PORTAR

- MSX I/O MAPPING -

Portar Specifications

About this Document

Portar Doc Version 1.7, last updated 21st March 2001 by Martin Korth.
This document describes the I/O Map of MSX computers, attempting to supply a very compact and mostly complete document about MSX1 and MSX2 hardware. It does not include information about the MSX firmware and operating systems, such like BIOS, BASIC, BDOS, MSXDOS, or CP/M related functions. Written by MAYER (1991-1995) and Martin Korth (1999-2001).

First Generation 1991-1995 by MAYER of WC HAKKERS
MAYER's SV738 X'press I/O MAP version 1.5. Thanks to Henrik Gilvad (Denmark) for info about the MSX-2 MVDP and the MSX-2 ROMs, to Pal F Hansen (Norway) for VRAM info and to Jonas Lindstroem (Sweden) for some MVDP info.

Second Generation 1999-2001 by Nocash/Martin Korth
Thanks to Enrique Sanchez for his collection of various docs, to Manuel Pazos for answering many questions and for V9958 specs, to Sean Young for his MEGAROMS doc and some TMS9918A details, to Konami Man and Ascii for MSX2 Technical Handbook, to Zelly of Mayhem for his V9958 summary, to Tomas Karlsson for WD1793 FDC specs, to Ascat/Takamichi for Kanji description, and to MAYER for the original doc & for comments about the updated version.

Other Formats and Updates and Help Wanted
This text is available in .TXT format (raw 7bit Ascii), and .HTM format (HTML). The .HTM version includes simple formatting for chapters, hyperlinks, and bold headlines.

```
  http://www.work.de/nocash/portar.txt
  http://www.work.de/nocash/portar.htm
```

Some sections in this doc are marked by question marks, if you can confirm or explain these parts, or if you know additional information, or if you find any information to be incorrect or incomplete, please drop a note to Martin Korth (http://www.work.de/nocash/email.htm).

I/O Port Summary

I/O Ports

Most internal MSX hardware is accessed by I/O instructions. In most (or all) cases only the lower 8 bits of the I/O addresses are relevant.

Port	R/W	Chip/Name/Function
00-3F	NC	Free for User
40-7B	NC	Reserved
7C	?	MSX MUSIC YM2413/OPLL (FM-PAC,FM-PAK,MSX2+) Index
7D	?	MSX MUSIC YM2413/OPLL (FM-PAC,FM-PAK,MSX2+) Data I/O
7E-7F	NC	Reserved
80	R/W	RS232 I8251 (ACIA) Data
81	R/W	RS232 I8251 (ACIA) Status/Command
82	R	RS232 Status for CTS,Timer/counter2,RI,CD
82	W	RS232 Interrupt mask register
83	R	RS232 ?Clock 0,1,2 read?
83	W	RS232 ?Receive ready interrupt enable?
84	R/W	RS232 I8253 (Baud gener.) Counter 0 Receive clock
85	R/W	RS232 I8253 (Baud gener.) Counter 1 Transmit clock
86	R/W	RS232 I8253 (Baud gener.) Counter 2 Used by programs
87	W	RS232 I8253 (Baud gener.) Mode register
88	?	?Modem enable?
88-8B	R/W,W	External VDP 9938 adaptor for MSX1 (similiar to Port 98-9B)
8C-8D	?	Reserved for modem
8E-8F	NC	Reserved
90	R	ULA5RA087 Centronic BUSY state (bit 1=1)
90	W	ULA5RA087 Centronic STROBE output (bit 0=0)
91	W	ULA5RA087 Centronic Printer Data
92-97	NC	Reserved
98	R/W	9918,9929,9938,9958,9978 VRAM Data Read/Write
99	R	9918,9929,9938,9958,9978 VDP Status Registers
99	W	2nd Byte b7=0: 99X8 VRAM Address setup
99	W	2nd Byte b7=1: 99X8 VDP Register write
9A	W	MVDP (MSX2) 9938,9958 Color Palette Register (2 bytes)
9B	W	MVDP (MSX2) 9938,9958 Register data
9C-9F	NC	Reserved
A0	W	I AY-3-8910 PSG Sound Generator Index
A1	W	I AY-3-8910 PSG Sound Generator Data write
A2	R	I AY-3-8910 PSG Sound Generator Data read
A3-A7	NC	Reserved
A8	R/W	I 8255A/ULA9RA041 PPI Port A Memory PSLOT Register (RAM/ROM)
A9	R	I 8255A/ULA9RA041 PPI Port B Keyboard column inputs
AA	R/W	I 8255A/ULA9RA041 PPI Port C Kbd Row sel,LED,CASo,CASm
AB	W	I 8255A/ULA9RA041 Mode select and I/O setup of A,B,C
AC-AF	NC	Reserved
B0-B3	?	External 8255 (SONY DataRamPack)
B4	W	RP 5C01 (Not in 738) RTC Register select
B5	R/W	RP 5C01 (Not in 738) RTC data
B6-B7	NC	Reserved
B8-BB	?	SANYO Light pen interface
BC-BF	?	VHD control
C0-C1	?	MSX audio (used in Music Module cartridge by Philips,

```
OPL1)
  C2-C7   NC     Reserved
  C8-CF   ?      MSX Interface (??)
  D0-D7   ?      External Floppy Disk Controller
  D8h     W      Kanji ROM Select Class 1 Code (lower 6 bits)
  D9h     W      Kanji ROM Select Class 1 Code (upper 6 bits)
  D9h     R      Kanji ROM Read Class 1 Data (32 bytes)
  DAh     W      Kanji ROM Select Class 2 Code (lower 6 bits)
  DBh     W      Kanji ROM Select Class 2 Code (upper 6 bits)
  DBh     R      Kanji ROM Read Class 2 Data (32 bytes)
  DC-F4   NC     Reserved
   E5-E7  ?      MSX-Engine chip (MSX2/2+/TurboR) ???
  F5      W      System Control (used to disable internal I/O ports)
  F6      ?      colour bus
  F7      R/W    Audio/Video control
  F8-FB   NC     Reserved (But somehow accessed by MSX2 BIOS ???)
  FC      R/W    Memory Mapper (RAM bank for 0000-3FFF)
  FD      R/W    Memory Mapper (RAM bank for 4000-7FFF)
  FE      R/W    Memory Mapper (RAM bank for 8000-BFFF)
  FF      R/W    Memory Mapper (RAM bank for C000-FFFF)
```

Memory mapped I/O

Most external hardware (and most external and internal disk controllers) are accessed by memory mapped I/O, ie. by LD commands rather than by IN and OUT commands. Common memory mapped I/O addresses are:

```
  DISK:7FXX     Floppy Disk Controller
  DISK:BFXX     Floppy Disk Controller
  CART:XXXX     Cartridge Memory Mappers
  CART:98XX     Cartridge SCC (Sound Custom Chip)
  SLOT:FFFF     Secondary Slot (select DISK ROM or MAIN RAM in a PSLOT)
```

To access these I/O 'Ports', the respective memory area must be selected into memory by the PSLOT register, and then data can be written (or read in some cases) to (or from) the memory addresses.

Memory

Port A8	Primary Slot Register (PPI Port A, PSLOT) (Read/Write)
Port FC-FF	Memory Mapper RAM page select (Read/Write)
Mem. SLOT:FFFF	Secondary Slot Register (Read Inverted/Write)
Mem. CART:XXXX	Cartridge Memory Mappers

Memory Map example for a diskless MSX1 machine
A typical MSX1 model includes 32K ROM, and 8K, 16K, 32K, or 64K RAM. External RAM or ROM Expansions/Games, or Disk Controller (with 16K DISK ROM) could be connected to either of the two cartridge slots.

Memory Address	PSLOT=0 MainROM	PSLOT=1 Cart. A	PSLOT=2 Cart. B	PSLOT=3 MainRAM
0000..3FFF	BIOS	<aux>	<aux>	RAM3 (*)
4000..7FFF	BASIC	<aux>	<aux>	RAM2 (*)
8000..BFFF	N/A	<aux>	<aux>	RAM1 (*)
C000..FFFF	N/A	<aux>	<aux>	RAM0

(*) If less than 64K internal RAM is installed, then RAM is located in the higher memory area only, ie. at address E000-FFFF for 8K, C000-FFFF for 16K, etc.

Memory Map example for a MSX2 machine with built-in Disk Drive
A typical MSX2 model includes 48K ROM (called BIOS, BASIC, and SUB), and in this example, 16K ROM for the DISK Drive. Typically 128K, 256K, or 512K RAM are built-in.

Memory	PSLOT=0	PSLOT=1	PSLOT=2	[...........PSLOT=3...........			]
Address	MainROM	Cart. A	Cart. B	SSLOT=0	SSLOT=1	SSLOT=2	SSLOT=3
0000..3FFF	BIOS	<aux>	<aux>	N/A	SUB	RAM[3]	N/A
4000..7FFF	BASIC	<aux>	<aux>	N/A	DISK	RAM[2]	N/A
8000..BFFF	N/A	<aux>	<aux>	N/A	N/A	RAM[1]	N/A
C000..FFFF	N/A	<aux>	<aux>	N/A	N/A	RAM[0]	N/A

Because of the additional internal ROM, Slot 3 is sub-classed into 4 slots by using a new Secondary Slot Register (Memory FFFFh, see below). The RAM is split into 16K banks, four of these banks can be mapped into memory at once by using a Memory Mapper (Port FCh-FFh, see below).

Port 0A8h, Primary Slot Register (PSLOT, PPI Port A) (Read/Write)
Used to select internal RAM, or ROM, or external memory (cartridges) into CPU address space, as shown in the examples above. In some cases also used to select 'memory mapped I/O ports' into memory.

```
  Bit   Expl.
  0-1   PSLOT number 0-3 for memory at 0000-3FFF
  2-3   PSLOT number 0-3 for memory at 4000-7FFF
  4-5   PSLOT number 0-3 for memory at 8000-BFFF
  6-7   PSLOT number 0-3 for memory at C000-FFFF
```

The PSLOT register is available in all MSX models, and Slot 0 is always used for Main ROM. However, there is no standard for the assignment of Slot 1-3. For example, RAM might be located in Slot 1, or Slot 2, or Slot 3.

Memory PSLOT:FFFFh, Secondary Slot Register (SSLOT) (Read Inverted, Write)
Used to subclass a Primary Slot into four Secondary Slots. Before accessing a secondary slot register its primary slot must be selected into memory at C000-FFFF (done by Bit 6-7 of Port A8h). The new SSLOT value may then be written to memory address FFFFh, reading from that address returns the current SSLOT value XORed by FFh (this behaviour might be used to detect the presence of a secondary slot).

```
  Bit   Expl.
  0-1   SSLOT number 0-3 for memory at 0000-3FFF of respective PSLOT
  2-3   SSLOT number 0-3 for memory at 4000-7FFF of respective PSLOT
  4-5   SSLOT number 0-3 for memory at 8000-BFFF of respective PSLOT
  6-7   SSLOT number 0-3 for memory at C000-FFFF of respective PSLOT
```

Theoretically each PSLOT could be subclassed by separate SSLOT registers. Commonly only one of the PSLOTs is subclassed though, and older computers with not more than 32K ROM do not contain any SSLOT registers at all. Beside for the internal usuage, SSLOTs might be found in Slot Expansion Cartridges which would then allow to connect up to four cartridges to a single cartride slot. As for PSLOTs, there is no standard that defines which PSLOT should contain an SSLOTs, and which of the SSLOTs should contain RAM...

Memory Mapper (RAM Banking)
A memory mapper is usually available in MSX computers with more than 64K internal RAM. The BIOS initializes these banks in reversed order, ie. the default values for Port FC-FF are 03-00.

```
  Port FC   RAM bank number to be mapped at 0000-3FFF
  Port FD   RAM bank number to be mapped at 4000-7FFF
  Port FE   RAM bank number to be mapped at 8000-BFFF
  Port FF   RAM bank number to be mapped at C000-FFFF
```

Usually the mapping circuit latches only the actually used lower 3, 4, or 5 bits of the bank numbers, depending on whether 128K, 256K, or 512K RAM are built-in. Theoretically (fully exanped) a total of 4MByte RAM (16Kbyte * 256 blocks) could be controlled by the mapping registers.
To enable/disable RAM, use the PSLOT register (Port A8h), and (if any) the SSLOT register (Address FFFFh).

Memory Mapped I/O
Beside for RAM and ROM some of the addresses of some slots might be also used as memory mapped I/O Ports. (For example for Disk Controllers, and Cartridge Memory Mappers, and the Secondary Slot register.)

Cartridge Memory Mappers
Cartridges with more than 64K ROM or RAM must include their own memory mappers (cartridges with 64K ROM often include a mapper also, even though it isn't actually required). Various different mappers exist, read the chapter about Cartridge Memory Mappers for more information about the must commonly used chips.

Further Memory
Video Memory is not part of the memory map because VRAM can be accessed through I/O ports only (as described in the chapter about the Video Display Processor). Some computers may also include japanese character set ROMs, these can be read out only through I/O ports either (as described in the chapter about Kanji ROM).

Peripheral Interface

8255A/ULA9RA041 PPI (Programmable Peripheral Interface)

Port A8 PPI Port A Memory PSLOT Register (RAM/ROM) (Read/Write)
Port A9 PPI Port B Keyboard column inputs (Used as Read Only)
Port AA PPI Port C Kbd Row sel,LED,CASo,CASm (Read/Write)
Port AB PPI Mode select and I/O setup of A,B,C (Write Only)

Port 0A8h, PPI Port A - Primary Slot Register (PSLOT) (Read/Write)

```
  Bit   Expl.
  0-1   PSLOT number 0-3 for memory at 0000-3FFF
  2-3   PSLOT number 0-3 for memory at 4000-7FFF
  4-5   PSLOT number 0-3 for memory at 8000-BFFF
  6-7   PSLOT number 0-3 for memory at C000-FFFF
```

Used to select internal RAM, or ROM, or external memory (cartridges) into CPU address space, for more info read the chapter about Memory.
In some cases also used to select 'memory mapped I/O ports' into memory.

Port 0A9h, PPI Port B - Keyboard Column Inputs (Read Only)
Reading this port will get back the state of the selected keyboard line (selected via port 0AAh).

```
  Bit  Name    Expl.
  0-7  KC0-7   Keyboard line status
```

Each bit corresponds to one of 8 keys on each line, as shown in the 'Keyboard Matrix' (see below).

Port 0AAh, PPI Port C - Keyboard Row,LED,Cassette (Read/Write)

```
  Bit  Name   Expl.
  0-3  KB0-3  Keyboard line               (0-8 on SV738 X'Press)
  4    CASON  Cassette motor relay        (0=On, 1=Off)
  5    CASW   Cassette audio out          (Pulse)
  6    CAPS   CAPS-LOCK lamp              (0=On, 1=Off)
  7    SOUND  Keyboard klick bit          (Pulse)
```

To generate a 50Hz sound, turn bit7 on and off at a rate of 50Hz.
For Keyboard input, see PPI Port B. For Keyboard Matrix, see below.
For Joystick input/output, and for Cassette input, see PSG Sound Generator.

Port 0ABh, PPI Control Register (Write Only)
This port is used in two ways depending on bit 7. If bit7 is zero, then this port can be used to set or reset a single bit in PPI register C. This is just an alternate method to modify bits in register C than by writing to it directly (via Port AAh).

```
  Bit  Name   Expl.
  0    B      Set/reset the bit           (0=Reset, 1=Set)
  1-3  N0-N2  Bit number                  (0-7)
  4-6  0      Not used
  7    SF     Must be "0" for bit set/reset function.
```

Otherwise, if bit7 is set, then this port is used as Mode Setup for PPI Port A-C. Theoretically these registers can be used as input or output ports. However, in the MSX, PPI Port A and C are always used as output, and PPI Port B as input. The BIOS initializes it like that by writing 82h to Port ABh on startup. Afterwards it makes no sense to change this setting, and thus needs no further explanatation here.

The MSX Keyboard Matrix

The names in this table refer to the original MSX1 keyboard map (which is mostly identical to the US-keymap on PCs).

Line	Bit_7	Bit_6	Bit_5	Bit_4	Bit_3	Bit_2	Bit_1	Bit_0
0	"7"	"6"	"5"	"4"	"3"	"2"	"1"	"0"
1	";"	"]"	"["	"\"	"="	"-"	"9"	"8"
2	"B"	"A"	???	"/"	"."	","	"'"	"`"
3	"J"	"I"	"H"	"G"	"F"	"E"	"D"	"C"
4	"R"	"Q"	"P"	"O"	"N"	"M"	"L"	"K"
5	"Z"	"Y"	"X"	"W"	"V"	"U"	"T"	"S"
6	F3	F2	F1	CODE	CAP	GRAPH	CTRL	SHIFT
7	RET	SEL	BS	STOP	TAB	ESC	F5	F4
8	RIGHT	DOWN	UP	LEFT	DEL	INS	HOME	SPACE
(9	NUM4	NUM3	NUM2	NUM1	NUM0	NUM/	NUM+	NUM*)
(10	NUM.	NUM,	NUM-	NUM9	NUM8	NUM7	NUM6	NUM5)

Line 9 and 10 are used/reserved for numeric keypad, the NUM/, NUM+, NUM* keys might be exchanged in some coutries. (Most MSX1 and MSX2 models do not include a numeric keypad at all though).

MSX2 models have been delivered with country-specific BIOSes and keymaps, in that case the logical keynames change. For example the key in Line 1, Bit 5 is always the key to the right of the "P"-key, but on german keyboards that key would be used as "UE" rather than "[".

About the five function keys (F1-F5), note that the key-combinations Shift+F1-F5 are often referred to as F6-F10 in the MSX world. The GRAPH key would be the LEFT ALT key on a PC keyboard, CODE would be RIGHT ALT, and SELECT and STOP could be mapped as PAGE-UP/DN and END.

Video Display Processor

VDP I/O Ports

Port 98-99 Internal VDP (V9918 for MSX1)
Port 98-9B Internal VDP (V9938 for MSX2, V9958 for MSX2+/turbo R)
Port 88-8B External VDP (V9938 upgrade for MSX1)

Internal VDP
Port 98 VRAM Data (Read/Write)
Port 99 VDP Status Registers (Read Only)
Port 99 2nd Byte b7=0: VRAM Address setup (Write Only)
Port 99 2nd Byte b7=1: VDP Register write (Write Only)
Port 9A MSX2 Only: Palette Register (2 bytes) (Write Only)
Port 9B MSX2 Only: Register Data (Write Only)

External VDP 9938
MSX1 computers could be upgraded to MSX2 by a special cartridge which includes a V9938 display processor and MSX2 BIOS ROMs. The V9938 is accessed through Port 88h-8Bh (in same way as Port 98h-9Bh of internal VDP). A programmer should always verify bytes at address 0006h and 0007h in Main-ROM which specify the VDP Port address.

Video Modes (Screens)

This chapter describes the standard VRAM map for the different MSX video modes (screens). Note that these default addresses can be changed by modifying VDP registers 2-6.

The VDP Video Modes (BASIC Screen 0-8) is selected by the Bits M1-M5 of VDP Register 0 and 1. The relationship between the bits and the screen is:

```
  M1 M2 M3 M4 M5  Screen format
  1  0  0  0  0   Text       40x24              (BASIC SCREEN 0)
  0  0  0  0  0   Half text  32x24              (BASIC SCREEN 1)
  0  0  1  0  0   Hi resolution 256x192         (BASIC SCREEN 2)
  0  1  0  0  0   Multicolour  4x4pix blocks    (BASIC SCREEN 3)
  ----Below MSX2 only----
  0  0  0  1  0   Screen2 with 8 Sprites/Line  (BASIC SCREEN 4)
  0  0  1  1  0   256*212, 16  colours/pixel    (BASIC SCREEN 5)
  0  0  0  0  1   512*212, 4   colours/pixel    (BASIC SCREEN 6)
  0  0  1  0  1   512*212, 16  colours/pixel    (BASIC SCREEN 7)
  0  0  1  1  1   256*212, 256 colours/pixel    (BASIC SCREEN 8)
  1  0  0  1  0   Text 80x24                    (BASIC SCREEN 0, WIDTH 80)
```

The above vertical default resolutions could be changed by modifying bit 7 of VDP register 9.

Screen 0-3 are available on both MSX1 and MSX2, the other screens are supported on MSX2 only.

SCREEN 0 - 40x24 text mode

```
  0000-03BF   BG Map
  0800-0FFF   BG Tiles
```

In screen 0, the tiles are defined as usually (8x8 pixels), but only the leftmost 6x8 pixels of each tile are visible.

SCREEN 1 - 32x24 coloured text mode

```
  0000-07FF   BG Tiles
  1800-1AFF   BG Map
  1B00-1B7F   OBJ Attributes
  2000-201F   BG Colors
  3800-3FFF   OBJ Tiles
```

The BG Colors array defines 32 colors (each 4 bit background, and 4 bit foreground, as in VDP register 7). The colors are assigned to the BG Tiles as follows: Tiles 00..07 share the first color, tiles 08..0F share the second color, etc, and tiles F8..FF share the last color.

SCREEN 2 - 256*192 Graphics mode

```
  0000-17FF   BG Tiles
  1800-1AFF   BG Map
  1B00-1B7F   OBJ Attributes
  2000-37FF   BG Colors
  3800-3FFF   OBJ Tiles
```

The BG Tiles array can be placed only at address 0000h or 2000h, ie. only bit 2 of VDP Register 4 is used. The BG Colors array is placed at the same address XORed by 2000h (ie. either at 2000h or 0000h).

In screen 2, the BG Tile memory consists of 300h tiles. The screen is vertically divided into 3 sections, all BG Map entries for the upper 8 character rows refer to tiles 00..FFh, the middle 8 rows to tiles 100..1FFh, and the bottom 8 rows to tiles 200..2FFh.

As usually, all TILE-BYTES define rows of eight pixels, each of these rows is colorized by a separate COLOR-BYTE in the BG Colors array, whereas each COLOR-BYTE defines the background color in bit 0-3 (for "0" bits in TILE-BYTE) and the foreground color in bit 4-7 (for "1" bits). That means there can be only 2 different colors in each row of 8 pixels!

SCREEN 3 (64*48 block graphics multicolour mode)

```
  0000-07FF   BG Tiles (block colors)
  0800-0AFF   BG Map
  1B00-1B7F   Sprite attribute table
  3800-3FFF   Sprite character patterns
```

The screen consists of 32x24 background tiles, each tile consists of 4 blocks, whereas each of these "pixels" can be colorized in any of the available 16 colors.

The BG Tile memory is organized as follows: It contains 8 color bytes for each of the 100h tiles. But obviously only two bytes are actually required (first byte for the upper half, and second byte for the lower half, in both cases most significant bits for the left 'pixels').

Which of the 8 bytes are used depends on the lower two bits of the vertical position (0-23), ie. tiles in lines 0,4,8,12,etc. use the first two bytes, tiles in lines 1,5,9,etc. use the next two bytes, and so on.

--- Below Screens 4-8 and the 80 column text screen exist on MSX2 only ---

SCREEN 4 (256*192 Graphics mode with multicolour sprites):

```
  0000-17FF   Charcter patterns
  1800-1AFF   Name table (char positions)
  1C00-1DFF   Sprite colours
  1E00-1E7F   Sprite attribute table
  1E80-1E9F   Palette
  2000-37FF   PixelByte colour table
  3800-3FFF   Sprite character patterns
```

This is mostly the same as Screen 2, except that the foregound sprites can have additional color attributes, and with the abilty to display a maximum of 8 sprites per line.

The "Palette" entry in the memory maps for screen 4-8 does not have a physical function. It is just a memory location where the MSX BIOS usually places a copy of the actual VDP palettes. For more information read the BASIC manual about the COLOR=RESTORE function.

SCREEN 5 (256*212 Graphic mode, 16 colours):

```
  0000-69FF   Matrix (Bitmap)
  7400-75FF   Sprite colours
  7600-767F   Sprite attribute table
  7680-769F   Palette
  7800-7FFF   Sprite character patterns
```

SCREEN 6 (512*212 Graphic mode, 4 colours):

```
  0000-69FF   Matrix (Bitmap)
  7400-75FF   Sprite colours
  7600-767F   Sprite attribute table
  7680-769F   Palette
  7800-7FFF   Sprite character patterns
```

SCREEN 7 (512*212 Graphic mode, 16 colours):

```
  0000-D3FF   Matrix (Bitmap)
  F000-F7FF   Sprite character patterns
  F800-F9FF   Sprite colours
  FA00-FA7F   Sprite attribute table
  FA80-FA9F   Palette
```

No$hackwork: (Even/Odd VRAM addressing)
In screen 7 and 8 (the video modes with 256 bytes per line), the video RAM is addressed differently as usually.

In these modes, the first 64K VRAM are used for even addresses, and the second 64K for ODD addresses. Ie. the address lines aren't A16..A0, rather it is A15..A0,A16. That method allows the video controller to increment the lower address lines at the same clock rate as in 128 byte/line modes.

The funny thing about that is, that it isn't visible for the programmer, ie. to set the 6th dot in first line in screen 8, (or the 12th and 13th, in screen 7), the program would still have to write to the (virtual) address 00005h, as if there would be no special even/odd feature. But physically the byte would be written to address 10002h!

As noted above, that is all done behind your back, and you don't have to care about it at all - EXCEPT if you write data into VRAM before you set up the desired video mode, ie. if you switch to screen 7/8 (or back), then any data that is already in VRAM changes it's position! Virtually at least...

SCREEN 8 (256*212 Graphic mode, 256 colours):

```
  0000-D3FF   RGB Matrix (Bitmap)
  F000-F7FF   Sprite character patterns
  F800-F9FF   Sprite colours                  (See note I)
  FA00-FA7F   Sprite attribute table
  FA80-FA9F   Palette (Huh?)
```

Each byte in the RGB Matrix defines a separate pixel. The bytes directly define the colors as follows:

```
  Bit 0-1   Blue, 0-3
  Bit 2-4   Red, 0-7
  Bit 5-7   Green, 0-7
```

Also read the description about even/odd VRAM addressing in screen 7 and 8. (See screen 7 description above.)

SCREEN 0 (Text mode, 80 column):

```
  0000-077F (086F)   Name table (char positions)
  0800-08EF (090D)   Character attribute (Blink)
  1000-17FF          Character patterns (font)
```

(Addresses in pharentheses is used in 26.5 lines mode. Observe that they are overlapping). The Characters attribute table is an array of bits, ie. the first byte of the table contains bits for the first 8 characters on the screen.

If the attribute-bit is zero, then the character is displayed as usually (colored as defined in VDP Reg 07h). If the bit is set, then the character is blinking (see VDP Reg 0Ch, and VDP Reg 0Dh).

Foreground Sprites

OBJ Attributes (Sprite attribute):
Defines 'OAM' data for up to 32 foreground sprites. Each entry consists of four bytes:

```
  0: Y-pos, Vertical position (FFh is topmost, 00h is second line, etc.)
  1: X-pos, Horizontal position (00h is leftmost)
  2: Pattern number
  3: Attributes. b0-3:Color, b4-6:unused, b7:EC (Early Clock)
```

If EC is set to 1, the X-pos is subtracted by 32 (can be used to place sprites particulary offscreen to the left.

When using 16x16 pixel sprites the lower two bits of the sprite number are ignored (should be zero). A 16x16 sprite logically consists of four 8x8 sprites, whereas the first 8x8 sprite is displayed in upperleft, the second one in lower left, the third in upper right, and the fourth in lower right.

If Y-pos is set to 208 (D0h), the sprite AND ALL FOLLOWING sprites are hidden! For MSX2 video modes (Screen 4-8), the same happens if Y-pos is set to 216 (D8h).

If the display is scrolled via VDP register 17h, this also affects the positions of the sprites! Ie. the actual visible position of the sprite would be YLOC+1-VDP(17h).
However, the special hide-the-sprites-value (YLOC=216) is hardcoded, independendly of the screen offset in VDP register 17h! If a sprite should be displayed in that line, either 215 or 217 must be used instead!

Sprite colours
In MSX2 video modes with colored sprites (screen 4-8), the fourth byte of the OAM entires is unused. Instead, the sprite attributes are stored in a separate 'color table'.
That table is always placed at the address of the above sprite attribute table minus 200h.
The color table contains 20h entries (one for each entry of the OAM table), and each entry is sized 10h bytes.
The 10h bytes of each entry specify color & attributes for each line of the displayed sprites (assuming that the sprite size is set to 16x16).

The bytes in that color table are used as follow:
```
  Bit 0-3    CL  Color Code (0-15)
  Bit 4      0   Unused
  Bit 5      IC  Ignore collisions with other sprites. (1=Ignore)
  Bit 6      CC  Mix color with sprite that has next higher priority.
  Bit 7      EC  Early clock (shift this line of the sprite 32 pixels to
left)
```
For screen 4-7 the color code specifies the desired palette color, for screen 8 the sprite colors are hardcoded as follows:
```
  Bit 0      Blue      (1=on, 0=off)
  Bit 1      Red       (1=on, 0=off)
  Bit 2      Green     (1=on, 0=off)
  Bit 3      Intensity (1=Light, 0=dark)
```
If the intensity-bit is set alone (with b0-2 cleared, ie. color 8), then something like bright pink or bright orange is displayed instead.

When CC is set, the color of the sprite is logically ORed with the pixels of the sprite "that has the next higher priority, and that has CC=0". In that case a collision just mixes colors, and does not causes a conflict, ie. bit 5 of status register 0 doesn't get set.

If a line of a sprite has the CC bit set, then it MUST collide with at least one pixel of another sprite with higher priority, otherwise the line of the sprite isn't displayed at all!

VRAM Data Read/Write

Port 98 VRAM Data (Read/Write)
Port 99 VRAM Address setup (2nd Byte b7=0) (Write Only)

Port 98h, Accessing VRAM Data
Read data from VRAM, or write data to VRAM. In either case the VRAM read/write pointer is automatically incremented, allowing to read/write a stream of bytes without having to setup the pointer each time.

Port 99h, VRAM Address Pointer Setup
The VRAM read/write pointer is initalized by writing TWO BYTES to port 99h with BIT 7 CLEARED in the second byte.

```
  Byte 1/Bit 0-7  Lower bits of VRAM Pointer
  Byte 2/Bit 0-5  Upper bits of VRAM Pointer
  Byte 2/Bit 6    Desired VRAM Direction (0=Reading, 1=Writing)
  Byte 2/Bit 7    Must be "0" for VRAM Pointer setup
```

This 14bit Pointer value allows to address 16Kbytes of VRAM (ie. the complete VRAM of MSX1 models). From what I understand, if the Direction is set up for Reading, VRAM data becomes latched immediately, and the pointer becomes incremented <before> data is actually read from Port 98h. If so, note that the latched byte might contain old data if the Pointer hasn't been set up previously.

Addressing more than 16K VRAM (MSX2 only)
On MSX2 the upper bits of the above 14bit pointer can be specified in VDP Register 0Eh, allowing to address the total of 128K VRAM. This register becomes automatically incremented when the 14bit read/write pointer overflows, for MSX1 compatibility this happens only in MSX2 video modes though. Additional 64K VRAM (if any) can be addressed by setting MXC bit of VDP Register 2Dh.

VDP Status Registers

Port 99 VDP Status Registers (Read Only)

MSX1 includes only one VDP Status Register (Register 0), for MSX2 VDP Status Registers 0-9 exist.

Status register 0 (default):

```
  Bit  Name  Expl.
  0-4  5/9th Number for the 5th sprite (9th in screen 4-8) on a line
  5    C     1 if overlapping sprites
  6    5D    1 if more than 4 sprites on a horizontal line (8 in screen
4-8)
  7    F     V-Blank IRQ Flag (1=interrupt) (See also IE0 flag)
```

In screen 1-3 only 4 sprites can be displayed per line, in screen 4-8 this number is doubled to 8 sprites per line. Bit 6 indicates if too many sprites have been (attempted to be) displayed. If the bit is set, Bit 0-4 indicate the number of the sprite that wasn't displayed properly. If more than one sprite haven't displayed properly, then Bit 0-4 specify the first of these bad sprites.

A sprite is overlapping another if a non-transparent pixel of a sprite hits a non-transparent pixel of another sprite. Whereas in MSX2 colored sprite modes this verfication can be made optional for specific sprites by CC and IC bits in the sprite OAM data.

The IRQ flag in bit 7 gets set at the beginning of the VBlank period, if IE0 in VDP Register 1 is set (or gets set at a later time, while the IRQ flag is still set) then an interrupt is generated.

The IRQ flag (bit 7) and the collision flag (bit 5) get cleared after reading Status register 0. BUG: When reading this register at the same time when IRQ changes from 0 to 1, this sometimes results in old value to be read (IRQ=0) before it becomes changed to 1, but the read-signal still acknowledges the IRQ (and sets it back to IRQ=0) - in that case the IRQ is lost.

Below Status Registers 1-9 exist on MSX2 only

To access these registers, first set the Status Register Index in VDP Register 0Fh, then read from port 99h, for compatibility to BIOS functions and MSX1 software the index should be always restored to zero after usage.

Status register 1: Interrupt Status

```
  Bit  Name Expl.
  0    FH   Horizontal Retrace IRQ Flag (See also: VDP Reg 13h and IE1 
flag)
  1-5  ID#  VDP Type    (0=V9938/MSX2, 2=V9958/MSX2+ and Turbo R, 
1=V9948?!)
  6    LPS  Light Pen ???               (MSX2 only, Not MSX2+)
  7    FL   Light Pen and/or Mouse ???  (MSX2 only, Not MSX2+)
```

Status register 2: VDP Command Status

```
  Bit  Name Expl.
  0    CE   Command Execute         (0=Finished, 1=VDP Command still 
executing)
  1    EO   Display field flag (??) (0=display first field)
  2-3  0    Not Used
  4    BO   Search Command Result   (0=Not found, 1=Found)
  5    HR   Horizontal Retrace Flag (1=HBlank)
  6    VR   Vertical Retrace Flag   (1=VBlank)
  7    TR   Data Ready (For CPU <--> VRAM Commands) (0=Not Ready, 
1=Ready)
```

Despite of its name, the Vertical Retrace Flag is set for the entire time while lower and upper screen borders are drawn and during actual vertical retrace.

The Horizontal Retrace Flag becomes set at the end of each scanline - including hidden/dummy scanlines during vertical retrace.

Status register 3: X-Coordinate+12 of sprite conflict (low)
Status register 4: X-Coordinate+12 of sprite conflict (high)
Status register 5: Y-Coordinate+8 of sprite conflict (low)
Status register 6: Y-Coordinate+8 of sprite conflict (high)
According to V9958.TXT above X/Y also used for mouse/lightpen (?)

Status register 7: Result from VRAM -> VDP/CPU Commands

```
  Bit 0-7  Color Code
```

Status register 8: Result of Search Command, X-Loc (low)
Status register 9: Result of Search Command, X-Loc (high)

VDP Register Write

Port 99 VDP Register Setup (2nd Byte b7=1) (Write Only)
Port 9A VDP Palette Register (MSX2 only) (2 bytes, Write Only)
Port 9B VDP Register Data (MSX2 only) (Write Only)

Port 99h, VDP Register Setup (Data, Index)
A VDP Register can be changed by writing TWO BYTES to port 99h with BIT 7 SET in the second byte.

```
  Byte 1, Bit 0-7  Data  (New value for the register)
  Byte 2, Bit 0-6  Index (VDP register number) (MSX1: 0-7, MSX2: 0-2E)
  Byte 2, Bit 7    Must be "1" for VDP Register setup
```

Port 9Ah, MVDP MSX2 Color Palette Register
Before writing to this port, select a the Color Number (0-15) by writing to VDP register 10h. Then output the RGB data for the selected color to Port 09Ah. The RGB data consists of two bytes:

```
  Byte 1, Bit 0-2  Blue data (0-7)
  Byte 1, Bit 4-6  Red data (0-7)
  Byte 2, Bit 0-2  Green data (0-7)
  Byte 1, Bit 3,7  Always 0 (no effect)
  Byte 2, Bit 3-7  Always 0 (no effect)
```

Note: The index in VDP register 10h becomes automatically incremented after the second byte has been written.

Port 9Bh, VDP Register Data (Raw Data, Write only)
This register offers an alternate method to access the VDP registers on MSX2 computers. (Normally VDP registers are set by writing TWO bytes (data, index) to Port 99h as described above).

Writing to port 9Bh directly sets the VDP register that is addressed by VDP Register 11h. Afterwards (if the auto increment bit was set) the value in register 11h gets incremented automatically. (See VDP register 11h description for more details).

Example for Port 9Bh: Initializing VDP registers 00h..17h
Write 00h (index 00h, increment=on) to VDP register 11h (using the oldstyle method through port 99h). Now write 18h bytes of data for registers 00h..17h to 9Bh (The data written to Register 11h is ignored, so that it can't damage itself).

VDP Registers 00h-07h: Basic MSX1/MSX2 Video Registers

The VDP register can be (on MSX 1) in the range of 0-7.
On the MSX2 VDP (the one in the SV738 X'Press) it can be in the range of 0-2Eh.

Register 0: Mode register 0

```
  Bit  Name  Expl.
  0    D     External video input       (0=input disable, 1=enable)
  1    M3    Mode M3 (Screen 2,5,7,8)
  2    M4    Mode M4 (Screen 4,5,8,0Hi) (MSX2 only)
  3    M5    Mode M5 (Screen 6,7,8)     (MSX2 only)
  4    IE1   H-Blank Interrupt Enable   (MSX2 only) (see also VDP Reg
13h)
  5    IE2   Light pen/mouse on ???     (MSX2 ONLY not MSX2+)
  6    DG    DiGitize mode              (MSX2 only)
  7    0     Not Used
```

Register 1: Mode register 1

```
  Bit  Name  Expl.
  0    MAG   Sprite zoom                (0=x1, 1=x2)
  1    SZ    Sprite size                (0=8x8, 1=16x16)
  2    0     Not Used
  3    M2    Mode M2 (Screen 3: Block)
  4    M1    Mode M1 (Screen 0: Text)
  5    IE0   V-Blank Interrupt Enable   (0=Disable, 1=Enable)
  6    BLK   Screen output control      (0=Disable, 1=Enable)
  7    416   VRAM size control          (0=4K, 1=16K) (No Function on
MSX)
```

Register 2: BG Map (Name table) base address

```
  Bit     7    6    5    4    3    2    1    0
  Name    0   A16  A15  A14  A13  A12  A11  A10
```

In screen 0-4 this register specifies the base address of the background map (that refers to the tiles that should be displayed as background). In screen 5-8 the register points to the base address of the background bitmap, whereas in screen 5 and 6 only A16 and A15 are used.
In screen 7 and 8 only Bit 5 of the register is used, in these two screens VRAM is

physically separated into two 64K blocks for even/odd addresses, so A15 is the most significant bit, for more info read the details in screen 7 description.

Register 3: BG Colors (Colour table) base address Low

```
  Bit     7    6    5    4    3    2    1    0
  Name   A13  A12  A11  A10  A09  A08  A07  A06
```

In screen 2 and 4 the bits for A12-A06 are ignored, making the address a multiple of 2000h. The register isn't used in screen 5-8, and screen 0/width 40. In screen 0/width 80 the bits for A06-A10 are ignored, making the address a multiple of 800h.

Register 4: BG Tiles (Pattern generator) base address

```
  Bit     7    6    5    4    3    2    1    0
  Name    0    0   A16  A15  A14  A13  A12  A11
```

In screen 2 and 4 the bits for A12 and A11 are ignored, making the address a multiple of 2000h. The register isn't used in screen 5-8.

Register 5: OBJ Attr (Sprite attribute table) base address Low

```
  Bit     7    6    5    4    3    2    1    0
  Name   A14  A13  A12  A11  A10  A09  A08  A07
```

In screen 4-8 (colored sprite mode) the bits for A8 and A7 ignored, making the address a multiple of 200h, in these video modes the register additionally specifies the address of the 'sprite color table' which is always placed 200h bytes before the sprite attribute table. Not used in screen 0.

Register 6: OBJ Tiles (Sprite pattern generator) base address

```
  Bit     7    6    5    4    3    2    1    0
  Name    0    0   A16  A15  A14  A13  A12  A11
```

Not used in screen 0.

Register 7: colour register.

```
  Bit  Name  Expl.
  0-3  TC0-3 Background colour in SCREEN 0 (also border colour in SCREEN
1-3)
  4-7  BD0-3 Foreground colour in SCREEN 0
```

The bits 0-3 and 4-7 can hold a number in the range of 0-15.
The corresponding colours are:

```
  0 = Transparent         8 = Medium red
  1 = Black               9 = Light red
  2 = Medium green        10= Dark yellow
  3 = Light green         11= Light yellow
  4 = Dark blue           12= Dark green
  5 = Light blue          13= Magenta
  6 = Dark red            14= Gray
  7 = Cyan                15= White
```

Color 0 is transparent only if used for foreground text/sprites. But it is visible when it is used for the background itself, whereas the color assigned to color 0 is just black - on MSX2 the palettes for all colors (including color 0) can be redefined though.

In Screen 8 the register contains a 8bit value that specifies the RGB values for the screen border (in same format as for the screen 8 pixels).
In screen 6 this register is having a rather exotic function:

```
  Bit  Name  Expl.
  0-1        Background color A
  2-3        Background color B
  4          Enable Background color B  (0=Use color A only, 1=use both)
```

If both color A & B are used, then the screen border (and any transparent background) becomes drawn as diagonal stripes which are slowly wandering over the screen.

VDP Registers 08h-17h: Additional MSX2 Video Registers

Registers below available in MSX2, MSX2+, and turbo R only.

Register 8: Mode register 2

```
  Bit  Name  Expl.
  0    BW    32 Greylevel MVDPmode out through CompositeVideo output.
             (Normally composite video and RGB are not generated from
this
             output but from another on the MVDP).
  1    SP    Disable OBJ Sprites          (0=On, 1=Disable)
  2-3  VRS   VRAM size and speed
(0=1*16KB,1=4*16KB,2=1*64KB,3=64KB/HighSpeed)
  4    CB    colour Bus direction         (0=Output, 1=Input)
  5    TP    Transparent from palette     (0=Normal, 1=Color 0 is solid)
  6    LCS   Lightpen Select (active 1) connected through colourbus (not
MSX2+)
  7    MSE   Mouse select    (active 1) connected through colourbus (not
MSX2+)
```

Register 9: Mode register 3

```
  Bit  Name  Expl.
  0    DC    Dot Clock Direction          (0=Output, 1=Input) (V9958.TXT)
  1    NT    NTSC                         (0=NTSC/60Hz, 1=PAL/50Hz)
  2    EO    Even Odd Display             (0=Normal, 1=Two screen)
  3    IL    Interlace                    (0=Off, 1=On)
  4-5  S#    Simultaneus mode             (0=Intern, 1=Mix,
2=Extern/Digitize)
  6    0     Always 0
  7    LN    Vertical heigth (pixels)     (0=192, 1=212)
```

Register 0Ah: colour table base address High

```
  Bit     7    6    5    4    3    2    1    0
  Name    0    0    0    0    0   A16  A15  A14
```

Register 0Bh: Sprite attribute base address High

```
  Bit     7    6    5    4    3    2    1    0
  Name    0    0    0    0    0    0   A16  A15
```

Register 0Ch: Inverse/Blink text colour

Bit	Name	Expl.
0-3	BC0-3	Inverse/Blink text background colour
4-7	T20-3	Inverse/Blink text forground colour

The bits 0-3 and 4-7 can hold a number in the range of 0-15.
The corresponding colours are the same as for register 7.

Register 0Dh: Blinking period

Bit	Name	Expl.
0-3	OF0-3	Off blink time (1/5 sec)
4-7	ON0-3	On blink time (1/5 sec)

Register 0Eh: VRAM access (VRAM address select, higher address lines)

Bit	7	6	5	4	3	2	1	0
Name	0	0	0	0	0	A16	A15	A14

Defines the 16K bank for VRAM reads/writes through port 98h.
For MSX2 video modes, this register is automatically incremented when the lower 14 bits in the VRAM read/write pointer overflow. However, for MSX1 compatibility, this does not happen in MSX1 video modes.
See also: Port 99h (with Bit7=0), and MXC bit in VDP Reg 2Dh

Register 0Fh: Status Register Index

Bit	Name	Expl.
0-3	S0-3	Status register number (0-9)
4-7	0	Always 0

For more information read the section about reading from port 99h.

Register 10h: Palette Index

Bit	Name	Expl.
0-3	C0-3	colour palette register number (0-15 for colour 0-15) to receive data from port 09Ah
4-7	0	Always 0

Register 11h: Register pointer
This register offers an alternate method to access the VDP registers. (Normally VDP registers are set by writing TWO bytes (data, index) to port 99h.)

Bit	Name	Expl.
0-5	R0-5	VDP Register Index (for writing data to Port 9Bh)
6	0	Always 0
7	AII	Auto increment VDP index (0=on, 1=off)

For example, to initializing VDP registers 00h..17h, send 00h (index 00h, increment=on) to VDP register 11h (through port 99h). Send 18h bytes data to port 9Bh for registers 00h..17h, whereas the byte written to register 11h will be ignored (the index can't damage itself).

Register 12h: Display adjust

```
Bit  Name  Expl.
0-3  H0-3  Horizontal adjust (0-0Fh, +-8 pixels)
4-7  V0-3  Vertical adjust   (0-0Fh, +-8 pixels)
```

Register 13h: Interrupt line
This register selects the scanline number in which the Horizontal Retrace Interrupt should occur. This may be a scanline that is part of the picture, or of the lower screen border.

```
Bit  Name  Expl.
0-7  IL0-7 Interrupt line (0-255 in 50 Hz mode)
```

In 60Hz mode, the sum of the picture plus lower border lines is less than 256, so that only line 0-EA (192 pix mode) or 0-F4 (212 pix mode) can be used.
Bit 4 in VDP Register 0 must be set to enable this function, and interrupts must be acknowledged by reading from Status Register 1.
The scanline number in this register must be relative to the Display Offset (Vertically Scrolling) in VDP Register 17h.

Register 14h: Colour burst register 1 - always set to 00h ?
Register 15h: Colour burst register 2 - always set to 3Bh ?
Register 16h: Colour burst register 3 - always set to 05h (portar: 15h) ?

Register 17h: Display offset

```
Bit  Name  Expl.
0-7  DO0-7 Display offset Y (0-255).
```

This register vertically scrolls the entire screen - including the foreground sprites. The picture wraps around to line 0 when it reaches the bottom, ie. if the offset is set to 200, then lines 200-255 are displayed (first 56 lines), followed by line 0-155 (remaining 156 lines, for a total of 212 visible lines).

VDP Registers 18h-1Fh: MSX2+/turbo R Video Registers

Register 18h-1Fh: Not used in MSX1 and MSX2
These eight registers haven't been used in MSX2 (and MSX1) models. However, the V9958 video chip of the MSX2+ and turbo R models includes three new registers in this previously unused area. The three new registers are all initialized to "0" upon reset, and the V9958 will function compatibly with V9938 in that state.

Register 19h: 9958 ONLY -- Horizontal Scroll Control

```
Bit Name  Expl.
0   SP2   H-Scroll Screen Width   (0=One page, 1=Two pages)
1   MSK   H-Scroll Mask 8 Pixels  (0=Normal, 1=Hide Leftmost Pixels)
2   WTE   VRAM Access Waitstates  (0=Normal, 1=Enable CPU Waitstate)
3   YJK   YJK Mode Enable         (0=Normal RGB, 1=YJK System)
4   YAE   YJK Attribute Enable    (0=No Attribute, 1=With Attribute)
5   VDS   Pin 8 Output selection  (0=Output CPUCLK, 1=Output /VDS)
```

6	CMD	Video Command Mode	(0=Normal, 1=Screen 2-4 as screen 8)
7	0	Not Used	

The meaning of these new bits is relative complicated, read on below for detailed information.

Register 1Ah: 9958 ONLY -- Horizontal Offset, High (character units)

Bit	7	6	5	4	3	2	1	0
Name	0	0	HO8	HO7	HO6	HO5	HO4	HO3

The screen is shifted TO THE LEFT as specified in 8-dot units (16-dot units in Screen 6-7).
When SP2=0: Scrolling is done within one page and the non-displayed left side of the page is displayed on the right hand side of the screen (HO8 is ignored in this mode).
When SP2=1: Scrolling is done within 2 pages, and when scrolled, the second page appears to the right habd side of the first page, and when scrolled more, the first page reappears to the right of the second page. Note: When SP2=1, the A15 bit of the Pattern Name table base address register should be set to "1" (VDP Register 02h, Bit 5), otherwise only the leftmost page would be displayed.

Register 1Bh: 9958 ONLY -- Horizontal Offset, Low (dot units)

Bit	7	6	5	4	3	2	1	0
Name	0	0	0	0	0	HO2	HO1	HO0

The screen is shifted TO THE RIGHT (unlike as for Register 1Ah which shifts to the left) as specified in 1-dot units (2-dot units in Screen 6-7). When this register is set to a non-zero value, the colors of the leftmost pixel(s) will be undefined, to avoid this dirt effect, the MSK bit must be set, the leftmost 8 pixels (16 pixels in Screen 6-7) will be then covered by the border color.

YJK=0 (YAE=Ignored) - Normal RGB Mode
When YJK is set to zero, colors are generated identically as on MSX2. Ie. Screen 0-7 Background and Sprites are colorized by the 3/3/3 bit RGB values that are defined in the Palette Registers. Screen 8 background is using hardcoded 3/3/2 bit RGB values for background, and hardcoded (1/1/1)*2 bit colors for sprites.

YJK=1 and YAE=0 - YJK Without Attribute
A group of 4 bytes defines the colors of 4 continuous dots (horizontally).

Bit	Dot 0	Dot 1	Dot 2	Dot 3	
0-2	KL	KH	JL	JH	Shared J and K values for all dots
3-7	Y1	Y2	Y3	Y4	Separate Y values for each dot

Each of the four pixels is having its own "Y" value (5 bits), the "J" and "K" values (6 bits each) are shared for all four pixels. It is possible to select between 131072 colors (17 bits), even though each 4 pixels must have similiar colors.
When YJK=1: Sprites in Screen 8 using RGB colors from Palette Registers!

YJK=1 and YAE=1 - YJK (and RGB) With Attribute
A group of 4 bytes defines the colors of 4 continuous dots (horizontally).

Bit	Dot 0	Dot 1	Dot 2	Dot 3	
0-2	KL	KH	JL	JH	Shared J and K values for all dots
3	A	A	A	A	Attribute for each dot

```
  4-7    Y1     Y2     Y3     Y4       Separate Y values for each dot
```

When A=0: Colors are generated as when YAE=0, except that only 65536 colors (16 bits) can be selected because Y-values are reduced to 4 bits.
When A=1: Y is used as color code, selecting one of the 16 RGB colors that are defined in the palette registers. K and L values are not used.
When YJK=1: Sprites in Screen 8 using RGB colors from Palette Registers!

Formulas for YJK / RGB conversion

```
  RGB to YJK:  J=R-Y,  K=G-Y,  Y=B/2+R/4+G/8
  YJK to RGB:  R=Y+J,  G=Y+K,  B=Y*5/4-J/2-K/4
```

Note: Even though up to 131072 colors could be selected by YJK codes, the resulting RGB values are cut down to 5 bits each, so that not more than 32768 colors can be displayed in practice.

WTE - Bit 2 - Wait Function Enable
When WTE is cleared, access to VRAM works in the same way as V9938, ie. the programmer must take care about inserting delays between each VRAM access.
The WAIT function is enabled when WTE is set. When the CPU accesses the VRAM, any following CPU access to any V9958 ports is offhold by a WAIT signal until the VRAM access is completed. (This feature is supposed to speed up the writing time of data from CPU to VRAM.)
However, WAIT function is provided after VRAM access only, not after access to any VDP registers, any VDP status registers, or VDP palette registers.

VDS - Bit 5 - Output selection between CPUCLK and /VDS
Unlike V9938, the V9958 video chip doesn't have two separate pins for CPUCLK and /VDS signals, instead a single pin is shared for both signals. By default CPUCLK is output, alternately, /VDS can be output by setting the VDS bit. (Using /VDS mode in MSX offers no known advantages, but it is reported to confuse some TV tuners and thus shouldn't be used.)

CMD - Bit 6 - Command Function
Normally MSX2 Video Commands (VDP Registers 20h-2Eh) can be used in Screen 5-8 only. However, by setting the CMD bit Video Commands may be used in all other Screen modes. In these 'additional' modes, Video Commands are accessing VRAM bytewise (as in Screen 8), therefore X/Y coordinates as in Screen 8 must be used. (The CMD bit has no influence in Screen 5-8.)

Deleted Functions
The V9958 does not output a Composite Video Signal, and does not include a Mouse/Lightpen Interface. In the result, the following bits have been removed (and should be set to zero): Bit 5 of VDP Reg 00h (IE2), Bit 6 and 7 of VDP Reg 08h (LP and MS), Bit 6 and 7 of VDP Status Reg 01h (LPS and FL).

VDP Registers 20h-2Eh: MSX2 Video Command Registers

The VDP Registers 20h-2Eh are related to MSX2 Video Commands.
Registers below available in MSX2, MSX2+, and turbo R only.

Register 20h: Source X Low byte (0-FF)
Register 21h: Source X High byte (0-1)
Register 22h: Source Y Low byte (0-FF)
Register 23h: Source Y High byte (0-3)
Register 24h: Destination X Low byte (0-FF)
Register 25h: Destination X High byte (0-1)
Register 26h: Destination Y Low byte (0-FF)
Register 27h: Destination Y High byte (0-3)
Register 28h: Number of X dots low byte (0-FF)
Register 29h: Number of X dots high byte (0-3)
Register 2Ah: Number of Y dots low byte (0-FF)
Register 2Bh: Number of Y dots high byte (0-3)

Register 2Ch: Data

```
  Bit  Name  Expl.
  0-7  CL    Color Code
```

For high speed commands, all bits are used, ie. in screen 5-7 more than one pixel can be transferred at once. For all other commands, only one pixel can be specified at once, so in screen 5-7 only the lower 2 or 4 bits of the register are used.

For VDP -> VRAM commands, all pixel(s) are colorized in the same color,
which must have been written to this register before the command started.

For CPU -> VRAM commands, each pixel (or byte in highspeed mode) must be written to this register separately, whereas the first pixel (or byte) should be written <before> the command gets started! When the command has been started, bit 7 of VDP status register 2 indicates whether the VDP controller is ready to receive the next value, and bit 0 indicates if the command has been completed.

Register 2Dh: Argument register

```
  Bit  Name  Expl.
  0    MAJ   Longest side (Line command)  (0=Normal, 1=Exchange X/Y-Len)
  1    EQ    Equal (For Search command)   (0=Repeat until equal, 1=not
equal)
  2    DIX   X step direction             (0=right, 1=left)
  3    DIY   Y step direction             (0=down, 1=up)
  4    MXS   Source external memory       (0=128K VRAM, 1=64K external)
  5    MXD   Destination external memory  (0=128K VRAM, 1=64K external)
  6    MXC   CPU Access external memory   (0=128K VRAM, 1=64K external)
  7    0     Always 0
```

By default MSX2 computers are equipped with 128K VRAM, but the system could be upgraded to additional 64K external VRAM. This additional RAM resides in the same address space as the 1st 64K normal VRAM, bit 4-6 are used to access external memory. The external VRAM cannot be displayed directly - but it can be used as additional workspace for VDP commands by setting the MXS and/or MXD bit(s). External memory could be also read/written through port 98h by setting the MXC bit.

Register 2Eh: Command register

```
  Bit  Name  Expl.
  0-2  AR0-2 Logical argument, see below.        ignored by highspeed
commands
  3    TRN   Transparent (1=Transparent Color 0) ignored by highspeed
commands
  4-7  C0-3  Command, see below.
```

Commands (C0-C3):

```
  Value:  Expl.:
  0       Stop
  4       Get Pixel,    VRAM -> VDP
  5       Put Pixel,    VDP  -> VRAM
  6       Search Pixel, VRAM -> VRAM
  7       Draw Line,    VDP  -> VRAM
  8       Logical Fill Rectangle, VDP  -> VRAM
  9       Logical Copy Rectangle, VRAM -> VRAM
  A       Logical Get Pixels,     VRAM -> CPU
  B       Logical Put Pixels,     CPU  -> VRAM
  C       Highspeed Fill Rectangle,  VDP  -> VRAM
  D       Highspeed Copy Rectangle,  VRAM -> VRAM
  E       Highspeed Copy Vertically, VRAM -> VRAM
  F       Highspeed Put Bytes,       CPU  -> VRAM
```

Arguments (AR0-AR2) (ignored by highspeed commands):

```
  Value:  Name:   Expl.:
  0       PSET    Set bits, set old bits to 0
  1       AND     Mask bits
  2       OR      Set bits, include old
  3       XOR     Swap new bits
  4       NOT     Set new bits to 0
```

Notes about above 'VDP' and 'CPU' Expressions

'CPU' in the above table means that the commands expects more than one value to be read/written manually by the program. And 'VDP' means that only one value is expected to be read/written by the program. In both cases, data is written through VDP Register 2Ch, or read through status register 07h.

Notes about X-Loc and X-Len

Highspeed commands always copy whole bytes, so in screen 6-7 the lower bit(s) of the X-Loc and X-Len operands have no effect.
For the Fill and Copy Commands, a horizontal length of zero is treated as maximum length (that is distance from origin to screen border).

Vertical Copy Command
For the 'Vertical' Copy command, the source X-Loc operand is ignored (the destination X-Loc operand is used for both source and destination). Also, the X-Len operand is ignored (Instead distance from origin to end of screen is used as horizontal length). Otherwise the command is quite the same as the Copy Rectangle command, probably a good bit faster though.

Notes about Line Command
For the 'Line' command, the greater value of the X-Len and Y-Len operands must be always placed into Register 28h and 29h (Major Length) and the smaller value must be placed into Register 2Ah and 2Bh (Minor Length).
For 'vertical' lines, bit 0 of VDP Register 2Dh must be set to indicate that Register 28h and 29h refer to the Y-Len.
Also note that the length values for the Line command specify the relocation from the origin, so a length of 0;0 would draw a single dot. The function allows to draw diagonal lines at any angle.

Display Timings

Horizontal Timings
```
  Scanline Time    : 227.75 cycles
  Scanline Rate    : 15716.99 Hz
```

Vertical Timings, 50Hz Mode
```
  Frame Height     : 313 scanlines
  Frame Time       : 71285.75 cycles
  Exact Frame Rate : 50.214 Hz
  Possible HBL IRQs: Line 00-FF (either 192/212 pix mode)
```

Vertical Timings, 60Hz Mode
```
  Frame Height     : 262 scanlines
  Frame Time       : 59670.5 cycles
  Exact Frame Rate : 59.9885 Hz
  Possible HBL IRQs: Line 00-EA (192 pix mode), 00-F4 (212 pix mode)
```

Above 'cycles' are meant to be counted in CPU clock cycle units, ie. 3.579545MHz units.

VDP Interrupts

Interrupt Sources
The MSX2 includes two interrupt sources: VBlank and HBlank interrupts. For MSX1 only the VBlank interrupt source exist.
A VBlank interrupt is requested each time when the VDP begins to draw the lower screen border (ie. in scanline 192 or 212, depending on the vertical screen resolution).
A HBlank interrupt is requested when the VDP draws the scanline specified in VDP register 13h, this might be any of the 192 or 212 scanlines of the picture - or one of the following scanlines in the lower screen border section. Multiple HBlank Interrupts can be generated by re-writing VDP Register 13h several times per frame.

Interrupt Enable Flags
VBlank and HBlank interrupts can be separately enabled/disabled by bits in VDP Registers 0 and 1. In case that the Z80 has disabled interrupts by clearing the IFF flag (for example by a "DI" instruction) then this becomes priority and VDP interrupt enable flags are ignored even if set.

Interrupt Requests
The VDP requests interrupts by setting the Vblank or Hblank IRQ flags in VDP Status Register 0 and 1 each time when an interrupt condition becomes true. This happens even if interrupts are disabled by the IFF flag and/or by the VBlank interrupt enable flags, however, HBlank IRQs are NOT requested if the HBlank interrupt flag is cleared.

Interrupt Execution
An interrupt is executed only when all of the following conditions are true: The CPU must have enabled interrupts by setting IFF=1. Either the VBlank or HBlank interrupt must be enabled in VDP register 0 or 1, and the respective VBlank or HBlank IRQ flag must be set in VDP Status Register 0 or 1.
The interrupt handler is executed at the time when the above conditions become true. This might directly upon IRQ request in case that the enable flags were set - or directly upon enabling interrupts in case that an IRQ flag was set.

Acknowleding Interrupts
The interrupt handler MUST manually acknowlede interrupts by reading from the respective VDP status register(s) which contain the IRQ flag(s). Otherwise the IRQ flag(s) are kept set, causing the same interrupt to be kept requested - which would be then executed again at the next time when interrupts are enabled.

Sound Generator

I AY-3-8910 Programmable Sound Generator (PSG)

Port A0 PSG Index 00-0Fh (Write Only)
Port A1 PSG Data Write
Port A2 PSG Data Read

The AY-3-8910 is a I/O chip with 3 sound generators.
It controls the three MSX standard audio channels, joystick and cassette.

PSG Registers 00-0Fh are:

00 = Frequency channel A, low (0-255)
01 = Frequency channel A, high (0-15)
02 = Frequency channel B, low (0-255)
03 = Frequency channel B, high (0-15)
04 = Frequency channel C, low (0-255)
05 = Frequency channel C, high (0-15)
The actual listened frequency in Hertz is calculated as follows:

```
  F = 3.579545MHz / 32 / nn      ;with nn in range 0..4095
```

06 = Noise period (0-31)
The actual noise frequency in Hertz is calculated as follows:

```
  F = 3.579545MHz / 32 / nn      ;with nn in range 0..31
```

07 = Mixer

```
  Bit Expl.
  0   Channel A tone enable       (0=Enable,1=Disable)
  1   Channel B tone enable       (0=Enable,1=Disable)
  2   Channel C tone enable       (0=Enable,1=Disable)
  3   Channel A noise enable      (0=Enable,1=Disable)
  4   Channel B noise enable      (0=Enable,1=Disable)
  5   Channel C noise enable      (0=Enable,1=Disable)
  6   I/O port A mode             (0=Input, 1=Output)
  7   I/O port B mode             (0=Input, 1=Output)
```

08 = Volume channel A (0-15, 16=Envelope)
09 = Volume channel B (0-15, 16=Envelope)
0A = Volume channel C (0-15, 16=Envelope)

0B = Envelope Frequency, low (0-255)
0C = Envelope Frequency, high (0-255)
Envelope step frequency (tone or noise) calculated as follows:

```
  F = 3.579545MHz / 32 / nn     ;with nn in range 0..65535
```

Depending on the envelope shape, the volume is incremented from 0 to 15, or decremted from 15 to 0. In either case it takes 16 steps to complete, the completion time for 16 steps is therefore:

```
  T = nn*512 / 3.579545MHz      ;with nn in range 0..65535 (0-9.37
seconds)
```

0D = Envelope shape (0-15)

```
  CONT ATT ALT HLD
    0   0   X   X  \_________  0-3 (same as 9)
    0   1   X   X  /_________  4-7 (same as F)
    1   0   0   0  \\\\\\\\\\  8   (Repeating)
    1   0   0   1  \_________  9
    1   0   1   0  \/\/\/\/\/  A   (Repeating)
    1   0   1   1  \"""""""""  B
    1   1   0   0  //////////  C   (Repeating)
    1   1   0   1  /"""""""""  D
    1   1   1   0  /\/\/\/\/\  E   (Repeating)
    1   1   1   1  /_________  F
```

0E = I/O port A (Joystick and Cassette Input)

```
  Bit Expl.
  0   Joystick Up                (0=Moved, 1=Not moved)
  1   Joystick Down              (0=Moved, 1=Not moved)
  2   Joystick Left              (0=Moved, 1=Not moved)
  3   Joystick Right             (0=Moved, 1=Not moved)
  4   Joystick button A          (0=Pressed, 1=Not pressed)
  5   Joystick button B          (0=Pressed, 1=Not pressed)
  6   Keyboard Switch            (Japanese SVI machines only ?)
  7   Cassette input
```

0F = I/O port B (Joystick Select Output)

```
  Bit Expl.
  0   1         (Used as handshaking output if touchpad)
  1   1         (Used as handshaking output if touchpad)
  2   1         (Used as handshaking output if touchpad)
  3   1         (Used as handshaking output if touchpad)
  4   Pulse 1   (Positive pulse starting a monostable timer)
  5   Pulse 2   (Positive pulse starting a monostable timer)
  6   Joystick select            (0=Connector 1, 1=Connector 2)
  7   LED       Code LED, if any  (0=On, 1=Off)
```

Bits 4 and 5 is used by a program which uses a paddle (analog-joystick). A short positive edge pulse on bit 4 (or 5) starts a monostable timer (in the attached paddle) and the paddle sets one of the joystick bits in register 14 low (FIRE A (FIRE B),L,R,D or U). When the monostable times out, the joystick bit in port 14 is set high again. The length of the counting period of the monostable timer is set (in the attached paddle) by a variable resistor. The computer can determine the position of the variable resistor by measuring the time while the joystick bit in register 14 is low.

The Code LED is included in models that have a locking function for the CODE key, such like Japanese, Russian, or Korean models which are enabling the native character set by that key, and by some US/European MSX1 BIOSes. Otherwise, if the Code key is active only when held down, no Code LED is included.

Cartridge Memory Mappers

Raw ROM without Mapper
Small cartridges with only 32Kbytes or less ROM aren't including memory mappers, these ROMs typically occupy the address space at 4000-BFFF.

```
Memory     Content (not mappable)
0000-3FFF  sometimes mirror of 1st 16KB of ROM
4000-7FFF  1st 16KB of ROM
8000-BFFF  2nd 16KB of ROM (eventually 1st 16K if size less than 32K)
```

Theoretically 64K ROMs aren't requiring a memory mapper as they could occupy the whole address space from 0000-FFFF, however, for whatever reason, most or all 64K ROMs are using Ascii 16K mappers.

Konami 8K without SCC
This type is used by Konami cartridges that do not have a SCC and some others.

```
Memory     Mapper I/O Address
4000-5FFF  fixed, always bank 0
6000-7FFF  select bank by writing to 6000
8000-9FFF  select bank by writing to 8000
A000-BFFF  select bank by writing to A000
```

Konami 8K with SCC
This type is used by Konami cartridges that do have a SCC and some cartridges not made by Konami (and that are NOT including a SCC ?).

```
Memory     Mapper I/O Address
4000-5FFF  select bank by writing to 5000-57FF (5000 used)
6000-7FFF  select bank by writing to 7000-77FF (7000 used)
8000-9FFF  select bank by writing to 9000-97FF (9000 used)
A000-BFFF  select bank by writing to B000-B7FF (B000 used)
```

If it is a Konami cartridge, you can use the SCC ("Sound Custom Chip") by writing a value with bits 0-5 set (3Fh, bits 6 and 7 do not matter) to 9000h-97FFh, you can then read and write to the SCC in the memory area 9800h-AFFFh.

ASCII 8KB
Used by various games. A few cartridges of this type may also contain 8K SRAM, which is selected by setting one of the upper bank number bits. (For example, bank=20h for Xanadu, bank=80h for Royal Blood.)

```
Memory     Mapper I/O Address
4000-5FFF  select bank by writing to 6000-67FF (6000 used)
6000-7FFF  select bank by writing to 6800-6FFF (6800 used)
8000-9FFF  select bank by writing to 7000-77FF (7000 used)
A000-BFFF  select bank by writing to 7800-7FFF (7800 used)
```

Note that writing to 6000-7FFF is used for mapper I/O, therefore SRAM can be only written to when mapped to 8000-9FFF or A000-BFFF (or maybe also 4000-5FFF ?).

ASCII 16KB

Among others this type is often used by many 64KB cartridges.

```
  Memory     Mapper I/O Address
  4000-7FFF  select bank by writing to 6000-67FF (or 6000-6FFF ?) (6000
used)
  8000-BFFF  select bank by writing to 7000-77FF (or 7000-7FFF ?) (7000
used)
```

The game Hydlide 2 also includes 2KB of SRAM (selected by bank number 10h), SRAM can be written to only when mapped to 8000-BFFF, the SRAM is mirrored to all 2K fragments of the 16K bank.

Other Mappers

A few cartridges are using their own exotic memory mappers which aren't described here, just note that the above standards do not apply for all cartridges.

Floppy Disk Controller

Disk based MSX computers are typically using a Western Digital FD1793 Floppy Disk Controller (FDC). MSX2 computers are (usually) including a built-in disk drive, and diskless MSX1 computers can be upgraded by external disk drives.

FDC I/O Addresses

Disk drives have not been part of the first MSX computers, and obviously different companies invented their own disk adapters independendly of each other. In result there are a couple of different 'world standards'.

Described below are different memory-mapped and I/O-based addresses, that are used by different adapters. (For memory mapped adapters the Disks PSLOT (and/or SSLOT) must be selected, the addresses may then be accessed by LD commands. Less popular I/O based adapters are accessed by IN and OUT commands.)

7FFX - Memory mapped IO addresses
Used by various distributors: Sharp, Philips, ACVS/CIEL.

```
  7FF8h    R    Status Register
  7FF8h    W    Command Register
  7FF9h    R/W  Track Register
  7FFAh    R/W  Sector Register
  7FFBh    R/W  Data Register
  7FFCh    R?/W Side (bit 0)     Motor here?
  7FFDh    R    ??
  7FFDh    W    Drive (bit 0)    Motor here?
  7FFEh    -    Unused
  7FFFh    R    Data Request (bit 7), Busy (bit 6)
```

Note: MSX-DOS/BarbarianLoader selects memory at 8000-BFFF into the disks PSLOT, and then accesses the disk via addresses BFFX rather than 7FFX.

7FBX - Memory mapped IO addresses for a SV738 (X'press) disk
These addresses are used by 'Technoahead' disk BIOS.

```
  7FB8h    R    Status Register
  7FB8h    W    Command Register
  7FB9h    R/W  Track Register
  7FBAh    R/W  Sector Register
  7FBBh    R/W  Data Register
  7FBCh    R    IRQ/Not Busy (bit 7), Data Request (bit 6)
  7FBCh    W    Select Drive 0/1 (bit 0/1), Side (bit 2), Motor (bit 3)
  7FBDh-Fh -    Unused
```

7F8X - Memory mapped IO addresses for Arabic Disk ROM
Same as above 7FB8-7FBC, but using addresses 7F80-7F84 instead.

I/O Ports for Brazilian Disk ROMs
Used by various (brazilian only) distributors: Digital Design Electronica Ltda, Conector Informatica, Microsol Tecnologia, Liftrom Informatica.

```
  D0h      R    Status Register
  D0h      W    Command Register
  D1h      R/W  Track Register
  D2h      R/W  Sector Register
  D3h      R/W  Data Register
  D4h      W    Drive (bit 1), Side (bit 4), Motor (bit ??)
  D4h      R    IRQ/Not Busy (bit 7), Data Request (bit 6) (V3.0 and up)
  D5h-D7h  -    Unused
```

Note: Reading from Port D4h required/supported by V3.0 or newer interfaces only, older devices used status register bits 0 and 1 which have identical meaning.
V2.7 and newer are reported to be "Mixed Port and Memory Based", however, these Disk BIOS versions appear to include Port Based code only, but possibly the hardware itself recognizes Memory Based software?

Note: As you may have noticed, the first four Memory Addresses or I/O Ports are always controlling the Status/Command, Track, Sector, and Data Registers, that is because these addresses are directly accessing the FDC registers. The four higher addresses control custom, non-standard circuits made by the drive manufacturer.

FDC Description

Command description
Commands should only be loaded in the Command Register when the Busy status bit is off (Status bit 0). The one exception is the Force Interrupt command. Whenever a command is being executed, the Busy status bit is set.
At the completion of every command an INTRQ is generated. INTRQ is reset by either reading the status register or by loading the command register with a new command. In addition, INTRQ is generated if a Force Interrupt command condition is met.
When a command is completed, an interrupt is generated and the busy status bit is reset. The Status Register indicates whether the completed command encountered an error or was fault free.

Command Summary (models 1791, 1792, 1793, 1794)

```
  Type Command        b7 b6 b5 b4 b3 b2 b1 b0
  I    Restore        0  0  0  0  h  V  r1 r0
  I    Seek           0  0  0  1  h  V  r1 r0
  I    Step           0  0  1  T  h  V  r1 r0
  I    Step-In        0  1  0  T  h  V  r1 r0
  I    Step-Out       0  1  1  T  h  V  r1 r0
  II   Read Sector    1  0  0  m  S  E  C  0
```

```
II   Write Sector    1  0  1  m  S  E  C  a0
III  Read Address    1  1  0  0  0  E  0  0
III  Read Track      1  1  1  0  0  E  0  0
III  Write Track     1  1  1  1  0  E  0  0
IV   Force Interrupt 1  1  0  1  i3 i2 i1 i0
```

Flag Summary

```
r1,r0  Stepping Motor Rate (0: 6ms, 1: 12ms, 2: 20ms, 3: 30 ms)
V      Track Number Verify Flag (0: no verify, 1: verify on dest track)
h      Head Load Flag (1: load head at beginning, 0: unload head)
T      Track Update Flag (0: no update, 1: update Track Register)
a0     Data Address Mark (0: FB, 1: F8 (deleted DAM))
C      Side Compare Flag (0: disable side compare, 1: enable side comp)
E      15 ms delay (0: no 15ms delay, 1: 15 ms delay)
S      Side Compare Flag (0: compare for side 0, 1: compare for side 1)
m      Multiple Record Flag (0: single record, 1: multiple records)
```

Interrupt Condition Flags

```
i3-i0  0 = Terminate with no interrupt (INTRQ)
i3     1 = Immediate interrupt, requires a reset
i2     1 = Index pulse
i1     1 = Ready to not ready transition
i0     1 = Not ready to ready transition
```

Type I commands (Restore, Seek, Step-Out, Step-In, Step)

Used to move the read/write head. The stepping motor rate should be usally set to 6ms (r1 and r0 bits set to zero) for a 3.5-inch floppy disk drive. An optional verification of head position can be performed by setting bit 2 (V=1) in the command word.

When V=1: When the seek is completed, the drive automatically goes into read mode, the track number from the first encountered Sector ID Field is then compared against the contents of the Track Register (or Data Register?), if the track numbers compare (and the ID Field CRC is correct) the verify operation is complete and an INTRQ is generated with no errors. If these track numbers do not match, the Seek Error bit of the status register is set.

When V=0: When the seek is completed, the track position is not verified (this mode is required for unformatted disks). The command ends once the last step pulse is output. Since the result is that there is not enough time for step stability, the host system must use its software to make the floppy disk wait a certain period before reading or writing the track just arrived at.

When the seek command is complete, the interrupt request is set and at the same time, the Busy bit in the status register is set to 0. When the CPU reads the status register, it resets the interrupt request signal.

Restore (Seek Track 0)

If TR00 is not active, stepping pulses are issued until the TR00 input is activated. The Track Register is set to zero, and an interrupt is generated when track 0 has been reached.

Seek
This command assumes that the Track Register contains the <current> position of the head, and that the Data Register contains the <desired> destination track number.The FD179X will update the Track Register and issue stepping pulses in the appropriate direction until the contents of the Track and Data Register are equal to each other. An interrupt is generated at the completion of the command.

Step-Out, Step-In, Step
Issues one stepping pulse to the disk drive. Step-Out: towards track 0. Step-In: towards track 76. Step: same direction as for previous step command. The track register is updated (ie. incremented or decremented) only if the "T" bit has been set in the command word. An interrupt is generated at the end of each command.

Type II commands (Read Sector, Write Sector)
Prior to loading the Type II command into the Command Register, the computer must load the Sector Register with the desired sector number. Upon receipt of the Type II command, the busy status bit is set. The FD179X must find an ID field with a matching Track number and Sector number, otherwise the Record not found status bit is set and the command is terminated with an interrupt.
Each of the Type II commands contains an m flag which determines if multiple records (sectors) are to be read or written. If m=0, a single sector is read or written and an interrupt is generated at the completion of the command. If m=1, multiple records are read or written with the sector register internally updated so that an address verification can occur on the next record. The FD179X will continue to read or write multiple records and update the sector register in numerical ascending sequence until the sector register exceeds the number of sectors on the track or until the Force Interrupt command is loaded into the Command Register.
The Type II commands for 1791-94 also contain side select compare flags. When C=0 (bit 1), no comparison is made. When C=1, the LSB of the side number is read off the ID Field of the disk and compared with the contents of the S flag.

Read Sector
Upon receipt of the command, the head is loaded, the busy status bit set and when an ID field is encountered that has the correct track number, correct sector number, correct side number, and correct CRC, the data field is presented to the computer.
An DRQ is generated each time when the CPU must read a byte from the data register, the Lost Data bit is set if the CPU didn't read data in time, and the Read operation continues until the end of sector is reached.
At the end of the Read operation, the type of Data Address Mark encountered in the data field is recorded in the Status Register (bit 5).

Write Sector
Upon receipt of the command, the head is loaded, the busy status bit set and when an ID field is encountered that has the correct track number, correct sector number, correct side number, and correct CRC, a DRQ is generated.
The FD179X counts off 22 bytes (in double density) from the CRC field and the Write Gate output is made active if the DRQ is serviced (ie. the DR has been loaded by the

computer). If DRQ has not been serviced, the command is terminated and the Lost Data status bit is set. If the DRQ has been serviced, 12 bytes of zeroes (in double density) are written to the disk, then the Data Address Mark as determined by the a0 field of the command.
The FD179X then writes the data field and generates DRQ's to the computer. If the DRQ is not serviced in time for continuous writing the Lost Data Status bit is set and a byte of zeroes is written on the disk and the command continues until the last byte of the sector is reached.
After the last data byte has been written on the disk, the two-byte CRC is computed internally and written on the disk followed by one byte of logic ones.

Type III commands

Read Address

Upon receipt of the Read Address command, the head is loaded and the Busy Status bit is set. The next encountered ID field is then read in from the disk, and the six data bytes of the ID field are assembled and transferred to the DR, and a DRQ is generated for each byte. The six bytes of the ID field are : Track address, Side number, Sector address, Sector Length, CRC1, CRC2. Although the CRC bytes are transferred to the computer, the FD179X checks for validity and the CRC error status bit is set if there is a CRC error. The track address of the ID field is written into the sector register so that a comparison can be made by the user. At the end of the operation, an interrupt is generated and the Busy status bit is reset.

Read Track

Upon receipt of the Read Track command, the head is loaded, and the busy status bit is set. Reading starts with the leading edge of the first encountered index pulse and continues until the next index pulse. All gap, header, and data bytes are assembled and transferred to the data register and DRQ's are generated for each byte. The accumulation of bytes is synchronized to each address mark encountered. An interrupt is generated at the completion of the command. The ID Address Mark, ID field, ID CRC bytes, DAM, Data and Data CRC bytes for each sector will be correct. The gap bytes may be read incorrectly during write-splice time because of synchronization.

Write Track (formatting a track)

Upon receipt of the Write Track command, the head is loaded and the Busy Status bit is set. Writing starts with the leading edge of the first encountered index pulse and continues until the next index pulse, at which time the interrupt is activated. The Data Request is activated immediately upon receiving the command, but writing will not start until after the first byte has been loaded into the DR. If the DR has not been loaded by the time the index pulse is encountered, the operation is terminated making the device Not Busy, the Lost Data status bit is set, and the interrupt is activated. If a byte is not present in the DR when needed, a byte of zeroes is substituted. This sequence continues from one index mark to the next index mark.
Normally, whatever data pattern appears in the data register is written on the disk with a normal clock pattern. However, if the FD179X detects a data pattern of F5 thru FE in the data register, this is interpreted as data address marks with missing clocks or CRC

generation. The CRC generator is initialized when an F5 data byte is about to be transferred (in MFM). An F7 pattern will generate two CRC bytes. As a consequence, the patterns F5 thru FE must not appear in the gaps, data fiels, or ID fiels. Tracks may be formatted with sector lengths of 128, 256, 512 or 1024 bytes. See "Formatting" below for more info and example.

Type IV command (Force Interrupt)
The Forced Interrupt command is generally used to terminate a multiple sector read or write command or insure Type I status register. This command can be loaded into the command register at any time. If there is a current command under execution (busy status bit set), the command will be terminated and the busy status bit reset.

Status Register
Upon receipt of any command, except the Force Interrupt command, the Busy Status bit is set and the rest of the status bits are updated or cleared for the new command.
The user has the option of reading the status register through program control or using the DRQ line with DMA or interrupt methods. When the DR is read the DRQ bit in the Status register and the DRQ line are automatically reset. A write to the DR also causes both DRQ's to reset. The busy bit in the status may be monitored with a user program to determine when a command is complete, in lieu of using the INTRQ line. When using the INTRQ, a busy status check is not recommended because a read of the status register to determine the condition of busy will reset the INTRQ line.

Status for Type I commands
```
  Bit Expl.
  0   Busy         (1=Command is in progress)
  1   Index        (1=Index mark detected from drive)
  2   Track 0      (1=Read/Write head is positioned to Track 0)
  3   CRC Error    (1=CRC encountered in ID field)
  4   Seek Error   (1=Desired track was not verified)  (reset 0 when
updated)
  5   Head Loaded  (1=Head loaded an engaged)
  6   Protected    (1=Disk write protected)
  7   Not Ready    (1=Drive not ready)
```

Status for type II & III commands
```
  Bit Expl.
  0   Busy         (1=Command is under execution)
  1   Data Request (1=CPU must read/write next data byte) (DRQ)
  2   Lost Data    (1=CPU did not respond to DRQ in one byte time)
  3-4 Error Code   (1=Bad Data CRC, 2=Sector not found, 3=Bad ID Field
CRC)
  5   Fault/Type   (Any Write:1=Write Fault, Read Sector:1=Deleted Data
Mark)
  6   Protected    (Any Write:1=Write Protect, Any Read:Not used)
  7   Not Ready    (1=Drive not ready)
```
Notes: Bit 1-6 are reset when updated. Bit 3-4 not used for read/write track.

Status for type IV command
If the Force Interrupt command is received while a command is under execution, the Busy status bit is reset and the rest of the status bits are unchanged.
If the Force Interrupt command is received when there is no command under execution, the Busy Status bit is reset and the rest of the status bits are updated or cleared. In this case, Status reflects the Type I commands.

External Circuit
The floppy disk controller itself cannot select the drive number, side, disk density, and it cannot turn on/off the drive motor(s). These settings must be set up by an external circuit, which are (as far as they aren't set to a fixed setting) controlled through separate I/O addresses. For details have a look at the MSX FDC I/O addresses.

Note that the FDC contains only one track register which is used for all drives. The current track position for each drive should be backed up in memory, and the track register should be updated each time when changing the current drive number.

The external circuit might also use the INTRQ and DRQ lines to handle FDC operations by DMA transfers, and/or to produce interrupts upon completion. In the MSX these methods are not used, however, most MSX adapters allow to read out the state of INTRQ and DRQ from a custom I/O address (this isn't actually required because INTRQ is just a inverted copy of the Status Busy bit, and DRQ can be read out from the normal Status register either.

Formatting
This table shows DATA PATTERNs and their FD179X interpretation in MFM.

```
00-F4   Write 00 thru F4
F5      Write A1, preset CRC
F6      Write C2
F7      Generate 2 CRC bytes
F8-FF   Write F8 thru FF
```

Formatting Example
The example below shows the data stream that must be presented to the Write Track command for the "IBM system 34 format" (256 bytes/sector), note that the MSX usually uses 512 bytes/sector. The left values in the tables below identify the write-repeat count (in decimal) for the values to the right. First the Track Header must be written, followed by Sector ID and Sector Data Fields (for each sector). Finally 4E bytes must be written until the command has completed.

Track Header

```
80 x 4E
12 x 00
 3 x F6 (writes C2)
 1 x FC (index mark)
50 x 4E
```

Sector ID Field

```
  12 x 00
   3 x F5 (writes A1, preset CRC)
   1 x FE (ID address mark)
   1 x Track number
   1 x Side number
   1 x Sector number
   1 x 01 (sector length=256)
   1 x F7 (write 2 CRC bytes)
  22 x 4E
```

Sector Data Field

```
  12 x 00
   3 x F5 (writes A1, preset CRC)
   1 x FB (data address mark)
 256 x DATA
   1 x F7 (write 2 CRC bytes)
  54 x 4E
```

Track End (Fill unused bytes)

```
  .. x 4E
```

Disk FAT Format

The usual MSX floppy format is compatible to that used under DOS on PCs, with the limitation that MSX BASIC doesn't support sub-directories. Typical formats are 3.5", Double Density, 80 Tracks/9 Sectors, either Single Sided (360KB) or Double Sided (720KB). The Sectors are logically numbered 01h..09h, and each sized 200h bytes.

Boot-Record
The first sector is always used as bootsector, giving information about the usage of the following sectors, and including the boot procedure (usually a short program that loads MSXDOS.SYS, or just a Z80 RET opcode (C9h) if the disk isn't bootable).

```
  00-02       jump to 80x86 boot procedure (not used for MSX, but see
below)
  03-0A       ascii disk name
  0B-0C       bytes / sector
  0D          sectors / cluster
  0E-0F       sectors / boot-record
  10          number of FAT-copys
  11-12       entrys / root-directory
  13-14       sectors / disk
  15          ID: F8=hdd, F9=3.5", FC=SS/9sec, FD=DS9, FE=SS8,FF=DS8
  16-17       sectors / FAT
  18-19       sectors / track
  1A-1B       heads / disk
  1C-1D       number of reserved sectors
  1E-1FF      MSX boot procedure (loaded to address C01Eh in RAM)
```

The first byte of the "jump to 80x86 boot procedure" entry must be either E9h or EBh, otherwise the MSX BIOS ignores the MSX boot procedure at 1Eh.

FAT and FAT copy(s)

The following sectors are occupied by the File Allocation Table (FAT), which contains 12- or 16-bit entries for each cluster:

```
  (0)000      unused, free
  (0)001      ???
  (0)002...   pointer to next cluster in chain (0)002..(F)FEF
  (F)FF0-6    reserved (no part of chain, not free)
  (F)FF7      defect cluster, don't use
  (F)FF8-F    last cluster of chain
```

Number and size of FATs can be calculated by the information in the boot sector.

Root directory

The following sectors are the Root directory, again, size depends on the info in bootsector. Each entry consists of 32 bytes:

```
  00-07       Filename (first byte: 00=free entry,2E=dir, E5=deleted
entry)
  08-0A       Filename extension
  0B          Fileattribute
  0C-15       reserved
  16-17       Timestamp: HHHHHMMM, MMMSSSSS
  18-19       Datestamp: YYYYYYYM, MMMTTTTT
  1A-1B       Pointer to first cluster of file
  1C-1F       Filesize in bytes
```

The 'cluster' entry points to the first used cluster of the file. The FAT entry for that cluster points to the next used cluster (if any), the FAT entry for that cluster points to the next cluster, and so on.

Reserved Sectors (if any)

Usually the number of reserved sectors is zero, but if it has been non-zero, then the following sector(s) are reserved (and could be used by the boot procedure for whatever purposes for example).

Data Clusters 0002..nnnn

Finally all following sectors are data clusters. The first cluster is called cluster number (0)002, followed by number (0)003, (0)004, and so on.

Real Time Clock

Port B4 RP 5C01 RTC Index Write
Port B5 RP 5C01 RTC Data Read/Write

This battery buffered Real Time Clock (RTC) isn't installed in all MSX computers, it could be often found in models with built-in (or external) disk drive.

Port 0B4h = RTC Index register (write only)

Bit	Name	Expl.
0-3	?	RTC register (0-15)
4-7	0	Not used

Port 0B5h = RTC data register (read/write)

Bit	Name	Expl.
0-3	?	RTC data read/write
4-7	0	Not used

The above index register only offers to access 16 RTC data registers, however, beside for the time and date, the RTC chip additionally stores other data (like the RTC/CMOS chips in PCs) and alltogether that exceeds the number of 16 4bit-registers.
The register at index D is used to sub-class registers at index 0-C into 4 blocks with contents as follows:

```
       Block 0       Block 1       Block 2       Block 3
Index  (BCD Timer)   (BCD Alarm)   (Screen)      (Ascii)
-----  ------------  ------------  ------------  ------------
0      Seconds, low  ---           Scratch       Type
1      Seconds, hi   ---           X-Adjust      Char 1, low
2      Minutes, low  Minutes, low  Y-Adjust      Char 1, hi
3      Minutes, hi   Minutes, hi   Screen        Char 2, low
4      Hours, low    Hours, low    Width, low    Char 2, hi
5      Hours, hi     Hours, hi     Width, hi     Char 3, low
6      Day of Week   Day of Week   Color, Text   Char 3, hi
7      Day, low      Day, low      Color, BG     Char 4, low
8      Day, hi       Day, hi       Color, Border Char 4, hi
9      Month, low    ---           Cas/Prn/Key   Char 5, low
A      Month, hi     12/24 hours   Beep Frq/Vol  Char 5, hi
B      Year, low     Leap Year     Color, Title  Char 6, low
C      Year, hi      ---           Native Code?  Char 6, hi
D   Mode Register  (Read/Write)
E   Test Register  (Write Only)
F   Reset Register (Write Only)
```

Mode Register (Index 0Dh)

```
Bit Name Expl.
0-1 M0-1 Select Block    (0-3)
 2  AE   Alarm enable    (0=Disable, 1=Enable) (not used in MSX)
 3  TE   Enable Seconds  (0=Freeze, 1=Active)
```

Test Register (Index 0Eh)

```
 Bit Name Expl.
  0  T0   Increment Seconds at a rate of 16384Hz
  1  T1   Increment Minutes ""
  2  T2   Increment Hours   ""
  3  T3   Increment Days    ""
```

This can be used to confirm that date and time carries are done correctly.

Reset Register (Index 0Fh)

```
 Bit Name Expl.
  0  AR   Reset all Alarm Registers to zero     (1=Reset)
  1  CR   Reset fractions smaller than 1 second (1=Reset)
  2  C16  Enable 16Hz clock output              (0=Enable?)
  3  C1   Enable 1Hz clock output               (0=Enable)
```

Note: The clock pulse outputs aren't used in MSX.

Block 0 and 1

These blocks are used by the clock, all values are stored in BCD format. Hours are counted from 0-23 when the 12/24 hour bit is set. Alternately hours could count from 0-11 (AM), and then from 20-31 (PM) when 12/24 hour bit is cleared. In the MSX the 24 hour mode is used (and any conversions to 12 hour clocks are done by software).
The Year entry is 1980 based, ie. values 00-99 mean 1980-2079. The RTC expects a leap year when the lower two bits of the leap year counter are zero, as the MSX counts from 1980 onwards (which has been a leap year), the leap year counter should be initialized identical to the lower bits of the year counter.

Block 2 and 3

These blocks may contain any custom data. The MSX2 BIOS them as follows: The scratch byte to detect the presence of the RTC chip, the other bytes in block 2 to set up the initial video mode settings, as well as the default cassette baud rate, key click, beep sound, etc.
Block 3 may be used either to define the Title (displayed in boot screen, type=0), or the Password (verified in boot screen, type=1), or the BASIC Prompt (usually "Ok" ,type=2). Type 0 and 2 are stored as 6 character ascii string, type 1 is stored in form of a 16bit CRC together with an optional 'key cartridge value'.

RS 232 Interface

The RS 232 interface (serial communications port) is accessed through Port 80h-87h, as described in the chapters below.

Note that most MSX computers do not include a built-in RS 232 adapter.
Beside for the actual hardware, a complete MSX RS232 interface should also include a RS232 ROM.

ACIA Data, Status, Mode, Command

I8251 Asynchronous Communication Interface Adapter (ACIA)

Port 80 I8251 (ACIA) Data Register (Read/Write)
Port 81 I8251 (ACIA) Status Register (Read Only)
Port 81 I8251 (ACIA) Mode/Command Register (Write Only)

Data Register (Port 80h, Read/Write)
This port holds the byte received from RS232.
This port is also used to send bytes to RS232.

Status Register (Port 81h, Read Only)
```
  Bit  Name    Desc.
  0    TxRDY   Transmit Ready
  1    RxRDY   Receive Ready    (1=Received byte may be read from Port
80h)
  2    TxEMPTY Transmit buffer Empty
  3    PE      Parity Error if 1
  4    OE      Overrun Error if 1 (CPU has not received character)
  5    FE      Framing Error if 1 (ASYNC only, STOP BIT not valid)
  6    SYNDET/BRKDET  SYNC/BREAK found
  7    DSR     Data Set Ready
```

Mode/Command Setup (Port 81h, Write Only)
As described below, this port is used both as Mode Setup and Command Setup port. The configuration procedure is as follows: Output one or more zero bytes (to ensure that the port is in Command state). Output 40h to reset the port into Mode Setup state. Output the desired Mode Setup value. Now the port is switched back into Command state. Output 10h to clear the error flags, and then initialize the desired state of the TX, RX, DTR bits.

Mode Setup Register (Port 81h, Write Only, After Reset Only)
Directly after a RESET command, the MODE SETUP byte may be written:

```
  Bit  Name     Expl.
  0-1  BAUDFACT Baud Fact  (0=Sync Mode, 1=1.8432MHz, 2=115.2kHz, 
3=28.8kHz)
  2-3  WORD     Number of Data Bits      (0=5bits, 1=6bits, 2=7bits, 
3=8bits)
  4    PAREN    Parity Bit Enable        (0=No Parity Bit, 1=One Parity 
Bit)
  5    PARITY   Parity Generation/Check  (0=Odd, 1=Even)
  6-7  STOP     Number of Stop Bits  (0=Invalid, 1=1bit, 2=1.5bits, 
3=2bits)
```

Command Setup Register (Port 81h, Write Only, Not after Reset)
Once the MODE SETUP byte has been written, COMMAND bytes may be written:

```
  Bit  Name     Expl.
  0    TX       Transmit Enable         (0=Disable, 1=Enable)
  1    DTR      Set Data Terminal Ready (0=No, 1=Ready)
  2    RX       Receive Enable          (0=Disable, 1=Enable)
  3    BRK      Break Operation         (0=No, 1=Send Break (TxD=Low))
  4    ERR_RES  Reset Error Flags       (0=No, 1=Reset Error Flags 
PE,OE,FE)
  5    RTS      Set Request to Send     (0=No, 1=Request to Send)
  6    RESET    Internal Reset          (0=No, 1=Reset and wait for MODE 
SETUP)
  7    HUNT     Enter Hunt Mode         (0=No, 1=Search for SYNC 
character)
```

Status, Interrupt Mask

Port 82 RS232 Status for CTS,Timer,RI,CD (Read Only)
Port 82 RS232 Interrupt mask register (Write Only)
Port 83 RS232 ?Clock 0,1,2 read? (Read Only)
Port 83 RS232 ?Receive ready interrupt enable? (Write Only

Status for CTS,Timer,RI,CD (Port 82h, Read Only)

```
  Bit Name  Expl.
  0   CD    Carrier Detect   (0=Active, 1=Not active)
  1   RI    Ring Indicator   (0=Active, 1=Not active) (N/C in MSX)
  6         Timer Output from i8253 Counter 2
  7   CTS   Clear to Send    (0=Active, 1=Not active)
```

Interrupt Mask Register (Port 82h, Write Only)

```
  Bit   Expl.
  0     Receive data ready    (0=Enable Interrupt, 1=Disable)
  1     Transmit data ready   (0=Enable Interrupt, 1=Disable) (N/C in 
MSX)
  2     Sync/Break found      (0=Enable Interrupt, 1=Disable) (N/C in 
MSX)
```

```
  3     i8253 channel 2 Timer (0=Enable Interrupt, 1=Disable) (N/C in
MSX)
```

??? Clock 0,1,2 read (Port 83h, Read Only)
??? Receive ready interrupt enable (Port 83h, Write Only)
Sorry, no info. TH-AP.TXT says that Port 83h is unused.

8253 Baud Rate Generator

Port 84 PCI i8253 Counter 0 (Receive clock)
Port 85 PCI i8253 Counter 1 (Transmit clock)
Port 86 PCI i8253 Counter 2 (Custom clock)
Port 87 PCI i8253 Mode register (Write Only)

Counter Registers (Port 84h, 85h, 86h)
The 8253 includes three 16bit counters. In the MSX RS 232 interface counter 0 and 1 are internally used to generate the Baud rates for receiving and transmitting data, counter 2 may be used for whatever purposes (for example as timeout counter).

The 8253 can operate at counting frequencies of up to 2MHz, in the MSX this might be 1.8432MHz, or 115.2kHz, or 28.8kHz (depending on the the Baud Fact value of Mode Setup byte (Port 81h). Counters are decremented at that frequency until they become zero, and are then automatically reloaded to the programmed reload value.

For example, a Baud Rate of 19200bps can be generated by using a counter reload value of 0006h. Ie. 115200Hz / 6 = 19200Hz (assuming that Baud Fact has been set to 115.2kHz). Keep in mind that the counters Read/Write mode must have been configured before outputting the counters reload value (see Port 87h below).

8253 Mode Register (Port 87h, Write Only)
This register actually includes three separate mode registers for each of the three counters (addressed by the upper two bits of the written value).

```
  Bit   Expl.
  0     Counter Format     (0=Binary, 1=BCD)
  1-3   Counter Mode       (See below) (6,7=Reserved)
  4-5   Prepare Read/Write (See below)
  6-7   Index              (Counter 0-2) (3=Reserved)
```

The 8253 is able to operate each counter in various modes, for the MSX RS232 interface Counter 0 and 1 (receive/transmit) should be set to mode 3 (Square Wave), and Counter 2 (custom) should be set to mode 0. The six possible modes are:

```
  0 = Interrupt Generator         3 = Square Wave Generator
  1 = Programmable Monoflop       4 = Trigger Output by Software
  2 = Clock Pulse Generator       5 = Trigger Output by Hardware
```

Before reading or writing to a counter, the Read/Write mode must be initialized. In most cases Write Mode 3 should be selected, then output both the low byte and high byte of the desired counter reload value to Port 84h, 85h, or 86h (depending on whether counter 0, 1, or 2 should be initialized). The four possible Read/Write modes are:

```
  0 = Latch Counter for Reading  2 = Load high byte
  1 = Load low byte              3 = Load low and high byte
```

Kanji ROM

Port D8h Select Class 1 Kanji Code (lower 6 bits)
Port D9h Select Class 1 Kanji Code (upper 6 bits)
Port D9h Read Data for Class 1 Kanji Code (32 bytes)
Port DAh Select Class 2 Kanji Code (lower 6 bits)
Port DBh Select Class 2 Kanji Code (upper 6 bits)
Port DBh Read Data for Class 2 Kanji Code (32 bytes)

Class 1 and Class 2 Kanji Characters
Class 1 and Class 2 Kanji are two japanese character sets. Note that not all MSX computers support both Class 1 and Class 2. Most european MSX computers probably don't include either one at all.
The Class 1 character set also contains various european, greek, and symbolic characters.
Note that different Kanji ROM versions exist, the characters may have slightly different appearances (and in some cases even different meanings).

Reading Kanji Codes
For Class 1 codes use Port D8h-D9h, for Class 2 codes use Port DAh-DBh.
Output the lower and upper bits of the Kanji Code (ie. the desired character number), then read 32 bytes from the respective Kanji Data port. These 32 bytes define a 16x16 pixel bitmap which is split into 4 tiles of 8x8 pixels each (first tile: upper left, second: upper right, third: lower left, and fourth: lower right).

Special I/O Registers

Port F5 System Control (Write Only)
Port F7 A/V Control (Read/Write)

Port F5h, System Control (Write only)
Setting bits to "1" enables available I/O devices.

```
  Bit  Expl.
   0   Kanji ROM Class 1
   1   Kanji ROM Class 2 (?)
   2   MSX-AUDIO
   3   Superimpose
   4   MSX interface
   5   RS-232C
   6   Lightpen
   7   CLOCK-IC (only on MSX2)
```

Bits to void the conflict between internal I/O devices or those connected by cartridge. The bits can disable the internal devices. When BIOS is initialised, internal devices are valid if no external devices are connected. Applications may not write to or read from here.

Port F7h, Audio/Video Control (A/V Control)

```
  Bit R/W Expl.
   0   W  Audio R                (mixing ON)
   1   W  Audio L                (mixing OFF)
   2   W  Select video input     (21p RGB)
   3  -R- Detect video input     (no input)
   4   W  AV control             (TV)
   5   W  Ym control             (TV)
   6   W  Inverse of bit 4 of VDP register 9
   7   W  Inverse of bit 5 of VDP register 9
```

Z80 CPU Specifications

Z80 Usage in MSX Models

Opcode Timings
Both MSX 1 and MSX 2 are using a Z80A CPU, operating at 3.579545MHz.
The execution time for each instruction is the number of clock cycles (specified in the instruction lists below) - plus the number of refresh cycles for that instruction.

One refresh cycle is performed for each normal instruction. A total of two refresh cycles is performed for all instructions which include the following prefix byte(s): CB, ED, DD, FD, DDCB, or FDCB.
For example, a NOP instruction counts 4 cycles plus 1 refresh cycle, resulting in a execution time of 5 cycles, ie. 5/3579545 seconds.

Interrupts
Interrupts are supplied by the Video Display Controller (VDP) only. Note that these interrupts must be manually acknowledged.
Non-maskable interrupts (NMIs) are not used. In Interrupt modes IM 0 and IM 2 an undefined parameter byte is supplied on the databus.

Z80 Register Set

Register Summary

16bit	Hi	Lo	Name/Function
AF	A	-	Accumulator & Flags
BC	B	C	BC
DE	D	E	DE
HL	H	L	HL
AF'	-	-	Second AF
BC'	-	-	Second BC
DE'	-	-	Second DE
HL'	-	-	Second HL
IX	IXH	IXL	Index register 1
IY	IYH	IYL	Index register 2
SP	-	-	Stack Pointer
PC	-	-	Program Counter/Pointer
-	I	R	Interrupt & Refresh

Normal 8bit and 16bit Registers
The Accumulator (A) is the allround register for 8bit operations. Registers B, C, D, E, H, L are normal 8bit registers, which can be also accessed as 16bit register pairs BC, DE, HL. The HL register pair is used as allround register for 16bit operations. B and BC are sometimes used as counters. DE is used as DEstination pointer in block transfer commands.

Second Register Set
The Z80 includes a second register set (AF',BC',DE',HL') these registers cannot be accessed directly, but can be exchanged with the normal registers by using the EX AF,AF and EXX instructions.

Refresh Register
The lower 7 bits of the Refresh Register (R) are incremented with every instruction. Instructions with at least one prefix-byte (CB,DD,ED,FD, or DDCB,FDCB) will increment the register twice. Bit 7 can be used by programmer to store data. Permanent writing to this register will suppress memory refresh signals, causing Dynamic RAM to lose data.

Interrupt Register
The Interrupt Register (I) is used in interrupt mode 2 only (see command "im 2"). In other modes it can be used as simple 8bit data register.

IX and IY Registers
IX and IY are able to manage almost all the things that HL is able to do. When used as memory pointers they are additionally including a signed index byte (IX+d). The disadvantage is that the opcodes occupy more memory bytes, and that they are less fast than HL-instructions.

Undocumented 8bit Registers
IXH, IXL, IYH, IYL are undocumented 8bit registers which can be used to access high and low bytes of the IX and IY registers (much like H and L for HL). Even though these registers do not officially exist, they seem to be available in all Z80 CPUs, and are quite commonly used by various software.

Z80 Flags

Flag Summary
The Flags are located in the lower eight bits of the AF register pair.

Bit	Name	Set	Clr	Expl.
0	C	C	NC	Carry Flag
1	N	-	-	Add/Sub-Flag (BCD)
2	P/V	PE	PO	Parity/Overflow-Flag
3	-	-	-	Undocumented
4	H	-	-	Half-Carry Flag (BCD)
5	-	-	-	Undocumented
6	Z	Z	NZ	Zero-Flag
7	S	M	P	Sign-Flag

Carry Flag (C)
This flag signalizes if the result of an arithmetic operation exceeded the maximum range of 8 or 16 bits, ie. the flag is set if the result was less than Zero, or greater than 255 (8bit) or 65535 (16bit). After rotate/shift operations the bit that has been 'shifted out' is stored in the carry flag.

Zero Flag (Z)
Signalizes if the result of an operation has been zero (Z) or not zero (NZ). Note that the flag is set (1) if the result was zero (0).

Sign Flag (S)
Signalizes if the result of an operation is negative (M) or positive (P), the sign flag is just a copy of the most significant bit of the result.

Parity/Overflow Flag (P/V)
This flag is used as Parity Flag, or as Overflow Flag, or for other purposes, depending on the instruction.
Parity: Bit7 XOR Bit6 XOR Bit5 ... XOR Bit0 XOR 1.
8bit Overflow: Indicates if the result was greater/less than +127/-128.
HL Overflow: Indicates if the result was greater/less than +32767/-32768.
After LD A,I or LD A,R: Contains current state of IFF2.
After LDI,LDD,CPI,CPD,CPIR,CPDR: Set if BC<>0 at end of operation.

BCD Flags (H,N)
These bits are solely supposed to be used by the DAA instruction. The N flag signalizes if the previous operation has be an addition or substraction. The H flag indicates if the lower 4 bits exceeded the range from 0-0Fh. (For 16bit instructions: H indicates if the lower 12 bits exceeded the range from 0-0FFFh.)
After adding/subtracting two 8bit BCD values (0-99h) the DAA instruction can be used to convert the hexadecimal result in the A register (0-FFh) back to BCD format (0-99h). Note that DAA also requires the carry flag to be set correctly, and thus should not be used after INC A or DEC A.

Undocumented Flags (Bit 3,5)
The content of these undocumented bits is by garbage by all instructions that affect one or more of the normal flags (for more info read the chapter Garbage in Flag Register), the only way to read out these flags would be to copy the flags register onto the stack by using the PUSH AF instruction.
However, the existence of these bits makes the AF register a full 16bit register, so that for example the code sequence PUSH DE, POP AF, PUSH AF, POP HL would set HL=DE with all 16bits intact.

Z80 Instruction Format

Commands and Parameters
Each instruction consists of a command, and optionally one or two parameters. Usually the leftmost parameter is modified by the operation when two parameters are specified.

Parameter Placeholders
The following placeholders are used in the following chapters:

```
  r      8bit  register A,B,C,D,E,H,L
  rr     16bit register BC, DE, HL/IX/IY, AF/SP   (as described)
  i      8bit  register A,B,C,D,E,IXH/IYH,IXL/IYL
  ii     16bit register IX,IY
  n      8bit  immediate 00-FFh                   (unless described else)
  nn     16bit immediate 0000-FFFFh
  d      8bit  signed offset -128..+127
  f      flag  condition nz,z,nc,c AND/OR po,pe,p,m  (as described)
  (..)   16bit pointer to byte/word in memory
```

Opcode Bytes
Each command (including parameters) consists of 1-4 bytes. The respective bytes are described in the following chapters. In some cases the register number or other parameters are encoded into some bits of the opcode, in that case the opcode is specified as "xx". Opcode prefix bytes "DD" (IX) and "FD" (IY) are abbreviated as "pD".

Clock Cycles
The clock cycle values in the following chapters specify the execution time of the instruction. For example, an 8-cycle instruction would take 2 microseconds on a CPU which is operated at 4MHz (8/4 ms). For conditional instructions two values are specified, for example, 17;10 means 17 cycles if condition true, and 10 cycles if false.
Note that in case that WAIT signals are sent to the CPU by the hardware then the execution may take longer.

Affected Flags
The instruction tables below are including a six character wide field for the six flags: Sign, Zero, Halfcarry, Parity/Overflow, N-Flag, and Carry (in that order). The meaning of the separate characters is:

s	Indicates Signed result
z	Indicates Zero
h	Indicates Halfcarry
o	Indicates Overflow
p	Indicates Parity
c	Indicates Carry
-	Flag is not affected
0	Flag is cleared
1	Flag is set
x	Flag is destroyed (unspecified)
i	State of IFF2
e	Indicates BC<>0 for LDX(R) and CPX(R), or B=0 for INX(R) and OUTX(R)

Z80 8bit Load Commands

Instruction	Opcode	Cycles	Flags	Notes
ld r,r	xx	4	------	r=r
ld i,i	pD xx	8	------	i=i
ld r,n	xx nn	7	------	r=n
ld i,n	pD xx nn	11	------	i=n
ld r,(HL)	xx	7	------	r=(HL)
ld r,(ii+d)	pD xx dd	19	------	r=(ii+d)
ld (HL),r	7x	7	------	(HL)=r
ld (ii+d),r	pD 7x dd	19	------	
ld (HL),n	36 nn	10	------	
ld (ii+d),n	pD 36 dd nn	19	------	
ld A,(BC)	0A	7	------	
ld A,(DE)	1A	7	------	
ld A,(nn)	3A nn nn	13	------	
ld (BC),A	02	7	------	
ld (DE),A	12	7	------	
ld (nn),A	32 nn nn	13	------	
ld A,I	ED 57	9	sz0i0-	A=I ;Interrupt Register
ld A,R	ED 5F	9	sz0i0-	A=R ;Refresh Register
ld I,A	ED 47	9	------	
ld R,A	ED 4F	9	------	

Z80 16bit Load Commands

```
 Instruction    Opcode  Cycles Flags  Notes
 ld   rr,nn     x1 nn nn    10 ------ rr=nn    ;rr may be BC,DE,HL or SP
 ld   ii,nn     pD 21 nn nn 13 ------ ii=nn
 ld   HL,(nn)   2A nn nn    16 ------ HL=(nn)
 ld   ii,(nn)   pD 2A nn nn 20 ------ ii=(nn)
 ld   rr,(nn)   ED xB nn nn 20 ------ rr=(nn)  ;rr may be BC,DE,HL or SP
 ld   (nn),HL   22 nn nn    16 ------ (nn)=HL
 ld   (nn),ii   pD 22 nn nn 20 ------ (nn)=ii
 ld   (nn),rr   ED x3 nn nn 20 ------ (nn)=rr  ;rr may be BC,DE,HL or SP
 ld   SP,HL     F9           6 ------ SP=HL
 ld   SP,ii     pD F9       10 ------ SP=ii
 push rr        x5          11 ------ SP=SP-2, (SP)=rr  ;rr may be
BC,DE,HL,AF
 push ii        pD E5       15 ------ SP=SP-2, (SP)=ii
 pop  rr        x1          10 (-AF-) rr=(SP), SP=SP+2  ;rr may be
BC,DE,HL,AF
 pop  ii        pD E1       14 ------ ii=(SP), SP=SP+2
 ex   DE,HL     EB           4 ------ exchange DE <--> HL
 ex   AF,AF     08           4 xxxxxx exchange AF <--> AF'
 exx            D9           4 ------ exchange BC,DE,HL <--> BC',DE',HL'
 ex   (SP),HL   E3          19 ------ exchange (SP) <--> HL
 ex   (SP),ii   pD E3       23 ------ exchange (SP) <--> ii
```

Z80 Blocktransfer- and Searchcommands

```
 Instruction    Opcode  Cycles Flags  Notes
 ldi            ED A0       16 --0e0- (DE)=(HL), HL=HL+1, DE=DE+1, BC=BC-
1
 ldd            ED A8       16 --0e0- (DE)=(HL), HL=HL-1, DE=DE-1, BC=BC-
1
 cpi            ED A1       16 szhe1- compare A-(HL), HL=HL+1, DE=DE+1,
BC=BC-1
 cpd            ED A9       16 szhe1- compare A-(HL), HL=HL-1, DE=DE-1,
BC=BC-1
 ldir           ED B0  bc*21-5 --0?0- ldi-repeat until BC=0
 lddr           ED B8  bc*21-5 --0?0- ldd-repeat until BC=0
 cpir           ED B1   x*21-5 szhe1- cpi-repeat until BC=0 or compare
fits
 cpdr           ED B9   x*21-5 szhe1- cpd-repeat until BC=0 or compare
fits
```

Z80 8bit Arithmetic/Logical Commands

```
 Instruction    Opcode  Cycles Flags  Notes
 daa            27           4 szxp-x decimal adjust akku
 cpl            2F           4 --1-1- A = A xor FF
 neg            ED 44        8 szho1c A = 00-A
 <arit>  r      xx           4 szhonc see below
 <arit>  i      pD xx        8 szhonc see below, UNDOCUMENTED
 <arit>  n      xx nn        7 szhonc see below
 <arit>  (HL)   xx          7 szhonc see below
 <arit>  (ii+d) pD xx dd   19 szhonc see below
 <cnt>   r      xx          4 szhon- see below
 <cnt>   i      pD xx       8 szhon- see below, UNDOCUMENTED
 <cnt>   (HL)   xx         11 szhon- see below
 <cnt>   (ii+d) pD xx dd   23 szhon- see below
 <logi>  r      xx          4 szhp00 see below
 <logi>  i      pD xx       8 szhp00 see below, UNDOCUMENTED
 <logi>  n      xx nn       7 szhp00 see below
 <logi>  (HL)   xx          7 szhp00 see below
 <logi>  (ii+d) pD xx dd   19 szhp00 see below
```

Arithmetic <arit> commands:

```
 add   A,op      see above 4-19 szho0c A=A+op
 adc   A,op      see above 4-19 szho0c A=A+op+cy
 sub   op        see above 4-19 szho1c A=A-op
 sbc   A,op      see above 4-19 szho1c A=A-op-cy
 cp    op        see above 4-19 szho1c compare, ie. VOID=A-op
```

Increment/Decrement <cnt> commands:

```
 inc   op        see above 4-23 szho0- op=op+1
 dec   op        see above 4-23 szho1- op=op-1
```

Logical <logi> commands:

```
 and   op        see above 4-19 sz1p00 A=A & op
 xor   op        see above 4-19 sz0p00 A=A XOR op
 or    op        see above 4-19 sz0p00 A=A | op
```

Z80 16bit Arithmetic Commands

```
 Instruction     Opcode  Cycles Flags  Notes
 add  HL,rr      x9          11 --h-0c HL = HL+rr    ;rr may be
BC,DE,HL,SP
 add  ii,rr      pD x9       15 --h-0c ii = ii+rr    ;rr may be
BC,DE,ii,SP (!)
 adc  HL,rr      ED xA       15 szho0c HL = HL+rr+cy ;rr may be
BC,DE,HL,SP
 sbc  HL,rr      ED x2       15 szho1c HL = HL-rr-cy ;rr may be
BC,DE,HL,SP
 inc  rr         x3           6 ------ rr = rr+1     ;rr may be
BC,DE,HL,SP
 inc  ii         pD 23       10 ------ ii = ii+1
 dec  rr         xB           6 ------ rr = rr-1     ;rr may be
BC,DE,HL,SP
 dec  ii         pD 2B       10 ------ ii = ii-1
```

Z80 Rotate and Shift Commands

```
 Instruction     Opcode  Cycles Flags  Notes
 rlca            07           4 --0-0c rotate akku left
 rla             17           4 --0-0c rotate akku left through carry
 rrca            0F           4 --0-0c rotate akku right
 rra             1F           4 --0-0c rotate akku right through carry
```

rld	ED 6F	18	sz0p0-	rotate left low digit of A through (HL)
rrd	ED 67	18	sz0p0-	rotate right low digit of A through (HL)
<cmd> r	CB xx	8	sz0p0c	see below
<cmd> (HL)	CB xx	15	sz0p0c	see below
<cmd> (ii+d)	pD CB dd xx	23	sz0p0c	see below
<cmd> r,(ii+d)	pD CB dd xx	23	sz0p0c	see below, UNDOCUMENTED modify and load

Whereas <cmd> may be:

rlc	rotate left
rl	rotate left through carry
rrc	rotate right
rr	rotate right through carry
sla	shift left arithmetic (b0=0)
sll	UNDOCUMENTED shift left (b0=1)
sra	shift right arithmetic (b7=b7)
srl	shift right logical (b7=0)

Z80 Singlebit Operations and CPU-Control Commands

Instruction	Opcode	Cycles	Flags	Notes
bit n,r	CB xx	8	xz1x0-	test bit n ;n=0..7
bit n,(HL)	CB xx	12	xz1x0-	
bit n,(ii+d)	pD CB dd xx	20	xz1x0-	
set n,r	CB xx	8	------	set bit n ;n=0..7
set n,(HL)	CB xx	15	------	
set n,(ii+d)	pD CB dd xx	23	------	
set r,n,(ii+d)	pD CB dd xx	23	------	UNDOCUMENTED set n,(ii+d) and ld r,(ii+d)
res n,r	CB xx	8	------	reset bit n ;n=0..7
res n,(HL)	CB xx	15	------	
res n,(ii+d)	pD CB dd xx	23	------	
res r,n,(ii+d)	pD CB dd xx	23	------	UNDOCUMENTED res n,(ii+d) and ld r,(ii+d)
ccf	3F	4	--h-0c	h=cy, cy=cy xor 1
scf	37	4	--0-01	cy=1
nop	00	4	------	no operation
halt	76	4	------	repeat until interrupt occurs
di	F3	4	------	iff=0 ;disable interrupts
ei	FB	4	------	iff=1 ;enable interrupts
im 0	ED 46	8	------	read opcode from databus on interrupt
im 1	ED 56	8	------	execute call 0038h on interrupt
im 2	ED 5E	8	------	execute call (i*100h+databus) on int.

Z80 Jumpcommands

Instruction	Opcode	Cycles	Flags	Notes
jp nn	C3 nn nn	10	------	jump to nn, ie. PC=nn

```
 jp   HL        E9           4 ------ jump to HL, ie. PC=HL
 jp   ii        pD E9        8 ------ jump to ii, ie. PC=ii
 jp   f,nn      xx nn nn 10;10 ------ jump to nn if nz,z,nc,c,po,pe,p,m
 jr   nn        18 dd       12 ------ relative jump to nn, ie. PC=PC+d
 jr   f,nn      xx dd     12;7 ------ relative jump to nn if nz,z,nc,c
 djnz nn        10 dd     13;8 ------ B=B-1 and relative jump to nn if
B<>0
 call nn        CD nn nn    17 ------ call nn ie. SP=SP-2, (SP)=PC, PC=nn
 call f,nn      xx nn nn 17;10 ------ call nn if nz,z,nc,c,po,pe,p,m
 ret            C9          10 ------ pop PC ie. PC=(SP), SP=SP+2
 ret  f         xx        11;5 ------ pop PC if nz,z,nc,c,po,pe,p,m
 reti           ED 4D       14 ------ pop PC, IFF2=IFF1 (ret from INT)
 retn           ED 45       14 ------ pop PC, IFF2=IFF1 (ret from NMI)
 rst  n         xx          11 ------ call n  ;n=00,08,10,18,20,28,30,38
```

Z80 I/O Commands

```
 Instruction    Opcode  Cycles Flags  Notes
 in   A,(n)     DB nn       11 ------ A=PORT(A*100h+n)
 in   r,(C)     ED xx       12 sz0p0- r=PORT(BC)
 in   (C)       ED 70       12 sz0p0- **undoc/illegal** VOID=PORT(BC)
 out  (n),A     D3 nn       11 ------ PORT(A*100h+n)=A
 out  (C),r     ED xx       12 ------ PORT(BC)=r
 out  (C),0     ED 71       12 ------ **undoc/illegal** PORT(BC)=00
 ini            ED A2       16 xexxxx MEM(HL)=PORT(BC), HL=HL+1, B=B-1
 ind            ED AA       16 xexxxx MEM(HL)=PORT(BC), HL=HL-1, B=B-1
 outi           ED A3       16 xexxxx B=B-1, PORT(BC)=MEM(HL), HL=HL+1
 outd           ED AB       16 xexxxx B=B-1, PORT(BC)=MEM(HL), HL=HL-1
 inir           ED B2   b*21-5 x1xxxx same than ini, repeat until b=0
 indr           ED BA   b*21-5 x1xxxx same than ind, repeat until b=0
 otir           ED B3   b*21-5 x1xxxx same than outi, repeat until b=0
 otdr           ED BB   b*21-5 x1xxxx same than outd, repeat until b=0
```

Z80 Interrupts

Lack of Information
Z80 interrupts are a mysterium, the official Z80 documentation basically denied the existence of interrupts in general. However, when describing opcodes such like IM 0, EI, and RETI the author couldn't fully avoid to mention the possibility that such a thing like interrupts might eventually exist by suggesting to refer to somewhat called an "application script about interrupt behaviour of Z80 systems".

In case that this document really exists, then it is probably been horribly expensive, made available to authorized developers only, and not available nowadays anymore. The content of this document might or might not confirm the existence of interrupts, and/or explain other details?

Judging from the fragments of information that leaked out in the Z80 docs, the CPU supports non-maskable interrupts (NMI) and maskable interrupts (INT). MSX and CPC homecomputers aren't using NMIs, so information below is reverse engineered guesswork for systems that use maskable interrupts only.

Interrupt Flip-Flop (IFF1,IFF2)
These internal flags are used to enable/disable interrupts, in a raw INT based system which isn't using NMIs, it appears to be safe to treat these flags as a single IFF flag, instead separate IFF1 and IFF2.

Interrupts can be enabled by the EI instruction (IFF=1) only, whereas the new IFF state isn't applied until the next instruction has completed (this ensures that an interrupt handler which is using the sequence "EI, RET" may return to the main program before the next interrupt is executed).
Interrupts can be disabled by the DI instruction (IFF=0), and are additionally automatically each time when an interrupt is executed.

Interrupt Execution
An interrupt is executed when an interrupt is requested by the hardware, and IFF is set. Whenever both conditions are true, the interrupt is executed after the completion of the current opcode.
Note that repeated block commands (such like LDIR) can be interrupted also, the interrupt return address on the stack then points to the interrupted opcode, so that the instruction may continue as normal once the interrupt handler returns.

Interrupt Modes (IM 0,1,2)
The Z80 supports three interrupts modes which can be selected by IM 0, IM 1, and IM 2 instructions. The table below describes the respective operation and execution time upon interrupt execution in each mode.

Mode	Cycles	Refresh	Operation
0	1+var	0+var	IFF=0, read and execute opcode from databus
1	12	1	IFF=0, CALL 0038h
2	18	1	IFF=0, CALL (I*100h+databus)

Mode 0 requires an opcode to be output to the databus by external hardware, in case that no byte is output, and provided that the 'empty' databus is free of garbage, then the CPU might tend to read a value of FFh (opcode RST 38h) - the clock cycles, refresh cycles, and executed operation are then fully identical as in Mode 1.
Mode 1 interrupts always perform a CALL 0038h operation. The downside is that many systems may have ROM located at this address, making it impossible to hook the interrupt handler directly.

Mode 2 calls to a 16bit address which is read from a table in memory, the table pointer is calculated from the "I" register (initialized by LD I,A instruction) multiplied by 100h, plus an index byte which is read from the databus. The following trick may be used to gain stable results in Mode 2 even if no index byte is supplied on the databus: For example, set I=40h the origin of the table will be then at 4000h on memory.

Now fill the entire area from 4000h to 4100h (101h bytes, including 4100h) by the value 41h. The CPU will then perform a CALL 4141h upon interrupt execution - regardless of whether the randomized index byte is a even or odd number.

RETI and RETN
These instructions are somewhat supposed to return from maskable and non-maskable interrupts. In a raw INT based system which isn't using NMIs they appear to behave 100% identical to normal RET instructions.

Z80 Meaningless and Duplicated Opcodes

Mirrored Instructions
NEG (ED44) is mirrored to ED4C,54,5C,64,6C,74,7C.
RETN (ED45) is mirrored to ED55,65,75.
RETI (ED4D) is mirrored to ED5D,6D,7D.

Mirrored IM Instructions
IM 0,X,1,2 (ED46,4E,56,5E) are mirrored to ED66,6E,76,7E.
Whereas IM X is an undocumented mirrored instruction itself which appears to be identical to either IM 0 or IM 1 instruction (?).

Duplicated LD HL Instructions
LD (nn),HL (opcode 22NNNN) is mirrored to ED63NNNN.
LD HL,(nn) (opcode 2ANNNN) is mirrored to ED6BNNNN.
Unlike the other instructions in this chapter, these two opcodes are officially documented.
The clock/refresh cycles for the mirrored instructions are then 20/2 instead of 16/1 as for the native 8080 instructions.

Mirrored BIT N,(ii+d) Instructions
Unlike as for RES and SET, the BIT instruction does not support a third operand, ie. DD or FD prefixes cannot be used on a BIT N,r instruction in order to produce a BIT r,N,(ii+d) instruction. When attempting this, the 'r' operand is ignored, and the resulting instruction is identical to BIT N,(ii+d).
Except that, not tested yet, maybe undocumented flags are then read from 'r' instead of from ii+d(?).

Non-Functional Opcodes
The following opcodes behave much like the NOP instruction.
ED00-3F, ED77, ED7F, ED80-9F, EDA4-A7, EDAC-AF, EDB4-B7, EDBC-BF, EDC0-FF.
The execution time for these opcodes is 8 clock cycles, 2 refresh cycles.
Note that some of these opcodes appear to be used for additional instructions by the R800 CPU in newer turbo R models.

Ignored DD and FD Prefixes
In some cases, DD-prefixes (IX) and FD-prefixes (IY) may be ignored by the CPU. This happens when using one (or more) of the above prefixes prior to instructions that already contain an ED, DD, or FD prefix, or prior to any instructions that do not support IX, IY, IXL, IXH, IYL, IYH operands. In such cases, 4 clock cycles and 1 refresh cycle are counted for each ignored prefix byte.

Z80 Garbage in Flag Register

Nocash Z80-flags description
This chapter describes the undocumented Z80 flags (bit 3 and 5 of the Flags Register), these flags are affected by ALL instructions that modify one or more of the normal flags - all OTHER instructions do NOT affect the undocumented flags.

For some instructions, the content of some flags has been officially documented as 'destroyed', indicating that the flags contain garbage, the exact garbage calculation for these instructions will be described here also.

All information below just for curiosity. Keep in mind that Z80 compatible CPUs (or emulators) may not supply identical results, so that it wouldn't be a good idea to use these flags in any programs (not that they could be very useful anyways).

Normal Behaviour for Undocumented Flags
In most cases, undocumented flags are copied from the Bit 3 and Bit 5 of the result byte. That is "A AND 28h" for:

```
  RLD; CPL; RLCA; RLA; LD A,I; ADD OP; ADC OP; XOR OP; AND OP;
  RRD; NEG; RRCA; RRA; LD A,R; SUB OP; SBC OP; OR OP ; DAA.
```

When other operands than A may be modified, "OP AND 28h" for:

```
  RLC OP; RL OP; SLA OP; SLL OP; INC OP; IN OP,(C);
  RRC OP; RR OP; SRA OP; SRL OP; DEC OP
```

For 16bit instructions flags are calculated as "RR AND 2800h":

```
  ADD RR,XX; ADC RR,XX; SBC RR,XX.
```

Slightly Special Undocumented Flags
For 'CP OP' flags are calculated as "OP AND 28h", that is the unmodified operand, and NOT the internally calculated result of the comparision.
For 'SCF' and 'CCF' flags are calculated as "(A OR F) AND 28h", ie. the flags remain set if they have been set before.
For 'BIT N,R' flags are calculated as "OP AND 28h", additionally the P-Flag is set to the same value than the Z-Flag (ie. the Parity of "OP AND MASK"), and the S-flag is set to "OP AND MASK AND 80h".

Fatal MEMPTR Undocumented Flags
For 'BIT N,(HL)' the P- and S-flags are set as for BIT N,R, but the undocumented flags are calculated as "MEMPTR AND 2800h", for more info about MEMPTR read on below.
The same applies to 'BIT N,(ii+d)', but the result is less unpredictable because the instruction sets MEMPTR=ii+d, so that undocumented flags are "<ii+d> AND 2800h".

Memory Block Command Undocumented Flags
For LDI, LDD, LDIR, LDDR, undocumented flags are "((A+DATA) AND 08h) + ((A+DATA) AND 02h)*10h".
For CPI, CPD, CPIR, CPDR, undocumented flags are "((A-DATA-FLG_H) AND 08h) + ((A-DATA-FLG_H) AND 02h)*10h", whereas the CPU first calculates A-DATA, and then internally subtracts the resulting H-flag from the result.

Chaotic I/O Block Command Flags
The INI, IND, INIR, INDR, OUTI, OUTD, OTIR, OTDR instructions are doing a lot of obscure things, to simplify the description a placeholder called DUMMY is used in the formulas.
```
  DUMMY = "REG_C+DATA+1"    ;for INI/INIR
  DUMMY = "REG_C+DATA-1"    ;for IND/INDR
  DUMMY = "REG_L+DATA"      ;for OUTI,OUTD,OTIR,OTDR
  FLG_C = Carry  of above "DUMMY" calculation
  FLG_H = Carry  of above "DUMMY" calculation (same as FLG_C)
  FLG_N = Sign   of "DATA"
  FLG_P = Parity of "REG_B XOR (DUMMY AND 07h)"
  FLG_S = Sign   of "REG_B"
  UNDOC = Bit3,5 of "REG_B AND 28h"
```
The above registers L and B are meant to contain the new values which are already incremented/decremented by the instruction.
Note that the official docs mis-described the N-Flag as set, and the C-Flag as not affected.

DAA Flags
Addition (if N was 0):
```
  FLG_H = (OLD_A AND 0Fh) > 09h
  FLG_C = Carry of result
```
Subtraction (if N was 1):
```
  FLG_H = (NEW_A AND 0Fh) > 09h
  FLG_C = OLD_CARRY OR (OLD_A>99h)
```
For both addition and subtraction, N remains unmodified, and S, Z, P contain "Sign", Zero, and Parity of result (A). Undocumented flags are set as (A AND 28h) as normal.

Mis-documented Flags
For all XOR/OR: H=N=C=0, and for all AND: H=1, N=C=0, unlike described else in Z80 docs. Also note C,N flag description bug for I/O block commands (see above).

Internal MEMPTR Register
This is an internal Z80 register, modified by some instructions, and usually completely hidden to the user, except that Bit 11 and Bit 13 can be read out at a later time by BIT N,(HL) instructions.

The following list specifies the resulting content of the MEMPTR register caused by the respective instructions.

```
Content Instruction
A*100h  LD (xx),A               ;xx=BC,DE,nn
xx+1    LD A,(xx)               ;xx=BC,DE,nn
nn+1    LD (nn),rr; LD rr,(nn)  ;rr=BC,DE,HL,IX,IY
rr      EX (SP),rr              ;rr=HL,IX,IY (MEMPTR=new value of rr)
rr+1    ADD/ADC/SBC rr,xx       ;rr=HL,IX,IY (MEMPTR=old value of rr+1)
HL+1    RLD and RRD
dest    JP nn; CALL nn; JR nn   ;dest=nn
dest    JP f,nn; CALL f,nn      ;regardless of condition true/false
dest    RET; RETI; RETN         ;dest=value read from (sp)
dest    RET f; JR f,nn; DJNZ nn ;only if condition=true
00XX    RST n
adr+1   IN A,(n)                ;adr=A*100h+n, memptr=A*100h+n+1
bc+1    IN r,(BC); OUT (BC),r   ;adr=bc
ii+d    All instructions with operand (ii+d)
```

Also the following might or might not affect MEMPTR, not tested yet:

```
OUT (N),A and block commands LDXX, CPXX, INXX, OUTXX
and probably interrupts in IM 0, 1, 2
```

All other commands do not affect the MEMPTR register - this includes all instructions with operand (HL), all PUSH and POP instructions, not executed conditionals JR f,d, DJNZ d, RET f (ie. with with condition=false), and the JP HL/IX/IY jump instructions.

Z80 Compatibility

The Z80 CPU is (almost) fully backwards compatible to older 8080 and 8085 CPUs.

Instruction Format
The Z80 syntax simplifies the chaotic 8080/8085 syntax. For example, Z80 uses the command "LD" for all load instructions, 8080/8085 used various different commands depending on whether the operands are 8bit registers, 16bit registers, memory pointers, and/or an immediates. However, these changes apply to the source code only - the generated binary code is identical for both CPUs.

Parity/Overflow Flag
The Z80 CPU uses Bit 2 of the flag register as Overflow flag for arithmetic instructions, and as Parity flag for other instructions. 8080/8085 CPUs are always using this bit as Parity flag for both arithmetic and non-arithmetic instructions.

Z80 Specific Instructions
The following instructions are available for Z80 CPUs only, but not for older 8080/8085 CPUs:
All CB-prefixed opcodes (most Shift/Rotate, all BIT/SET/RES commands).
All ED-prefixed opcodes (various instructions, and all block commands).
All DD/FD-prefixed opcodes (registers IX and IY).
As well as DJNZ nn; JR nn; JR f,nn; EX AF,AF; and EXX.

8085 Specific Instructions
The 8085 instruction set set includes two specific opcodes in addition to the 8080 instruction set, used to control 8085-specifc interrupts and SID and SOD input/output signals. These opcodes, RIM (20h) and SIM (30h), are not supported by Z80 instruction set.

Z80 vs Z80A
Both Z80 and Z80A are including the same instruction set, the only difference is the supported clock frequency (Z80 = max 2.5MHz, Z80A = max 4MHz).

Different MSX Models

MSX computers have been manufactured by almost 40 different companies spread all over the world, and most of these companies have assembled their own unique hardware combination(s), so that there should be about 100 different MSX models.
All models can be split into four categories: MSX, MSX2, MSX2+, and turbo R. Listed below are the most common configurations for each category (not including special hardware/firmware expansions that might have been included in some models).

MSX1
Z80 CPU (8bit) 3.579545MHz
PSG Sound: three sound & noise channels
Keyboard: 72 Keys, CAPs LED, Keyclick Sound
Video chip: V9918, 40x24 text, 256x192 pix, 16 colors, 32 sprites
Memory: 32K ROM, 8/16/32/64K RAM, 16K VRAM, Memory Control: Primary Slot
Connectors: 2 Joysticks, Cassette, RGB and TV, 2 cartridge slots

MSX2
More or less backwards compatible to MSX1
Video chip: V9938, 80x24 text, 512x212 pix, 256 colors, VRAM DMA operations
Memory: 48K ROM, 64/128/256K RAM, 64/128K VRAM
Memory Control: Primary Slot, Secondary Slot, Memory Mapper (for >=128K RAM)
Built-in disk drive, disk controller, and 16K disk ROM (optional)
Built-in battery buffered real time clock (optional)

MSX2+
Mostly backwards compatible to MSX2
Video chip: V9958, horizontal scrolling, max 32768 colors in YJK mode
FM sound chip -OPLL- YM2413 (additionally to old PSG chip)
Japanese Kanji ROM charcter set (optional)

turbo R
Don't know. Somewhat using a new Z80 compatible CPU.

External Connectors

Below information is taken from my Philips MSX1 and MSX2 manuals, but some or all connectors might differ for other MSX models.

Cartridge Connectors (2)

1	Out	/CS1	ROM addresses 4000-7FFF select signal
2	Out	/CS2	ROM addresses 8000-BFFF select signal
3	Out	/CS12	ROM addresses 4000-BFFF select signal
4	Out	/SLTSL	Slot select signal
5	-	Reserved	Reserved, don't use
6	Out	/RFSH	Refresh cycle signal
7	In	/WAIT	CPU's WAIT request signal
8	In	/INT	Interrupt request signal to CPU
9	Out	/M1	Signal expressing CPU fetch cycle
10	In	/BUSDIR	External data bus buffer direction
11	Out	/IORQ	I/O request signal
12	Out	/MERQ	Memory request signal
13	Out	/WR	Write timing signal
14	Out	/RD	Read timing signal
15	Out	/RESET	System reset signal
16	-	Reserved	Reserved, don't use
17-32	Out	Ax	Address bus (A9,15,11,10,7,6,12,8,14,13,1,0,3,2,5,4)
33-40	I/O	Dx	Data bus (D1,D0,D3,D2,D5,D4,D7,D6)
41	-	GND	Ground
42	Out	CLOCK	CPU clock 3.579545MHz
43	-	GND	Ground
44	-	SW1	For insertion/removal protect
45	-	+5V	+5V power source
46	-	SW2	For insertion/removal protect
47	-	+5V	+5V power source
48	-	+12V	+12V power source
49	In	SOUNDIN	Sound input signal (-5bdm)
50	-	-12V	-12V power source

Data Recorder Connector

1-3		GND
4	Out	Data Output (Record)
5	In	Data Input (Play)
6	Out	Remote+
7	Out	Remote-
8		GND

Joystick Connectors (2)

1	In	Up
2	In	Down
3	In	Left
4	In	Right
5		+5V
6	I/O	Trigger 1
7	I/O	Trigger 2

8	Out	/Select
9		GND

Printer Connector

1	Out	/Strobe
2-9	Out	D0-D7
10	N/C	N/C
11	In	Busy
12-14	N/C	NC

Monitor Connector

Pin	PAL-version		RGB-version (french only)	
1	-	+5V	Out	Status RGB
2	-	GND	-	GND
3	Out	Audio	Out	Blue
4	Out	Luminance	Out	Luminance
5	Out	Video	Out	Red
6	-	+12V	-	+12V
7	N/C	Not Used	Out	Sound
8	N/C	Not Used	Out	Green

Audio/Video Out (MSX2)

1	Out	Audio Out, right
2	N/C	Audio In, right
3	Out	Audio Out, left
4		Audio GND
5		Blue GND
6	N/C	Audio In, left
7	Out	Blue Out
8	Out	Status CVBS
9		Green GND
10	N/C	NC
11	Out	Green Out
12	N/C	NC
13		Red GND
14		GND
15	Out	Red Out
16	Out	Status RGB
17		CVBS GND
18		RBG Status GND
19	Out	CVBS Out
20	N/C	CVBS In
21		Socket GND (Shield)

TV Connector

Ext. Drive Connector

4	/In Use	Pin 20	/Step
6	/Drive Select 3	Pin 22	/Write Data
8	/Index	Pin 24	/Write Gate
10	/Drive Select 0	Pin 26	/Track 00
12	/Drive Select 1	Pin 28	/Write Protect

```
  14  /Drive Select 2    Pin 30  /Read Data
  16  /Motor On          Pin 32  /(Head Select)
  18  /Direction         Pin 34  /Ready
```

Pin 1 and 2 are not connected. All other pins are connected to ground (that are all pins with odd numbers, in range from 3-33).

Power Supply AC 220V (MSX2 with built-in power supply)

Power Supply DC (MSX1 with external power supply)

```
  1   N/C   (middle)
  2   GND   (upper left)
  3   +12V  (lower left)
  4   +5V   (lower right)
  5   -12V  (upper right)
```

Data Structures and Formats

Disk Images

MSX Disk Images (.DSK files) are just a raw copy of the data of all sectors from a real disk. The DSK format does not contain any information about the physical format of the disk, and thus doesn't support uncommon 'copy protected' formats.

Usually a disk image contains data for 80 tracks, 9 sectors (sized 200h bytes each, and numbered from 01h to 09h). Resulting in an image size of 360Kbytes or 720KBytes, depending on whether the disk is single sided, or double sided.

However, most MSX disks include basic information in their boot sectors, which could be used (if it does exist, and if it is filled out correctly) to determine the number of heads, sectors, and tracks of the disk.

Disk FAT Format (alias)

See chapter Disk FAT Format in FDC description

Disk File Formats

Binary Files
Binary files are invented by a 7 bytes header:

```
  1 byte   File type     (FEh=binary)
  2 bytes  Load Address  (Destination Address for Data in RAM)
  2 bytes  Last Address  (Load Address + Data Length - 1)
  2 bytes  Start Address (Used if started as BLOAD"FILE",R)
```

Then followed by the actual data.

Basic Files
A basic file is invented by a 1 byte header which contains the file type (FFh=basic), the format of the following data is identical as for Basic programs which are saved on tapes. Note that Basic programs could be also saved (and loaded) in Ascii format.

Ascii/Text Files
Ascii files are saved as raw data without any header. Lines are usually ended by CR,LF (0Dh,0Ah) characters. A clean Ascii file should be terminated by an EOF (1Ah) character.

COM Files
CP/M and MSXDOS programs are saved as .COM files, these files do not have any headers, the load- and startaddress is always 100h. Because of the CP/M filesystem, the filesize of CP/M files must be a multiple of 80h.

Cassette File Formats

Each MSX file consists of two blocks: a header block, and a data block. (Except for ascii files, which may have multiple data blocks - but only one header block).
All data is saved without any checksums, readfails could be particulary recognized by verifying that each byte has proper Start- and Stopbits. The baud rate can be determined by measuring the length of the synchronization bits of each block.

Format of MSX Header Blocks

```
  1E00h  "1"-bits  synchronisation bits
     10  bytes     file type (repeated 10 times the same value)
      6  bytes     file name (unused entries filled by SPCs)
```

The file-type could be binary (D0h), basic (D3h), or ascii (EAh).

Format of MSX Basic Data Blocks (Type D3h)

```
   780h  "1"-bits  synchronisation bits
   <nn>  lines     data, format for each line as follows:
                         2  bytes   origin of next line (lower byte
first)
                      <xx>  bytes   data (xx = nextlineorg-
currentlineorg-2)
      2  bytes     0000h zero origin, no further lines following
    7-8  bytes     terminator, seven or eight 00h bytes
```

Note: The first line is assumed to begin at origin 8001h.

Format of MSX Binary Data Blocks (Type D0h)

```
   780h  "1"-bits  synchronisation bits
      2  bytes     load address  (lower byte first)
      2  bytes     end address   ("")
      2  bytes     start address ("")
  <len>  bytes     data (length = endadr-loadadr+1)
```

Format of MSX Ascii Data Blocks (Type EAh)

```
   780h  "1"-bits  synchronisation bits
   100h  bytes     data (ended by EOF/1Ah, unused bytes filled up by 1Ah)
```

More data blocks follow as long as the data block contains no EOF (1Ah).

Format of Bits and Bytes

The MSX BIOS generates start and stop bits for each byte:

```
  1  start bit ("0")
  8  data bits (lower bit first)
  1  first stop bit ("1")
  1  second stop bit ("1") (but might be unreadable if short delay
follows)
```

And finally, the MSX bits themselves are encoded as follows:

```
  "0"  =   \_______/"""""""\
  "1"  =   \___/"""\___/"""\
```

Physically the signal looks less square, rather like a sine-wave.
(Note: The signal could be inverted, ie. high-pulse first...)

ROM Headers

The first 16 bytes of each Cartridge or Expansion ROM are used as header. In most cases (ie. in game cartridges) only the first two entries are used, and the remaing entries are set to zero.

```
  Address Name       Expl.
  X000    ID         Identification Code (4241h=ROM Cartridge,
4443h=SUBROM)
  X002    INIT       Start Address (could be anywhere 0000-BFFF)
  X004    Statement  Statement expansion routine address. For creating
new
                     CALL statement (For example CALL MUSIC used in FM
PAC)
  X006    DEV        For creating new devices (CAS:, MEM:, GRP:, etc...)
  X008    TEXT       Pointer to BASIC program in ROM, must be in 8000-
BFFF.
  X00A-F  N/A        Reserved (0)
```

Upon startup the BIOS scans all SLOTs for Cartridge ROMs (ID 4241h) at addresses 4000h and 8000h, and for SUBROMs (ID 4443h) at address 0000h.
Most Cartridge ROMs occupy the area from 4000h-BFFFh, but might be also using (or being mirrored to) the whole address space from 0000h to FFFFh.

Other DOCs

Original doc by Mayer of WC Hakkers version 1.5 (PORTAR.DOC)
Documentation about MSX1/MSX2 and optional external hardware (IO ports).
ftp://ftp.funet.fi/pub/msx/txt/tech/portar.doc

TMS9918A VDP description/undocumented features by Sean Young
Documentation about MSX1 video display processor (VDP)
http://www.msxnet.org/tech/tms9918a.txt

The V9958-Registers by Zelly/Mayhem (V9958.TXT)
Brief (but VERY complete) summary about MSX2(+) VDP registers.

The MSX Red Book by Avalon Software (TREDBOOK.TXT)
Documentation about
- MSX1 hardware (IO ports)
- MSX1 firmware (BIOS functions)

ftp://ftp.funet.fi/pub/msx/txt/tech/tredbook.arj

MSX2 Technical Handbook by Ascii Corp/Konami Man (TH-XX.TXT)
Verbose documentation about
- MSX2 hardware (IO ports)
- MSX2 firmware (BIOS functions)
- MSX2 software (BASIC and MSX-DOS commands)

http://www.geocities.com/SiliconValley/Bay/9797/msx2.htm#MSX2TH

Western Digital FD1793 Floppy Disk Controller (TECH_WD1793.HTM)
Unofficial Spectravideo Homepage by tomas.k (docs about FDC and other stuff)
http://home.swipnet.se/~w-16418/tech_wd1793.htm (old URL)
http://www.abysscrew.nu/spectravideo/ (new URL)

A22i CPC/Gameboy/MSX Assembler & Docs by Martin Korth (A22I.ZIP)
My MSX assembler for DOS, including a brief summary about
- Z80 instruction set (Opcodes, flags, clock cycles)

http://www.work.de/nocash/a22i.zip

No$msx Docs (an OLDER version of this file) by Martin Korth
http://www.work.de/nocash/msx-docs.zip

Kanji ROM Description (KANROM.TXT)
MSX Technical Guidebook from ASCAT / TAKAMICHI / G&T Soft International Div.
http://www.msxnet.org/gtinter/kanrom.htm

Others
PC-Profibuch by Martin Althaus (Sybex): 8253 Mode Register Info

END