the origin 0 = below, 1 = above

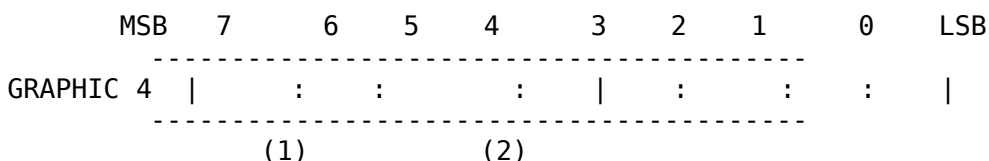
DX: destination origin X-coordinate (0 to 511)\*

DY: destination origin Y-coordinate (0 to 1023)

CLR (R#44:Colour register): 1st byte of data to be transferred

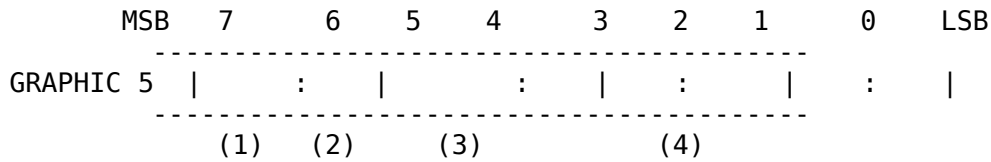
\* The one low-order bit for GRAPHIC 4 and 6 modes, or two low-order bits for GRAPHIC 5 mode of the DX and NX registers are ignored.

Figure 4.75 Dots not to be referred to

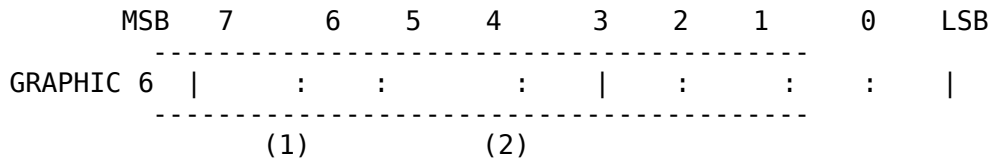


Since 1 VRAM byte represents 2 dots, 1 low order bit of X-coordinate is not

referred to.



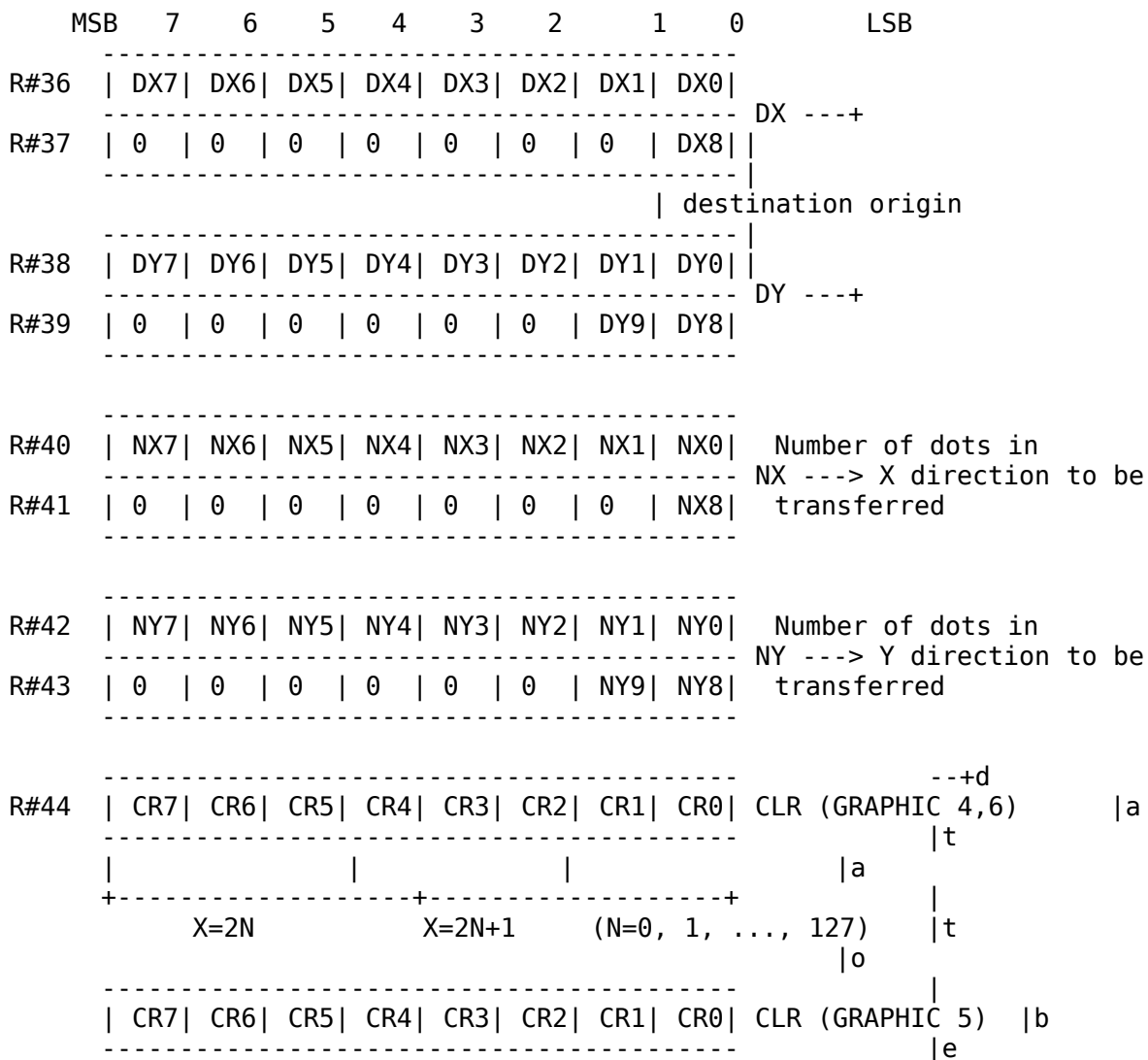
Since 1 VRAM byte represents 4 dots, 2 low order bits of X-coordinate are not referred to.

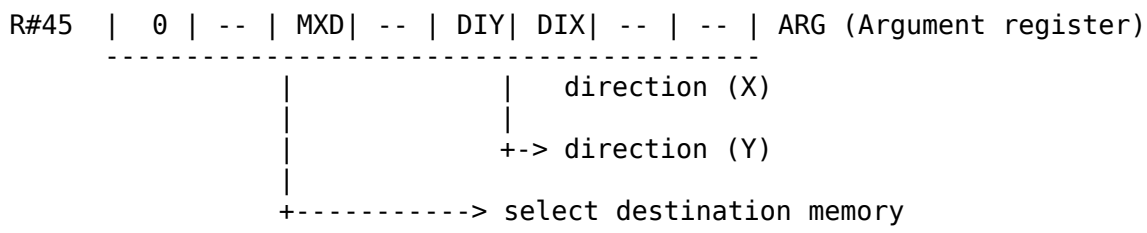
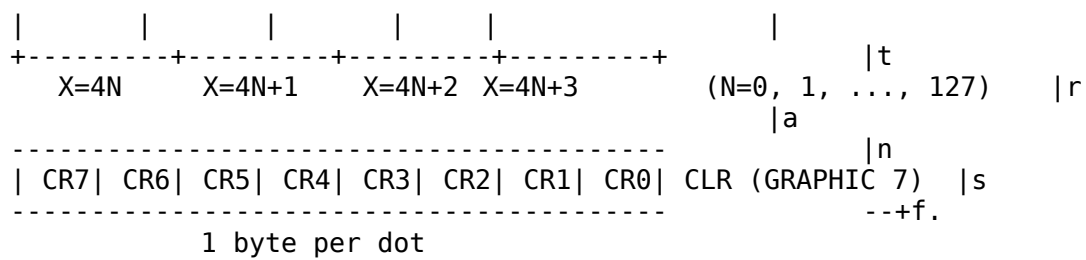


Since 1 VRAM byte represents 2 dots, 1 low order bit of X-coordinate is not referred to.

Figure 4.76 Register settings of HMMC command

> HMMC register setup





> HMMC command execution

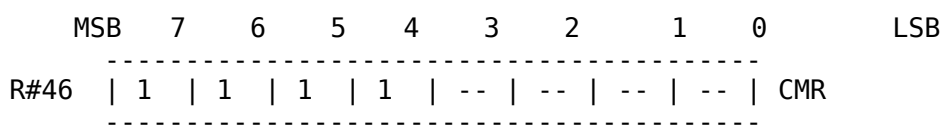
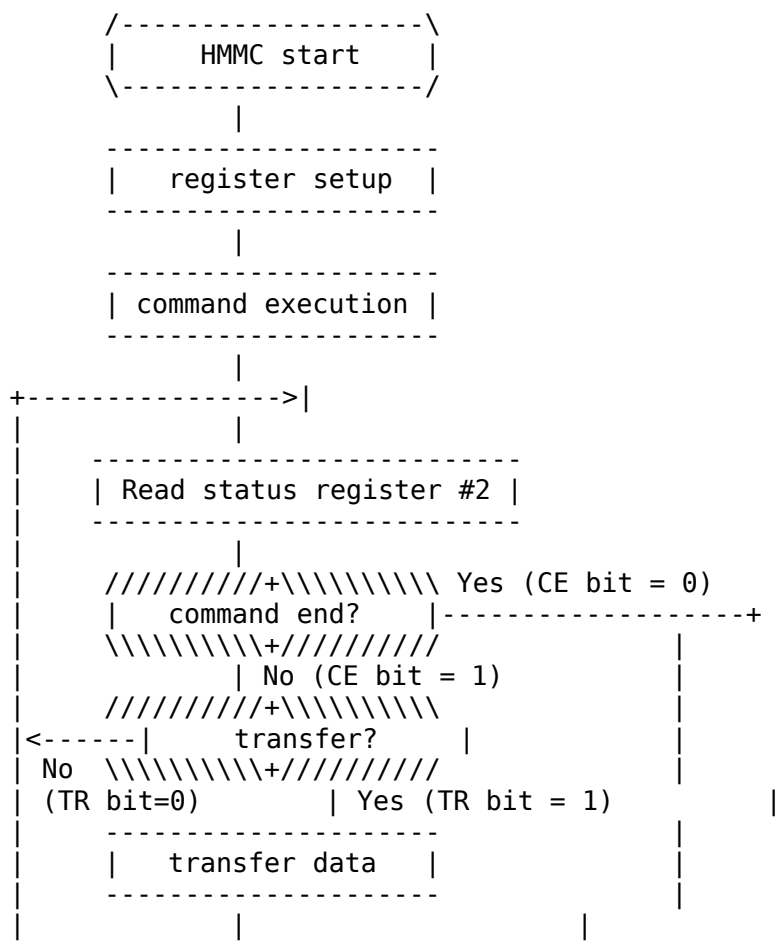
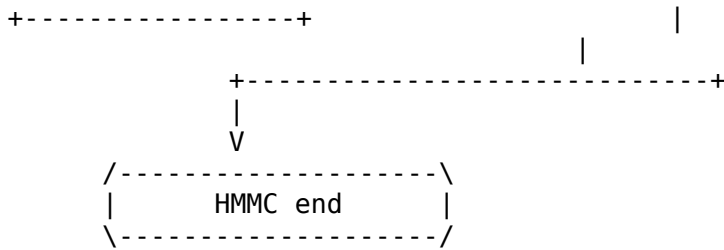


Figure 4.77 HMMC command execution flow chart







List 4.8 Example of HMMC command execution

```

;*****
; List 4.8 HMMC sample
; to use, set H, L, D, E, IX and go
; RAM (IX) ---> VRAM (H,L)-(D,E)
;*****
;
RDVDP: EQU 0006H
WRVDP: EQU 0007H

;----- program start -----

HMMC: DI ;disable interrupt
CALL WAIT.VDP ;wait end of command

LD A,(WRVDP)
LD C,A
INC C ;C := PORT#1's address
LD A,36
OUT (C),A
LD A,17+80H
OUT (C),A ;R#17 := 36

INC C
INC C ;C := PORT#3's address
XOR A
OUT (C),H ;DX
OUT (C),A
OUT (C),L ;DY
OUT (C),A

LD A,H ;make NX and DIX
SUB A
LD D,00000100B
JR NC,HMMC1
LD D,00000000B
NEG

HMMC1: LD H,A ;H := NX , D := DIX

LD A,L
SUB A
LD E,00001000B
JR NC,HMMC2
LD E,00000000B
NEG

HMMC2: LD L,A ;L := NY , E := DIY

```

```

XOR    A
OUT    (C),H           ;NX
OUT    (C),A
OUT    (C),L           ;NY
OUT    (C),A
LD     H,(IX+0)
OUT    (C),H           ;first DATA
LD     A,D
OR     E
OUT    (C),A           ;DIX and DIY
LD     A,0F0H
OUT    (C),A           ;HMMC command

LD     A,(WRVDP)
LD     C,A             ;C := PORT#1's address
INC    C
LD     A,44+80H
OUT    (C),A
LD     A,17+80H
OUT    (C),A
INC    C
INC    C

LOOP:  LD     A,2
CALL   GET.STATUS
BIT    0,A             ;check CE bit
JR     Z,EXIT
BIT    7,A             ;check TR bit
JR     Z,LOOP
INC    IX
LD     A,(IX+0)
OUT    (C),A
JR     LOOP

EXIT:  LD     A,0
CALL   GET.STATUS     ;when exit, you must select S#0
EI
RET

GET.STATUS:
;read status register specified by A
PUSH   BC
LD     BC,(WRVDP)
INC    C
OUT    (C),A
LD     A,8FH
OUT    (C),A
LD     BC,(RDVDP)
INC    C
IN     A,(C)
POP    BC
RET

WAIT.VDP:
;wait VDP ready
LD     A,2
CALL   GET.STATUS
AND    1
JR     NZ,WAIT.VDP

```

```

XOR  A
CALL GET.STATUS
RET

END

```

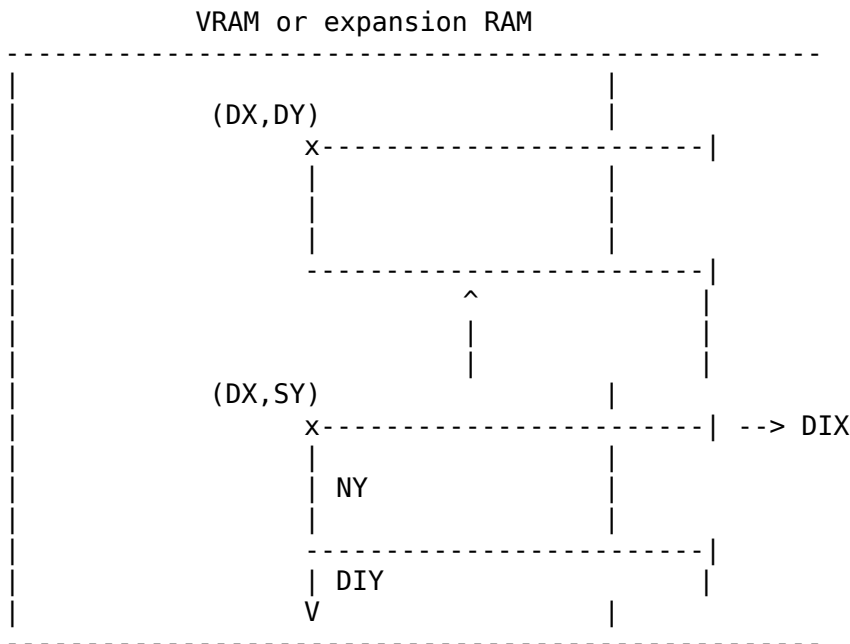
---

### 6.5.2 YMMM (high-speed transfer between VRAM in Y direction)

Data from a specified VRAM area is transferred into another area in VRAM. Note that transfers using this command can only be done in the Y direction (see Figure 4.78).

After setting the data as shown in Figure 4.79 in the proper registers, writing command code E0H in R#46 causes the command to be executed. When the CE bit of S#2 is "1", it indicates that the command is currently being executed. List 4.9 shows an example of using YMMM.

Figure 4.78 Actions of YMMM command



MXD: select the destination memory 0 = VRAM, 1 = expansion RAM

SY: source origin Y-coordinate (0 to 1023)

NY: number of dots to be transferred in Y direction (0 to 1023)

DIX: set which to be transferred, to the right end or to the left end of the screen from the source origin 0 = right, 1 = left

DIY: direction of NY from the origin 0 = below, 1 = above

DX: destination origin X-coordinate (0 to 511)\*

DY: destination origin Y-coordinate (0 to 1023)

\* The one low-order bit for GRAPHIC 4 and 6 modes, or two low-order bits for GRAPHIC 5 mode of the DX register are ignored.

Figure 4.79 Register settings of YMMM command

> YMMM register setup

	MSB	7	6	5	4	3	2	1	0	LSB							
R#34		SY7		SY6		SY5		SY4		SY3		SY2		SY1		SY0	
R#35		0		0		0		0		0		0		SY9		SY8	SY --> source origin
R#36		DX7		DX6		DX5		DX4		DX3		DX2		DX1		DX0	
R#37		0		0		0		0		0		0		0		DX8	DX --> destination and source origin
R#38		DY7		DY6		DY5		DY4		DY3		DY2		DY1		DY0	
R#39		0		0		0		0		0		0		DY9		DY8	DY --> destination origin
R#42		NY7		NY6		NY5		NY4		NY3		NY2		NY1		NY0	number of dots to
R#43		0		0		0		0		0		0		NY9		NY8	NY ---> be transferred in Y direction
R#45		0		--		MXD		--		DIY		DIX		--		--	ARG (Argument register)
																	direction (X)
																	+--> direction (Y)
																	+-----> select destination memory

> YMMM command execution

	MSB	7	6	5	4	3	2	1	0	LSB							
R#46		1		1		1		0		--		--		--		--	CMR

List 4.9 Example of YMMM command execution

```

=====
;*****
; List 4.9 YMMM sample
; to use, set L, E, B, C, D(bit 2) and go
; VRAM (B,L)-(*,E) ---> VRAM (B,C)

```

```

;          DIX must be set in D(bit 2)
;*****
;
RDVDP:    EQU    0006H
WRVDP:    EQU    0007H

;----- program start -----

YMMM: DI          ;disable interrupt
      PUSH BC     ;save destination
      CALL WAIT.VDP ;wait end of command

      LD  A,(WRVDP)
      LD  C,A
      INC C        ;C := PORT#1's address
      LD  A,34
      OUT (C),A
      LD  A,17+80H
      OUT (C),A    ;R#17 := 34

      INC C
      INC C        ;C := PORT#3's address
      XOR A
      OUT (C),L    ;SY
      OUT (C),A

      LD  A,L      ;make NY and DIY
      SUB A
      LD  E,00001000B
      JP  NC,YMMM1
      LD  E,00000000B
      NEG

YMMM1: LD  L,A      ;L := NY , D := DIY

      LD  A,D
      OR  E

      POP DE       ;restore DX,DY
      PUSH AF      ;save DIX,DIY
      XOR A
      OUT (C),D    ;DX
      OUT (C),A
      OUT (C),E    ;DY
      OUT (C),A
      OUT (C),A    ;dummy
      OUT (C),A    ;dummy
      OUT (C),L    ;NY
      OUT (C),A
      OUT (C),A    ;dummy
      POP AF
      OUT (C),A    ;DIX and DIY
      LD  A,11100000B ;YMMM command
      OUT (C),A

      EI
      RET

```

```
GET.STATUS:
```

```

PUSH BC
LD BC, (WRVDP)
INC C
OUT (C), A
LD A, 8FH
OUT (C), A
LD BC, (RDVDP)
INC C
IN A, (C)
POP BC
RET

```

```

WAIT.VDP:
LD A, 2
CALL GET.STATUS
AND 1
JP NZ, WAIT.VDP
XOR A
CALL GET.STATUS
RET

END

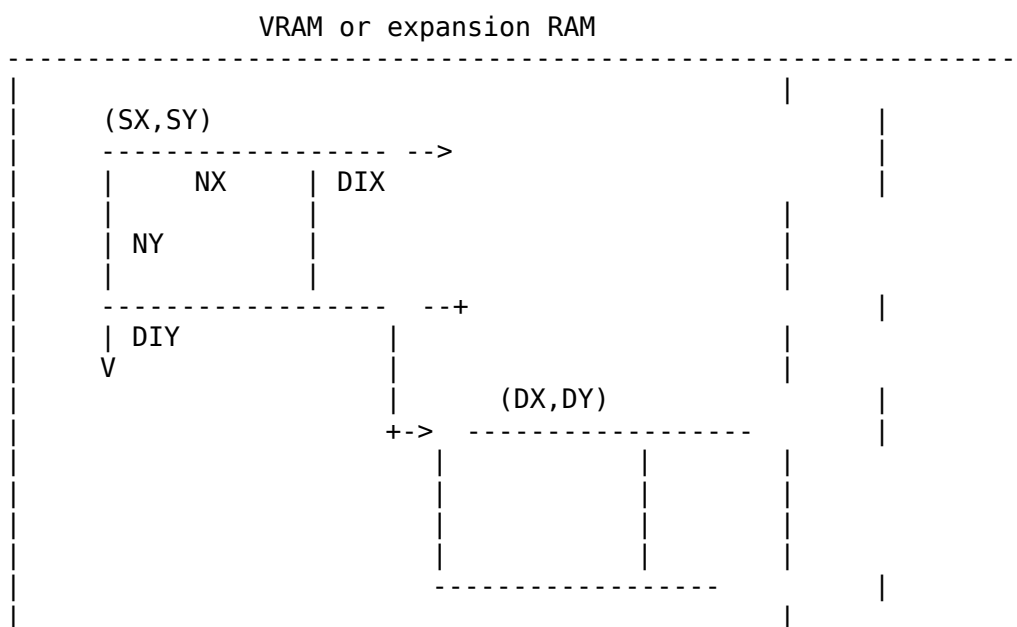
```

### 6.5.3 HMMM (high-speed transfer between VRAM)

Data of specified VRAM area is transferred into another area in VRAM (see Figure 4.80).

After setting the parameters as shown in Figure 4.81, writing D0H in R#46 causes the command to be executed. While the command is being executed, CE bit of S#2 is "1". List 4.10 shows an example of using HMMM.

Figure 4.80 Actions of HMMM command



```

-----
MXS: select the source memory      0 = VRAM, 1 = expansion RAM
MXD: select the destination memory 0 = VRAM, 1 = expansion RAM

SX:  source origin X-coordinate (0 to 511)*
SY:  source origin Y-coordinate (0 to 1023)

NX:  number of dots to be transferred in X direction (0 to 511)*
NY:  number of dots to be transferred in Y direction (0 to 1023)

DIX: direction of NX from the origin    0 = right, 1 = left
DIY: direction of NY from the origin    0 = below, 1 = above

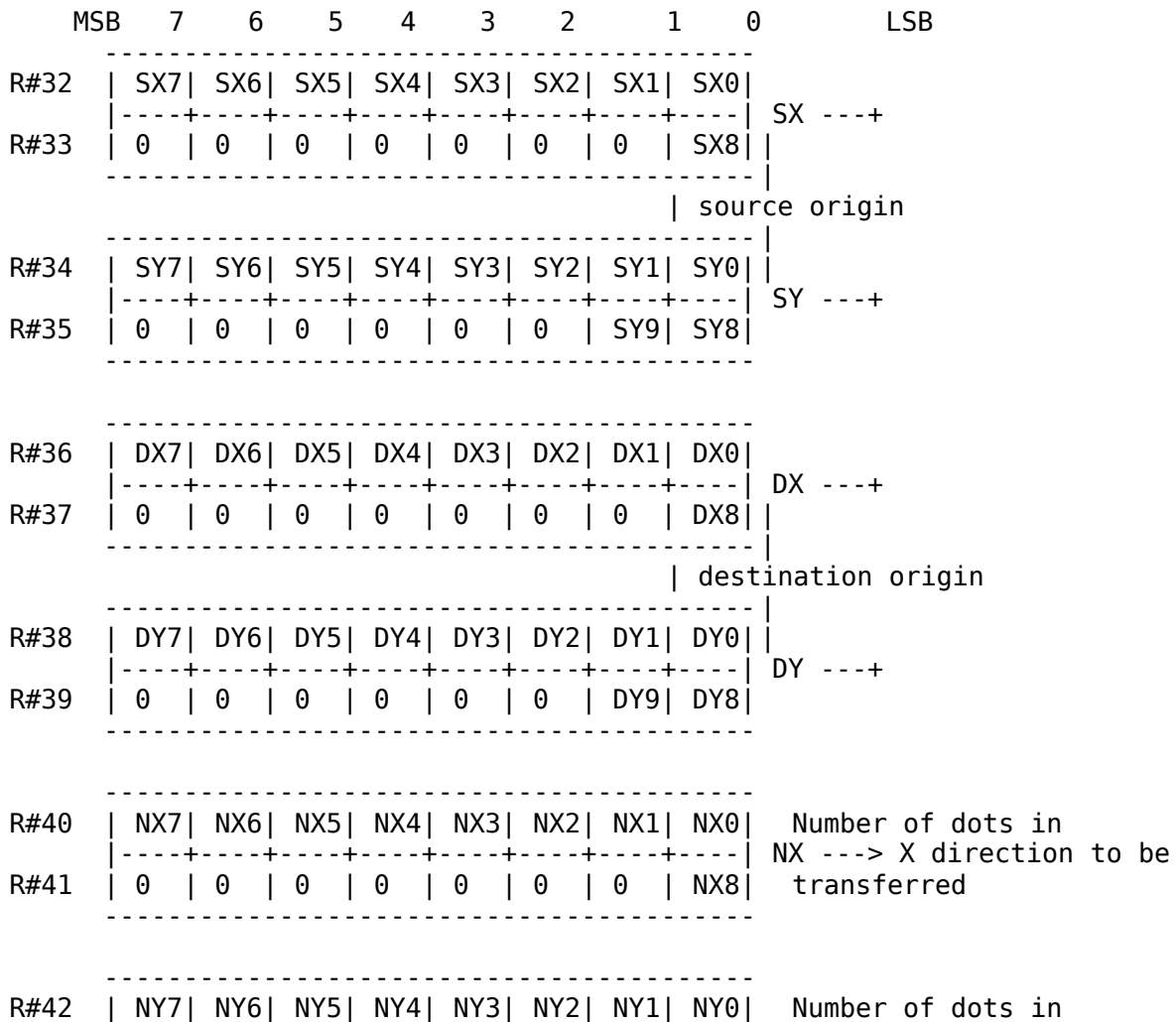
DX:  destination origin X-coordinate (0 to 511)*
DY:  destination origin Y-coordinate (0 to 1023)

```

\* The one low-order bit for GRAPHIC 4 and 6 modes, or two low-order bits for GRAPHIC 5 mode of the SX, DX, and NX register are ignored.

Figure 4.81 Register settings of HMMM command

> HMMM register setup







```

LD    A,H                ;make NX and DIX
SUB   A
LD    D,00000100B
JP    NC,HMMM1
LD    D,00000000B
NEG
HMMM1: LD    H,A                ;H := NX , D := DIX

LD    A,L                ;make NY and DIY
SUB   A
LD    E,00001000B
JP    NC,HMMM2
LD    E,00000000B
NEG
HMMM2: LD    L,A                ;L := NY , E := DIY

LD    A,D
OR    E
POP   DE                ;restore DX,DY
PUSH  AF                ;save DIX,DIY
XOR   A
OUT   (C),D            ;DX
OUT   (C),A
OUT   (C),E            ;DY
OUT   (C),A
OUT   (C),H            ;NX
OUT   (C),A
OUT   (C),L            ;NY
OUT   (C),A            ;dummy
POP   AF
OUT   (C),A            ;DIX and DIY

LD    A,11010000B      ;HMMM command
OUT   (C),A

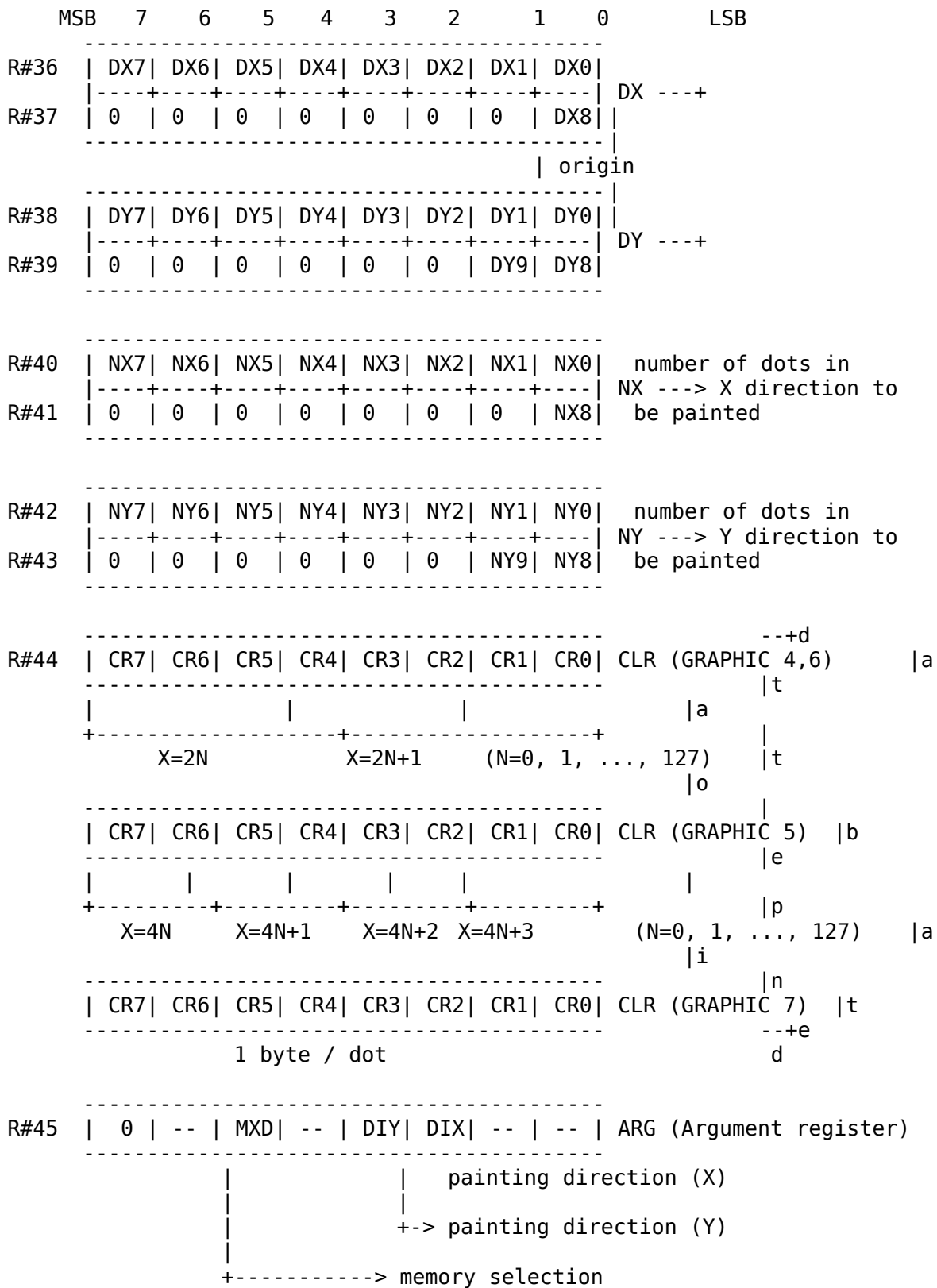
EI
RET

GET.STATUS:
PUSH  BC
LD    BC,(WRVDP)
INC   C
OUT   (C),A
LD    A,8FH
OUT   (C),A
LD    BC,(RDVDP)
INC   C
IN    A,(C)
POP   BC
RET

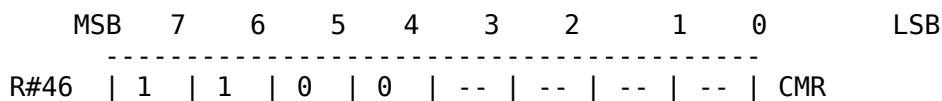
WAIT.VDP:
LD    A,2
CALL  GET.STATUS
AND   1
JP    NZ,WAIT.VDP
XOR   A

```





> HMMV command execution



-----  
 List 4.11 Example of HMMV command execution  
 =====

```

;*****
; List 4.11  HMMV sample
;           to use, set H, L, D, E, B and go
;           B ---> VRAM (H,L)-(D,E) fill
;*****
;
RDVDP:      EQU   0006H
WRVDP:      EQU   0007H

;----- program start -----

HMMV: DI          ;disable interrupt
      CALL  WAIT.VDP      ;wait end of command

      LD    A,(WRVDP)
      LD    C,A
      INC   C              ;C := PORT#1's address
      LD    A,36
      OUT   (C),A
      LD    A,80H+17
      OUT   (C),A          ;R#17 := 36

      INC   C
      INC   C              ;C := PORT#3's address
      XOR   A
      OUT   (C),H          ;DX
      OUT   (C),A
      OUT   (C),L          ;DY
      OUT   (C),A

      LD    A,H            ;make NX and DIX
      SUB   A
      LD    D,00000100B
      JP    NC,HMMV1
      LD    D,00000000B
      NEG

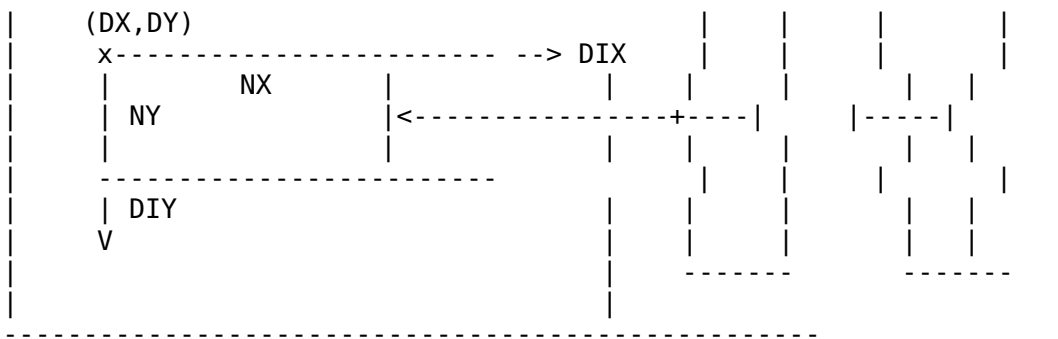
HMMV1:  LD    H,A          ;H := NX

      LD    A,L            ;make NY and DIY
      SUB   A
      LD    E,00001000B
      JP    NC,HMMV2
      LD    E,00000000B
      NEG

HMMV2:  OUT   (C),H
      LD    H,A            ;H := NY

      XOR   A
      OUT   (C),A
      OUT   (C),H
      OUT   (C),A
      OUT   (C),B          ;fill data
  
```





MXD: select destination memory                    0 = VRAM, 1 = expansion RAM

NX: number of dots to be transferred in X direction (0 to 511)

NY: number of dots to be transferred in Y direction (0 to 1023)

DIX: direction of NX from the origin            0 = right, 1 = left

DIY: direction of NY from the origin            0 = below, 1 = above

DX: destination origin X-coordinate (0 to 511)

DY: destination origin Y-coordinate (0 to 1023)

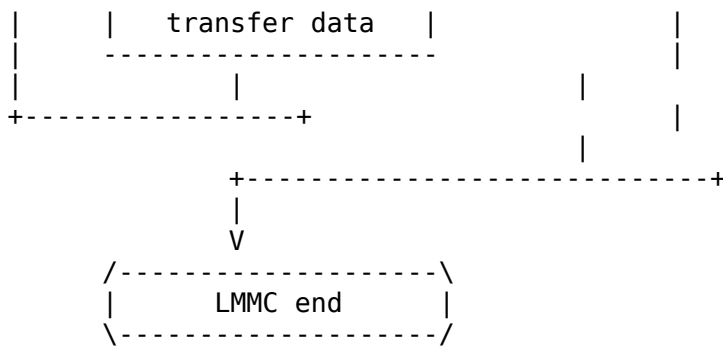
CLR (R#44:Colour register): 1st byte of data to be transferred

Figure 4.85 Register settings of LMMC command

> LMMC register setup

	MSB	7	6	5	4	3	2	1	0	LSB	
R#36	DX7   DX6   DX5   DX4   DX3   DX2   DX1   DX0									DX ----+	
R#37	0   0   0   0   0   0   0   0   DX8										
destination origin											
R#38	DY7   DY6   DY5   DY4   DY3   DY2   DY1   DY0									DY ----+	
R#39	0   0   0   0   0   0   0   DY9   DY8										
-----											
R#40	NX7   NX6   NX5   NX4   NX3   NX2   NX1   NX0									NX ----> X direction to be transferred	
R#41	0   0   0   0   0   0   0   0   NX8										
-----											
R#42	NY7   NY6   NY5   NY4   NY3   NY2   NY1   NY0									NY ----> Y direction to be transferred	
R#43	0   0   0   0   0   0   0   NY9   NY8										
-----											
R#44	--	--	--	--	CR3	CR2	CR1	CR0	CLR (GRAPHIC 4,6)		





List 4.12 Example of LMMC command execution

```

;*****
; List 4.12      LMMC sample
;               to use, set H, L, D, E, IX, A and go
;               RAM (IX) ---> VRAM (H,L)-(D,E) (logi-OP : A)
;*****
;
RDVDP:    EQU    0006H
WRVDP:    EQU    0007H

;----- program start -----

LMMC: DI          ;disable interrupt
      LD    B,A          ;B := LOGICAL OPERATION
      CALL WAIT.VDP      ;wait end of command

      LD    A,(WRVDP)
      LD    C,A
      INC  C              ;C := PORT#1's address
      LD    A,36
      OUT  (C),A
      LD    A,80H+17
      OUT  (C),A          ;R#17 := 36

      INC  C
      INC  C              ;C := PORT#3's address
      XOR  A
      OUT  (C),H          ;DX
      OUT  (C),A
      OUT  (C),L          ;DY
      OUT  (C),A

      LD    A,H          ;make NX and DIX
      SUB  A
      LD    D,00000100B
      JR   NC,LMMC1
      LD    D,00000000B
      NEG

LMMC1:   LD    H,A          ;H := NX , D := DIX

      LD    A,L
      SUB  A
      LD    E,00001000B

```



```

        JR    NC,LMMC2
        LD    E,00000000B
        NEG
LMMC2:  LD    L,A                ;L := NY , E := DIY

        XOR  A
        OUT  (C),H              ;NX
        OUT  (C),A
        OUT  (C),L              ;NY
        OUT  (C),A
        LD   A,(IX+0)
        OUT  (C),A              ;first DATA
        LD   A,D
        OR   E
        OUT  (C),A              ;DIX and DIY

        LD   A,B                ;A := LOGICAL OPERATION
        OR   10110000B         ;LMMC command
        OUT  (C),A

        DEC  C
        DEC  C

LOOP:   LD   A,2
        CALL GET.STATUS
        BIT  0,A                ;check CE bit
        JP  Z,EXIT
        BIT  7,A                ;check TR bit
        JP  Z,LOOP
        INC  IX
        LD   A,(IX+0)
        OUT  (C),A
        JR   LOOP

EXIT:   LD   A,0
        CALL GET.STATUS

        EI
        RET

GET.STATUS:
        PUSH BC
        LD   BC,(WRVDP)
        INC  C
        OUT  (C),A
        LD   A,8FH
        OUT  (C),A
        LD   BC,(RDVDP)
        INC  C
        IN   A,(C)
        POP  BC
        RET

WAIT.VDP:
        LD   A,2
        CALL GET.STATUS
        AND  1
        JR   NZ,WAIT.VDP

```

```

XOR  A
CALL GET.STATUS
RET

END

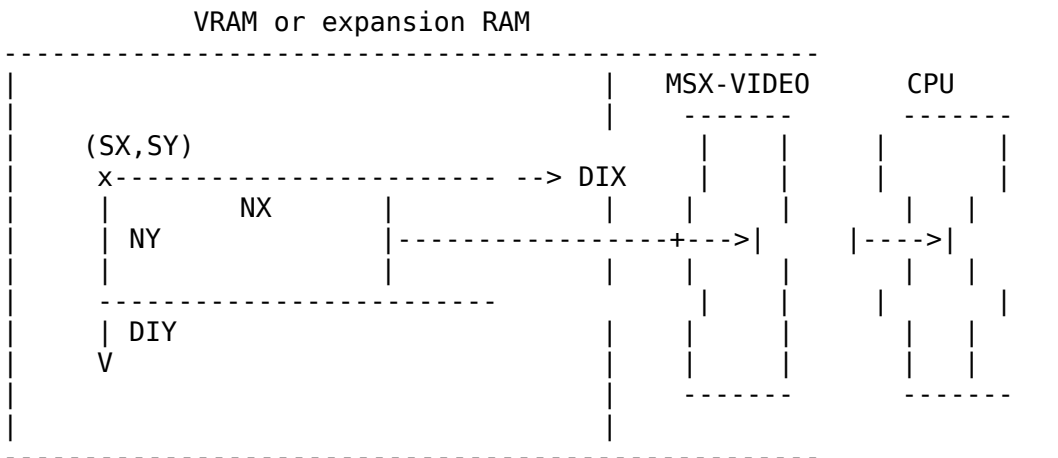
```

### 6.5.6 LMCM (VRAM - CPU logical transfer)

Data is transferred from the specified VRAM area to CPU in dots (see Figure 4.87)

After setting the parameters as shown in Figure 4.88, writing command code A0H in R#46 causes the command to be executed and data to be transferred from MSX-VIDEO. The CPU refers to the TR bit of S#2 and, since data of MSX-VIDEO has been prepared if this bit is "1", the CPU reads data from S#7. When CE bit of S#2 is "0", data comes to the end (see Figure 4.89). List 4.13 shows an example of using LMCM.

Figure 4.87 Action of LMCM command



MXS: select source memory                    0 = VRAM, 1 = expansion RAM

SX: source origin X-coordinate (0 to 511)

SY: source origin Y-coordinate (0 to 1023)

NX: number of dots to be transferred in X direction (0 to 511)

NY: number of dots to be transferred in Y direction (0 to 1023)

DIX: direction of NX from the origin        0 = right, 1 = left

DIY: direction of NY from the origin        0 = below, 1 = above

Figure 4.88 Register settings of LMCM command

> LMCM register setup

```

MSB  7   6   5   4   3   2   1   0           LSB
-----

```

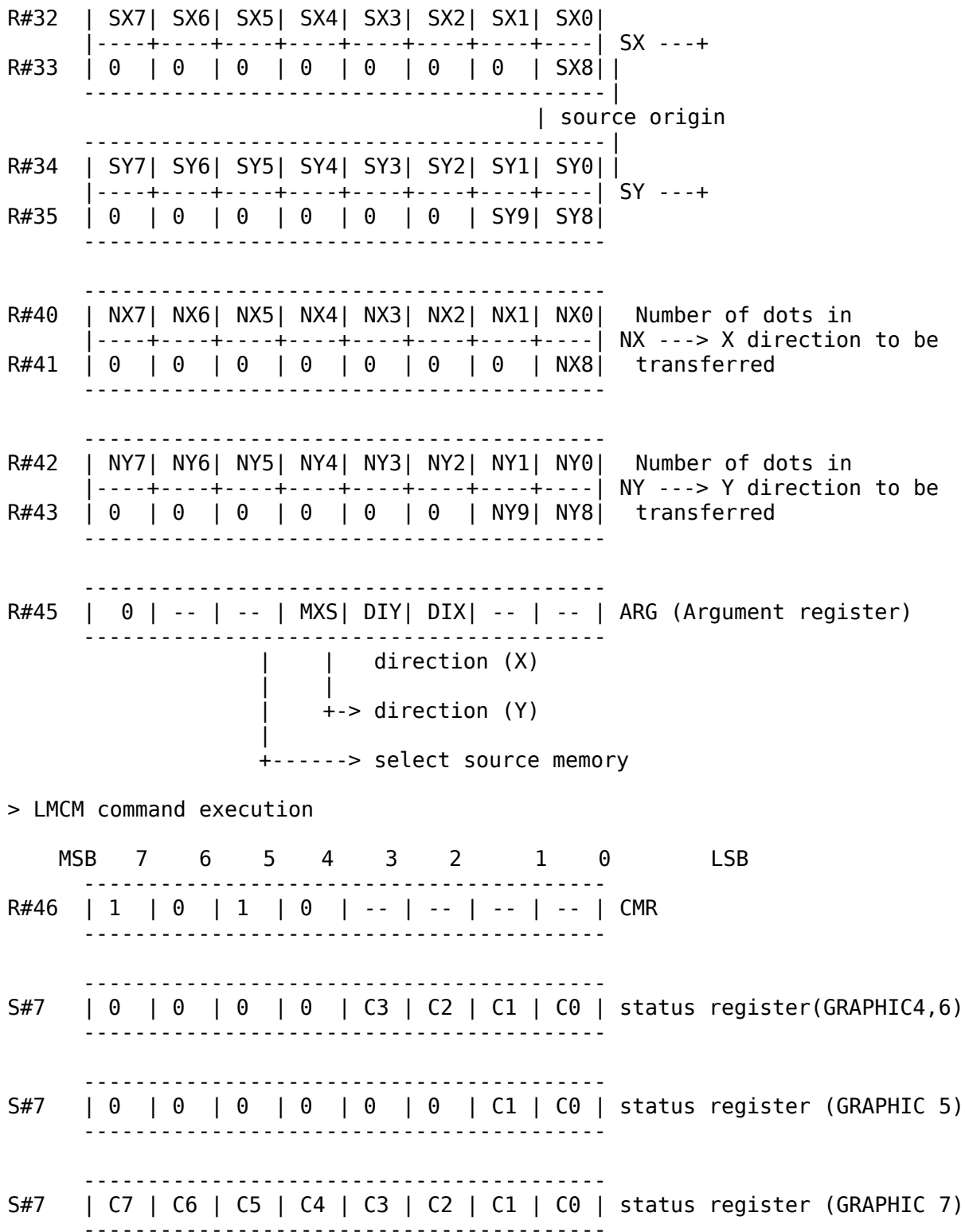
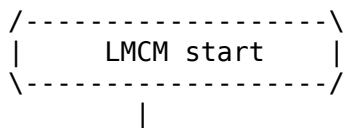
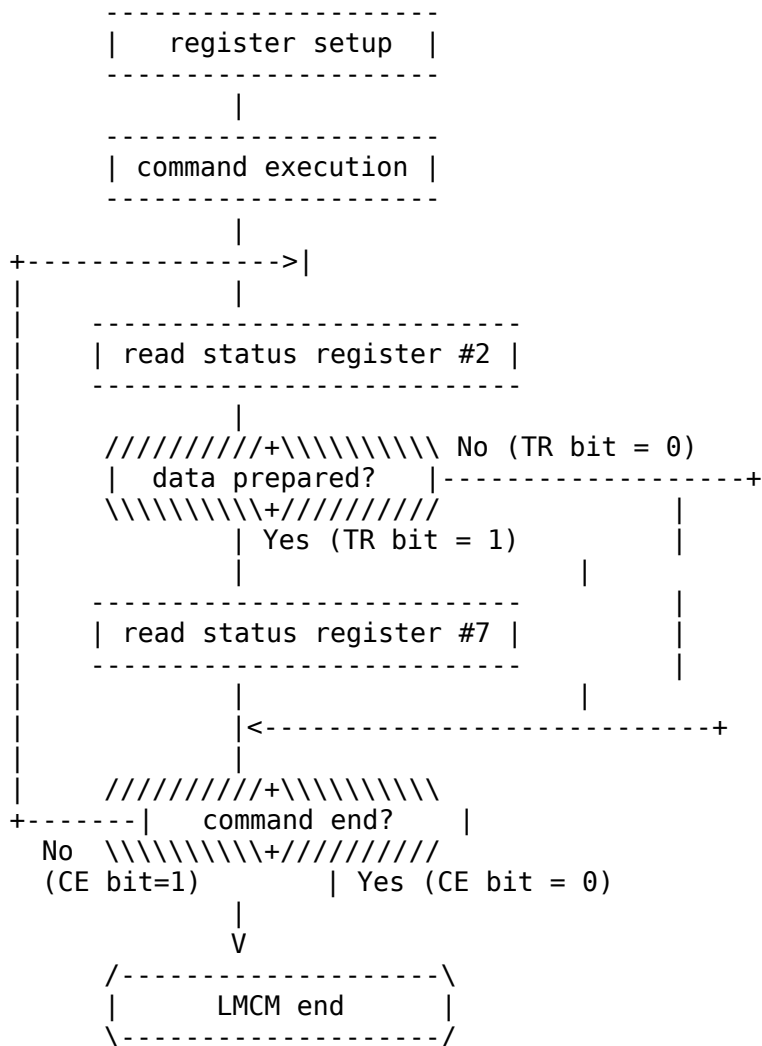


Figure 4.89 LMCM command execution flow chart





\* Note 1: Read status register #7 in "register setup", since TR bit should be reset before the command execution.

\* Note 2: Though last data was set in register #7 and TR bit was 1, the command would end inside of the MSX-VIDEO and CE would be zero.

List 4.13 Example of LCMC command execution

```

=====
;*****
; List 4.13 LCMC sample
; to use, set H, L, D, E, IX, A and go
; VRAM (H,L)-(D,E) ---> RAM (IX)
;*****
;
RDVDP: EQU 0006H
WRVDP: EQU 0007H

;----- program start -----

LMCM: DI ;disable interrupt
LD B,A ;B := LOGICAL OPERATION

```

```

CALL WAIT.VDP          ;wait end of command

LD A,(WRVDP)
LD C,A
INC C                  ;C := PORT#1's address
LD A,32
OUT (C),A
LD A,80H+17
OUT (C),A             ;R#17 := 32
INC C
INC C                  ;C := PORT#3's address
XOR A
OUT (C),H             ;SX
OUT (C),A
OUT (C),L             ;SY
OUT (C),A
OUT (C),A             ;dummy
OUT (C),A             ;dummy
OUT (C),A             ;dummy
OUT (C),A             ;dummy
LD A,H                ;make NX and DIX
SUB A
LD D,00000100B
JR NC,LMCM1
LD D,00000000B
NEG
LMCM1: LD H,A          ;H := NX , D := DIX

LD A,L
SUB A
LD E,00001000B
JR NC,LMCM2
LD E,00000000B
NEG
LMCM2: LD L,A         ;L := NY , E := DIY

XOR A
OUT (C),H             ;NX
OUT (C),A
OUT (C),L             ;NY
OUT (C),A
LD A,(IX+0)
OUT (C),A             ;dummy
LD A,D
OR E
OUT (C),A             ;DIX and DIY
LD A,7
CALL GET.STATUS
LD A,B                ;A := LOGICAL OPERATION
OR 10100000B         ;LMCM command
OUT (C),A
LD A,(RDVDP)
LD C,A                ;C := PORT#1's address
LOOP: LD A,2
CALL GET.STATUS
BIT 0,A                ;check CE bit
JP Z,EXIT
BIT 7,A                ;check TR bit

```

```

        JP    Z,LOOP
        LD    A,7
        CALL  GET.STATUS
        LD    (IX+0),A
        INC  IX
        JR   LOOP

EXIT: LD    A,0
      CALL  GET.STATUS
      EI
      RET

GET.STATUS:
      PUSH  BC
      LD    BC,(WRVDP)
      INC  C
      OUT  (C),A
      LD    A,8FH
      OUT  (C),A
      LD    BC,(RDVDP)
      INC  C
      IN   A,(C)
      POP  BC
      RET

WAIT.VDP:
      LD    A,2
      CALL  GET.STATUS
      AND  1
      JR   NZ,WAIT.VDP
      XOR  A
      CALL  GET.STATUS
      RET

      END

```

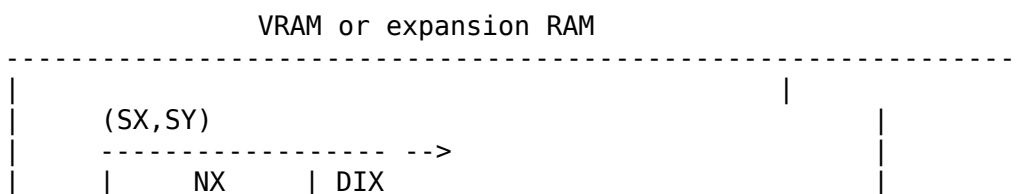
---

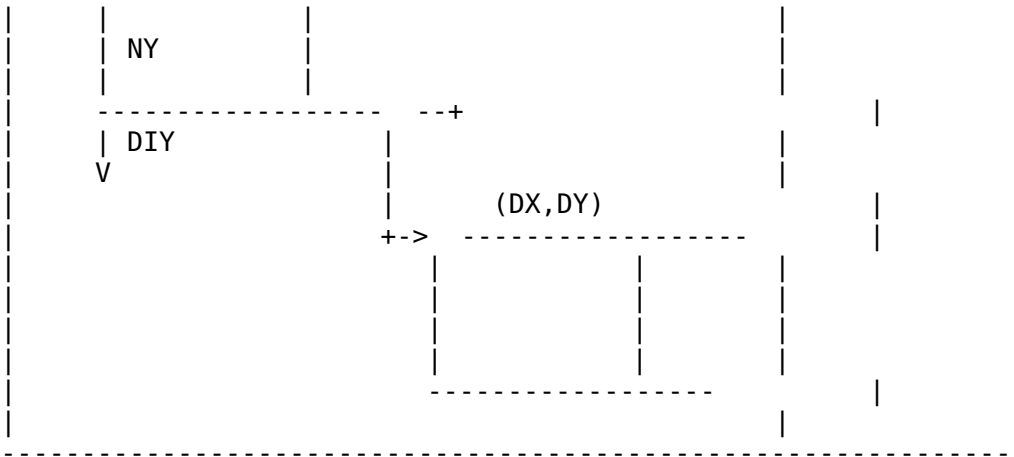
### 6.5.7. LMMM (VRAM->VRAM logical transfer)

Data of the specified VRAM area is transferred into another VRAM area in dots (see figure 4.9)

After setting the parameters as shown in Figure 4.91, writing command code 9XH (X means a logical operation) in R#46 causes the command to be executed. While the CE bit of S#2 is "1", the command is being executed. List 4.14 shows an example of using LMMM.

Figure 4.90 Actions of LMMM command





MXS: select the source memory      0 = VRAM, 1 = expansion RAM  
 MXD: select the destination memory 0 = VRAM, 1 = expansion RAM  
  
 SX: source origin X-coordinate (0 to 511)  
 SY: source origin Y-coordinate (0 to 1023)  
  
 NX: number of dots to be transferred in X direction (0 to 511)  
 NY: number of dots to be transferred in Y direction (0 to 1023)  
  
 DIX: direction of NX from the origin      0 = right, 1 = left  
 DIY: direction of NY from the origin      0 = below, 1 = above  
  
 DX: destination origin X-coordinate (0 to 511)  
 DY: destination origin Y-coordinate (0 to 1023)

Figure 4.91 Register settings of LMMM command

> LMMM register setup

	MSB	7	6	5	4	3	2	1	0	LSB
R#32	SX7   SX6   SX5   SX4   SX3   SX2   SX1   SX0								SX ----+	
R#33	0	0	0	0	0	0	0	0	SX8	
source origin										
R#34	SY7   SY6   SY5   SY4   SY3   SY2   SY1   SY0								SY ----+	
R#35	0	0	0	0	0	0	0	SY9	SY8	
destination origin										
R#36	DX7   DX6   DX5   DX4   DX3   DX2   DX1   DX0								DX ----+	
R#37	0	0	0	0	0	0	0	DX8		
destination origin										
R#38	DY7   DY6   DY5   DY4   DY3   DY2   DY1   DY0								DY ----+	





```

INC    C                ;C := PORT#1's address
LD     A,32
OUT    (C),A
LD     A,80H+17
OUT    (C),A            ;R#17 := 32

INC    C
INC    C                ;C := PORT#3's address
XOR    A
OUT    (C),H            ;SX
OUT    (C),A
OUT    (C),L            ;SY
OUT    (C),A

LD     A,H              ;make NX and DIX
SUB    A
LD     D,00000100B
JP     NC,LMMM1
LD     D,00000000B
NEG
LMMM1: LD     H,A                ;H := NX , D := DIX

LD     A,L              ;make NY and DIY
SUB    A
LD     E,00001000B
JP     NC,LMMM2
LD     E,00000000B
NEG
LMMM2: LD     L,A                ;L := NY , E := DIY

LD     A,D
OR     E
POP    DE                ;restore DX,DY
PUSH   AF                ;save DIX,DIY
XOR    A
OUT    (C),D            ;DX
OUT    (C),A
OUT    (C),E            ;DY
OUT    (C),A
OUT    (C),H            ;NX
OUT    (C),A
OUT    (C),L            ;NY
OUT    (C),A            ;dummy
POP    AF
OUT    (C),A            ;DIX and DIY

POP    AF                ;A := LOGICAL OPERATION
OR     10010000B        ;LMMM command
OUT    (C),A

EI
RET

GET.STATUS:
.      PUSH   BC
LD     BC,(WRVDP)
INC    C

```

```

OUT   (C),A
LD    A,8FH
OUT   (C),A
LD    BC,(RDVDP)
INC   C
IN    A,(C)
POP   BC
RET

```

```

WAIT.VDP:
LD    A,2
CALL  GET.STATUS
AND   1
JP    NZ,WAIT.VDP
XOR   A
CALL  GET.STATUS
RET

END

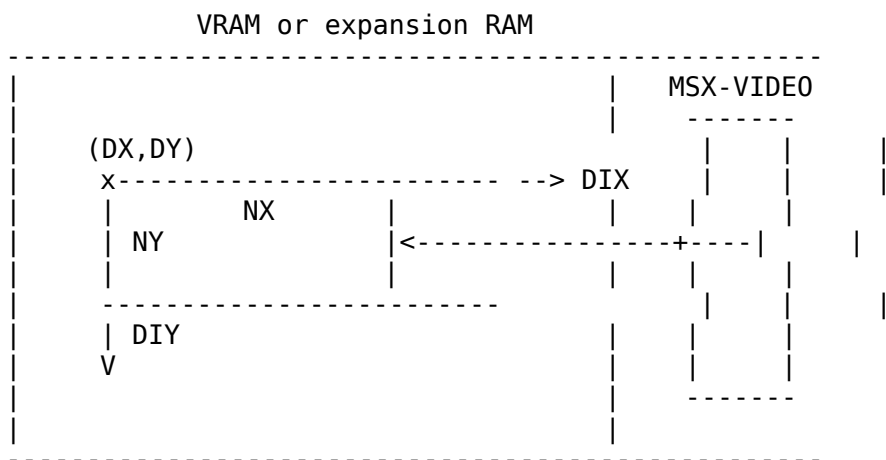
```

### 6.5.8 LMMV (VRAM logical paint)

The specified VRAM area is painted by the colour code in dots (see Figure 4.92). Logical operations between data in VRAM and the specified data are allowed.

After setting the parameters as shown in Figure 4.93, writing command code 8Xh (X means a logical operation) in R#46 causes the command to be executed. While the CE bit of S#2 is "1", the command is being executed. List 4.15 shows an example of using LMMV.

Figure 4.92 Actions of LMMV command



MXD: select memory                      0 = VRAM, 1 = expansion RAM

NX: number of dots to be painted in X direction (0 to 511)

NY: number of dots to be painted in Y direction (0 to 1023)

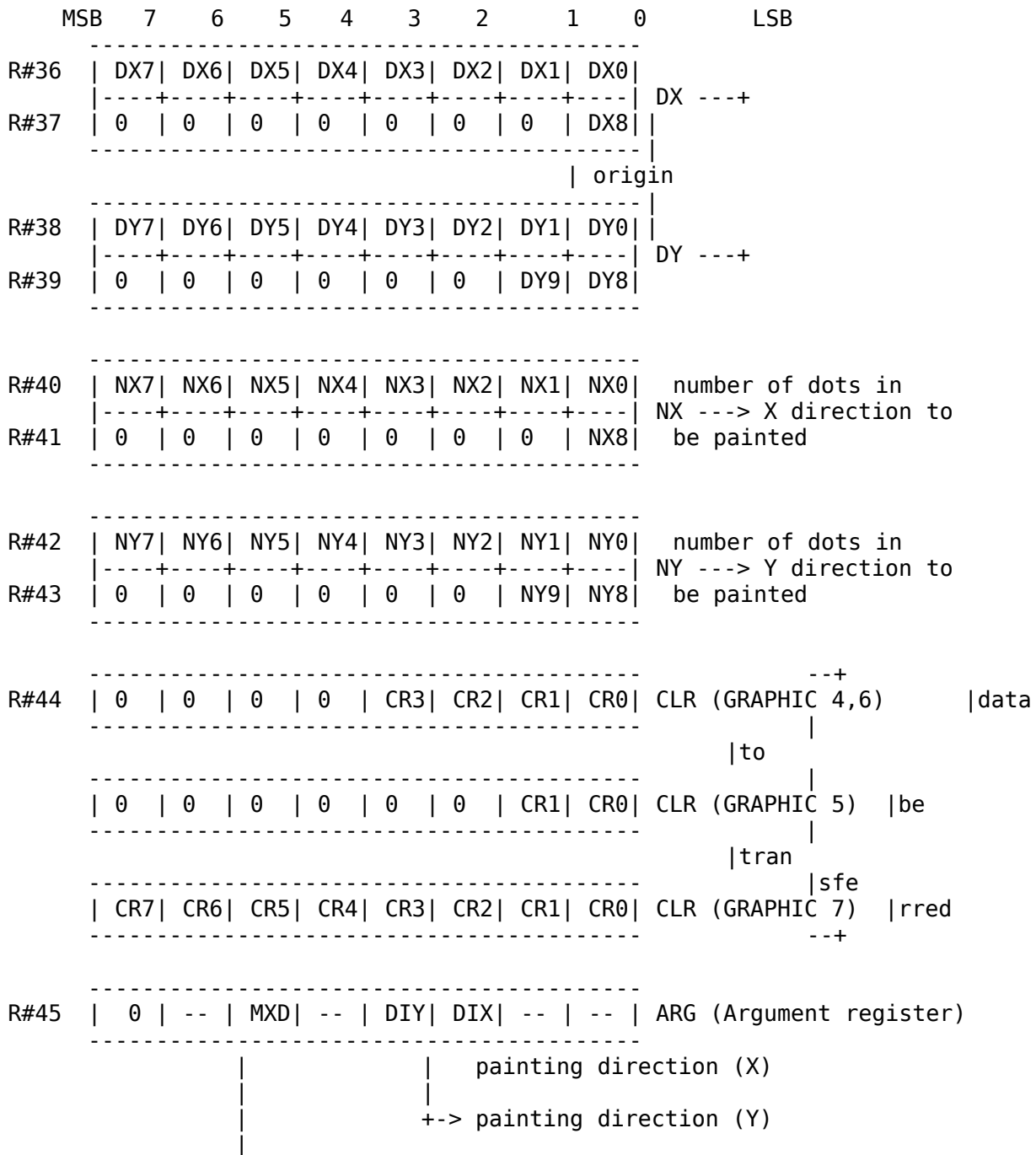
DIX: direction of NX from the origin      0 = right, 1 = left  
 DIY: direction of NY from the origin      0 = below, 1 = above

DX: origin X-coordinate (0 to 511)  
 DY: origin Y-coordinate (0 to 1023)

CLR (R#44:Colour register): Painted data

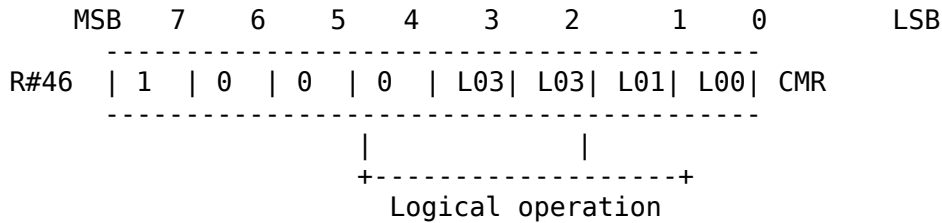
Figure 4.93 Register settings of LMMV command

> LMMV register setup



+-----> memory selection

> LMMV command execution



List 4.15 Example of LMMV command execution

```
=====
;*****
; List 4.15 LMMV sample
; to use, set H, L, D, E, B, A and go
; data B ---> fill VRAM (H,L)-(D,E) (logi-op : A)
;*****
;
RDVDP: EQU 0006H
WRVDP: EQU 0007H

;----- program start -----

LMMV: DI ;disable interrupt
PUSH AF ;save LOGICAL OPERATION
PUSH BC ;save FILL DATA
CALL WAIT.VDP ;wait end of command

LD A,(WRVDP)
LD C,A
INC C ;C := PORT#1's address
LD A,36
OUT (C),A
LD A,80H+17
OUT (C),A ;R#17 := 36

INC C
INC C ;C := PORT#3's address
XOR A
OUT (C),H ;DX
OUT (C),A
OUT (C),L ;DY
OUT (C),A

LD A,H ;make NX and DIX
SUB A
LD D,00000100B
JP NC,LMMV1
LD D,00000000B
NEG

LMMV1: LD H,A ;H := NX , D := DIX

LD A,L ;make NY and DIY
SUB A
```

```

LD    E,00001000B
JP    NC,LMMV2
LD    E,00000000B
NEG
LMMV2: LD    L,A                ;L := NY , E := DIY

XOR   A
OUT   (C),H                    ;NX
OUT   (C),A
OUT   (C),L                    ;NY
OUT   (C),A
POP   AF
OUT   (C),A                    ;FILL DATA
LD    A,D
OR    E
OUT   (C),A                    ;DIX and DIY

POP   AF                        ;restore LOGICAL OPERATION
OR    A,10000000B              ;LMMV command
OUT   (C),A

EI
RET

```

```

GET.STATUS:
PUSH  BC
LD    BC,(WRVDP)
INC   C
OUT   (C),A
LD    A,8FH
OUT   (C),A
LD    BC,(RDVDP)
INC   C
IN    A,(C)
POP   BC
RET

```

```

WAIT.VDP:
LD    A,2
CALL  GET.STATUS
AND   1
JP    NZ,WAIT.VDP
XOR   A
CALL  GET.STATUS
RET

END

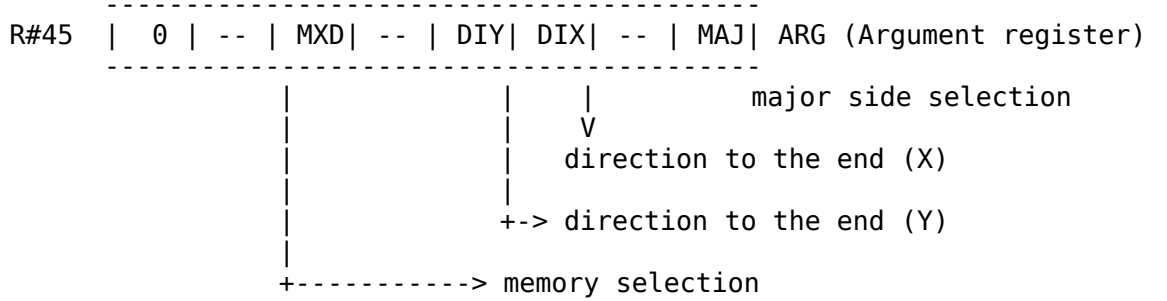
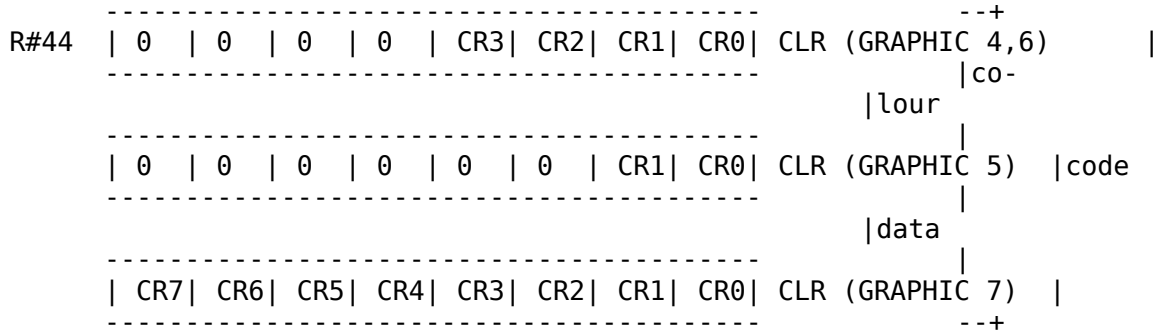
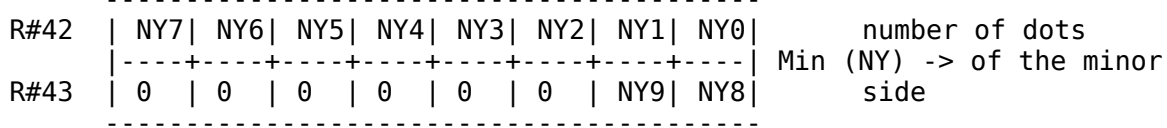
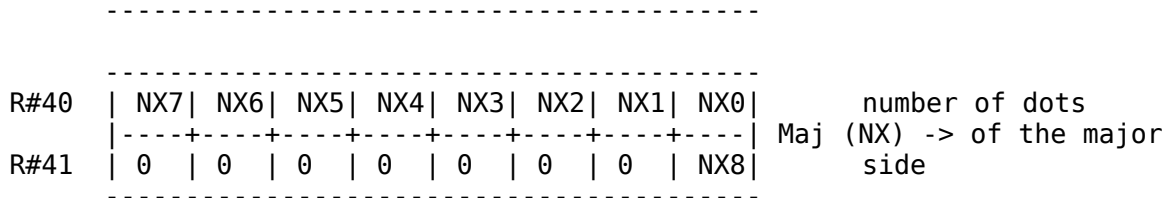
```

=====

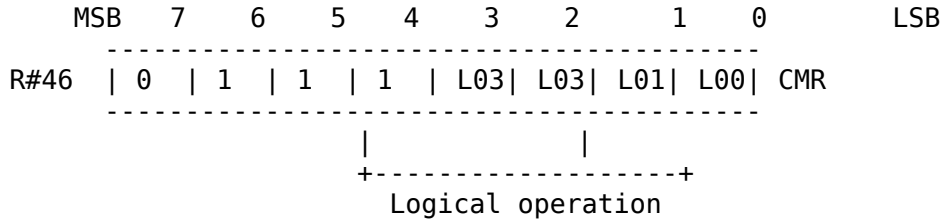
### 6.5.9 LINE (drawing a line)

Lines can be drawn between any coordinates in VRAM. The parameters to be specified include the (X,Y) coordinates of the starting point and the X and Y lengths in units to the ending point (see Figure 4.94). Logical operations between data in VRAM and the specified data are allowed.





> LINE command execution



List 4.16 Example of LINE command execution

```

=====
;*****
; List 4.16 LINE sample
; to use, set H, L, D, E, B, A and go
; draw LINE (H,L)-(D,E) with color B, log-op A
;*****
;
RDVDP: EQU 0006H

```

WRVDP: EQU 0007H

;----- program start -----

```
LINE: DI ;disable interrupt
      PUSH AF ;save LOGICAL OPERATION
      PUSH BC ;save COLOR
      CALL WAIT.VDP ;wait end of command
      LD A,(WRVDP)
      LD C,A
      INC C ;C := PORT#1's address
      LD A,36
      OUT (C),A
      LD A,80H+17
      OUT (C),A ;R#17 := 36

      INC C
      INC C ;C := PORT#3's address
      XOR A
      OUT (C),H ;DX
      OUT (C),A
      OUT (C),L ;DY
      OUT (C),A

      LD A,H ;make DX and DIX
      SUB D
      LD D,00000100B
      JP NC,LINE1
      LD D,00000000B
      NEG
LINE1: LD H,A ;H := DX , D := DIX

      LD A,L ;make DY and DIY
      SUB E
      LD E,00001000B
      JP NC,LINE2
      LD E,00000000B
      NEG
LINE2: LD L,A ;L := DY , E := DIY

      CP H ;make Maj and Min
      JP C,LINE3
      XOR A
      OUT (C),L ;long side
      OUT (C),A
      OUT (C),H ;short side
      OUT (C),A
      LD A,00000001B ;MAJ := 1
      JP LINE4

LINE3: XOR A
      OUT (C),H ;NX
      OUT (C),A
      OUT (C),L ;NY
      OUT (C),A
      LD A,00000000B ;MAJ := 0

LINE4: OR D
```



```

OR    E            ;A := DIX , DIY , MAJ
POP   HL          ;H := COLOR
OUT   (C),H
OUT   (C),A
POP   AF          ;A := LOGICAL OPERATION
OR    01110000B
OUT   (C),A
LD    A,8FH
OUT   (C),A
EI
RET

```

```

GET.STATUS:
PUSH  BC
LD    BC,(WRVDP)
INC   C
OUT   (C),A
LD    A,8FH
OUT   (C),A
LD    BC,(RDVDP)
INC   C
IN    A,(C)
POP   BC
RET

```

```

WAIT.VDP:
LD    A,2
CALL  GET.STATUS
AND   1
JP    NZ,WAIT.VDP
XOR   A
CALL  GET.STATUS
RET

```

```

END

```

---

#### 6.5.10 SRCH (colour code search)

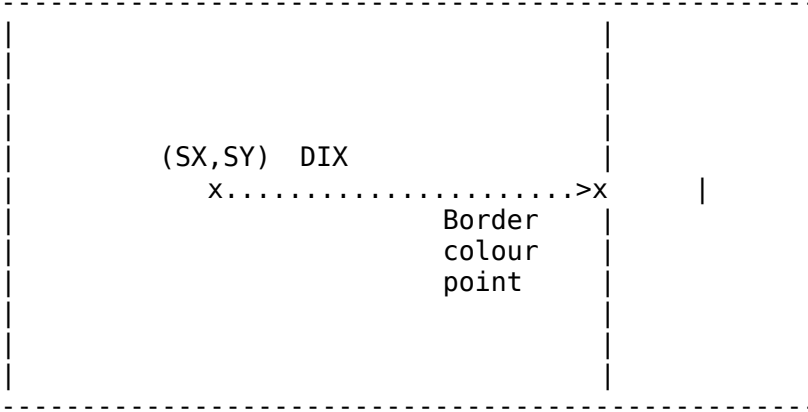
SRCH searches for the existence of the specified colour from any coordinate on VRAM to the right or the left (see figure 4.96). This is very useful for paint routines.

After setting the parameters as shown in Figure 4.97, writing 60H in R#46 causes the command to be executed. The command terminates when the objective colour is found or when it cannot be found after searching for it to the screen edge. While the CE bit of S#2 is "1", the command is being executed (see Figure 4.98).

After the command ends, the objective colour code is stored in S#8 and S#9. List 4.17 shows an example of using SRCH.

Figure 4.96 Actions of SRCH command

VRAM or expansion RAM



MXD: memory selection for the search      0 = VRAM, 1 = expansion RAM

SX: search origin X-coordinate (0 to 511)  
 SY: search origin Y-coordinate (0 to 1023)

DIX: direction for the search from the origin    0 = right, 1 = left

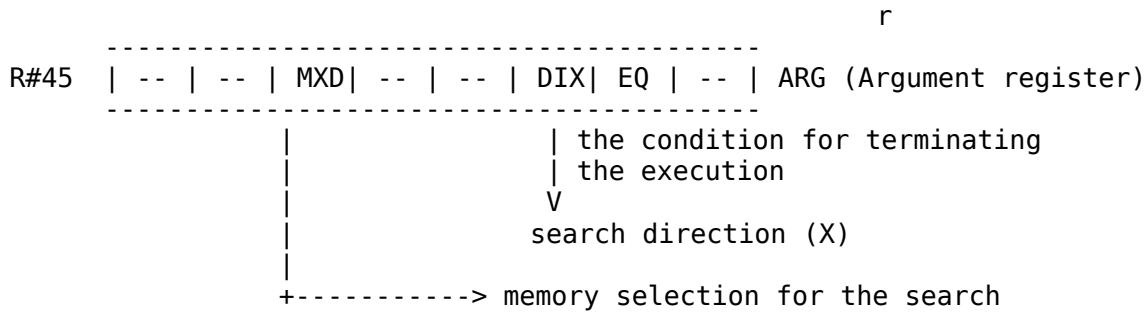
EQ: 0 = ends the execution when the border colour is found  
 1 = ends the execution when the colour is found other than the border colour

CLR (R#44:Colour register): border colour

Figure 4.97 Register settings of SRCH command

> SRCH register setup

	MSB	7	6	5	4	3	2	1	0	LSB	
R#32	SX7   SX6   SX5   SX4   SX3   SX2   SX1   SX0										
R#33	0   0   0   0   0   0   0   0								SX8		SX ----+
search origin											
R#34	SY7   SY6   SY5   SY4   SY3   SY2   SY1   SY0										
R#35	0   0   0   0   0   0   0   0								SY9   SY8		SY ----+
R#44	0   0   0   0				CR3   CR2   CR1   CR0				CLR (GRAPHIC 4,6)		r
									CLR (GRAPHIC 5)		
									CLR (GRAPHIC 7)		o



> SRCH command execution

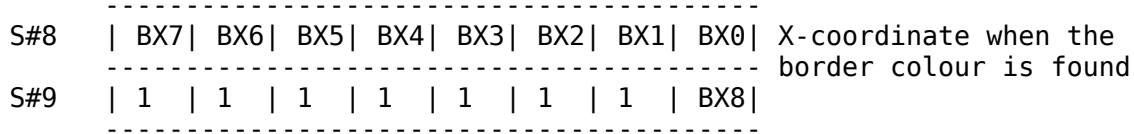
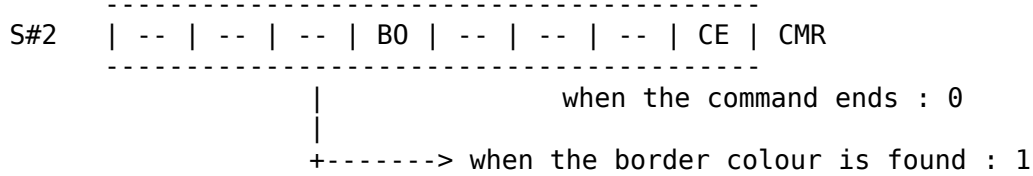
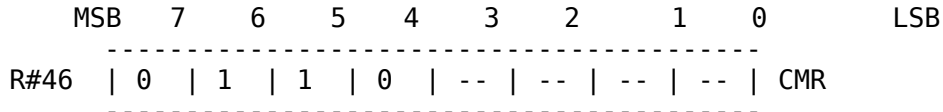
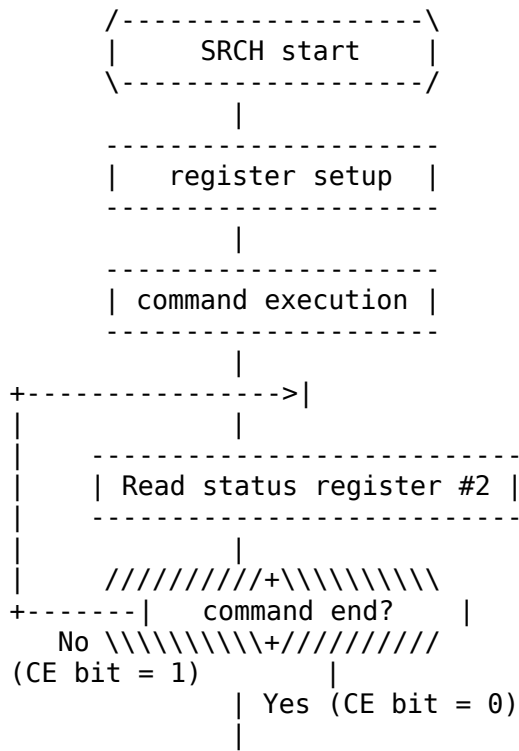


Figure 4.98 SRCH command execution flowchart





```

OUT   (C),A           ;R#35 := 0
LD    A,44+80H
OUT   (C),E
OUT   (C),A           ;R#44 := E
INC   A
LD    E,A
POP   AF              ;A := ARG
OUT   (C),A
OUT   (C),E           ;R#45 := A

LD    A,01100000B
OUT   (C),A
INC   E
OUT   (C),E           ;R#46 := SRCH command

LOOP: LD    A,2
      CALL GET.STATUS
      BIT   0,A
      JP   NZ,LOOP
      LD   E,A
      LD   A,8
      CALL GET.STATUS
      LD   D,A
      LD   A,9
      CALL GET.STATUS
      LD   A,D
      BIT   4,E

      EI
      RET

GET.STATUS:
      PUSH BC
      LD   BC,(WRVDP)
      INC  C
      OUT  (C),A
      LD   A,8FH
      OUT  (C),A
      LD   BC,(RDVDP)
      INC  C
      IN   A,(C)
      POP  BC
      RET

WAIT.VDP:
      LD   A,2
      CALL GET.STATUS
      AND  1
      JP   NZ,WAIT.VDP
      XOR  A
      CALL GET.STATUS
      RET

      END

```

---

List 4.18 Simple PAINT routine using SRCH and LINE

```

;*****
; List 4.18  SRCH and LINE sample
;          search color to right and left,
;          then draw line between the two points
;*****
;
;          EXTRN SRCH
;          EXTRN LINE

Y      EQU   0A800H
X      EQU   0A801H
COL    EQU   0A802H
ARG    EQU   0A803H
PCOL   EQU   0A804H

;----- program start -----

MAIN: LD    (STK),SP
      LD    SP,AREA
      LD    HL,(Y)
      LD    A,(COL)
      LD    E,A
      LD    A,(ARG)
      PUSH HL
      PUSH DE
      SET  2,A
      CALL SRCH
      POP  DE
      POP  HL
      JP   NZ,S1
      LD   A,(X)
      DEC  A
S1:   INC  A
      PUSH AF
      LD   A,(ARG)
      RES 2,A
      CALL SRCH
      JP   NZ,S2
      LD   A,(X)
      INC  A
S2:   DEC  A
      LD   D,A
      POP  AF
      LD   H,A
      LD   A,(Y)
      LD   L,A
      LD   E,A
      LD   A,(PCOL)
      LD   B,A
      LD   A,0           ;PSET
      CALL LINE
      LD   SP,(STK)
      RET

;----- work area -----

```

```
STK: DS 2
      DS 200
AREA: $

      END
```

=====  
List 4.19 Example of the use of simple PAINT routine  
=====

```
1000 '*****
1010 ' list 4.19 SRCH and LINE sample
1020 ' Operate cursor while holding down the space bar.
1030 '*****
1040 '
1050 SCREEN 5
1060 FOR I=0 TO 50:LINE -(RND(1)*255,RND(1)*211),15:NEXT
1070 I=&HA000 :DEF USR=I
1080 READ A$
1090 IF A$="END" THEN 1130
1100 POKE I,VAL("&H"+A$):I=I+1
1110 READ A$
1120 GOTO 1090
1130 X=128:Y=100:COL=15:PCOL=2:ARG=0
1140 CURS=0
1150 A=STICK(0)
1160 CURS=(CURS+1) AND 1
1170 LINE (X-5,I)-(X+5,I),15,,XOR
1180 LINE (X,Y-5)-(X,Y+5),15,,XOR
1190 IF CURS=1 THEN 1290
1200 IF A=1 THEN Y=Y-1
1210 IF A=2 THEN Y=Y-1:X=X+1
1220 IF A=3 THEN X=X+1
1230 IF A=4 THEN X=X+1:Y=Y+1
1240 IF A=5 THEN Y=Y+1
1250 IF A=6 THEN Y=Y+1:X=X-1
1260 IF A=7 THEN X=X-1
1270 IF A=8 THEN X=X-1:Y=Y-1
1280 IF STRIG(9) THEN GOSUB 1300
1290 GOTO 1150
1300 POKE &HA800,Y
1310 POKE &HA801,X
1320 POKE &HA802,COL
1330 POKE &HA803,ARG
1340 POKE &HA804,PCOL
1350 A=USR(0)
1360 RETURN
1370 DATA ED,73,80,A8,31,4A,A9,2A,00,A8,3A,02
1380 DATA A8,5F,3A,03,A8,E5,D5,CB,D7,CD,AD
1390 DATA A0,D1,E1,C2,21,A0,3A,01,A8
1400 DATA 3D,3C,F5,3A,03,A8,CB,97,CD,AD,A0,C2
1410 DATA 32,A0,3A,01,AB,3C,3D,57,F1,67,3A
1420 DATA 00,A8,6F,5F,3A,04,A8,47,3E
1430 DATA 00,CD,49,A0,ED,7B,80,A8,C9,F3,F5,CD
1440 DATA 0D,A1,C5,3A,06,00,4F,0C,3E,24,ED
```

```

1450 DATA 79,3E,91,ED,79,0C,0C,AF,ED
1460 DATA 61,ED,79,ED,69,ED,79,7C,92,16,04,D2
1470 DATA 72,A0,16,00,ED,44,67,7D,93,1E,08
1480 DATA D2,7E,A0,1E,00,ED,44,BC,DA
1490 DATA 90,A0,ED,79,AF,ED,79,ED,61,ED,79,26
1500 DATA 01,C3,9C,A0,ED,61,67,AF,ED,79,ED
1510 DATA 61,ED,79,26,00,7C,B2,B3,E1
1520 DATA ED,61,ED,79,F1,E6,0F,F6,70,ED,79,FB
1530 DATA C9,F5,F3,CD,0D,A1,ED,4B,06,00,0C
1540 DATA 3E,A0,16,00,ED,61,ED,79,3C
1550 DATA ED,51,ED,79,3C,ED,69,ED,79,3C,ED,51
1560 DATA ED,79,3E,AC,ED,59,ED,79,3C,5F,F1
1570 DATA ED,79,ED,59,3E,60,ED,79,1C
1580 DATA ED,59,3E,02,CD,FD,A0,CB,47,C2,E2,A0
1590 DATA 5F,3E,08,CD,FD,A0,57,3E,00,CD,FD
1600 DATA A0,7A,CB,63,FB,C9,C5,ED,4B
1610 DATA 06,00,0C,ED,79,3E,8F,ED,79,ED,78,C1
1620 DATA C9,3E,02,CD,FD,A0,E6,01,C2,0D,A1
1630 DATA AF,CD,FD,A0,C9,END

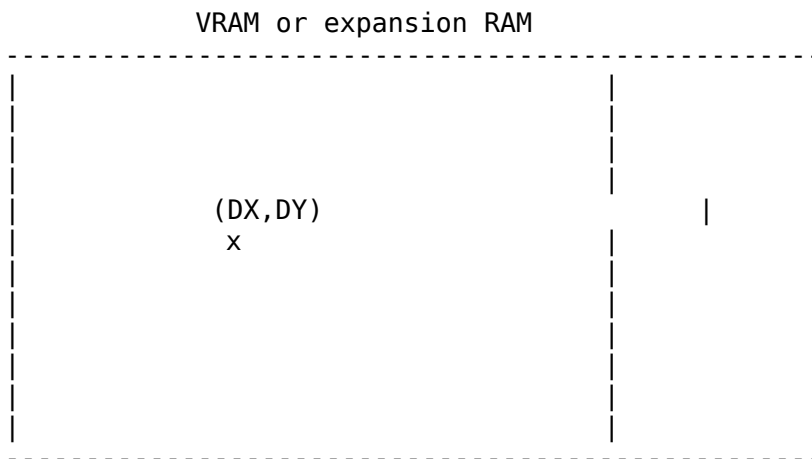
```

#### 6.5.11 PSET (drawing a point)

A point is drawn at any coordinate in VRAM (see figure 4.99).

After setting the parameters as shown in Figure 4.100, writing 5XH (X means a logical operation) in R#46 causes the command to be executed. While the CE bit of S#2 is "1", the command is being executed. List 4.20 shows an example of using PSET.

Figure 4.99 Actions of PSET command



MXD: memory selection                      0 = VRAM, 1 = expansion RAM

DX: origin X-coordinate (0 to 511)

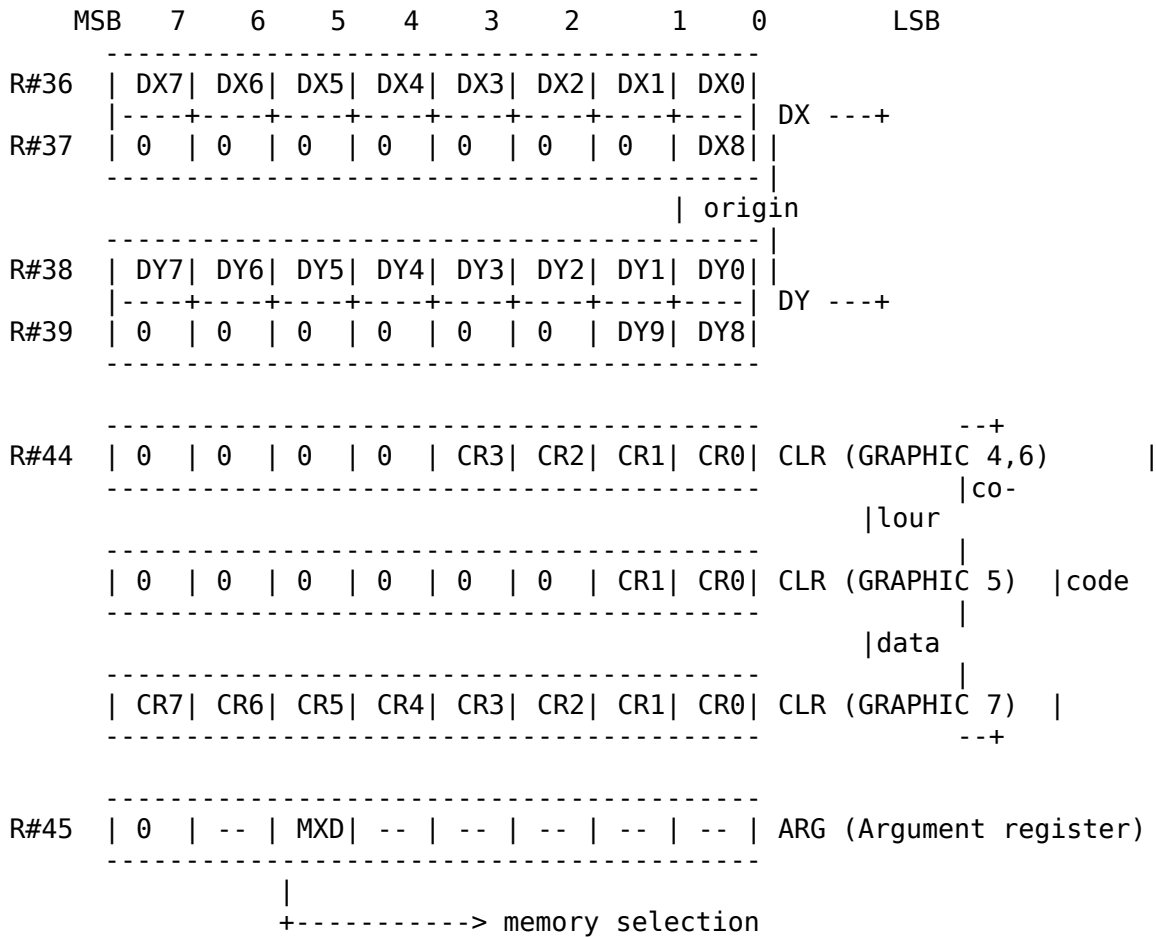
DY: origin Y-coordinate (0 to 1023)

CLR (R#44:Colour register): point colour

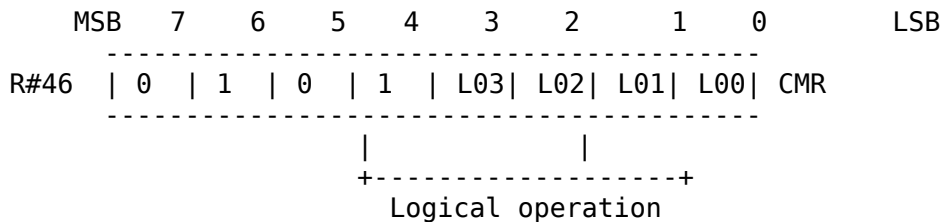


Figure 4.100 Register settings of PSET command

> PSET register setup



> PSET command execution



List 4.20 Example of PSET command execution

```

=====
;*****
; List 4.20 PSET sample
; to use, set H, L, E, A as follows
; pset (x:H, y:L), color:E, logi-OP:A
;*****
;
PUBLIC PSET

```

```
RDVDP:    EQU    0006H
WRVDP:    EQU    0007H
```

```
;----- program start -----
```

```
PSET: DI
      PUSH AF
      CALL WAIT.VDP
      LD BC, (WRVDP)

      INC C
      LD A, 36
      OUT (C), A
      LD A, 80H+17
      OUT (C), A

      PUSH BC
      INC C
      INC C
      XOR A
      OUT (C), H
      OUT (C), A
      OUT (C), L
      OUT (C), A
      POP BC

      LD A, 44
      OUT (C), A
      LD A, 80H+17
      OUT (C), A

      INC C
      INC C
      OUT (C), E
      XOR A
      OUT (C), A

      LD E, 01010000B
      POP AF
      OR E
      OUT (C), A

      EI
      RET
```

```
GET.STATUS:
      PUSH BC
      LD BC, (WRVDP)
      INC C
      OUT (C), A
      LD A, 8FH
      OUT (C), A
      LD BC, (RDVDP)
      INC C
      IN A, (C)
      POP BC
      RET
```

```

WAIT.VDP:
  LD   A,2
  CALL GET.STATUS
  AND  1
  JP   NZ,WAIT.VDP
  XOR  A
  CALL GET.STATUS
  RET

  END

```

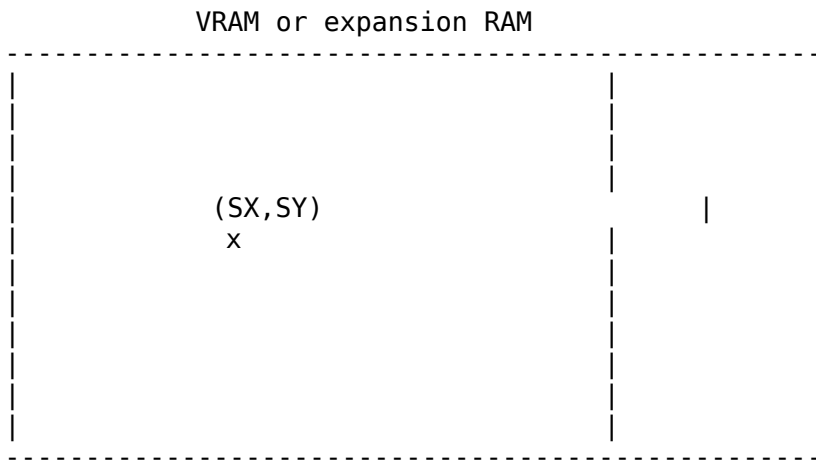
=====

### 6.5.12 POINT (reading a colour code)

POINT reads the colour code in any coordinate of VRAM (see Figure 4.101).

After setting the parameters as shown in Figure 4.102, writing 40H in R#46 causes the command to be executed. While the CE bit of S#2 is "1", the command is being executed. After the command terminates, the colour code of the specified coordinate is set in S#7. List 4.21 shows an example of using POINT.

Figure 4.101 Actions of POINT command

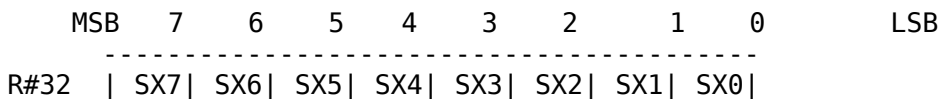


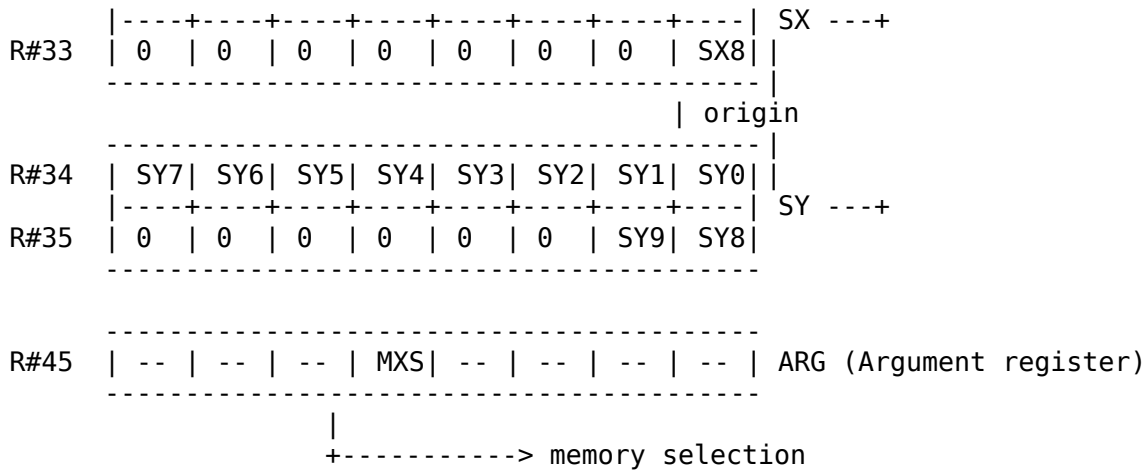
MXD: memory selection                      0 = VRAM, 1 = expansion RAM

SX: origin X-coordinate (0 to 511)  
 SY: origin Y-coordinate (0 to 1023)

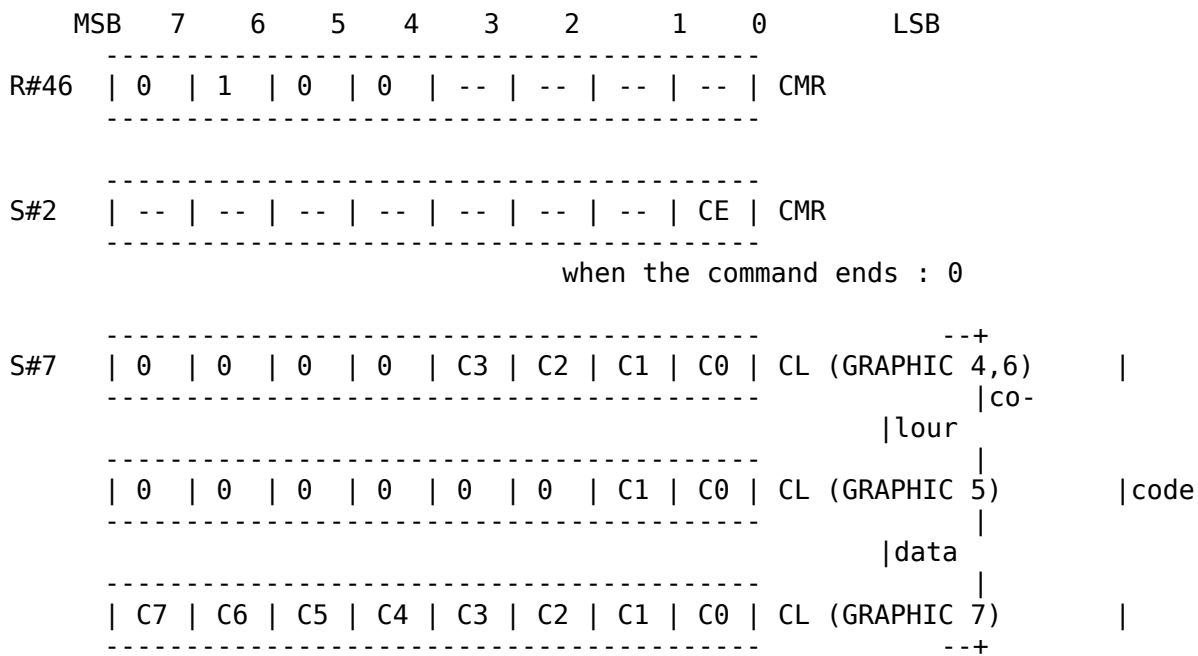
Figure 4.102 Register settings of POINT command

> POINT register setup





> POINT command execution



List 4.21 Example of POINT command execution

```

=====
;*****
; List 4.21 POINT sample
; to use, set H, L as follows
; POINT ( x:H, y:L )
; returns: A := COLOR CODE
;*****
;
PUBLIC POINT

RDVDP: EQU 0006H
WRVDP: EQU 0007H

;----- program start -----

```

```

POINT:      DI
           CALL WAIT.VDP

           LD  A,(WRVDP)
           LD  C,A

           INC C
           LD  A,32
           OUT (C),A
           LD  A,80H+17
           OUT (C),A

           INC C
           INC C
           XOR A
           OUT (C),H
           OUT (C),A
           OUT (C),L
           OUT (C),A

           DEC C
           DEC C
           OUT (C),A
           LD  A,80H+45
           OUT (C),A
           LD  A,01000000B
           OUT (C),A
           LD  A,80H+46
           OUT (C),A
           CALL WAIT.VDP
           LD  A,7
           CALL GET.STATUS
           PUSH AF
           XOR A
           CALL GET.STATUS
           POP  AF

           EI
           RET

```

```

GET.STATUS:
           PUSH BC
           LD  BC,(WRVDP)
           INC C
           OUT (C),A
           LD  A,8FH
           OUT (C),A
           LD  BC,(RDVDP)
           INC C
           IN  A,(C)
           POP BC
           RET

```

```

WAIT.VDP:
           LD  A,2
           CALL GET.STATUS
           AND 1
           JP  NZ,WAIT.VDP

```

```

XOR  A
CALL GET.STATUS
RET

END

```

```

=====
List 4.22  PAINT routine using PSET and POINT
=====

```

```

;*****
; List 4.22  paint routine using PSET and POINT
; ENTRY:    X:H, Y:L, BORDER COLOR:D, PAINT COLOR:E
;*****
;
; EXTRN PSET
; EXTRN POINT

Q.LENGTH EQU 256*2*2
MAX.Y EQU 211

;----- paint main routine -----

PAINT: CALL POINT
      CP D
      RET Z
      CALL INIT.Q
      LD (COL),DE
      CALL PUT.Q
      LD A,(COL)
      LD E,A
      XOR A ;logi-0P : PSET
      CALL PSET
PAINT0: CALL GET.Q
      RET C
      INC H
      CALL NZ,PAINT.SUB
      DEC H
      JP Z,PAINT1
      DEC H
      CALL PAINT.SUB
      INC H
PAINT1: DEC L
      LD A,-1
      CP L
      CALL NZ,PAINT.SUB
      INC L
      INC L
      LD A,MAX.Y
      CP L
      CALL NC,PAINT.SUB
      JP PAINT0

```

```

;----- check point and pset -----

```

```

PAINT.SUB:

```

```

CALL POINT
LD D,A
LD A,(BORD)
CP D
RET Z
LD A,(COL)
CP D
RET Z
LD E,A
XOR A
CALL PSET
CALL PUT.Q
RET

```

;----- init Q.BUFFER pointer -----

```

INIT.Q:
PUSH HL
LD HL,Q.BUF
LD (Q.TOP),HL
LD (Q.BTM),HL
POP HL
RET

```

;----- put point to Q.BUF (X:H , Y:L) -----

```

PUT.Q:
EX DE,HL
LD HL,(Q.TOP)
LD BC,Q.BUF+Q.LENGTH+1
OR A ;clear CARRY
PUSH HL
SBC HL,BC
POP HL
JP C,PUT.Q1
LD HL,Q.BUF
PUT.Q1:
LD (HL),D
INC HL
LD (HL),E
INC HL
LD (Q.TOP),HL
EX DE,HL
RET

```

;----- take point data to D, E -----  
; returns: NC H:x, L:y  
; C buffer empty

```

GET.Q: LD HL,(Q.BTM)
LD BC,(Q.TOP)
OR A
SBC HL,BC
JP NZ,GET.Q0
SCF
RET

```

```

GET.Q0: LD HL,(Q.BTM)

```

```

        LD    BC,Q.BUF+Q.LENGTH+1
        OR    A
        PUSH HL
        SBC  HL,BC
        POP  HL
        JP   C,GET.Q1
        LD   HL,Q.BUF
GET.Q1: LD   D,(HL)
        INC  HL
        LD   E,(HL)
        INC  HL
        LD   (Q.BTM),HL
        OR   A
        EX  DE,HL
        RET

```

;----- work area -----

```

COL    DS    1
BORD   DS    1
Q.TOP  DS    2
Q.BTM  DS    2
Q.BUF  DS    Q.LENGTH

```

END

---

List 4.23 Example of using the PAINT routine

---

```

1000 '*****
1010 ' list 4.23  paint routine using POINT and PSET
1020 ' Position cursor at beginnig of paint area and press the space bar.
1030 '*****
1040 '
1050 SCREEN 5
1060 FOR I=0 TO 50
1070 LINE -(RND(1)*255,RND(1)*211),15
1080 NEXT
1090 I=&HA000 :DEF USR=I
1100 READ A$
1110 IF A$="END" THEN 1150
1120 POKE I,VAL("&H"+A$):I=I+1
1130 READ A$
1140 GOTO 1110
1150 X=128:Y=100:COL=15:PCOL=2
1160 CURS=0
1170 A=STICK(0)
1180 CURS=(CURS+1) AND 1
1190 LINE (X-5,I)-(X+5,I),15,,XOR
1200 LINE (X,Y-5)-(X,Y+5),15,,XOR
1210 IF CURS=1 THEN 1310
1220 IF A=1 THEN Y=Y-1
1230 IF A=2 THEN Y=Y-1:X=X+1
1240 IF A=3 THEN X=X+1
1250 IF A=4 THEN X=X+1:Y=Y+1

```



```

1260 IF A=5 THEN Y=Y+1
1270 IF A=6 THEN Y=Y+1:X=X-1
1280 IF A=7 THEN X=X-1
1290 IF A=8 THEN X=X-1:Y=Y-1
1300 IF STRIG(9) THEN GOSUB 1320
1310 GOTO 1170
1320 POKE &HA8CA,Y
1330 POKE &HA8CB,X
1340 POKE &HA8CD,COL
1350 POKE &HA8CC,PCOL
1360 A=USR(0)
1370 RETURN
1380 DATA ED,73,00,A8,31,CA,A8,2A,CA,A8,ED,5B,CC,A8,CD,67
1390 DATA A0,ED,7B,00,A8,C9,E5,21,D4,A8,22,D0,A8,22,D2,A8
1400 DATA E1,C9,EB,2A,D0,A8,01,D5,AC,B7,E5,ED,42,E1,DA,34
1410 DATA A0,21,D4,A8,72,23,73,23,22,D0,A8,EB,C9,2A,D2,A8
1420 DATA ED,4B,D0,A8,B7,ED,42,C2,4C,A0,37,C9,2A,D2,A8,01
1430 DATA D5,AC,B7,E5,ED,42,E1,DA,5D,A0,21,D4,A8,56,23,5E
1440 DATA 23,22,D2,A8,B7,EB,C9,CD,B8,A0,BA,C8,CD,16,A0,ED
1450 DATA 53,CE,A8,CD,22,A0,3A,CE,A8,5F,AF,CD,F4,A0,CD,3D
1460 DATA A0,D8,24,C4,A1,A0,25,CA,8F,A0,25,CD,A1,A0,24,2D
1470 DATA 3E,FF,BD,C4,A1,A0,2C,2C,3E,D3,BD,D4,A1,A0,C3,7E
1480 DATA A0,CD,B8,A0,57,3A,CF,A8,BA,C8,3A,CE,A8,BA,C8,5F
1490 DATA AF,CD,F4,A0,CD,22,A0,C9,F3,CD,3A,A1,ED,4B,06,00
1500 DATA 0C,3E,20,ED,79,3E,91,ED,79,0C,0C,AF,ED,61,ED,79
1510 DATA ED,69,ED,79,0D,0D,ED,79,3E,AD,ED,79,3E,40,ED,79
1520 DATA 3E,AE,ED,79,CD,3A,A1,3E,07,CD,2A,A1,F5,AF,CD,2A
1530 DATA A1,F1,FB,C9,F3,F5,CD,3A,A1,ED,4B,06,00,0C,3E,24
1540 DATA ED,79,3E,91,ED,79,C5,0C,0C,AF,ED,61,ED,79,ED,69
1550 DATA ED,79,C1,3E,2C,ED,79,3E,91,ED,79,0C,0C,ED,59,AF
1560 DATA ED,79,1E,50,F1,B3,ED,79,FB,C9,C5,ED,4B,06,00,0C
1570 DATA ED,79,3E,8F,ED,79,ED,78,C1,C9,3E,02,CD,2A,A1,E6
1580 DATA 01,C2,3A,A1,AF,CD,2A,A1,C9
1590 DATA END

```

---

## 6.6 Speeding Up Commands

MSX-VIDEO performs various screen management duties in addition to executing the specified commands. Sometimes the command execution speed seems to be a bit slow because of this. Thus, by discarding these operations, the speed of the command executions can be made faster. This can be done using the following method.

### 1. Sprite display inhibition

This method is useful since speedup can be realised while the screen remains displayed. Set "1" to bit 1 of R#8.

### 2. Screen display inhibition

This method cannot be used frequently except in the case of initialising the screen, since the screen fades out in this mode. Set "1" to bit 6 of R#1.

## 6.7 Register Status at Command Termination

Table 4.7 shows the register status at the command termination for each command.

When the number of dots to be executed in Y direction assumes N, the values of SY\*, DY\*, and NYB can be calculated as follows:

SY\*=SY+N, DY\*=DY+N ..... when DIY bit is 0  
 SY\*=SY-N, DY\*=DY-N ..... when DIY bit is 1  
 NYB=NY-N

Note: when MAJ bit is 0 in LINE, N = N - 1.

Table 4.7 Register status at command termination

command name	SX	SY	DX	DY	NX	NY	CLR	CMR H	CMR L	ARG
HMMC	---	---	---	.	---	#	---	0	---	---
YMMM	---	.	---	.	---	#	---	0	---	---
HMMM	---	.	---	.	---	#	---	0	---	---
HMMV	---	---	---	.	---	#	---	0	---	---
LMMC	---	---	---	.	---	#	---	0	---	---
LMCM	---	.	---	---	---	#	.	0	---	---
LMMM	---	.	---	.	---	#	---	0	---	---
LMMV	---	---	---	.	---	#	---	0	---	---
LINE	---	---	---	.	---	---	---	0	---	---
SRCH	---	---	---	---	---	---	---	0	---	---
PSET	---	---	---	---	---	---	---	0	---	---
POINT	---	---	---	---	---	---	.	0	---	---

--- : no change

. : coordinate (SY\*, DY\*) and the colour code at the command termination

# : the number of counts (NYB), when the screen edge is fetched