MSX

MSX Technical Data Book

Hardware/Software Specifications

Presented by

Robsy's MSX Workshop

Originaly scanned by

Ivan Latorre

Converted to PDF

Eduardo Robsy

[September 2004]

SONY

MSX

MSX Technical Data Book

Hardware/Software Specifications

SONY

Sony Corporation
4 14 1, Asahi-cho, Atsugi-shi,
Kanagawa-ken, 243 Japan

Copyright © 1984 Microsoft Corporation
Produced by ASCII Corporation

Printed in Japan

PREFACE

The Microsoft MSX standard was invented to provide end users and software developers with a standardized computer so that programs could run on any computer even though they were made by different manufacturers.

This book presents the MSX specifications in detail. It is intended to be a reference for advanced programmers and software developers. The information is generally divided four parts.

Part A, MSX HARDWARE SPECIFICATIONS, presents the specifications for the MSX system hardware.

Chapter 1, Hardware Specification, covers the MSX standard hardware configuration in terms of the requirements for the LSIs, memory size, interrupts, screen, keyboard, and sound used in the main unit; and the various (cassette, floppy, printer, serial, and slot) interfaces and connectors. It also covers topics such as cartridges, expansion, ports, and memory maps.

Part B, MSX SYSTEM SOFTWARE, contains a reference guide for MSX-BASIC and information for advanced programming.

Chapter 2, Language Specification, is a guide to MSX-BASIC and is for use with advanced programming requiring machine language routines.

Part C, EXPANDED MSX SYSTEM SOFTWARE, is about the advanced features of MSX, including Expanded Disk BASIC and MSX-DOS.

Chapter 3, MSX-DOS, contains a user's guide to MSX-DOS and Disk BASIC, and includes information needed for the advanced programmer.

Chapter 4, Other Expansion, covers the serial (RS-232C) expansion and BIOS calls available in the extended version.

Part D, SOFTWARE DEVELOPMENT GUIDE, contains information for software developers.

Chapter 5, International MSX Versions and their Differences, is for manufacturers or programmers who wish to make the hardware or software be usable internationally.

Chapter 6, Notes for MSX Software Developers, contains information that software developers should consider when programming for MSX computers.

Syntax Notation in Reference Sections

Wherever the format for a statement/command or a function is given, the following rules apply:

CAPS Items in capital letters must be input as shown.

< > Items in lowercase letters enclosed in angle brackets (< >) are to be supplied by the user.

[ ] Items in square brackets ([ ]) are optional.

... Items followed by an ellipsis (...) may be repeated any number of times (up to the length of the line).

{ } Braces indicate that the user has a choice between two or more entries. At least one of the entries enclosed in braces must be chosen unless the entries are also enclosed in square brackets.

| Vertical bars separate the choices within braces. At least one of the entries separated by bars must be chosen unless the entries are also enclosed in square brackets.

All punctuation except angle brackets and square brackets (i.e., commas, parentheses, semicolons, hyphens, equal signs) must be included where shown.

Arguments to functions are always enclosed in parentheses. In the formats given for the functions in this book, the arguments are abbreviated as follows:

X and Y	Represent any numeric expressions.
I and J	Represent integer expressions.
X$ and Y$	Represent string expressions.

C O N T E N T S

PART A MSX HARDWARE SPECIFICATIONS

1. Hardware Specifications

1.1 MSX Standard 8
1.2 MSX System Configuration 9
1.3 Main Unit 10
1.3.1 LSIs 10
1.3.2 Memory 10
1.3.3 Interrupts 11
1.3.4 Screen 12
1.3.5 Keyboard 13
1.3.6 Sound 14
1.4 Interfaces 15
1.4.1 Cassette Interface 15
1.4.2 Floppy Disk Interface 18
1.4.3 Printer Interface 19
1.4.4 RS-232C Interface 20
1.4.5 Peripheral I/O Port(s) 25
1.4.6 Joysticks 27
1.4.7 Paddles 28
1.4.8 Connectors 29
1.4.9 Slots 30
1.5 Cartridges 31
1.5.1 Cartridge Standard 31
1.5.2 Cartridge Bus 32
1.5.3 Cartridge Bus Connenction Conditions 34
1.5.4 Cartridge Power Capacity 34
1.5.5 Sample Circuit Diagram of Expanded Slot Select Signal35
1.6 Notes for System Expansion 36
1.6.1 RAM Expansion 36
1.6.2 Slot Expansion 36
1.6.3 I/O Expansion 37
1.7 Address Maps 38
1.7.1 Memory Map 38
1.7.2 I/O Address Map 40
1.7.3 Printer Port 41
1.7.4 VDP Port 41
1.7.5 PSG Port 41
1.7.6 PPI Port 41
1.7.7 External Memory (SONY) 41
1.7.8 Light Pen (SANYO) 41
1.7.9 Audio/Video Control 42
1.7.10 Notes on I/O Address Assignments 42
1.7.11 8255 (PPI) Bit Assignments 43
1.7.12 PSG Bit Assignments 44

PART B MSX SYSTEM SOFTWARE

2. Language Specifications

2.1 MSX BASIC Reference Guide 46
2.1.1 Modes of Operation 46

2.1.2 Line Format 47
2.1.3 Character Set 47
2.1.4 Constants 48
2.1.5 Variables 50
2.1.6 Type Conversion 51
2.1.7 Expressions and Operators 53
2.1.8 Program Editing 57
2.1.9 Special Keys 62
2.1.10 Error Messages 63
2.1.11 Commands and Statements except those doing I/O 63
2.1.12 Functions except those doing I/O 79
2.1.13 Device Specific Statements 84
2.1.14 I/O Functions 100
2.1.15 Special Variables 102
2.1.16 Machine Dependent Statements and Functions 104
2.1.17 Summary of Error Codes and Messages 105
2.1.18 MSX BASIC Reserved Words 109
2.2 Advanced Programming Guide 110
2.2.1 BIOS Entry List 110
2.2.2 Work Area 135
2.2.3 Slot Control 161
2.2.4 Cassette I/O Mechanism 172
2.2.5 MSX Printer Specifications 177

PART C EXPANDED MSX SYSTEM SOFTWARE

3. MSX-DOS

3.1 MSX-DOS User's Guide 182
3.1.1 System Requirements 182
3.1.2 Getting Started 182
3.1.3 Wild Cards 184
3.1.4 Illegal File Names 185
3.1.5 Directories 186
3.1.6 Types of MSX-DOS Commands 186
3.1.7 Command Options 187
3.1.8 Information Common to All MSX-DOS Commands 188
3.1.9 Batch Processing 189
3.1.10 The AUTOEXEC.BAT File 190
3.1.11 How to Create a Batch File 191
3.1.12 Replaceable Parameters in .BAT File 192
3.1.13 MSX-DOS Editing and Function Keys 194
3.1.14 Instructions for Users with Single-drive Systems ...200
3.1.15 Disk Errors 201
3.2 MSX-DOS Command Guide 202
3.3 MSX Disk BASIC Reference Guide 219
3.3.1 Commands and Statements 219
3.3.2 Functions 246
3.3.3 Error Codes and Error Messages 252
3.4 MSX-DOS and Disk BASIC Boot Procedure 255
3.5 MSX-DOS and Disk BASIC Disk Drivers 256
3.6 MSX-DOS System Calls 267

4. Other Expansion

4.1 MSX-RS232C Support 290
4.1.1 Extended BASIC for RS-232C Communication 291
4.1.2 Extended BIOS Calls Handling RS-232C 300
4.2 Other MSX Extended BIOS Calls 309
4.2.1 Extended BIOS Calls 309
4.2.2 Extended BIOS Maker ID Number 313
4.3 Tenkey Support on MSX 314

PART D SOFTWARE DEVELOPMENT GUIDE

5. International MSX Versions and their Differences

5.1 Introduction 316
5.2 Keyboard 316
5.2.1 Keyboard Hardware 316
5.2.2 Character Set 317
5.2.3 Keyboard Layout 319
5.2.4 CAPS Lock 319
5.2.5 DEAD-Key Functions 331
5.3 Screen Mode 333
5.4 Other Differences among Versions 334
5.5 ID Bytes 335

6. Notes for MSX Software Developers 336

PART A

MSX HARDWARE SPECIFICATIONS

MSX HARDWARE SPECIFICATIONS

1. Hardware Specifications

1.1 MSX Standard

o CPU
  Z80A compatible

o MEMORY
  ROM: 32K bytes (MSX system software)
  RAM: 16K bytes (Minimum)

o SCREEN DISPLAY
  Text display: 32 x 24 (See Section 2.4)
  Graphics: 256 x 192
  Colors: 16

o CASSETTE TAPE
  FSK format, 1200/2400 Baud

o SOUND
  8 Octaves, 3 Voices

o KEYBOARD VERSIONS
  Alphanumerics, Japanese, Graphics (Japanese)
  Alphanumerics, European, Graphics (International)

o FLOPPY DISK DRIVES
  Hardware depends on the manufacturer
  Disk format MS-DOS-compatible

o PRINTER *
  8 bit parallel

o ROM CARTRIDGE AND I/O BUS
  Software cartridge and expansion BUS slots

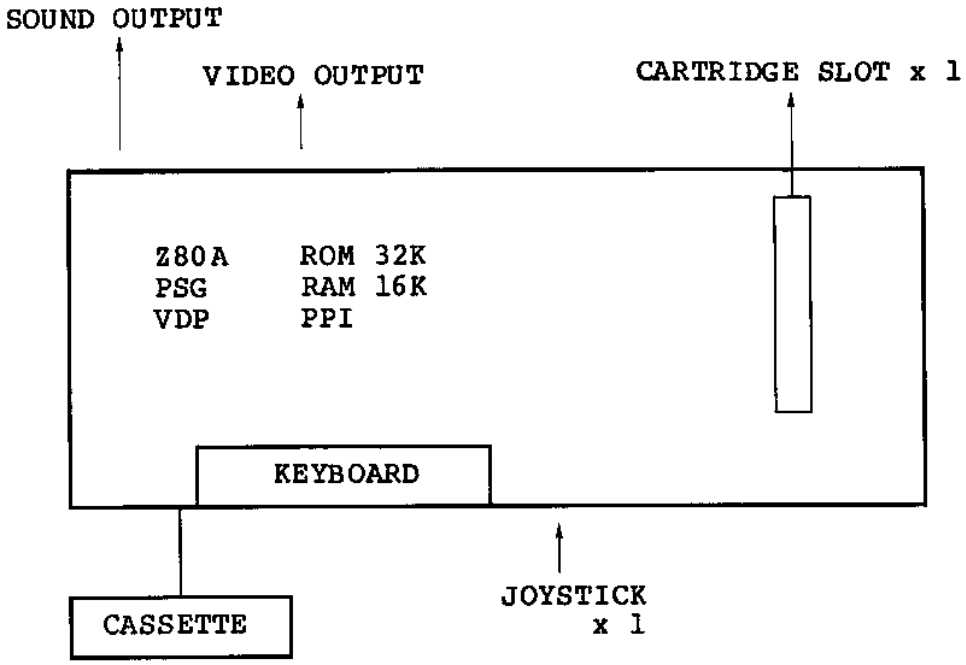
o JOYSTICKS *
  1 or 2

o CHINESE CHARACTERS *
  At manufacturer's disgression

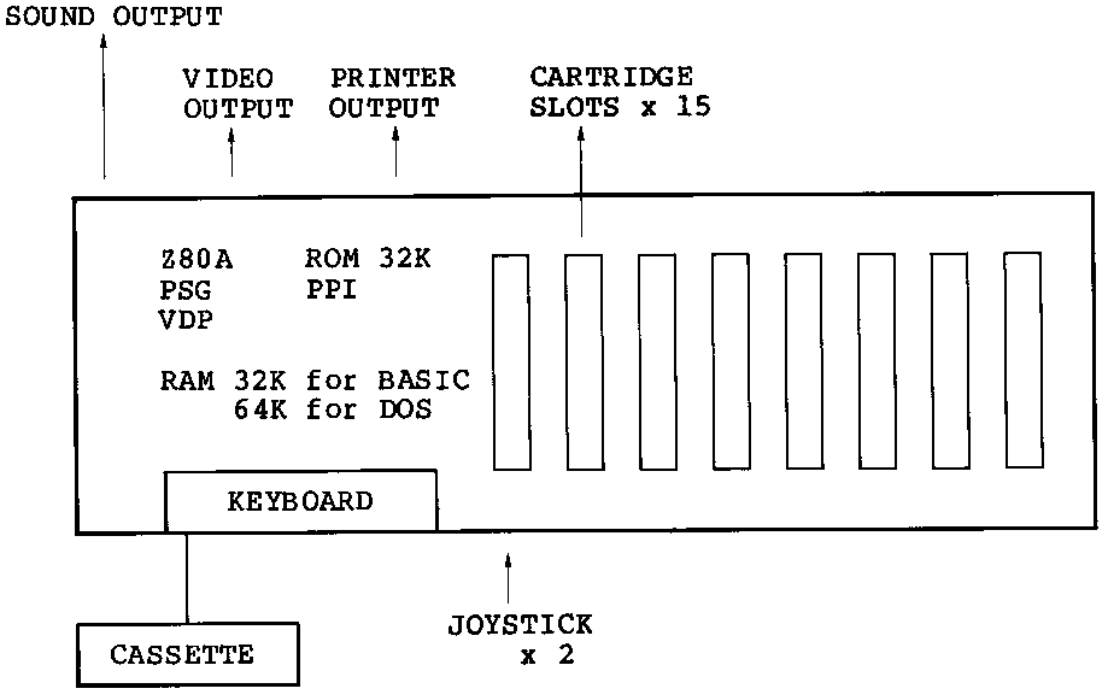
* The items with asterisks may not be provided in the basic system configuration.

1.2 MSX System Configuration

o MINIMUM CONFIGURATION



o SOFTWARE SUPPORT LIMIT



1.3 Main Unit

1.3.1 LSIs

- o CPU
  Z80A compatible
  Clock 3.579545MHz (NTSC Color sub-carrier frequency)
  1 WAIT in M1 CYCLE

- o VDP
  TI TMS-9918A compatible

- o PSG
  GI AY-3-8910 compatible

- o PPI
  Intel i-8255 compatible

1.3.2 Memory

- o ROM
  MSX-BASIC, 32K bytes

- o RAM
  Minimum 16K bytes

NOTE

Since the minimum system configuration contains four slots, the memory area may be expanded up to 256K bytes. Each slot can be further expanded to have four slots, for a total of 16 slots. Thus the maximum memory space is 1 megabyte.

The BASIC ROM interpreter occupies addresses 0000 to 7FFF, and the RAM addresses start at FFFF and grows downward on the memory map.

See the memory map in Section 1.7 for details.

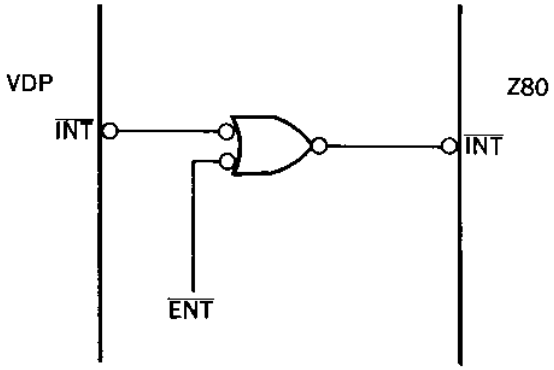
1.3.3 Interrupts

o NMI

Not used. MSX ROM only provides a RAM hook.

o INT

Interrupts are accepted from the VDP and the cartridges. The interrupt mode is 1. (Branch to 38H) The MSX system software uses an interrupt from the VDP. The interrupt intervals are 60 Hz in the NTSC version and 50 Hz in the PAL/SECAM version.



NOTE

It is not possible to support NMI under MSX-DOS because the address 66H (an entry vector for the NMI) is used by the MSX-DOS FCB data.

1.3.4 Screen

o LSI
TI TMS9918A Compatible

o Character set
Alphanumerics + Japanese (European) + Graphics
256 patterns, 8x8 dots

o Color
16 colors

o Sprites
32 sprites, with a maximum of four sprites on the same horizontal line.

o Display modes

MODE		RES.	SIZE	NO. *	COLOR	SPRITE AVAIL.	NO. OF CHARS.
Graphic I	LSI Spec.	256 x192	8 x 8	256	16 colors	Yes	32x24
	Suggested value	240 x192					29X24
Graphic II	LSI Spec.	256 x192	8 x 8	768	16 colors	Yes	32x24
	Suggested value	240 x192					29X24
Multi-color	LSI Spec.	64 x48blk	4 x 4 /Block	-	16 colors	Yes	32x24
	Suggested value	64 x40blk					29x24
Text	LSI Spec.	256 x192	8 x 6	256	2 out of 16 colors	No	40x24
	Suggested value	240 x193					39x24

* Number of patterns

Suggested values : The eight pixels from the left and right of the horizontal line are not used by the software.

1.3.5 Keyboard

o Layout

Alphanumerics	: ASCII standard
Japanese syllables	: JIS standard syllable layout
European	: International versions
Graphic Characters	: Depending on international version (Selected by jumper connection)

o Scanning

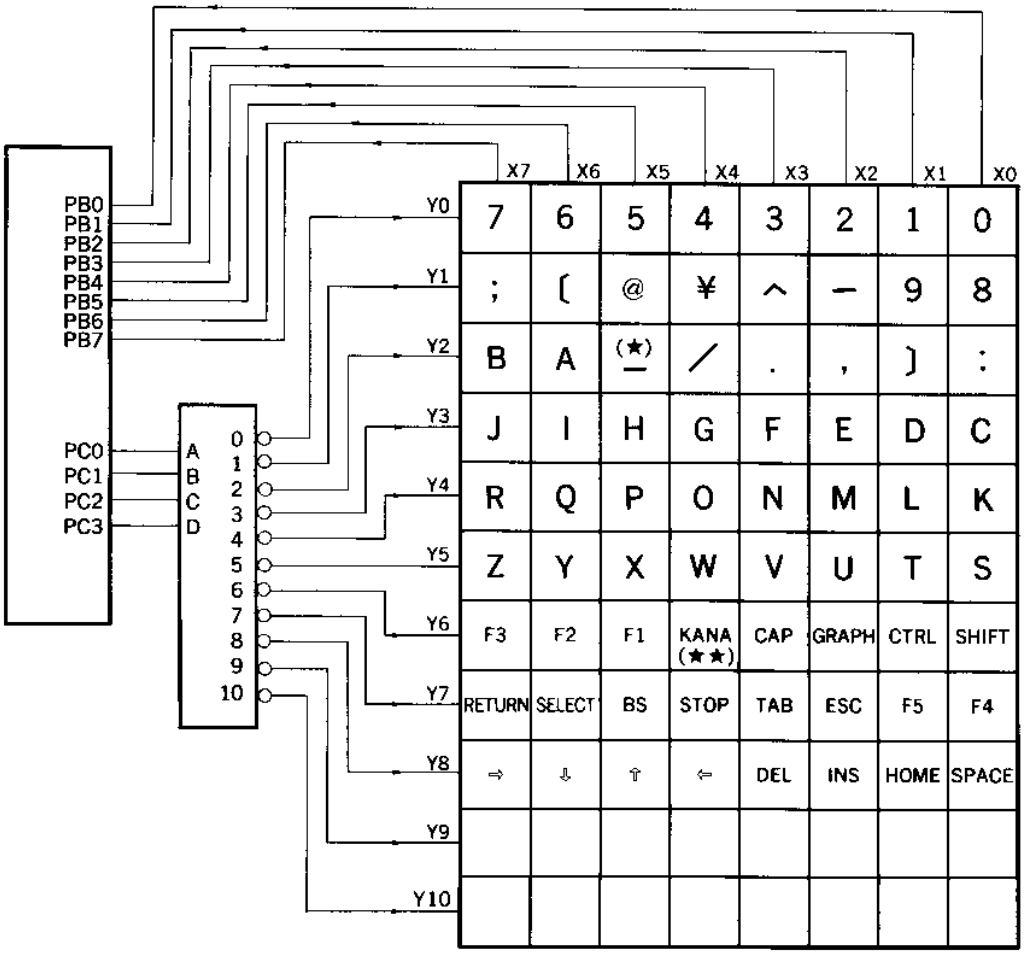
Software scanning driven by VDP interrupt

o Number of keys

72

See section 5.2.2/5.2.3 for details.

o Matrix diagram



★ Underscore character.
★★ Code Lock key in international versions.

1.3.6 Sound

- LSI
  GI AY-3-8910 Compatible.Clock 1.7897725 MHz (1/2 CPU clock)

- OCTAVES
  8 Octaves (3 Voices)

- SOUND EFFECTS
  Available

- SOFTWARE SOUND OUTPUT
  1 bit from output port

- OUTPUT LEVEL
  -5dbm (Providing the system has an output connector)

- CONNECTOR
  RCA 2 pins (Providing the system has an output connector)

1.4 Interfaces

1.4.1 Cassette Interface

o INPUT
From the earphone terminal of the tape recorder

o OUTPUT
To the microphone terminal of the tape recorder

o SYNCHRONIZATION
Asynchronous, software-controlled

o BAUD RATES
1200 baud (1200Hz - 1 wave "0", 2400Hz - 2 waves "1") (Default)

2400 baud (2400Hz - 1 wave "0", 4800Hz - 2 waves "1"), software-selected
(The tape recorder to be used may have to be specified by the manufacturer when using 2400 baud)

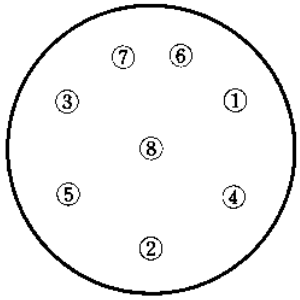
o MODULATION
FSK (Frequency Shift Keying), software-controlled

o DEMODULATION
Software-controlled. The system software automatically detects the baud rate upon receiving the data.

o MOTOR CONTROL
Available

o CONNECTOR
DIN 45326 (8 pins)

o TABLE OF SIGNAL PINS

PIN NO.	SIGNAL NAME	DIRECTION	PIN CONNECTION
1	GND	---	
2	GND	---	
3	GND	---	
4	CMTOUT	OUTPUT	
5	CMTIN	INPUT	
6	REMOTE +	OUTPUT	
7	REMOTE -	OUTPUT	
8	GND	---	

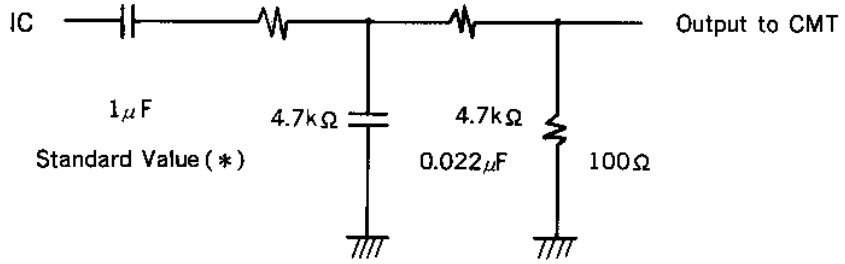
o SAVE Level
The constants in the SAVE circuit should be adjusted so as to perform the output level as follows:

Output level -45 dBm ± 5 dBm (0 dBm = 0.775 V)

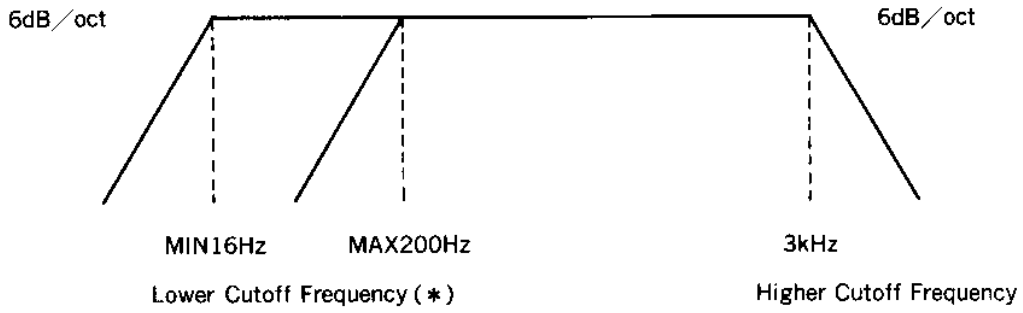
The output should be 22 mVp-p ~ 7 mVp-p at 1200 Hz input signal.



o Sample Circuit for SAVE



o Frequency Characteristics



* Note that the lower cutoff capacitor is to protect the IC of MSX. Cassette tape recorders themselves will not be harmed even if it is not there. The capacitance may be in the range 0.1 ~ 2.2μF. Adjust the capacitor to limit the lower cutoff frequency in the range 16 ~ 200Hz, if the output inpedance of the IC is too high.

1.4.2 Floppy Disk Interface

- The Floppy Disk Interface contains 16K bytes of ROM beginning at 4000H that includes the following modules:

  * MSX-DOS KERNEL
  * MSX DISK BASIC
  * PHYSICAL DISK I/O DRIVER (Supplied by manufacturer)

- The hardware interface is not specified. The physical disk I/O driver supplied by the manufacturer should resolve the hardware differences.

- Ideally, the mechanism in the disk drive should detect that the drive door has been opened. This reduces the number of disk accesses required to check if the system disk been replaced.

- Floppy disk format: MS-DOS compatible

8-inch	Single-density	128 Bytes/Sector
8-inch	Double-density	1024 Bytes/Sector
5.25-inch	Double-density	512 Bytes/Sector
3.5-inch	CFD	512 Bytes/Sector
3-inch	CFD	512 Bytes/Sector

1.4.3 Printer Interface

o SPECIFICATIONS
8 bit parallel, handshakes by BUSY and STROBE

o LEVEL
TTL

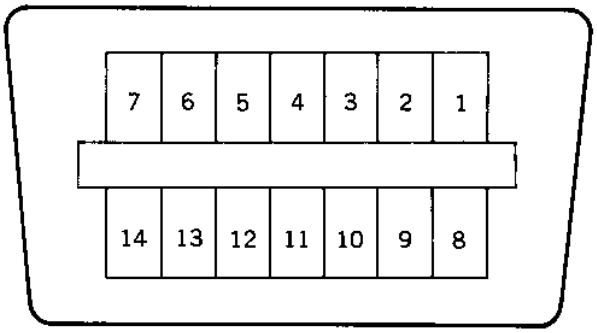
o CHARACTER CODES
Same as the MSX display codes

o CONNECTOR
14-pin AMP compatible

o LIST OF PINS

PIN NO.	SIGNAL NAME	I/O
1	$\overline{\text{PSTB}}$	O
2	PDB0	O
3	PDB1	O
4	PDB2	O
5	PDB3	O
6	PDB4	O
7	PDB5	O
8	PDB6	O
9	PDB7	O
10	N.C.	-
11	BUSY	I
12	N.C.	-
13	N.C.	-
14	GND	-

PIN CONNECTION



1.4.4 RS-232C Interface

o LSI COMPONENTS

i-8251 Communications Interface
i-8253 Programmable Interval Timer

At least 4K bytes of ROM is required for software support.

o PORT ADDRESSES

80H	R/W	8251 Data Port
81H	R/W	8251 Command/Status Port
82H	R	Status Sense Port for CTS, Timer/Counter 2, RI, and CD
82H	W	Interrupt Mask Register
83H		Reserved
84H	R/W	8253 Counter 0
85H	R/W	8253 Counter 1
86H	R/W	8253 Counter 2
87H	W	8253 Mode Register

* The port at address 83H is reserved for use by the manufacturer.

o USING THE PORT AT ADDRESS 82H

82H Read: Get System Status

Data Bit	Description
D7	CTS (Clear To Send)
	0: CTS Asserted
	1: CTS Negated
D6	Timer/Counter Output-2 from i8253
D5	Reserved
D4	
D3	
D2	
D1	+ RI (Ring Indicator)
	0: RI Asserted
	1: RI Negated
D0	+ CD (Carrier Detect)
	0: CD Asserted
	1: CD Negated

NOTE: The signals with the plus (+) sign are optional. If only one signal is chosen, it must be 'CD'.

NOTE

The CTS signal is sensed through the port instead of through the 8251 because of a problem in the CTS logic in some versions of the 8251. Software handling is thus made possible.

82H Write: Interrupt Mask Register

Data Bit	Description
D7	Reserved
D6	Reserved
D5	Reserved
D4	Reserved
D3	+ Timer Interrupt from i8253 channel-2 1: Mask Interrupt (Initial value) 0: Enable Interrupt
D2	+ Sync character detect/Break detect 1: Mask Interrupt (Initial value) 0: Enable Interrupt
D1	+ Transmit Data Ready (Tx Ready) 1: Mask Interrupt (Initial value) 0: Enable Interrupt
D0	Receive Data Ready (Rx Ready) 1: Mask Interrupt (Initial value) 0: Enable Interrupt

NOTE: The signals above with the plus (+) sign are optional. The minimum requirement for the interrupt signal is thus Rx Ready.

o USING THE 8253 TO GENERATE BAUD RATE CLOCK FOR THE 8251

A. CRYSTAL FREQUENCY

The crystal frequency is 1.8432 MHz.

Baud rate (Baud)	Scale Factor and Error (x16)		
50	2304		
75	1536		
110	1047	110.0287	+0.3%
150	768		
300	384		
600	192		
1200	96		
1800	64		
2000	58	1986.2	-0.7%
2400	48		
3600	32		
4800	24		
7200	16		
9600	12		
19200	6		

B. USING THE COUNTER CHANNEL

CH0: Rx Baud rate clock
CH1: Tx Baud rate clock
CH2: Used by application (Interrupt generated optionally)

o PINS OF DB25 CONNECTOR

Pin	Signal
1	Frame Ground
2	Transmit Data
3	Receive Data
4	Request To Send
5	Clear To Send
6	Data Set Ready
7	Signal Ground
8	Carrier Detect
9	
10	
11	
12	
13	

Pin	Signal
14	
15	
16	
17	
18	
19	
20	Data Terminal Ready
21	
22	Ring Indicator
23	
24	
25	

1.4.5 Peripheral I/O Port(s) (1 or 2)*

o LSI
AY-3-8910 compatible

o I/O
Input 4 bits, Output 1 bit, Bidirectional 2 bits per port

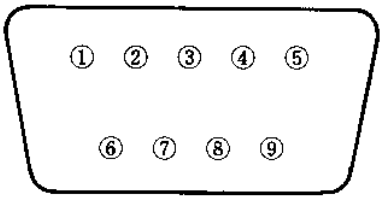
o LOGIC
Active high

o LEVEL
TTL

o CONNECTOR
9-pin AMP compatible

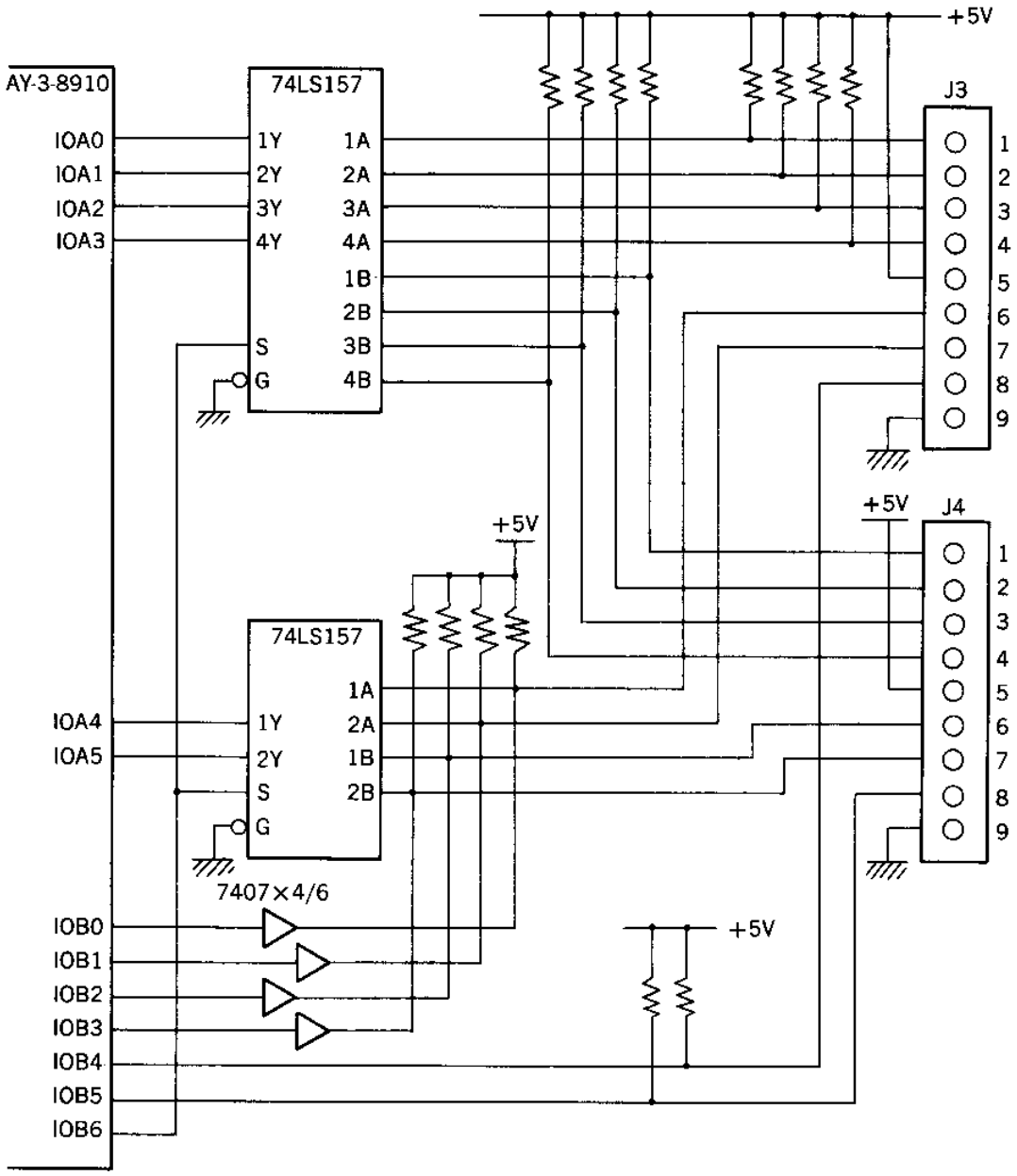
o LIST OF PINS

PIN NO.	SIGNAL NAME	DIRECTION	PIN CONNECTION
1	FWD	Input	
2	BACK	Input	
3	LEFT	Input	
4	RIGHT	Input	
5	+ 5V *	---	
6	TRG 1	Input/ Output	
7	TRG 2	Output	
8	OUTPUT	Output	
9	GND	---	



* Current capacity: 50mA each

o Circuit Diagram



All resistors are 10k ohm typically.

1.4.6 Joysticks

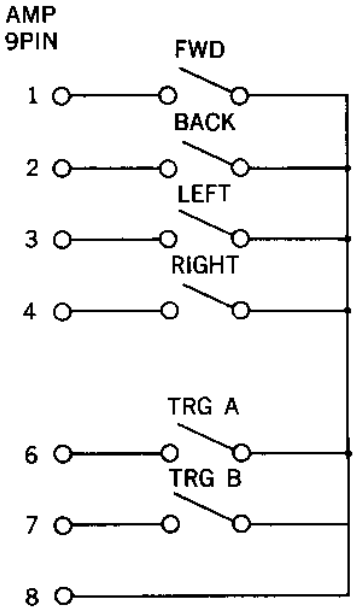
o There are two types of joysticks.

Joystick Type A has one trigger button, or if there is more than one trigger button, the software cannot distinguish between them.

Joystick Type B has two independent trigger buttons.

The joysticks produced from now on should show which type they are and software that needs to have Type B should say so on the package.

o Circuit Diagram



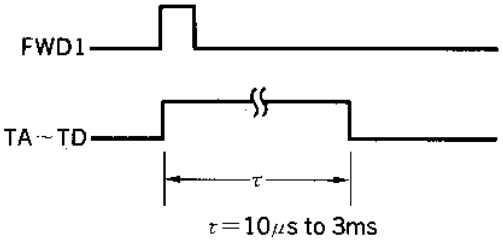
Option as described above.

1.4.7 Paddles

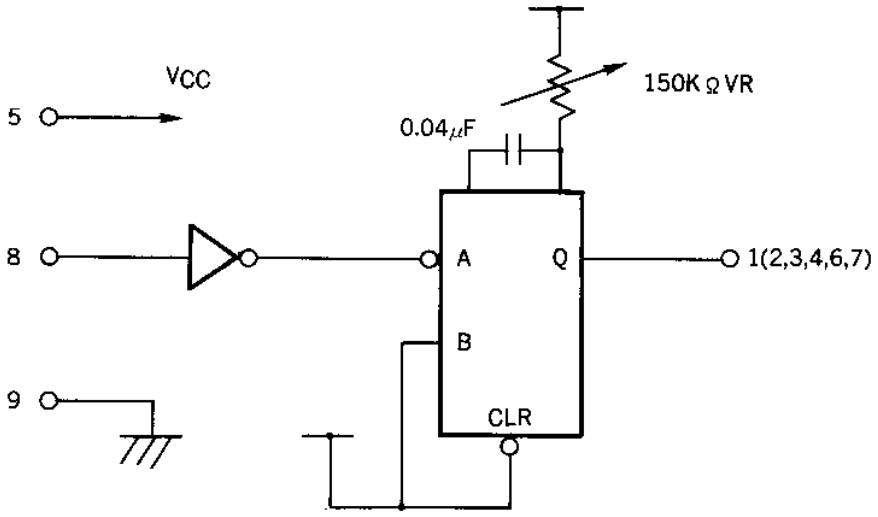
o A trigger pulse is sent to the 8 pin of the peripheral I/O port every time the PDL function is called. The paddle circuit, triggers the monostable multivibrator with this pulse. A pulse of the length corresponding to the level of the volume is returned to the port.

A maximum of 6 channels of paddles can be attached to each I/O port.

Paddle timing diagram



Circuit diagram (for 1 channel)



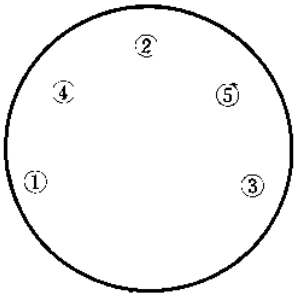
LS 123 or EQUIV.

NOTE: The volume (or the capacitance) should be adjustable as to justify the function of the paddle.

1.4.8 Connectors

PIN NAME	SPECIFICATIONS
1. Video output and composite video	DIN 5-Pin Connector *, or RCA 2-Pin Connector
2. RF modulated signal	RCA 2-Pin Connector
Cassette	DIN 8-Pin Connector (DIN-45326)
I/O Port	AMP 9-Pin Connector
Printer	UNPHENOL 14-Pin Connector
Cartridge Bus	2.54 PACE, 50-Pin Connector
Audio	RCA 2-Pin Connector

* DIN 5-PIN CONNECTOR SIGNAL PIN ASSIGMENTS

PIN NO.	NAME	PIN CONNECTION
1	+5V	
2	GND	
3	Audio	
4	Monitor	
5	RF Video	

1.4.9 Slots

o CONCEPT OF SLOTS

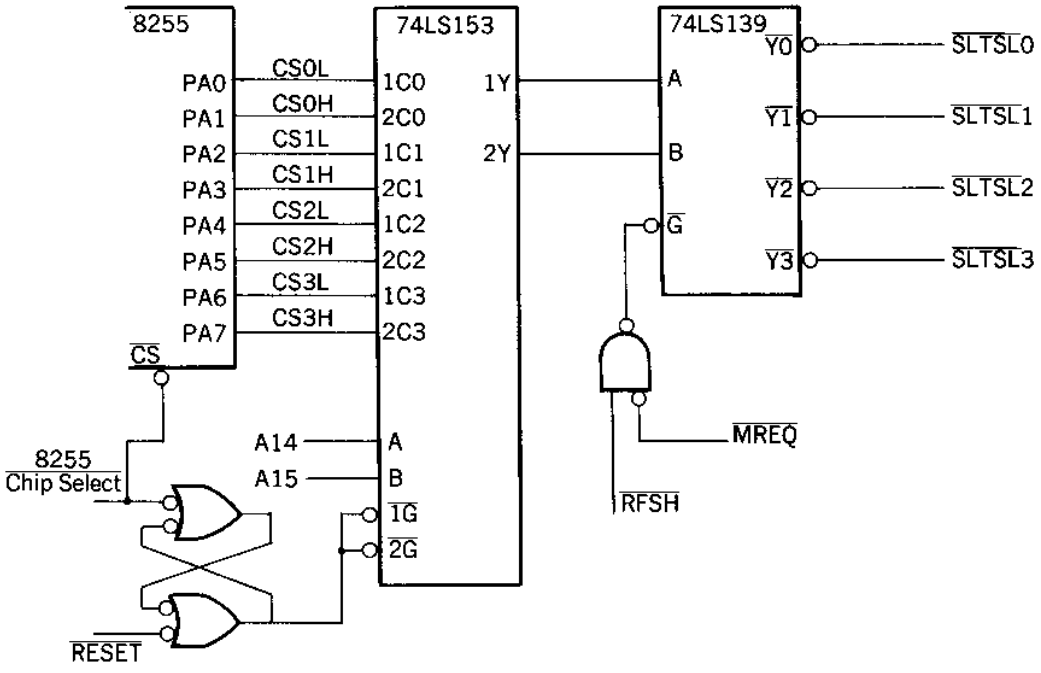
For computers having 64K bytes of memory, the concepts of slots and memory banking are nearly identical. The CPU can directly choose the cartridge by its slot number.

The slot concept originated from a desire to support the maximum amount of software. Using the slots, the software can be run, regardless of the number of physical slots available to the computer.

o ADVANTAGES OF SLOT STRUCTURE

In a common bus structure, when there is an even number of memory banks, the device select signal connected to the bus cannot distinguish between the different devices by using the same memory area. If this were to occur, the system would not only be unusable, but the hardware would quickly deteriorate. By using the slot select signal to choose the memory devices, the above problem is avoided, and programs that handle two or more devices having the same memory area are made possible. This is a favorable point, considering the system's flexibility and expandability.

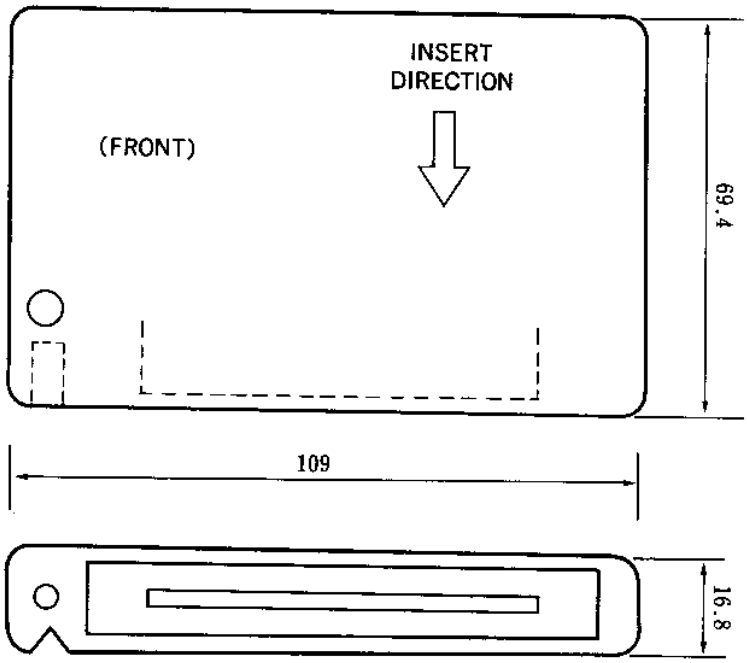
o Circuit diagram



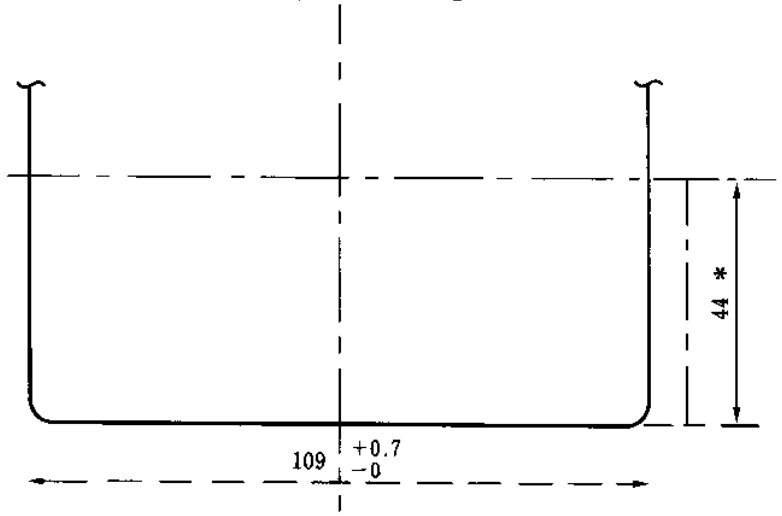
1.5 Cartridges

1.5.1 Physical Cartridge Specifications

o Physical dimension of the standard cartridge



o Physical dimension of the expanded cartridge



1.5.2 Cartridge Bus

o LIST OF SIGNAL PINS

PIN NO.	NAME	* I/O	PIN NO.	NAME	* I/O
1	$\overline{CS1}$	O	2	$\overline{CS2}$	O
3	$\overline{CS12}$	O	4	$\overline{SLTSL}$	O
5	Reserved #	-	6	$\overline{RFSH}$	O
7	$\overline{WAIT}$%	I	8	$\overline{INT}$%	I
9	$\overline{M1}$	O	10	$\overline{BUSDIR}$	I
11	$\overline{IORQ}$	O	12	$\overline{MERQ}$	O
13	$\overline{WR}$	O	14	$\overline{RD}$	O
15	$\overline{RESET}$	O	16	Reserved #	-
17	A9	O	18	A15	O
19	A11	O	20	A10	O
21	A7	O	22	A6	O
23	A12	O	24	A8	O
25	A14	O	26	A13	O
27	A1	O	28	A0	O
29	A3	O	30	A2	O
31	A5	O	32	A4	O
33	D1	I/O	34	D0	I/O
35	D3	I/O	36	D2	I/O
37	D5	I/O	38	D4	I/O
39	D7	I/O	40	D6	I/O
41	GND	-	42	CLOCK	O
43	GND	-	44	SW1	-
45	+5V	-	46	SW2	-
47	+5V	-	48	+12V	-
49	SOUNDIN	I	50	-12V	-

* The Input/Output directions are relative to the main unit.
# Do not use the Reserved PINs.
% OPEN COLLECTOR output

o LIST OF SIGNAL PINS

PIN NO.	NAME	DESCRIPTION
1	$\overline{CS1}$	ROM 4000 to 7FFF, selected signal
2	$\overline{CS2}$	ROM 8000 to BFFF, selected signal
3	$\overline{CS12}$	ROM 4000 to BFFF, selected signal (for 256K ROM)
4	$\overline{SLTSL}$	Slot select signal
5	Reserved	Reserved for future expansion. Do not use this pin.
6	$\overline{RFSH}$	Refresh signal
7	$\overline{WAIT}$	Wait signal to CPU
8	$\overline{INT}$	Interrupt request signal
9	$\overline{M1}$	Fetch cycle signal of CPU
10	$\overline{BUSDIR}$	This signal controls the direction of the external data bus buffer when the cartridge is selected. It is LOW when the data is sent by the cartridge.
11	$\overline{IORQ}$	I/O request signal
12	$\overline{MERQ}$	Memory request signal
13	$\overline{WR}$	Write signal
14	$\overline{RD}$	Read signal
15	$\overline{RESET}$	System reset signal
16	Reserved	Reserved for future expansion. Do not use this pin.
17~32	A0~A15	Address bus
33~40	D0~D7	Data bus
41	GND	Ground
42	CLOCK	CPU clock, 3.579 MHz
43	GND	Ground
44, 46	SW1, SW2	Detect Insert/Remove for protection
45, 47	+5V	+5V power supply
48	+12V	+12V power supply
49	SOUNDIN	Sound input (-5 dbm)
50	-12V	-12V power supply

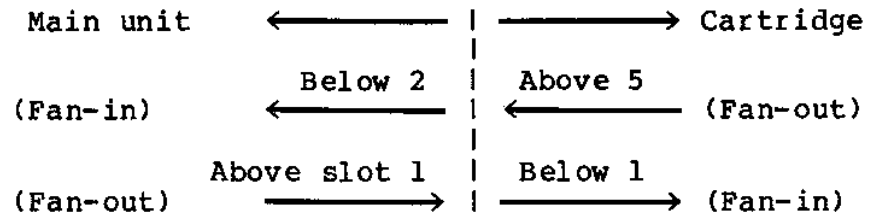
NOTE

The CS signals imply a memory request and a read signal. Thus they cannot be used as chip select for writable devices such as RAMs.

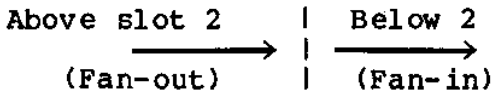
1.5.3 Cartridge Bus Connection Conditions

o FAN-IN, FAN-OUT (LS-TTL load)

Data and Address bus



o CONTROL SIGNALS



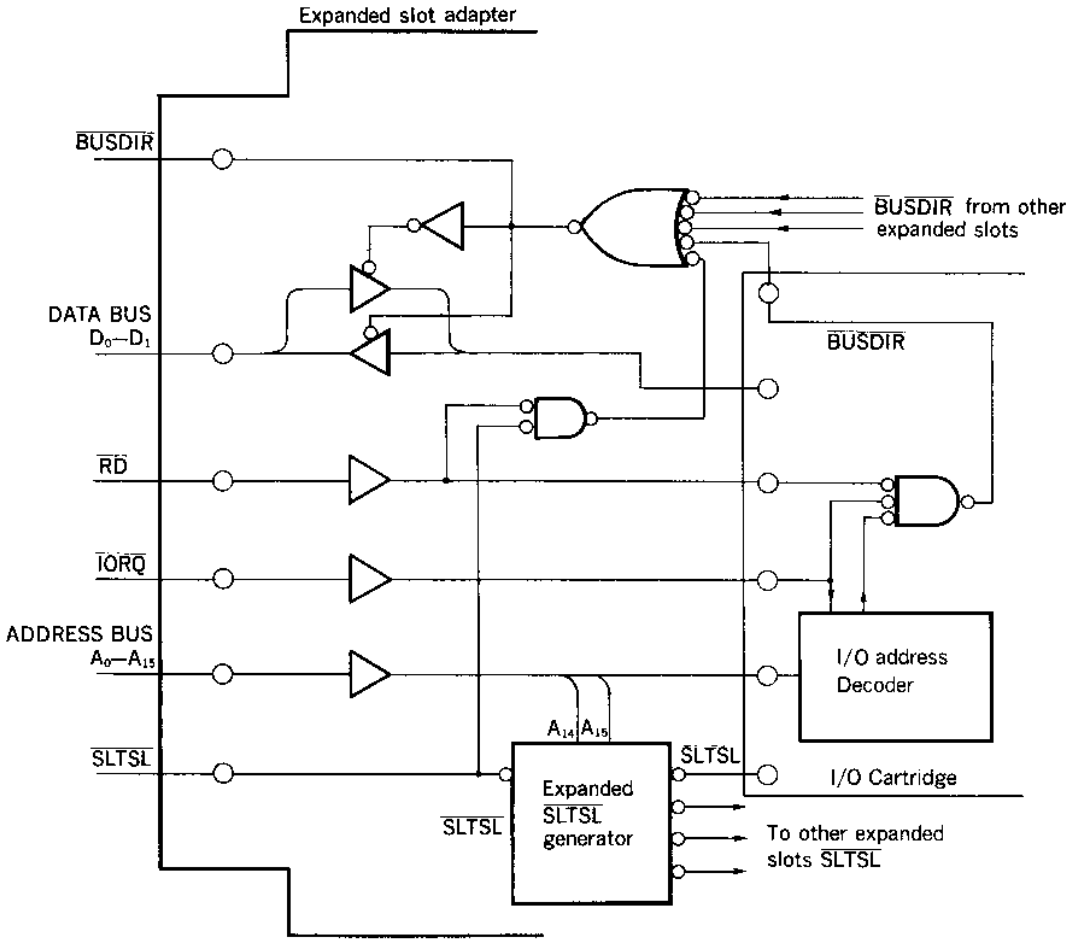
o VOLTAGE LEVEL

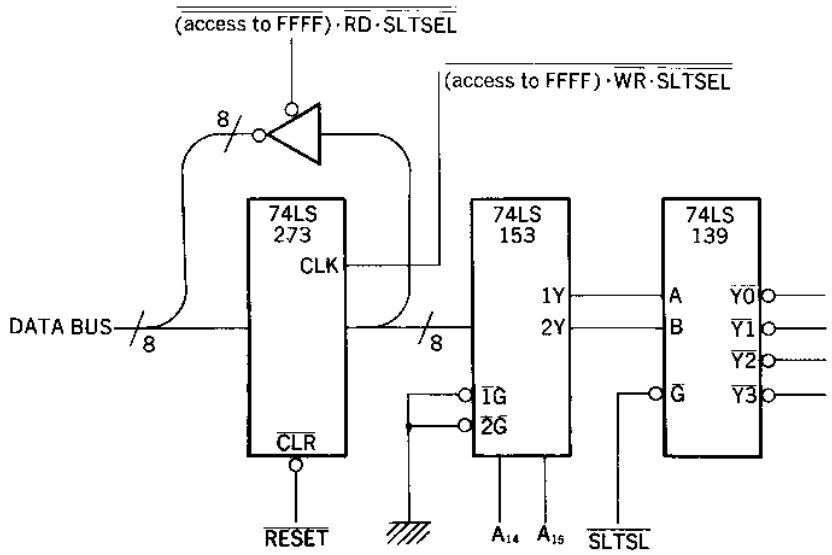
TTL level

1.5.4 Cartridge Power Capacity

+5V	300 mA/slot
+12V	50 mA
-12V	50 mA

1.5.5 Sample Circuit Diagram of Expanded Slot Select Signal





1.6 Notes for System Expansion

1.6.1 RAM Expansion

- Since MSX-BASIC needs a contiguous RAM area from 8000 to FFFF, the additional RAM should be added to the existing RAM so as to be contiguous.

- Since the MSX-BASIC software requires only RAM from 8000 to FFFF, RAM installed from 0000 to 7FFF cannot be used by it.

1.6.2 Slot Expansion

- When slots are expanded, the expanded slots must be expanded from a primary slot. Primary slots are those slots managed by the slot select register provided in port A of the 8255. Thus, to select an expansion slot, first select the primary slot to which the expansion slot is connected, then select the desired slot.

- The slots directly attached to the MSX computer itself must be primary slots. Because there are significant differences between the primary and secondary slots, there must be a clear indication of which kind of slot is placed in an expansion adapter.

- The location of the slot select register for the additional slots is address FFFF of the primary slot. To make it possible to differentiate the register from ordinary RAM, take the complement of the output of the register. That is, when the register is read, the data is the complement of the value of the register.

- A maximum of four cartridges can be connected to the cartridge bus. Therefore, buffers are necessary if the system is to support more than five slots. The BUSDIR signal controls the direction of those buffers. Devices placed in expansion slots that send signals to the CPU must also send the BUSDIR signal to change the direction of the expansion slots to the CPU. However, for memory accesses, it is possible to determine the direction of the bus by using the slot select signal sent to the primary slot, the memory request signal, and the read/write signal. The direction of the buffer should thus be controlled around the buffer circuit; cartridges containing only ROM or RAM thus do not have to manage the BUSDIR signal, and expansion RAM cartridges do not have to be expensive.

  Cartridges containing devices to send signals to the CPU (those devices responding to the INP instruction or supplying an address in response to mode 2 interrupts) must force BUSDIR to the 'L' level when sending data to CPU.

1.6.3 I/O Expansion

- In Z-80 based system, it is common to place I/O devices in the I/O address space. Since the MSX system was designed to be flexible and expandable, it is possible to add I/O devices using cartridges that share the same address space. If this is the case, those devices will not be able to be accessed properly.

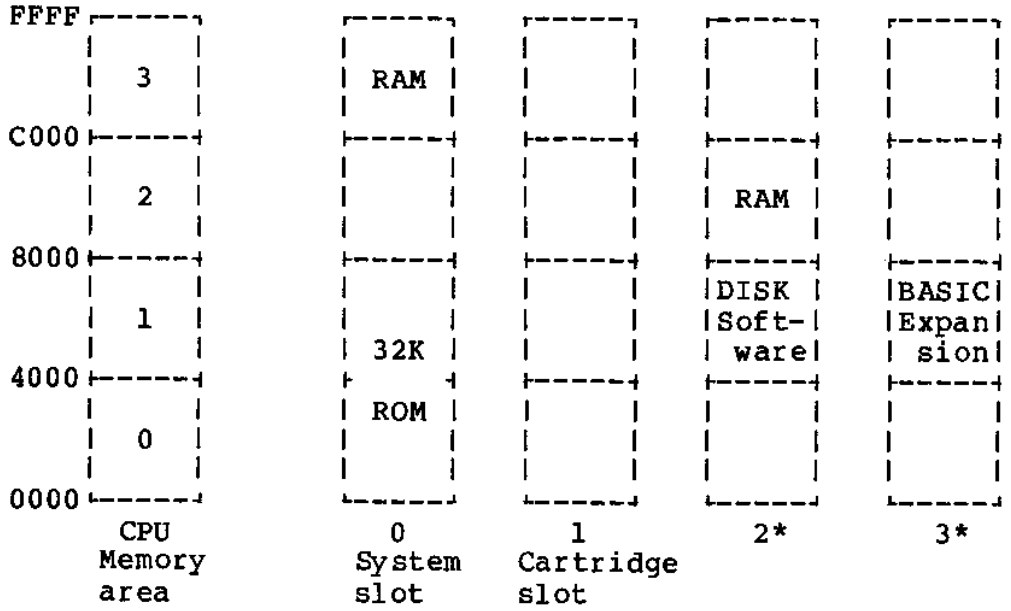
  To avoid the above situation, the I/O devices should be placed in the memory area because they will be managed by slot select logic and the memory cannot be accessed simultaneously when placed in different slots, since devices placed in the memory area cannot be accessed by software running in different slots. General devices such as the VDP must therefore be placed in the I/O address space. Note also that in some cases it is more economical to use the I/O address space, because only eight bits of address information have to be decoded.

  The MSX system specifications define the system device I/O address space to be addresses from 40 to FF. The addresses below 3F are left free. While other devices may use this address space, other manufacturers may use the same addresses for other purposes. Thus, we recommend that memory addresses be used instead of the I/O area. In later MSX versions it is possible that standard devices will use the unassigned (reserved) addresses.

1.7 Address Maps

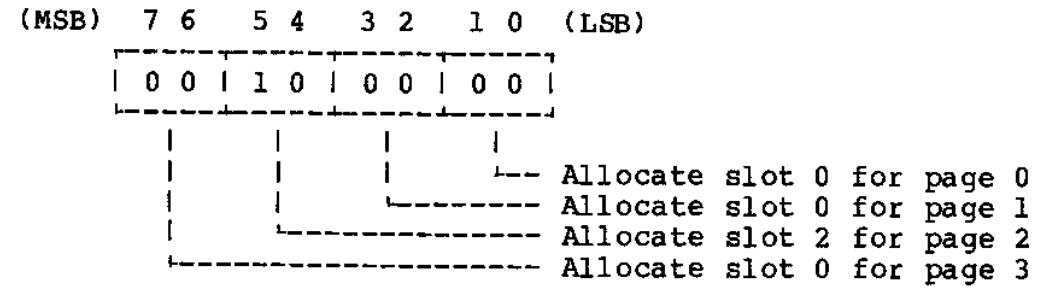
1.7.1 Memory Map

o The following is an example memory map.



o MSX BASIC uses the largest contiguous available RAM area installed from 8000 to FFFF for its system working RAM area. This RAM may be placed in any slot, including the expansion slots.

o The slot select register, port A of the 8255, maps the physical memory space to the logical CPU memory space in 16K-byte units (pages). For example, the following value in the slot select register allocates pages 0 and 1 from slot 0, page 2 from slot 2, and page 3 from slot 0.



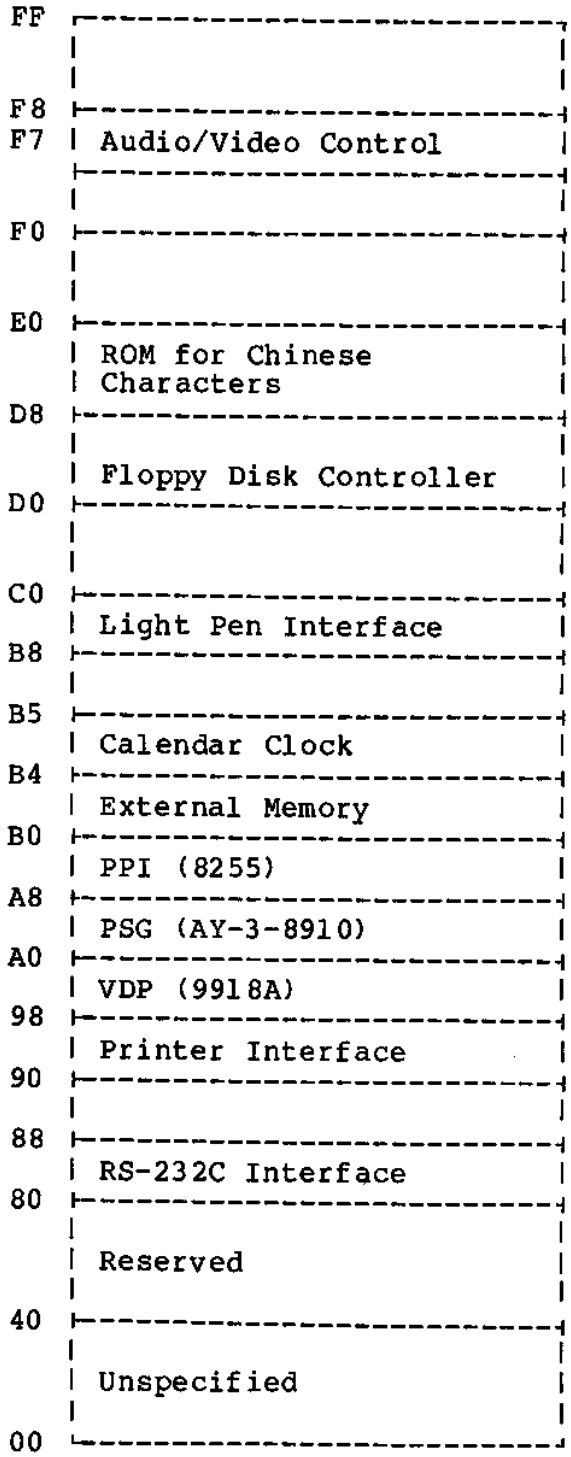
The physical memory is always allocated to the same memory page in the CPU memory space. It is not possible to allocate it to a different page, as in allocating page 3 of slot 3 to page 0 of the CPU memory space.

- The minimum system must have two slots, one for the system, and the other for the cartridge.

NOTE

The meaning of "slot" does not imply that it must have a cartridge connector; however, the cartridge slot must have the cartridge connector.

1.7.2 I/O Address Map



1.7.3 Printer Port

90H	R	Busy state:	Bit 1
90H	W	Strobe output:	Bit 0
91H	W	Print data	

1.7.4 VDP Port

98H	R/W	Video RAM data
99H	R/W	Command and status register

1.7.5 PSG Port

A0H	W	Address latch
A1H	W	Data write
A2H	R	Data read

1.7.6 PPI Port

A8H	R/W	Port A
A9H	R/W	Port B
AAH	R/W	Port C
ABH	R/W	Mode register

1.7.7 External Memory (Sony)

B0H thruogh B3H

1.7.8 Light Pen (Sanyo)

B8H through BBH

1.7.9 Audio/Video Control

F7H	W	BIT4 - AV Control	L - TV
	W	BIT5 - Ym Control	L - TV
	W	BIT6 - Ys Control	L - Super
	W	BIT7 - Video select	L - TV

1.7.10 Notes on I/O Address Assignments

o I/O addresses 40-FF are assigned for system use. The unused empty area is also reserved for system use.

  Although I/O addresses are defined above, the software must not access those devices directly using the above ports. All I/O accesses must be done using BIOS calls, in order to make the software independent of hardware differences. MSX manufacturers may change some of the hardware from the standard MSX system and maintain software compatibility by rewriting BIOS. The hardware differences would thus be transparent to the software.

  The only exception to the above is the access to the VDP. Locations 6 and 7 of the MSX system ROM contains the Read and Write addresses of the VDP register. Software that must access the VDP quickly may access the VDP directly by using the addresses stored in ROM.

o Addresses 00 to 3F are free. Different devices using the same address must not be accessed simultaneously. In general, the I/O devices that are not defined here should be placed in the memory space as memory-mapped I/O. See section 1.6.3 for further details.

% The FDC may be placed in the I/O area; however, it must have a mechanism to disable it, and it must be enabled only if the system does accesses to the FDC. This makes it possible for the system to have more than one FDC interface for handling different media types.

1.7.11 8255 (PPI) Bit Assignments

PORT	BIT	I/O	SIGNAL NAME	DESCRIPTION
A	0		CS0L	
				0000-3FFF Address slot select signal
	1	O	CS0H	
	2	U	CS1L	
				4000-7FFF Address slot select signal
	3	T	CS1H	
	4	P	CS2L	
				8000-BFFF Address slot select signal
	5	U	CS2H	
	6	T	CS3L	
				C000-FFFF Address slot select signal
	7		CS3H	
B	0	I		Keyboard return signal
	|	N		
	|	P		
	|	U		
	7	T		
C	0		KB0	
	1		KB1	Keyboard scan signal
	2	O	KB2	
	3		KB3	
		U		
	4	T	CASON	Cassette control signal (L=ON)
		P		
	5		CASW	Cassette write signal
		U		
	6	T	CAPS	CAPS lamp signal (L=ON)
	7		SOUND	Software-controlled sound output

1.7.12 PSG Bit Assignments

PORT	BIT	I/O	CONNECTOR PIN NO.		NOTES
A	0		J3-PIN 1	#1	FWD1
			J4-PIN 1 *	#2	FWD2
	1	I	J3-PIN 2	#1	BACK1
			J4-PIN 2 *	#2	BACK2
	2	N	J3-PIN 3	#1	LEFT1
			J4-PIN 3 *	#2	LEFT2
	3	P	J3-PIN 4	#1	RIGHT1
			J4-PIN 4 *	#2	RIGHT2
	4	U	J3-PIN 6	#1	TRGA1
			J4-PIN 6 *	#2	TRGA2
	5	T	J3-PIN 7	#1	TRGB1
			J4-PIN 7 *	#2	TRGB2
	6		KEY LAYOUT Select	#4	Japanese version only
	7		CSAR (Cassette tape READ)		
B	0		J3-PIN 6	#3	--
	1	O	J3-PIN 7	#3	"H" Level
	2	U	J4-PIN 6 *	#3	
	3	T	J4-PIN 7 *	#3	--
	4	P	J3-PIN 8		
	5	U	J4-PIN 8 *		
	6	T	PORT A INPUT SELECT		Selects J3 or J4
	7		KLAMP (KANA LAMP L=ON)		Japanese version only

#1 Available if bit 6 of port B is LOW and is used by JOYSTICK1
#2 Available if bit 6 of port B is HIGH and is used by JOYSTICK2
#3 Set these pins to "H" when using them as an input port.
Connect an open collector buffer to the output.
#4 JIS layout - "H", syllable layout - "L"

<Remark> PIN 5: +5V
PIN 9: GND

o On the minimum MSX system, there is no J4 connector.

PART B

MSX SYSTEM SOFTWARE

2. Language Specifications

2.1 MSX-BASIC Reference Guide

MSX-BASIC is an extended version of Microsoft Standard BASIC Version 4.5, and includes support for graphics, music, and various peripherals attached to MSX Personal Computers. Generally, MSX-BASIC is designed to follow GW-BASIC, which is one of the standard BASICs running on 16-bit computers. During the creation of MSX-BASIC, a major effort was made to make the system as flexible and expandable as possible.

MSX-BASIC also features a BCD-arithmetic function with a double-precision accuracy of up to 14 digits. Arithmetic operations thus do not generate rounding errors that tend to confuse new programmers. In addition, all trancendental functions are calculated with 14-digit accuracy. 16-bit, signed, integers are also available for faster execution.

2.1.1 Modes of Operation

When MSX-BASIC is initialized, it displays the "OK" prompt. "Ok" indicates MSX-BASIC is at command level; that is, it is ready to accept commands. At this point, MSX-BASIC may be used in either of two modes: direct mode or indirect mode.

In the direct mode, MSX-BASIC statements and commands entered as they are without preceeding line numbers. They are executed immediately, and the results of arithmetic and logical operations may thus be determined quickly. While these results may also be stored for later use, the instructions themselves are lost after execution. Direct mode is thus useful for debugging and for using MSX-BASIC as a "calculator" for quick computations not requiring a complete program.

The indirect mode is used for entering programs. Program lines are preceded by line numbers and are stored in memory. The program stored in memory is executed by entering the RUN command.

2.1.2 Line Format

The program lines of MSX-BASIC programs must be in the following format. Square brackets denote statements that are optional.

nnnnn BASIC statement[:BASIC statement...] <Carriage Return>

An MSX-BASIC program line always begins with a line number and ends with a carriage return. A logical line may contain a maximum of 255 characters. More than one BASIC statement may be placed on a logical line, but the statements must be separated by a colon.

The line numbers indicate the order in which the program lines will be stored in memory, and in MSX-BASIC, they must be between 0 and 65529. They are also used as references during branching and editing.

During editing, a period (.) may be used with the LIST, AUTO, and DELETE commands to refer to the current line.

2.1.3 Character Set

The MSX-BASIC character set consists of alphabetic characters, numeric characters, special characters, graphic characters, and both (Japanese) hiragana and katakana characters. See section 5.2.2 for details.

The alphabetic characters in MSX-BASIC are the uppercase and lowercase letters of the alphabet.

The MSX-BASIC numeric characters are the digits 0 through 9.

In addition, the following special characters are recognized by MSX-BASIC:

Character	Action
	Blank
=	Equals sign or assignment symbol
+	Plus sign
-	Minus sign
*	Asterisk or multiplication symbol
/	Slash or division symbol
^	Up arrow or exponentiation symbol
(	Left parenthesis
)	Right parenthesis
%	Percent
#	Number (or pound) sign
$	Dollar sign
!	Exclamation point
[	Left bracket
]	Right bracket
,	Comma
.	Period or decimal point

'	Single quotation mark (apostrophe)
;	Semicolon
:	Colon
&	Ampersand
?	Question mark
<	Less than
>	Greater than
¥	Yen sign or integer division symbol (back slash in international versions)
@	At sign
_	Underscore
<Rubout>	Deletes last character typed.
<Escape>	Escapes
<Tab>	Moves print position to next tab stop. Tab stops are set every eight columns.
<Line feed>	Moves to next physical line.
<Carriage return>	Terminates input of a line.

2.1.4 Constants

Constants are the values MSX-BASIC uses during execution. There are two types of constants: string and numeric.

A string constant is a sequence of up to 255 alphanumeric characters enclosed in double quotation marks.

Examples:

```
"HELLO"
"$25,000.00"
"Number of Employees"
```

Numeric constants are positive or negative numbers. MSX-BASIC numeric constants cannot contain commas. There are six types of numeric constants:

1.	Integer constants	Whole numbers between -32768 and 32767. Integer constants do not contain decimal points.
2.	Fixed-point constants	Positive or negative real numbers, i.e., numbers that contain decimal points.
3.	Floating-point constants	Positive or negative numbers represented in exponential form (similar to scientific notation). A floating-point constant consists of an optionally signed integer or fixed-point number (the mantissa) followed by the letter E and an optionally signed integer (the exponent). The allowable range for floating-point constants is 10E-64 to 10E+63.

Examples:

235.988E-7 = .0000235988
2359E6 =2359000000

(Double-precision floating-point constants are denoted by the letter D instead of E.)

4. Hex constants — Hexadecimal numbers, denoted by the prefix &H.

Examples:

&H76
&H32F

5. Octal constants — Octal numbers, denoted by the prefix &O.

Examples:

&O347
&O1234

6. Binary constants — Binary numbers, denoted by the prefix &B.

Examples:

&B01110110
&B11100111

o Single- And Double-Precision Numeric Constants

Numeric constants may be either single-precision or double-precision numbers. Single-precision numeric constants are stored with 6 digits of precision, and are printed with up to 6 digits of precision. Double-precision numeric constants are stored with 14 digits of precision and printed with up to 14 digits. Double-precision is the default for constants in MSX-BASIC.

A single-precision constant is any numeric constant that has one of the following characteristics:

1. Exponential form using E.

2. A trailing exclamation point (!).
   Examples:

   -1.09E-06
   22.5!

A double-precision constant is any numeric constant that has one of these characteristics:

1. Any digits of number without any exponential or type specifier.

2. Exponential form using D.

3. A trailing number sign (#).

Examples:

```
 3489
 345692811
-1.09432D-06
 3489.0#
 7654321.1234
```

2.1.5 Variables

Variables are names used to represent values used in a BASIC program. The value of a variable may be assigned explicitly by the programmer, or it may be assigned as the result of calculations in the program. Before a variable is assigned a value, its value is assumed to be zero.

o Variable Names And Declaration Characters

MSX-BASIC variable names may be of any length. Up to 2 characters are significant. Variable names can contain letters and numbers; however, the first character must be a letter. Special type declaration characters are also allowed--see the discussion below.

A variable name may not be a reserved word and may not contain embedded reserved words. Reserved words include all MSX-BASIC commands, statements, function names, and operator names (See appendix for the list). If a variable begins with FN, it is assumed to be a call to a user-defined function.

Variables may represent either a numeric value or a string. String variable names are written with a dollar sign ($) as the last character, for example: A$ = "SALES REPORT".

The dollar sign is a variable type declaration character; that is, it "declares" that the variable will represent a string.

Variable names may also inherently declare the variables to be integer, single-precision, or double-precision. The last character in these variables must be one of the following variable-type declaration characters:

%	Integer variable
!	Single-precision variable
#	Double-precision variable

The default type for a numeric variable name is double-precision.

Examples of MSX-BASIC variable names:

PI#	Declares a double-precision value.
MINIMUM!	Declares a single-precision value.
LIMIT%	Declares an integer value.
N$	Declares a string value.
ABC	Represents a double-precision value.

Variable types may also be declared within a program by using the MSX-BASIC DEFINT, DEFSTR, DEFSNG, and DEFDBL statements. For details, refer to the descriptions of these statements.

o Array Variables

An array variable is a group or a table of values that is organized with the same variable name. Each element in an array is referenced by an array variable (having an integer or an integer expression as a subscript). Names for array variables may have as many subscripts as there are dimensions in the array. For example, V(10) would be the name of a variable in a one-dimension array, T(1,4) would be the name of a variable in a two-dimension array, and so on. MSX-BASIC supports a maximum number of 255 dimensions for an array. The maximum number of elements depends on the size of the computer's memory.

o Space Requirements

The following table lists the number of bytes that each variable occupies in memory.

Variables:	Type	Bytes
	Integer	2
	Single-Precision	4
	Double-Precision	8

Arrays:	Type	Bytes
	Integer	2 per element
	Single-Precision	4 per element
	Double-Precision	8 per element

Strings: 3 bytes for bookkeeping plus the length of the string.

2.1.6 Type Conversion

When necessary, MSX-BASIC will convert a numeric constant from one type to another. The following rules and examples should be kept in mind.

1. If a numeric constant of one type is set to a numeric variable of a different type in a LET statement, the number is converted and stored as the type declared by the new variable name, unless an attempt to set a string variable to a numeric variable is done. The latter case results the occurrence of a "Type mismatch" error.)

   Example:

```
10 A%=23.42
20 PRINT A%
RUN
 23
```

2. During the evaluation of an expression, all operands of the arithmetic or relational operation are converted to a uniform precision to match the most precise operand. The operation also results in the precision of the most precise operand.

   Examples:

```
10 D=6/7!
20 PRINT D
RUN
 .85714285714286
```

The operation was done in double-precision and the result, returned in D, is double-precision.

```
10 D!=6/7
20 PRINT D!
RUN
 .857143
```

The operation was done in double-precision and the result, returned to D! (a single-precision variable) was rounded and printed as single-precision.

3. Logical operators convert their operands to integers and return integer results. Operands must be between -32768 and 32767, or an "Overflow" error occurs.

4. When a floating-point value is converted to an integer, the fractional portion is truncated.

   Example:

```
10 C%=55.88
20 PRINT C%
RUN
 55
```

5. If a double-precision variable is set to a single-precision value, only the first six digits of the double-precision are valid. Single-precision variables support only a maximum of six digits.

Example:

```
10 A!=SQR(2)
20 B=A!
30 PRINT A!,B

RUN
 1.41421         1.41421
```

2.1.7 Expressions and Operators

An expression may be a string or numeric constant, a variable, or a combination of constants and variables with operators which produces a single value.

Operators perform mathematical or logical operations on values. MSX-BASIC operators may be divided into four categories:

1. Arithmetic
2. Relational
3. Logical
4. Functional

These categories will be described in the following sections.

o Arithmetic Operators

Arithmetic operators in MSX-BASIC have a defined order of precedence. The operators are listed below in order of precedence.

Operator	Operation	Example
^	Exponentiation	X^Y
-	Negation	-X
*,/	Multiplication, Floating-point Division	X*Y X/Y
+,-	Addition, Subtraction	X+Y

To change the above order of evaluation of operations, use parentheses. The operations embedded within parentheses will be evaluated first. Within the parentheses themselves, the above evaluation order is followed.

o Integer Division And Modulus Arithmetic

The following two additional operations, integer division and modulus arithmetic, are also available in MSX-BASIC:

Integer division is denoted by the yen symbol (or the backslash in international versions). The operands are truncated to integers (between -32768 and 32767) before division is done. The quotient

is truncated to an integer.

Examples:

```
10¥4=2
25.68¥6.99=4
```

Integer division follows both multiplication and floating-point division in the above order of precedence.

Modulus arithmetic is denoted by the operator MOD. Modulus arithmetic yields the (integer) remainder of integer division.

Example:

```
10.4 MOD 4=2      (10/4=2 with a remainder of 2)
25.68 MOD 6.99=1  (25/6=4 with a remainder of 1)
```

Modulus arithmetic follows integer division in the above order of precedence.

o Overflow Or Division By Zero

During the evaluation of an expression, if a division by zero is attempted, a "Division by zero" message is displayed, and the execution of the program is terminated. Also, if an overflow occurs during the evaluation of an expression, an "Overflow" message is displayed and the execution of the program is terminated.

o Relational Operators

Relational operators are used to compare two values. The result of the comparison is either "true" (-1) or "false" (0). The result can then be used to make decisions for program logic. (See the description on the "IF" statement.)

The relational operators are as follows:

Operator	Relationship	Example
=	Equality	X=Y
<>	Inequality	X<>Y
<	Less than	X<Y
>	Greater than	X>Y
<=	Less than or equal to	X<=Y
>=	Greater than or equal to	X>=Y

(The equals sign is also used to assign a value to a variable.)

When both arithmetic and relational operators are used in a single expression, the arithmetic operation is done first. For example,

X+Y<(T-1)/Z is true if the value of X + Y is less than the value of T-1 divided by Z.

More examples:

```
IF SIN(X)<0 GOTO 1000
IF I MOD J<>0 THEN K=K+1
```

o Logical Operators

Logical operators test multiple relationships, bit manipulation, or Boolean operations. The logical operator returns a one-bit result which is either "true" (not zero) or "false" (zero). Logical operations are performed after arithmetic and relational operations in expressions. The outcome of a logical operation is determined as shown in Table 1. The operators are listed in their order of precedence.

Table 1. Truth Table of MSX-BASIC Relational Operators

NOT

X	NOT X
1	0
0	1

AND

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

XOR

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

EQV

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

IMP

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1

Besides using relational operators to make decisions on program flow, logical operators can connect two or more relations and return true or false to be used in decisions.

Examples:

```
IF D<200 AND F<4 THEN 80
IF I>10 OR K<0 THEN 50
IF NOT P THEN 100
```

Logical operators convert their operands to 16-bit, signed, two's complement integers between -32768 and 32767. If the operands are not in this range, an error results. If both operands are supplied as 0 or -1, the logical operators return 0s or -1s. The given operation is done on the integers by the results of the corresponding bits in the two operands.

It is thus possible to use logical operators to test bytes for a particular bit pattern. For instance, the AND operator may be used to "mask" bits of a status byte for an I/O port. The OR operator may be used to "unmask" bits of a status byte for an I/O port. The following are examples of how the logical operators work.

63 AND 16=16	63 = binary 111111, and 16 = binary 10000, so 63 AND 16 = 16.
15 AND 14=14	15 = binary 1111, and 14 = binary 1110, so 15 AND 14 = 14 (binary 1110).
-1 AND 8=8	-1 = binary 1111111111111111, and 8 = binary 1000, so -1 AND 8=8.
4 OR 2=6	4 = binary 100, and 2 = binary 10, so 4 OR 2 = 6 (binary 110).
10 OR 10=10	10 = binary 1010, so 1010 OR 1010 = 1010 (decimal 10).
-1 OR -2=-1	-1 = binary 1111111111111111 and -2 = binary 1111111111111110, so -1 OR -2 = -1. The bit complement of sixteen zeros is sixteen ones (the two's complement representation of -1).
NOT X=-(X+1)	The two's complement of any integer is its bit complement plus one.

o Functional Operators

In MSX-BASIC, functions are used in expressions to call previously defined operations such as SQR (square root) and SIN (sine) for use in evaluating operands. Some are resident functions provided already in the MSX-BASIC interpreter.

Functions may also be defined within programs if they are not provided with the MSX-BASIC system. These functions may be defined by using the "DEF FN" statement. For a more detailed discussion, refer to the descriptions for "DEF FN".

o String Operations

Two or more strings may be concatenated by using a plus sign (+).

Example:

```
10 A$="FILE" : B$="NAME"
20 PRINT A$+B$
30 PRINT "NEW "+A$+B$
RUN
FILENAME
NEW FILENAME
```

Two strings may also be compared by using the same relational operators used for numbers, as shown below:

```
=    <>    <    >    <=    >=
```

Strings are compared by comparing the ASCII codes of both strings, comparing one character at a time. If all of the ASCII codes are the same, the strings are considered equal. If some of the ASCII codes are different, the string having the ASCII code with the lower code number will precede the other string. If the end of one of the strings is reached before the end of the other string is reached, the shorter string precedes the other string. During comparison, leading and trailing spaces are significant.

Examples:

```
"AA"<"AB"
"FILENAME"="FILENAME"
"X&">"X#"
"CL ">"CL"
"kg">"KG"
"SMYTH"<"SMYTHE"
B$<"9/12/83"      where B$="8/12/83"
```

Strings can thus be compared for alphabetization or for determining branching of program logic. Note that when string variables are compared, the expressions must be enclosed in quotation marks.

2.1.8 Program Editing

MSX-BASIC also includes a Full Screen Editor to allow the programmer to enter program lines and edit them using the entire screen.

The MSX-BASIC Full Screen Editor supports special keys for moving the cursor, for inserting or deleting characters, and for erasing lines or screens. These time-saving special functions and their key assignments will be discussed in the following sections.

With the Full Screen Editor, programmers can move the cursor anywhere on the screen and make the necessary corrections. To make changes, the cursor is placed on the first line to be changed, and after the changes are entered, the <RETURN> key is pressed at the beginning of each line. Lines in the stored programs will not be changed unless a <RETURN> is entered somewhere within the line.

Writing Programs

When MSX-BASIC is used and the "Ok" prompt is issued, the system is in the direct mode and is ready to receive a RUN command to execute the program or Editor commands. Except for commands to execute programs, lines that are entered are processed by the Full Screen Editor. All lines of text beginning with numbers are considered as program statements. The Editor processes the program statements in one of the following ways:

1. A new line is added to the program if the line number is valid (between 0 and 65529) and at least one non-blank character follows the line number.

2. An existing program line is modified if the line number already exists in the program and at least one non-blank character follows the line number. The new line replaces the text of the previously existing line.

3. An existing program line is deleted if the line number already exists in the program and the new line contains only a line number.

4. An error is generated.

   Am attempt to delete a non-existent line will result in an "Undefined line number" error.

   If the new line causes the program memory to be entirely filled, no line is added and "Out of memory" is displayed.

More than one statement may be placed after a line number, except the statements must be separated by colons (:). (These colons do not require spaces.) A logical program line may have a maximum of 255 characters, including the line number.

Editing Programs

The LIST command displays all or a part of the program currently residing in memory on the screen so that they can be edited with the Full Screen Editor. To modify the program, move the cursor to the location requiring change and do one of the following:

1. Type over existing characters
2. Delete characters to the right of the cursor
3. Delete characters to the left of the cursor
4. Insert characters
5. Append characters to the end of the logical line

These actions are performed by special keys assigned to the Full Screen Editor (see the next section).

Program lines are changed if a carriage return is entered while the cursor is located somewhere on the line. This action changes all editing done to the logical line, regardless of the number of physical lines the program line encompasses. The cursor can be located anywhere in the program line.

Full Screen Editor Functions

The following table lists the hexadecimal codes for the MSX-BASIC control characters and summarizes their functions. The Control-key sequence normally assigned to each function is also listed. These conform as closely as possible to ASCII standards.

A discussion of the individual control follows the table.

Table 1. MSX-BASIC Control Functions. Control characters are entered by holding down CTRL and pressing the character key.

Hex Code	Control Key	Special Key	Function
01	A		Ignored
02 *	B		Move cursor to start of previous word
03 *	C		Break if MSX-BASIC is waiting for input
04 *	D		Ignored
05 *	E		Erase text to end of logical line
06 *	F		Move cursor to start of next word
07 *	G		Beep
08	H	Back Space	Backspace, deleting characters passed over
09	I	Tab	Tab to next TAB stop
0A *	J		Line feed
0B *	K	Home	Move cursor to home position
0C *	L	CLS	Clear screen
0D *	M	Return	Carriage return (enter current logical line)
0E *	N		Append at end of line
0F *	O		Ignored
10 *	P		Ignored
11 *	Q		Ignored
12 *	R	INS	Toggle between insert and typeover modes

13 *	S		Ignored
14 *	T		Ignored
15 *	U		Erase logical line
16 *	V		Ignored
17 *	W		Ignored
18 *	X	Select	Ignored
19 *	Y		Ignored
1A *	Z		Ignored
1B	[	ESC	Ignored
1C *	¥	Right arrow	Move cursor right(back slash in int. ver.)
1D *	]	Left arrow	Move cursor left
1E *	^	Up arrow	Move cursor up
1F *	_	Down arrow	Move cursor down
7F	DEL	DEL	Delete character at cursor

Note: The keys marked with asterisks (*) cancel the insert mode if the Full Screen Editor is in insert mode.

PREVIOUS WORD

The cursor is moved left to the first character of the previous word. A word is defined as a character string composed of A-Z, a-z, or 0-9.

BREAK

Returns the control to MSX-BASIC direct mode without changing the line that was being edited.

ERASE TO END OF LINE

The cursor is moved to the end of the logical line, and the characters passed over are deleted. Additional characters at the new cursor position are appended to the line.

NEXT WORD

The cursor is moved right to the first character of the next word. A word is defined as a character string composed of A-Z, a-z, or 0-9.

BEEP

Produces the beep tone.

BACKSPACE

Deletes the character to the left of the cursor. All characters to the right of the cursor are moved to the left one position. Any subsequent characters and lines within the current logical line are moved up (wrapped).

TAB

TAB moves the cursor to the next tab stop, overwriting all spaces. Tab stops occur every 8 characters.

CURSOR HOME

Moves the cursor to the upper left corner of the screen. The screen is not erased.

CLEAR SCREEN

Moves the cursor to home position and clears the entire screen, regardless of where the cursor is positioned when the key is entered.

CARRIAGE RETURN

A carriage return ends the logical line and saves it as part of the MSX-BASIC program.

APPEND

Moves the cursor to the end of the line, without deleting the characters passed over. All characters typed at the new position are appended to the logical line until a carriage return is encountered.

INSERT

Toggle switch for insert mode. When insert mode is on, the cursor size is reduced and characters are inserted at the current cursor position. Characters to the right of the cursor move right as new characters are typed. Line wrap is done on characters going beyond the physical line. If the insert mode is off, the size of cursor returned to normal, and the typed characters replace any existing characters on the line.

CLEAR LOGICAL LINE

Erases entire logical line when this key is entered anywhere in the line.

CURSOR RIGHT

Moves the cursor one position to the right. Line wrap is done on characters going beyond the physical line.

CURSOR LEFT

Move the cursor one position to the left. Line wrap is done on characters going beyond the physical line.

CURSOR UP

Moves the cursor up one physical line at the current position.

CURSOR DOWN

Moves the cursor down one physical line at the current position.

o Logical line Definition with INPUT

A logical line ordinarily consists of all the characters on all of its physical lines. During the execution of an INPUT or LINE INPUT statement, however, this definition is modified slightly to allow for formatted input. When either statement is executed, the logical line is restricted to characters typed or passed over by the cursor. The insert mode and the delete function only move characters within the logical line, and DELETE decrements the size of the line.

The insert mode increments the logical line, except when the characters moved will write over non-blank characters that are on the same physical line but not part of the logical line. If this occurs, the non-blank characters that are not part of the logical line are preserved, and the characters at the end of the logical line are erased. This is to preserve labels existing prior to the INPUT statement. If an incorrect character is entered as the line is being typed, it can be deleted using the <Back Space> key or with a Control-H. Once the undesired character(s) have been deleted, simply continue typing the line.

To delete the current line being typed, type Control-U.

To correct program lines of the program currently in memory, simply type a new line using the same line number. MSX-BASIC will automatically replace the old line with the new line.

To delete the entire program currently in memory, enter a NEW command. Usually the NEW command is only used to clear the memory before entering a new program.

2.1.9 Special keys

MSX-BASIC supports several special keys (function keys and the STOP key) as follows.

o Function Keys

MSX-BASIC has ten predefined function keys. The current settings of these keys are displayed on the last line on the screen and can be redefined within a program with the KEY statement. The initial settings for the keys are as follows:

F1	color[b]	
F2	auto[b]	Meanings of abbreviations:
F4	goto[b]	
F5	list[b]	[b] = blank character
F5	run[cr]	[cr]= carriage return
F6	color 15,4,7[cr]	[u] = cursor up character
F7	cload"	[cls]=clear screen character
F8	cont[cr]	
F9	list.[cr][u][u]	(F6 color 15,4,4[cr]
F10	[cls]run[cr]	in international versions)

The function keys can also be used as event trap keys. Refer to the ON KEY GOSUB and KEY ON/OFF/STOP statements for details.

o STOP key

When MSX-BASIC is in the direct mode, the STOP key has no effect on the current operation, and MSX-BASIC simply ignores its input.

If MSX-BASIC is executing a program and the STOP key is pressed, program execution is suspended and the cursor is displayed to indicate that execution was suspended. If the STOP key is pressed again, execution is resumed. If the CTRL key is held down and the STOP key is pressed, MSX-BASIC stops executing the program and returns to the direct mode with the following message.

```
Break in nnnn
```

The nnnn is the line number of the program that was being executed when the execution was aborted.

2.1.10 ERROR MESSAGES

If an error is encountered during program execution, execution terminates, and the appropriate error message is displayed. Refer to 2.1.17 for a complete list of MSX-BASIC error codes and error messages.

2.1.11 Commands and Statements except those doing I/O

AUTO [<line number>[,<increment>]]
Automatically generates line numbers after each carriage return.

AUTO begins numbering at <line number> and increments each subsequent line number by <increment>. The default for both values is 10. If <line number> is followed by a comma and <increment> is not specified, the last increment specified in an AUTO command is assumed.

If AUTO generates an existing line number, an asterisk is printed after the line number as a warning that the existing line will be replaced. If a carriage return is instead immediately entered, the existing line is preserved and the next line number is generated.

The AUTO command is terminated by typing Control-C or Control-STOP, and MSX-BASIC returns to the direct mode. The line being input when Control-C is typed is not saved.

CONT
Continues program execution after a BREAK or STOP.

DELETE [<line number>][-<line number>]
Deletes program lines.

BASIC always returns to the direct mode after a DELETE is entered. If the <line number> does not exist, an 'Illegal function call' error occurs.

LIST [<line number>[-[<line number>]]]
Lists all or a part of the program.

If both <line number> parameters are omitted, the program is listed beginning at the lowest line number.

If only the first <line number> parameter is specified, only that line is listed.

If the first <line number> parameter and a "-" are specified, that line and all lines following it are listed.

If "-" and the second <line number> parameter are specified, all lines beginning at the lowest line number are listed until the specified number is reached.

If both <line number> parameters are specified, the lines in the range from the first <line number> through the second <line number> are listed.

The displayed listing can be terminated by holding down "CTRL" and pressing the "STOP" key. The listing can be temporarily suspended by pressing the "STOP" key, and resumed by pressing the "STOP" key again.

LLIST [<line number>[-[<line number>]]]
Lists all or part of the program on the printer, with the use of the parameters being identical for the LIST command.

NEW
Deletes the current program in memory and resets all variables.

RENUM [[<new number>][,[<old number>][,<increment>]]]
Renumbers program lines.

The <new number> parameter is the first line that will be used in the renumbered program, with the default being 10. The <old number> is the line of the current program where renumbering is to begin, with the default being the first line of the program. The <increment> is the increment used in renumbering, and the default is 10.

RENUM also changes all line number references following GOTO, GOSUB, THEN, ELSE, ON..GOTO, ON..GOSUB and ERL statements to reflect the new line numbers. If a nonexistent line number appears after one of the above statements, an 'Undefined line nnnn in mmmm' is displayed. The reference to the incorrect line number (nnnn) is not changed by RENUM, but line number mmmm may be changed.

NOTE: RENUM can neither be used to change the order of program lines (for example, entering RENUM 15,30 for a program having the three lines numbered 10, 20 and 30), nor can it be used to generate line numbers greater than 65529. In either case, an 'Illegal function call' error results.

RUN [<line number>]
Executes the current program.

Execution begins at the first line of the program unless the <line number> parameter is specified, in which case, execution begins at that line.

TRON/TROFF
Traces the execution of program statements.

The TRON statement can be executed in either the direct or indirect mode to print the line number being executed when the program is RUN. The line numbers are displayed within square brackets. The TRON function continues until a TROFF statement or a NEW command is executed.

CLEAR [<string space>[,<highest address>]]
Sets all numeric variables to zero, all string variables to null, and closes all open files; and optionally sets the end of memory.

The <string space> parameter sets the memory size allocated for string variables, with the default being 200 bytes. The <highest address> parameter sets the highest memory address to be used by MSX-BASIC.

DATA <list of constants>
Used to set the constants to be used by the program's READ statements.

DATA statements are not executable and they may be placed anywhere in a program. If a DATA statement is used to define more than one constant, the constants must be delimited by commas. The maximum number of constants that may be placed on a logical line is limited only by the size of the logical line. READ statements replace the constants for the variables used by the program in the sequence listed in the DATA statement(s).

The <list of constants> may contain numeric constants in any format: fixed point, floating point, or integer. Numeric expressions are not allowed in DATA statements. String constants may also be used in DATA statements. If the string contains commas, colons, or significant leading or trailing spaces, the string must be embedded in quotation marks.

The variable type required (numeric or string) required by a READ statement must match the type specified in its DATA statement. The RESTORE statement may be used to set the data to be read from a specific line. If the RESTORE statement is not used, the data is read from the program's first DATA statement.

DIM <list of subscripted variables>
Specifies the maximum size of array variables.

If no DIM statement is specified, the maximum size allocated in memory for the array is 10. If a subscript greater than the maximum size is used, a 'Subscript out of range' error occurs. The subscripts always begin at 0.

DEFINT <range(s) of letters>
DEFSNG <range(s) of letters>
DEFDBL <range(s) of letters>
DEFSTR <range(s) of letters>
Declares the variable type to be integer, single-precision, double-precision, or string.

The DEFINT/SNG/DBL/STR statements declare that variable names beginning with the letter(s) specified will always be that type of variable. An exception to this rule is when a variable type declaration character is used for a variable. Section 2.1.5 lists the variable declaration characters.

DEF FN<name>[(<parameter list>)]=<function definition>
Defines and names a user-programmed function.

The <name> must be a legal variable name and is preceded by FN. The <name> becomes the name of the defined function. The <parameter list> comprises the variable names in the function definition that are replaced when the function is called, and they must be separated by commas. The <function definition> is an expression performing the function, and is limited to to one line. Variable names appearing in the expression serve only to define the function; they do not affect program variables having the same name. The <parameter list> may have a variable name used, and if so, the value of the variable is supplied when the function is called, otherwise, the current value of the variable is used.

The variables in the parameter list represent, on a one-to-one basis, the argument variables or values that will be given in the function call.

If the function specifies a variable type, the expression's value takes on that type befor being returned to the calling statement. If the types specified in the function name and its argument do not match, a 'Type mismatch' error occurs.

The DEFFN statement must be executed before the defined function is used, if not, an 'Undefined user function' error occurs. Note that DEFFN cannot be used in the direct mode.

DEFUSR[<digit>]=<integer expression>
Specifies the entry point of a machine language subroutine.

The <digit> may be any digit from 0 to 9, and corresponds to the number of the USR routine whose address is being specified. If <digit> is omitted, DEFUSR0 is assumed. The value of <integer expression> is the entry point of the USR routine.

DEFUSR statements may be reused as many times as necessary within a program to redefine the entry points of subroutines.

ERASE <list of array variables>
Eliminates arrays from a program.

Arrays may be reDIMensioned after they are ERASEd, or the previously allocated array space in memory may be used for other purposes. If an attempt is made to reDIMension an array without a prior ERASE, a 'Redimensioned array' error occurs.

END
Terminates program execution, closes all files and returns to direct mode.

An END statement may be placed anywhere in a program to end its execution. Unlike STOP, the END statement does not cause a BREAK message to be displayed. An END statement located at the end of a program is optional.

ERROR <integer expression>
Simulates the occurrence of an error or allows error codes to be defined by the user.

The value of <integer expression> must be greater than 0 and less than 255. If the value of <integer expression> equals an error code already in use by BASIC, the ERROR statement will simulate the occurrence of that error, and the corresponding error message will be printed.

To define an error code, use a value that is greater than that used by BASIC. Section 2.1.17 lists the error codes and messages. Use the highest available codes to maintain compatibility in case more error codes are added to later versions of BASIC. The new user-defined error code may then be handled in an error trap routine. One such example follows.

```
10 ON ERROR GOTO 1000
.
.
120 IF A$="Y" THEN ERROR 250
.
.
1000 IF ERR=250 THEN PRINT "Sure?"
.
.
```

If an ERROR statement specifyinge a code for which no error message is defined or an ERROR statement having no error trap routine is executed, MSX-BASIC will respond with an 'Unprintable error', and execution will be terminated.

FOR <variable>=x TO y [STEP z]
.
.
NEXT [<variable>][,<variable>...]

Allows a series of instructions to be performed in a loop a given number of times.

The <variable> is used as a counter for the FOR...NEXT loop. It may be integer, single-precision, or double-precision, where x, y, and z are numeric expressions. The first numeric expression (x) is the initial value of the counter. The second numeric expression (y) is the final value of the counter. The program lines following the FOR statement are executed until the NEXT statement is encountered. Then the counter is incremented by the value of STEP. The value of the counter is then compared with the final value (y), and if it is not greater, execution is branched back to the statement immediately following the FOR statement and the statements within the loop are repeated. If the counter is exceeded, execution continues with the statement following the NEXT statement. If STEP is not specified, the default is one.

If STEP is negative, the final value of the counter must be less than the initial value. The counter is decremented each time through the loop, and the loop is executed until the counter is less than the final value.

The loop is executed at least once if the initial value of the loop times the sign of the step exceeds the final value times the sign of the step.

FOR...NEXT loops may be nested, that is, a FOR...NEXT loop may be placed within another FOR...NEXT loop. When loops are nested, each loop must have a different variable name for its counter. The NEXT statement for the inside loop must appear before the NEXT for the outside loop. If nested loops share the same end point, a single NEXT statement may be used for all of them. The depth of nesting of FOR...NEXT loops is limited only by the available memory.

The variable(s) in the NEXT statement may be omitted, in which case the NEXT statement will match the most recent FOR statement. If a NEXT statement is encountered before its corresponding FOR statement, a 'NEXT without FOR' error message is issued and execution is terminated.

GOSUB <line number>
.
.
RETURN [<line number>]

Branches to the subroutine beginning at <line number> and returns from a subroutine.

The <line number> is the first line of the subroutine. A subroutine may be called any number of times in a program, and

a subroutine may be called from within another subroutine. Nesting of subroutines is limited only by the available memory.

RETURN statements in subroutines cause BASIC to branch back to the statement following the most recent GOSUB statement. A subroutine may contain more than one RETURN statement if it is required by the program logic. Subroutines may be placed anywhere in the program, but should be readily distinguishable from the main program for greater understandability. To prevent accidental entry into a subroutine, it may be preceded by a STOP, END, or GOTO statement that directs program control around the subroutine. Otherwise, a 'RETURN without GOSUB' error will occur and execution terminates.

GOTO <line number>

Branches unconditionally out of the normal program sequence to a specified <line number>.

If <line number> is an executable statement, that statement and those following are executed. If it is a nonexecutable statement, execution proceeds at the first executable statement encountered after <line number>.

IF <expression> THEN <statement(s)|<line number>
[ELSE <statement(s)|<line number>]
IF <expression> GOTO <line number>
[ELSE <statement(s)|<line number>]

Changes the program flow based on the result returned by an expression.

If the result of <expression> is not zero, the THEN or GOTO clause is executed. THEN may be followed by either a line number for branching or one or more statements to be executed. GOTO is always followed by a line number. If the result of <expression> is zero, the THEN or GOTO clause is ignored and the ELSE clause, if present, is executed. Execution continues with the next executable statement.

Example:

```
A=1:B=2  -> A=B is zero     (FALSE).
A=2:b=2  -> A=B is not zero (TRUE).
```

IF...THEN...ELSE statements may be nested. Nesting is limited only by the length of the line. If the statement does not contain the same number of ELSE and THEN clauses, each ELSE is matched with the closest unmatched THEN. For example, the following statement will not print "A<>C" when A<>B.

```
IF A=B THEN IF B=C THEN PRINT "A=C"
        ELSE PRINT "A<>C"
```

The statement will print "A<>C" when A=B and B<>C.

If an IF...THEN statement is followed by a line number in the

direct mode, an 'Undefined line' error results unless a statement with the specified line number had previously been entered in the indirect mode.

INPUT ["<prompt string>";]<list of variables>
Allows input from the keyboard during program execution.

When an INPUT statement is encountered, program execution pauses and a question mark is printed to indicate that the program is waiting for data. If a "<prompt string>" is included, the string is printed before the question mark. The required data is entered by the keyboard.

The data that is entered is assigned to the variable(s) given in <variable list>. The number of data items supplied must be the same as the number of variables in the list. The data must be separated by commas.

The variables named in the <list of variables> may be numeric or string variables (including subscripted variables). The entered data type must agree with the type specified by the variable name. Strings entered in response INPUT statements do not need to be embedded in quotation marks.

If the wrong variable type is input (a string variable instead of a numeric variable, etc.), a "?Redo from start" message is displayed. No value is assigned until an acceptable response is given. An example of this follows.

```
list
10 INPUT "A and B";A,B
20 PRINT A+B
Ok
run
A and B? 10,00
?Redo from start
A and B? 10,20
 30
Ok
```

If the response to the INPUT statement has too many items, an "?Extra ignored" message is displayed, and the next statement is executed. One such example follows.

```
list
10 INPUT "A and B";A,B
20 PRINT A+B
Ok
run
A and B? 10,20,30
?Extra ignored
 30
Ok
```

Responding to an INPUT statement with too few items causes two question marks to be printed and a wait for the next data item.

```
Example:
list
10 INPUT "A and B";A,B
20 PRINT A+B
Ok
run
A and B? 10 (The 10 was typed in by the user)
?? 20       (The 20 was typed in by the user)
 30
Ok
```

The program can be suspended at the INPUT statement by typing Control-C or by holding down the "CTRL" key and pressing "STOP". MSX-BASIC will return to the direct mode and respond with "Ok". To resume execution, type CONT.

LINE INPUT ["<prompt string>";]<string variable>

Inputs an entire line (up to 254 characters) to a string variable, without the use of delimiters.

The <prompt string> is displayed on the console before input is accepted. No question mark is printed unless it is a part of the <prompt string>. All input typed to the console before a carriage return is assigned to <string variable>.

The program can be suspended at the LINE INPUT statement by typing Control-C or by holding down the "CTRL" key and pressing "STOP". MSX-BASIC will return to the direct mode and respond with "Ok". To resume execution, type CONT.

[LET] <variable>=<expression>

Assigns the value of an expression to a variable.

Note that the word LET is optional.

LPRINT [<list of expressions>]
LPRINT USING <string expression>;<list of expressions>

Prints data on the line printer. (Refer to the PRINT and PRINT USING statements below for details.)

MID$(<string exp. 1>),n[,m])=<string exp.2 >

Replaces a portion of one string with another string.

The characters in <string exp.1>, beginning at position n, are replaced by the characters in <string exp.2>. The optional m refers to the number of characters from <string exp.2> that will be used in the replacement. If m is omitted or included, the characters replaced does not go beyond the original length of <string exp.1>.

ON ERROR GOTO <line number>

Enables error trapping and specifies the first line of the

error handling subroutine.

Once error trapping has been enabled, all errors detected, including direct mode errors (e.g., Syntax errors), will cause a jump to the specified error handling subroutine. If <line number> does not exist, an 'Undefined line number' error occurs. To disable error trapping, execute an ON ERROR GOTO 0. Subsequent errors will then display error messages and halt execution. An ON ERROR GOTO 0 statement appearing in an error trapping subroutine will cause BASIC to stop and display the error message for the error that caused the trap. It is recommended that all error trapping subroutines execute an ON ERROR GOTO 0 if an error is encountered for which there is no recovery action.

If an error occurs during execution of an error handling subroutine, the BASIC error message is printed and execution terminates. Error trapping does not occur within the error handling subroutine.

ON <expression> GOTO <list of line numbers>
ON <expression> GOSUB <list of line numbers>
Branches to one of several specified line numbers, depending on the value returned when an expression is evaluated. The value of <expression> determines which line number in the list will be used for branching. For example, if the value is three, the third line number in the list will be the destination of the branch. If the value is not an integer, the fractional portion is discarded.)

In the ON...GOSUB statement, each line number in the list must be the first line number of a subroutine.

If the value of <expression> is either zero or is greater than the number of items in the list (and <= 255), MSX-BASIC continues with the next executable statement. If the value of <expression> is either negative or is greater than 255, an 'Illegal function call' error occurs.

POKE <memory address>,<integer expression>
Writes a (decimal) byte to a (decimal) memory location.

The <memory address> is the address of the memory location to be written to (POKEd). The <integer expression> is the data (byte) to be POKEd. It must be in the range 0 to 255. The <memory address> must be in the range -32768 to 65535. If this value is negative, the address is computed by subtraction from 65536. For example, a -1 is the same as 65535 (65536-1=65535). Otherwise, an 'Overflow' error occurs.

PRINT [<list of expressions>]
Displays data to the console.

If the <list of expressions> is omitted, a blank line is printed. If the <list of expressions> is included, the values

of the expressions are displayed on the console. An expression in the list may be a numeric and/or a string expression. Strings must be enclosed in quotation marks.

The position of each displayed item is determined by the punctuation used to separate the items in the list. MSX-BASIC divides the line into print zones of 14 spaces each. In the <list of expressions>, a comma causes the next value to be displayed at the beginning of the next zone. A semicolon causes the next value to be displayed immediately after the last value. One or more spaces between the expressions are treated as semicolons.

If a comma or a semicolon terminates the <list of expressions>, the next PRINT statement begins printing on the same line, spacing accordingly. If the <list of expressions> terminates without a comma or a semicolon, a carriage return is printed at the end of the line. If the printed line is longer than the console width, MSX-BASIC goes to the next physical line and continues printing.

A displayed number is always followed by a space. Positive numbers are preceded by a space. Negative numbers are preceded by a minus sign.

A question mark may be used instead of the word PRINT.

PRINT USING <string expression>;<list of expressions>
Displays strings or numerics using a specified format.

The <list of expressions> is comprised of the string expressions or numeric expressions that are to be printed, separated by semicolons. The <string expression> is a string literal (or variable) comprising special formatting characters. These formatting characters (see below) determine the field and the format of the printed strings or numbers.

When PRINT USING is used to print strings, one of the following three formatting characters may be used to format the string field:

"!"

Specifies that only the first character in the given string is to be printed.

```
Example:
A$="Japan"
Ok
PRINT USING "!";A$
J
Ok
```

"& n spaces &" (Japanese. Refer to 5.4 for other versions.)

Specifies that 2+n characters from the string are to be printed. If the '&' signs are typed with no spaces, two characters will be printed; with one space three characters will be printed, and so on. If the string is longer than the field, the extra characters are ignored. If the field is longer than the string, the string will be left-justified in the field and padded with spaces on the right.

```
Example:
A$="Japan"
Ok
PRINT USING "&  &";A$
Japa
Ok
```

"@" (Japanese. Refer to 5.4 for other versions.)

Specifies that the whole character in the given string is to be printed.

```
Example:
A$="Japan"
Ok
PRINT USING "I love @ very much.";A$
I love Japan very much.
Ok
```

When PRINT USING is used to print numbers,the following special characters may be used to format the numeric field:

"#"

A number sign is used to represent each digit position. The digit positions are always filled. If the number to be printed has fewer digits than positions specified, the number will be right-justified (preceded by spaces) in the field.

A decimal point may be inserted at any position in the field. If the format string specifies that a digit is to precede the decimal point, the digit will always be printed (as 0 if necessary). Numbers are rounded as necessary.

```
Example:
PRINT USING "###.##";10.2,2,3.456,.24
 10.20  2.00  3.46  0.24
Ok
```

"+"

A plus sign at the beginning or end of the format string will cause the sign of the number (plus or minus) to be printed before or after the number.

Example:

```
PRINT USING "+###.##";1.25,-1.25
  +1.25  -1.25
Ok
PRINT USING "###.##+";1.25,-1.25
  1.25+  1.25-
Ok
```

"_"

A minus sign at the end of the format field will cause negative numbers to be printed with a trailing minus sign.

Example:

```
PRINT USING "###.##-";1.25,-1.25
  1.25   1.25-
Ok
```

"**"

A double asterisk at the beginning of the format string causes leading spaces in the numeric field to be filled with asterisks. The ** also specifies positions for two or more digits.

Example:

```
PRINT USING "**#.##";1.25,-1.25
**1.25*-1.25
Ok
```

"¥¥" (Japanese. Refer to 5.4 for other versions.)

A double yen sign causes a yen sign to be printed to the immediate left of the formatted number. The ¥¥ specifies two more digit positions, one of which is the yen sign. The exponential format cannot be used with ¥¥. Negative numbers cannot be used unless the minus sign trails to the right.

Example:

```
PRINT USING "¥¥###.##";12.35,-12.35
  ¥12.35 -¥12.35
Ok
PRINT USING "¥¥###.##-";12.35,-12.35
  ¥12.35   ¥12.35-
Ok
```

"**¥" (Japanase. Refer to 5.4 for other versions.)

The **¥ at the beginning of a format string combines the effects of the above two symbols. Leading spaces will be filled with asterisks and a yen sign will be printed before the number. **¥ specifies three more digit positions, one of which is the yen sign.

Example:

```
PRINT USING "**¥#.##";12.35
*¥12.35
```

```
Ok
```

","

A comma that is to the left of the decimal point in a formatting string causes a comma to be printed to the left of every third digit to the left of the decimal point. A comma that is at the end of the format string is printed as part of the string. A comma specifies another digit position. The comma has no effect if used with the exponential format.

```
Example:
PRINT USING "####,.##";1234.5
1,234.50
Ok
PRINT USING "####.##,";1234.5
1234.50,
Ok
```

"^^^^"

Four carats may be placed after the digit position characters to specify exponential format. The four carats allow space for E+xx to be printed. Any decimal point position may be specified. The significant digits are left-justified, and the exponent is adjusted. Unless a leading + or trailing + or - is specified, one digit position will be used to the left of the decimal point to print a space or minus sign.

```
Example:
PRINT USING "##.##^^^^";234.56
 2.35E+02
Ok
PRINT USING "#.##^^^^-";-12.34
1.23E+01-
Ok
PRINT USING "+#.##^^^^";12.34,-12.34
+1.23E+01-1.23E+01
Ok
```

"%"

If the number to be printed is larger than the specified numeric field, a percent sign is printed in front of the number. Also, if rounding causes the number to exceed the field, a percent sign will be printed in front of the rounded number.

```
Example:
PRINT USING "##.##";123.45
%123.45
Ok
PRINT USING ".##";.999
%1.00
Ok
```

If the number of digits specified exceed 24, an 'Illegal function call' error will result.

READ <list of variables>

Reads values from a DATA statement and assigns them to variables.

A READ statement must always be used in conjunction with a DATA statement. READ statements assign variables to DATA statement values on a one-to-one basis. READ statement variables may be numeric or string, and the values read must agree with the variable types specified. If they do not agree, a 'Syntax error' will result.

A single READ statement may access one or more DATA statements (they will be accessed in order), or several READ statements may access the same DATA statement. If the number of variables in <list of variables> exceeds the number of elements in the DATA statement(s), an 'Out of DATA' error will result. If the number of variables specified is fewer than the number of elements in the DATA statement(s), subsequent READ statements will begin reading data at the first unread element. If there are no subsequent READ statements, the extra data is ignored.

To reread DATA statements from the start, use the RESTORE statement.

REM <remark>

Allows explanatory remarks to be inserted in a program.

REM statements are not executed but are output exactly as entered when the program is listed.

REM statements may be branched to (from a GOTO or GOSUB statement), and execution will continue with the first executable statement after the REM statement.

Remarks may be added to the end of a line by preceding the remark with a single quotation mark instead of :REM.

Do not use the above in a DATA statement as it would be considered legal data.

RESTORE [<line number>]

Allows DATA statements to be reread from a specified line.

After a RESTORE statement is executed, the next READ statement accesses the first item in the first DATA statement in the program. If <line number> is specified, the next READ statement accesses the first item in the specified DATA statement. If a nonexistent line number is specified, an 'Undefined Line number' error will result.

RESUME
RESUME 0
RESUME NEXT
RESUME <line number>

Continues program execution after an error recovery procedure has been performed.

Any one of the four formats shown below may be used, depending upon where execution is to resume:

RESUME or RESUME 0
Execution resumes at the statement which caused the error.

RESUME NEXT
Execution resumes at the statement immediately following the one which caused the error.

RESUME <line number>
Execution resumes at <line number>

A RESUME statement that is not in an error trap subroutine causes a 'RESUME without' error.

STOP
Terminates program execution and returns to command level.

A STOP statement may be used anywhere in a program to terminate execution. When a STOP statement is encountered, the following message is printed:

Break in nnnn (nnnn is a line number)

Unlike the END statement, the STOP statement does not close files.

Execution is resumed by issuing a CONT command.

SWAP <variable>,<variable>
Exchanges the values of two variables.

Any type of variable may be SWAPped (integer, single-precision, double-precision, string), but the two variables must be of the same type, or a 'Type mismatch' error results.

2.1.12 Functions except those doing I/O

ABS(X)
Returns the absolute value of the expression X.

ASC(X$)
Returns a numerical value that is the ASCII code of the first character of the string X$. If X$ is null, a 'Illegal function call' error is returned.

ATN(X)
Returns the arctangent of X in radians. Result is in the range -pi/2 to pi/2. The expression X may be any numeric type, but the evaluation of ATN is always performed in double precision.

BIN$(n)
Returns a string which represents the binary value of the decimal argument. The numeric expression, n, must be between -32768 and 65535. If n is negative, the two's complement is used. That is, BIN$(-n) is the same as BIN$(65536-n).

CDBL(X)
Converts X to a double precision number.

CHR$(I)
Returns a string whose one element is the ASCII code for I. CHR$ is commonly used to send a special character to the console.

CINT(X)
Converts X to an integer number by truncating the fractional portion. If X is not between -32768 and 32767, an 'Overflow' error occurs.

COS(X)
Returns the cosine of X in radians. COS(X) is calculated to double precision.

CSNG(X)
Converts X to a single precision number.

CSRLIN
Returns the vertical coordinate of the cursor.

ERL/ERR
The ERR and ERL variables are usually used in IF-THEN statements to direct program flow in the error trap routine. When an error handling subroutine is entered, the variable ERR contains the error code for error,and the variable ERL contains the line number of the line in which the error was detected.

If the statement that caused the error was a direct mode

statement, ERL will contain 65535. To test if an error occurred in a direct statement, use the following statement.

```
IF 65535=ERL THEN .....
```

Otherwise, use the following statements.

```
IF ERL=<line number> THEN ....
IF ERR=<error code> THEN....
```

Because ERL and ERR are reserved variables, neither may appear to the left of the equals sign in a LET (assignment) statement.

EXP(X)
Returns e to the power of X. X must be <=145.06286085862. If EXP overflows, the 'Overflow' error message is printed.

FIX(X)
Returns the integer part of X (fraction truncated). FIX(X) is equivalent to SGN(X)*INT(ABS(X)). The major difference between FIX and INT is that FIX does not return the next lower number for negative X.

FRE(0)
FRE("")
Arguments to FRE are dummy arguments. FRE returns the number of bytes in memory not being used by BASIC.

FRE(0) returns the number of bytes in memory which can be used for BASIC programs, text files, and machine language program files. FRE("") returns the number of bytes in memory for string space.

HEX$(X)
Returns a string which represents the hexadecimal value of the decimal argument. The numeric expression, n, must be between -32768 and 65535. If n is negative, the two's complement form is used. That is, HEX$(-n)is the same as HEX$(65536-n).

INKEY$
Returns either a one-character string containing a character read from the keyboard or a null string if no key is pressed. No characters will be echoed and all characters are passed through to the program, except for Control-STOP, which terminates the program.

INPUT$(X)
Returns a string of X characters, read from the keyboard. No character will be echoed and all characters are passed through, except for Control-STOP, which terminates the program.

INSTR([I,]X$,Y$)
Searches for the first occurrence of string Y$ in X$ and returns the position at which the match is found. the optional

offset I sets the position for starting the search. I must be in the range 0 to 255. If I>LEN(X$) or if X$ is null or if Y$ cannot be found or if X$ and Y$ are null, INSTR returns 0. If only Y$ is null, INSTR returns I or 1. X$ and Y$ may be string variables, string expressions, or string literals.

INT(X)
Returns the largest integer <=X.

LEFT$(X$,I)
Returns a string comprising the leftmost I characters of X$. I must be in the range 0 to 255. If I is greater than LEN(X$), the entire string (X$) is returned. If I=0, a null string (length zero) is returned.

LEN(X$)
Returns the number of characters in X$. Nonprinting characters and blanks are counted.

LOG(X)
Returns the natural logarithm of X, X being greater than zero.

LPOS(X)
Returns the current position of the line printer print head within the line printer buffer, not necessarily giving the physical position of the print head. X is a dummy argument.

MID$(X$,I[,J])
Returns a string of length J characters from X$ beginning with the Ith character. I and J must be in the range 1 to 255. If J is omitted or if there are fewer than J characters to the right of the Ith character, all rightmost characters beginning with the Ith character are returned. If I>LEN(X$), MID$ returns a null string.

OCT$(n)
Returns a string which represents the octal value of the decimal argument.

The numeric expression, n, must be between -32768 and 65535. If n is negative, the two's complement form is used, for example, OCT$(-n) is the same as OCT$(65536-n).

PEEK(I)
Returns the byte (decimal integer in the range 0 to 255) read from memory location I. I must be in the range -32768 to 65535. PEEK is the complementary function to the POKE statement.

POS(I)
Returns the current cursor position. The leftmost position is 0. I is a dummy argument.

RIGHT$(X$,I)
Returns the rightmost I characters of string X$. If I=LEN(X$), return X$. If I=0, a null string (length zero) is returned.

RND(X)
Returns a random number between 0 and 1. The same sequence of random number is generated each time the program is RUN. If X<0, the random generator is reseeded for any given X. X=0 repeats the last number generated. X>0 generates the next random number in the sequence.

SGN(X)
Returns 1 (for X>0), 0 (for X=0), -1 (for X<0).

SIN(X)
Returns the sine of X in radians. SIN(X) is calculated to double-precision.

SPACE$(X)
Returns the string of spaces of length X. The expression X discards the fractional portion and must be range 0 to 255.

SPC(I)
Prints I blanks on the screen. SPC may only be used with PRINT and LPRINT statements. I must be in the range 0 to 255.

SQR(X)
Returns the square root of X. X must be >=0.

STR$(X)
Returns a string representation of the value of X.

STRING$(I,J)
STRING$(I,X$)
Returns a string of length I whose characters all have ASCII code J or the first character of the string X$.

TAB(I)
Spaces to position I on the console. If the current print position is already beyond space I, TAB does nothing. Space 0 is the leftmost position, and the rightmost position is the width minus one. I must be in the range 0 to 255. TAB may only be used with PRINT and LPRINT statements.

TAN(X)
Returns the tangent of X in radians. TAN(X) is calculated to double precision. If TAN overflows, an 'Overflow' error will occur.

USR[<digit>](X)
Calls the user's assembly language subroutine with the argument X. <digit> is in the range 0 to 9 and corresponds to the digit supplied with the DEFUSR statement for that routine. If <digit> is omitted, USR0 is assumed.

VAL(X$)
Returns the numerical value of the string X$. The VAL function also strips leading blanks, tabs, and linefeeds from the

argument string. The following is an example.

```
PRINT VAL("   -7")
-7
Ok
```

VARPTR(<variable name>)
VARPTR(#<file number>)

Returns the address of the first byte of data identified with <variable name>. A value must be assigned to <variable name> prior to execution of VARPTR. Otherwise, an 'Illegal function call' error results. Any type of variable name may be used (numeric, string, array), and the address returned will be an integer in the range -32768 to 32767. If a negative address is returned, add it to 65536 to obtain the actual address.

VARPTR is usually used to obtain the address of a variable or array so it may be passed to a machine language subroutine. A function call of the form VARPTR(A(0)) is usually specified when passing an array, so that the lowest-address element of the array is returned.

All simple variables should be assigned before calling VARPTR for an array because the address of the arrays change whenever a new simple variable is assigned. If #<file number> is specified, VARPTR returns the starting address of the file control block.

--- Expanded Statements and Functions for MSX ---

2.1.13 Device Specific Statements

SCREEN [<mode>][,<sprite size>][,<key click switch>]
[,<cassette baud rate>][,<printer option>]

Assigns the screen mode, sprite size, key click, cassette baud rate, and printer option.

<mode> should be set to 0 to select 40x24 text mode, 1 to select 32x24 text mode, 2 to select high resolution mode, 3 to select multi-color (low-resolution mode).

0: 40x24, text mode
1: 32x24, text mode
2: high-resolution mode
3: multi-color mode

<sprite size> determines the size of sprite. Should be set to 0 to select 8x8 unmagnified sprites, 1 to select 8x8 magnified sprites, 2 to select 16x16 unmagnified sprites, 3 to select 16x16 magnified sprites. NOTE: If <sprite size> is specified, the contents of SPRITE$ will be cleared.

0: 8x8, unmagnified
1: 8x8, magnified
2: 16x16, unmagnified
3: 16x16, magnified

<key click switch> determines whether to enable or disable the key click. Should be set to 0 to disable it.

0: disable key click
non-zero: enable key click

Note that in text mode, all graphics statements except 'PUT SPRITE' generate an 'Illegal function call' error. Note also that the mode is forced to text mode when an 'INPUT' statement is encountered or BASIC returns to command level.

<cassette baud rate> determines the default baud rate for succeeding write operations, 1 for 1200 baud, and 2 for 2400 baud. The baud rate can also be determined using CSAVE command with baud rate option.

Note that when reading cassette, the baud rate is automatically determined, so that users do not need to know the baud rate the cassette is written. <printer option> determines if the printer in operation is 'MSX printer' (which has 'graphics symbol' and 'Hiragana' capability) or not. Should be non-0 if the printer does not have such capability. In this case,

graphics symbols are converted to spaces, and Hiragana are converted to Katakana in the Japanese version.

WIDTH <width of screen in text mode>
Sets the width of the display during text mode. Valid values are 1 to 40 in 40x24 text mode, and 1 to 32 in 32x24 text mode.

CLS
Clears the screen. Valid in all screen modes.

LOCATE [<x>][,<y>][,<cursor display switch>]
Locates character the position for PRINT. <cursor display switch> can be specified only in text mode.

0: Disable the cursor display
1: Enable the cursor display

COLOR [<foreground color>][,<background color>][,<border color>]
Defines the color, the default being 15,4,7 in the Japanese version. Refer to 5.4 for other versions. The argument can be in the range of 0 to 15. The color correspondences follow.

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

PUT SPRITE <sprite plane number>[,<coordinate specifier>]
[,<color>][,<pattern number>]
Sets up sprite attributes.

<sprite plane number> may range from 0 to 31.

<coordinates specifier> always ca come in one of two forms:

STEP (x offset, y offset) or
(absolute x, absolute y)

The first form is a point relative to the most recent point referenced. The second, more common, form is directly refers to a point without regard to the last point referenced. The follwing are some examples.

```
(10,10)       absolute form
STEP (10,0)   offset 10 in x and 0 in y
(0,0)         origin
```

Note that when BASIC scans coordinate values it will allow them to be beyond the edge of the screen, however values outside the integer range (-32768 to 32767) will cause an overflow error. And the values outside of the screen will be substituted with the nearest possible value. For example, 0 for any negative coordinate specification.

Note that (0,0) is always the upper left-hand corner. Although numbering y at the top causes the bottom left corner to be (0,191) in both high- and medium-resolution, this is standard.

The above description can be applied wherever graphic coordinates are used.

The X coordinate <x> may range from -32 to 255. The Y coordinate <y> may range from -32 to 191. If 208 (&HD0) is given to <y>, all sprite planes behind disappears until a value other than 208 is given to that plane. If 209 (&HD1) is specified to <y>, that sprite disappears from the screen. Refer to the VDP manual for further details.

When a field is omitted, the current value is used. At start up, the color defaults to the current foreground color.

<pattern number> specifies the pattern of sprite, and must be less than 256 when size of sprites if 0 or 1, and must be less than 64 when the size of sprites is 2 or 3. <pattern number> defaults to the <sprite plane number>. See also the SCREEN statement and the SPRITE$ variable.

CIRCLE <coordinate specifier>,<radius>[,<color>]
[,<start angle>][,<end angle>][,<aspect ratio>]

Draws an ellipse with a center and radius as indicated by the first of its arguments.

<coordinate specifier> specifies the coordinate of the center of the circle on the screen. For details on <coordinate specifier>, see the description of the PUT SPRITE statement.

The <color> defaults to foreground color.

The <start angle> and <end angle> parameters are radian arguments between 0 and 2*PI which allow you to specify where drawing of the ellipse will begin and end. If the start or end angle is negative, the ellipse will be connected to the center point with a line, and the angles will be treated as if they were positive. Note that this is different than adding 2*PI.

The <aspect ratio> is for horizontal and vertical ratio of the ellipse.

DRAW <string expression>

Draws figure according to the graphic macro language.

The graphic macro language commands are contained in the string expression string. The string defines an object, drawn when BASIC executes the DRAW statement. During execution, BASIC examines the value of string and interprets single-letter commands from the contents of the string. These commands are described in detail below:

The following movement commands begin movement from the last point referenced. After each command, last point referenced is the last point the command draws.

U n	Moves up
D n	Moves down
L n	Moves left
R n	Moves right
E n	Moves diagonally up and right
F n	Moves diagonally down and right
G n	Moves diagonally down and left
H n	Moves diagonally up and left

The n in each of the preceding commands indicates the distance to move. The number of points moved is n times the scaling factor set by the S command.

M x,y	Moves absolute or relative. If x has a plus sign(+) or a minus sign(-) in front of it, it is relative. Otherwise, it is absolute.

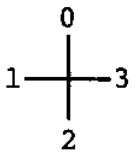
The aspect ratio of the screen is 1. Thus, 8 horizontal points are equal to 8 vertical points.

The following two prefix commands may precede any of the above movement commands.

B	Moves, but doesn't plot any points.
N	Moves, but returns to the original position when finished.

The following commands are also available:

A n	Sets angle n. n may range from 0 to 3, where 0 is 0 degrees, 1 is 90, 2 is 180, and 3 is 270.



C n	Sets color n, being between 0 and 15.

S n Sets the scale factor, n being betwen 0 and 255. The scale factor is n/4. For example, if n = 1, the scale factor is 1/4. The scale factor is multiplied by the distance given with U,D,L,R,E, F,G,and H; and relative M commands give the distance moved. The default value is 0, meaning 'no (i.e., it is the same as S4).

X<string variable>:
Executes substring. This allows you to execute a second string from within a string.

```
Example  A$="U80R80D80L80":DRAW "XA$;"
         ->Draws a square
```

In all of these commands, the n, x, or y argument can be a constant like 123 or it can be '=<variable>;' where <variable> is the name of a numeric variable. The semicolon (;) is required if the variable is used this way, or in the X command. Otherwise, a semicolon is optional between commands. Spaces are ignored in string. For example, variables in a move command in this way:

```
X1=40:X2=50
DRAW "M+=x1;,-=X2"
```

The X command can be a very useful part of DRAW, because you can define a part of an object separate from the entire object and also can use X to draw a string of commands more than 255 characters long.

LINE [<coordinate specifier>]-<coordinate specifier>[,<color>] [,<B|BF>]

Draws a line connecting the two specified coordinate. For the details on the <coordinate specifier>, see the description of the PUT SPRITE statement.

If 'B' is specified, a rectangle is drawn. If 'BF' is specified, the rectangle is filled.

PAINT <coordinate specifier>[,<paint color>][,<border color>]

Fills in a bordered figure with the specified fill color from the <coordinate specifier>. See the description on PUT SPRITE for details of the <coordinate specifier>. The PAINT statement does not allow <coordinate specifier> to be off the screen.

Note that PAINT must not have a border for high-resolution graphics, border can be specified only in multicolor mode. In high-resolution graphics mode, the paint color is regarded as the border color.

PSET<coordinate specifier>[,<color>]
PRESET<coordinate specifier>[,<color>]

Sets/resets the specified coordinate. For details of the <coordinate specifier>, see the description on PUT SPRITE.

The only difference between PSET and PRESET is that if no <color> is given in PRESET statement, the background color is selected. When a <color> argument is given, PRESET is identical to PSET.

KEY <function key #>,<string expression>
Sets a string to specified function key. <function key #> must be in the range 1 to 10. <string expression> must be within 15 characters.

Example:

```
KEY 1,"PRINT TIME$"+CHR$(13)
A$="Japan"
KEY 2,A$
```

KEY LIST
Lists the contents of all function keys.

Example:

```
KEY LIST
color
auto
goto
list
run
color 15,7,7
cload"
cont
list .
 run
Ok
```

"color" corresponds to key "f1", "auto" with "f2", "goto" with "f3", and so on. Position in the list reflects the key assignments. Note that control characters assigned to function keys are converted to spaces.

KEY ON|OFF
Turns on/off function key display on 24th line of text screen.

ON KEY GOSUB <list of line numbers>
Sets up a line numbers for BASIC to trap to when the function keys is pressed.

Example:

```
ON KEY GOSUB 100,200,,400,,500
```

When a trap occurs, an automatic KEY(n)STOP is executed so receive traps can never take place. The RETURN from the trap routine will automatically do a KEY(n)ON unless an explicit KEY(n)OFF has been performed inside the trap routine.

Event trapping does not take place when BASIC is not executing a program. When an error trap (resulting from an ON ERROR statement) takes place this automatically disables all trapping (including ERROR, STRIG, STOP, SPRITE, INTERVAL and KEY).

KEY (<function key #>) ON/OFF/STOP
Activates/deactivates trapping of the specified function key in a BASIC program.

A KEY(n)ON statement must be executed to activate trapping of function key. After a KEY(n)ON statement, if a line number is specified in the ON KEY GOSUB statement then every time BASIC starts a new statement it will check to see if the specified key was pressed. If so, it will perform a GOSUB to the line number specified in the ON KEY GOSUB statement.

If a KEY(n)OFF statement has been executed, no trapping takes place and the event is ignored.

If a KEY(n)STOP statement has been executed, no trapping will take place, but if the specified key is pressed, this is remembered so trapping is done if KEY(n)ON is executed.

KEY(n)ON does not affect the function key assignments displayed at the bottom of the console.

ON STRIG GOSUB <list of line numbers>
Sets up a line numbers for BASIC to trap to when the trigger button is pressed.

Example:
```
ON STRIG GOSUB ,200,,400
```

When the trap occurs an automatic STRIG(n)STOP is executed so receive traps can never take place. The RETURN from the trap routine will automatically do a STRIG(n)ON unless an explicit STRIG(n)OFF has been performed inside the trap routine.

Event trapping does not take place when BASIC is not executing a program. When an error trap (resulting from an ON ERROR statement) takes place, all trapping (including ERROR, STRIG, STOP, SPRITE, INTERVAL and KEY)is automatically disabled.

STRIG (<n>) ON/OFF/STOP
Activates or deactivates trapping of joystic trigger buttons in BASIC programs.

<n> can be between 0 and 4. If <n> = 0, the space bar is used for a trigger button. If <n> is either 1 or 3, the trigger of a joystick 1 is used. When <n> is either 2 or 4, joystick 2 is used.

A STRIG(n)ON statement must be executed to activate trapping

of the trigger button. After an STRIG(n)ON statement is executed, if a line number is specified in the ON STRIG GOSUB statement, then every time BASIC starts a new statement, it will check to see if the trigger button was pressed. If so, it will perform a GOSUB to the line number specified in the ON STRIG GOSUB statement.

If a STRIG(n)OFF statement has been executed, no trapping takes place and the event is not remembered even if it does take place.

If a STRIG(n)STOP statement has been executed, no trapping will take place, but if the trigger button is pressed this is remembered so an immediate trap will take place when STRIG(n)ON is executed.

ON STOP GOSUB <line number>
Sets up a line numbers for BASIC to trap to when the Control-STOP key is pressed.

When the trap occurs, the STOP STOP statement is executed so receive traps can never take place. The RETURN from the trap routine will automatically do a STOP ON unless an explicit STOP OFF has been performed inside the trap routine.

Event trapping does not take place when BASIC is not executing a program. When an error trap (resulting from an ON ERROR statement) takes place, all trapping (including ERROR, STOP, STRIG, SPRITE, INTERVAL and KEY) are automatically disabled.

Use caution when using this statement. For example, the following program cannot be aborted, and the only way out is to reset the system!

```
10 ON STOP GOSUB 40
20 STOP ON
30 GOTO 30
40 RETURN
```

STOP ON/OFF/STOP
Activates/deactivates trapping of control-STOP.

A STOP ON statement must be executed to activate trapping of a control-STOP. After STOP ON statement, if a line number is specified in the ON STOP GOSUB statement, then every time BASIC starts a new statement, it will check to see if a control-STOP was pressed. If so, it will perform a GOSUB to the line number specified in the ON STOP GOSUB statement.

If a STOP OFF statement has been executed, no trapping takes place and the event is not remembered even if it does take place.

If a STOP STOP statement has been executed, no trapping will take place. But if a Control-STOP is pressed,this is remembered,

so an immediate trap will take place when STOP ON is executed.

ON SPRITE GOSUB <line number>
Sets up a line number for BASIC to trap to when the sprites coincide.

When the trap occurs an automatic SPRITE STOP is executed so receive traps can never take place. The RETURN from the trap routine will automatically do a SPRITE ON unless an explicit SPRITE OFF has been performed inside the trap routine.

Event trapping does not take place when BASIC is not executing a program. When an error trap (resulting from an ON ERROR statement) takes place this automatically disables all trapping (including ERROR, STRIG, STOP, SPRITE, INTERVAL and KEY).

SPRITE ON/OFF/STOP
Activates/deactivates trapping of sprite in a BASIC program.

A SPRITE ON statement must be executed to activate trapping of sprite. After SPRITE ON statement, if a line number is specified in the ON SPRITE GOSUB statement then every time BASIC starts a new statement it will check to see if the sprites coincide. If so, it will perform a GOSUB to the line number specified in the ON SPRITE GOSUB statement.

If a SPRITE OFF statement has been executed, no trapping takes place and the event is not remembered even if it does take place.

If a SPRITE STOP statement has been executed, no trapping will take place. But if the sprites coincide, this is remembered so an immediate trap will take place when SPRITE ON is executed.

ON INTERVAL=<time interval> GOSUB <line number>
Sets up a line number for BASIC to trap to time interval.

Generates a timer interrupt every <time interval>/60 second.

When the trap occurs an automatic INTERVAL STOP is executed so receive traps can never take place. The RETURN from the trap routine will automatically do a INTERVAL ON unless an explicit INTERVAL OFF has been performed inside the trap routine.

Event trapping does not take place when BASIC is not executing a program. When an error trap (resulting from an ON ERROR statement) takes place this automatically disables all traps (including ERROR, STRIG, STOP, SPRITE, INTERVAL and KEY).

INTERVAL ON/OFF/STOP
Activates/deactivates trapping of time intervals.

A INTERVAL ON statement must be executed to activate trapping of time interval. After INTERVAL ON statement, if a line number is specified in the ON INTERVAL GOSUB statement, then

every time BASIC starts a new statement it will check the time interval. If so, it will perform a GOSUB to the line number specified in the ON INTERVAL GOSUB statement.

If a INTERVAL OFF statement has been executed, no trapping takes place and the event is not remembered even if it does take place.

If a INTERVAL STOP statement has been executed, no trapping will take place. But if the timer interrupt occurs, this is remembered so an immediate trap will take place when INTERVAL ON is executed.

VPOKE <address of VRAM>,<value to be written>
Pokes a value to specified location of VRAM. <address of VRAM> can be between 0 and 16383. <value to be written> should be a byte value.

BEEP
Generates a beep sound, as for the output of CHR$(7).

MOTOR [<ON|OFF>]
Changes the status of cassette motor switch. When no argument is given, flips the motor switch. Otherwise, enables/disables motor of cassette.

SOUND <register of PSG>,<value to be written>
Writes value directly to the <register of PSG>.

PLAY <string exp for voice 1>[,<string exp for voice 2>
[,<string exp for voice 3>]]
Plays music according to the music macro language.

PLAY implements a concept similar to DRAW by embedding a "music macro language" into a character string. <string exp for voice n> is a string expression consisting of single character music commands. When a null string is specified, the voice channel remains silent. The single character commands in PLAY are:

A to G with optional #,+,or -
Plays the indicated note in the current octave. A number sign(#) or plus sign(+) afterwards indicates a sharp, a minus sign(-) indicates a flat. The #,+, or - is not allowed unless it corresponds to a black key on a piano. For example, B# is an invalid note.

O n Octave. Sets the current octave for the following notes. There are 8 octaves, numbered 1 to 8. Each octave goes from C to B. Octave 4 is the default octave.

N n Plays note n. n may range from 0 to 96. n=0 means rest. This is an alternative way of

selecting notes besides specifying the octave(O n) and the note name (A-G). (The C of octave 4 is 36.)

L n — Sets the length of the following notes. The actual note length is 1/n. n may range from 1 to 64. The following table may help explain this:

Length	Equivalent
L1	Whole note
L2	Half note
L3	One of a triplet of three half notes (1/3 of a 4 beat measure)
L4	Quarter note
L5	One of a quintuplet (1/5 of a measure)
L6	One of a quarter note triplet
.	
.	
L64	Sixty-forth note

The length may also follow the note when you want to change the length only for the note. For example, A16 is equivalent to L16A. The default is 4.

R n — Pause (rest). n may range from 1 to 64, and figures the length of the pause in the same way as L(length). The default is 4.

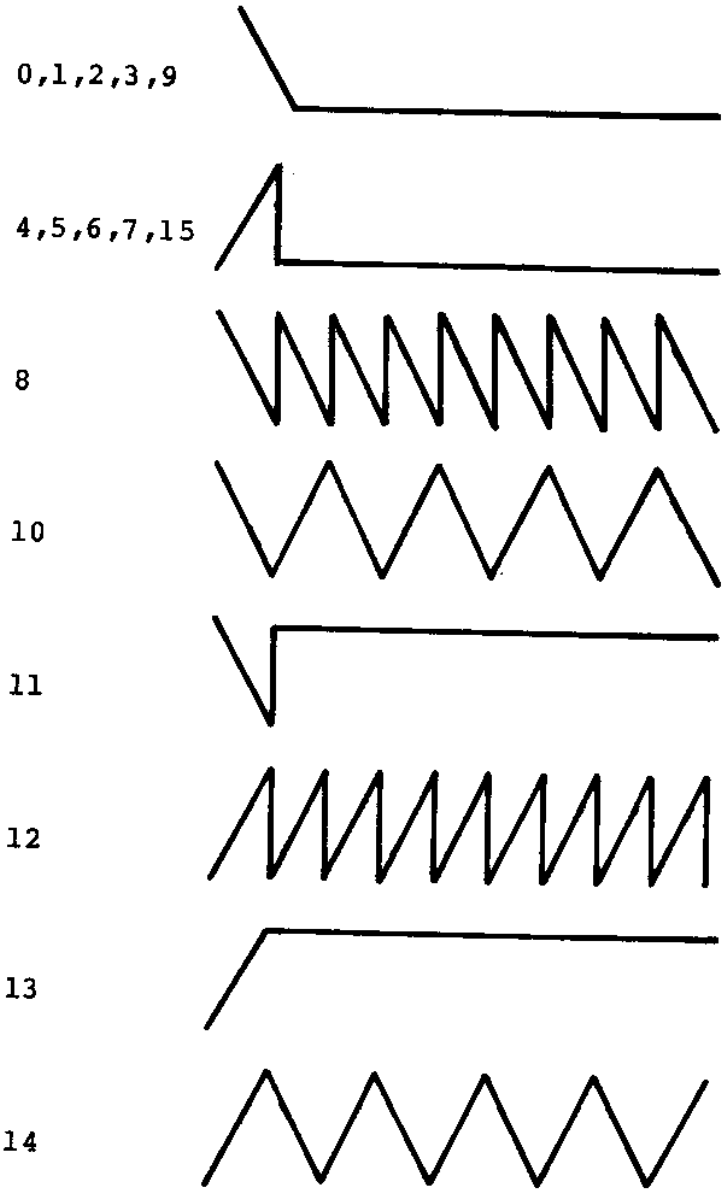
. — (Dot or period) After a note, causes the note to be played as a dotted note. That is, its length is multiplied by 3/2. More than one dot may appear after the note, 1/(2^n) is added per one dot. For example, "A..." will play 15/8 as long, etc. Dots may also appear after the pause(P) to scale the pause length in the same way.

T n — Tempo. Sets the number of quarter notes in a minute. n may range from 32 to 255. The default is 120.

V n — Volume. Sets the volume of output. n may range from 0 to 15. The default is 8.

M n — Modulation. Sets period of envelope. n may range from 1 to 65535. The default is 255.

S n — Shape. Sets shape of envelope. n may range from 1 to 15. The default is 1. The pattern set by this command are as follows:



X<variable>;
;Executes specified string.

In all of these commands, the n argument can be a constant like 12 or it can be "=<variable>;" where variable is the name of a variable. The semicolon(;) is required when you use a variable in this way, and when you use the X command. Otherwise, a semicolon is optional between commands. Note that the values specified in the above commands will be reset to the system default when a beep sound is generated.

MAXFILES=<expression>
Specifies the maximum number of files opened at a time.

<expression> can be in the range of 0 to 15. When 'MAXFILES=0' is executed, only SAVE and LOAD can be performed. The default value assigned is 1.

OPEN "<device_descriptor>[<file name>]" [FOR <mode>]
AS [#]<file number>

Allocates a buffer for I/O and set the mode that will be used with the buffer.

This statement opens a device for further processing. Currently, the following devices are supported.

CAS:	Cassette
CRT:	CRT screen
GRP:	Graphic screen
LPT:	Line printer

Device descriptors can be added using the ROM cartridge. See section 2.2.3 for further details.

<mode> is one of the following:

OUTPUT	: Sequential output mode
INPUT	: Sequential input mode
APPEND	: Sequential append mode

<file number> is an integer expression whose value is between one and the maximum number of files specified in a MAXFILES= statement.

<file number> is the number that is associated with the file for as long as it is OPEN and is used by other I/O statements to refer to the file.

An OPEN must be executed before any I/O may be done to the file using any of the following statements, or any statement or function requiring a file number:

PRINT #, PRINT # USING
INPUT #, LINE INPUT #
INPUT$, GET, PUT

PRINT #<file number>,<exp>
PRINT #<file number>,USING <string expression>;<list of expression>

Writes data to the specified channel. Refer to the PRINT and PRINT USING statements for details.

INPUT #<file number>,<variable list>

Reads data items from the specified channel and assigns them to program variables.

The type of data in the file must match the type specified by the <variable list>. Unlike the INPUT statement, no question mark is printed with INPUT# statement.

The data items in the file should appear just as they would if data were being typed in response to an INPUT statement. With numeric values, the leading spaces, carriage returns, and line feeds are ignored. The first character encountered that is not a space, carriage return, or line feed is assumed to be start of a number. The number terminates on a space, carriage return, line feed, or comma.

Also, if BASIC is scanning the data for a string item, leading spaces, carriage returns, and line feeds are ignored. The first character encountered that is not a space, carriage return, or line feed is assumed to be the start of a string item. If this first character is a double-quotation mark ("), the string item will consist of all characters read between the first quotation mark and the second. Thus, a quoted string may not contain a quotation mark as a character.

If the first character of the string is not a quotation mark, the string is an unquoted string, and will terminate on a comma, carriage return, line feed, or after 255 characters have been read. If end of file is reached when a numeric or string item is being INPUT, the item is terminated.

LINE INPUT #<file number>,<string variable>
Reads an entire line (up to 254 characters), without delimiters, from a sequential file to a string variable.

<file number> is the number which the file was OPENed.

<string variable> is the name of a string variable to which the line will be assigned.

LINE INPUT# reads all characters in the sequential file up to a carriage return. It then skips over the carriage return/line feed sequence, and the next LINE INPUT# reads all characters up to the next carriage return. (If a line feed/carriage return sequence is encountered, it is preserved. That is, the line feed/carriage return characters are returned as part of the string.)

LINE INPUT# is especially useful if each line of a file has been broken into fields, or if a BASIC program saved in ASCII mode is being read as data by another program.

INPUT$(n,[#]<file number>)
Returns a string of n characters, read from the file. <file number> is the number which the file was OPENed.

CLOSE [[#]<file number>[,<file number>]]
Closes the channel and releases the buffer associated with it. If no <file number> is specified, all open channels are closed.

SAVE "<device descriptor>[<file name>]"
Saves a BASIC program file to the device. Control-Z is treated as end-of-file.

LOAD "<device_descriptor>[<file name>]"
Loads a BASIC program file from the device.

LOAD closes all open files and deletes the current program from memory. However, with the "R" option, all data files remain OPEN and execute the loaded program.

If the <file name> is omitted, the next program, which should be an ASCII file, encountered on the tape is loaded. Control-Z is treated as end-of-file.

MERGE "<device descriptor>[<file name>]"
Merges the lines from an ASCII program file into the program currently in memory.

If any lines in the file being merged have the same line number as lines in the program in memory, the lines from the file will replace the corresponding lines in memory.

After the MERGE command, the MERGEd program resides in memory, and BASIC returns to command level.

If <file name> is omitted, the next program file, which should be in ASCII format, encountered on cassette tape is MERGEd. A Control-Z is treated as end-of-file character.

BSAVE "<device descriptor>[<file name>]",<top adrs>,<end adrs>
[,<execution adrs>]
Saves a memory image at the specified memory location to the device. (Currently, only CAS: is supported.)

<top adrs> and <end adrs> are the top address and the end address of the area to be saved.

If <execution adrs> is omitted, <top adrs> is regarded as <execution adrs>.

Examples:

```
BSAVE "CAS:TEST",&HA000,&HAFFF
BSAVE "CAS:GAME",&HE000,&HE0FF,&HE020
```

BLOAD "<device_descriptor>[<file name>]"[,R][,<offset>]
Loads a machine language program from the specified device. (Currently only CAS: is supported.)

If R option is specified, after the loading, program begins execution automatically from the address which is specified at BSAVE.

The loaded machine language program will be stored at the memory location which is specified at BSAVE. If <offset> is specified, all addresses which are specified at BSAVE are offset by that value.

If the <file name> is omitted, the next machine language program file encountered is loaded.

CSAVE "<file name>"[,<baud rate>]
Saves a BASIC program in binary format on cassette tape.

BASIC saves the file in a compressed binary (tokenized) format. ASCII files take up more space, but some types of access require that files be in ASCII format. For example, a file to be later MERGEd must be saved in ASCII format. Programs saved in ASCII may be read as BASIC data files and text files. Use the SAVE command instead for ASCII format.

<baud rate> is a parameter from 1 to 2, which determines the default baud rate for every cassette write operations. 1 for 1200 baud, 2 for 2400 baud. The default baud rate can also be set with SCREEN statement.

CLOAD ["<file name>"]
Loads a BASIC program file from the cassette tape.

CLOAD closes all open files and deletes the current program from memory. If the <file name> is omitted, the next program file encountered on the tape is loaded. For all cassette read operations, the baud rate is determined automatically.

CLOAD? ["<file name>"]
Checks if the program on cassette matches the one in memory.

CALL <name of expanded statement>[(<argument list>)]
Invokes an expanded statement supplied by ROM cartridge. See section 2.2.3 for further details. '_' is an abbreviation for 'CALL', so the next 2 statements have the same meaning.

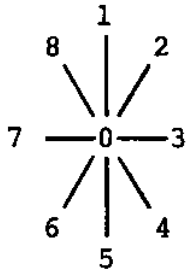
```
CALL TALK("Yamashita","Hayashi","Suzuki GSX400FW")
_TALK("Yamashita","Hayashi","Suzuki GSX400FW")
```

2.1.14 I/O Functions

POINT(<X coordinate>,<Y coordinate>)
Returns the color of a specified pixel.

VPEEK(<address of VRAM>)
Returns a value of VRAM specified. <address of VRAM> can be in the range of 0 to 16383.

STICK(<n>)
Returns the direction of a joystick. <n> can be in the range of 0 to 2. If <n>=0, the cursor key is used as a joystick. If <n> is either 1 or 2, the joystick connected to proper port is used. When neutral, 0 is returned. Otherwise, the value corresponding to direction is returned.



STRIG(<n>)
Returns the status of a trigger button of a joystick. <n> can be in the range of 0-4. If <n>=0, the space bar is used for a trigger button. If <n> is either 1 or 3, the trigger of a joystick 1 is used. When <n> is either 2 or 4, joystick 2. 0 is returned if the trigger is not being pressed, -1 is returned otherwise.

PDL(<n>)
Returns the value of a paddle. <n> can be in the range of 1 to 12. If <n> is either 1, 3, 5, 7, 9 or 11, the paddle connected to port 1 is used. When 2, 4, 6, 8, 10 or 12, the paddle connected to port 2 is used.

PAD(<n>)
Returns various status of touch pad. <n> can be in the range of 0 to 7.

When 0 to 3 is specified, the touch pad connected to joystick port 1 is selected, if between 4 to 7, port 2 is selected.

When <n>=0 or 4, the status of touch pad is returned, -1 when touched, 0 when released.

When <n>=1 or 5, the X-coordinate is returned, when <n>=2 or 6, Y-coordinate is returned.

When <n>=3 or 7, the status of switch on the pad is returned,

-1 when being pushed, 0 otherwise.

Note that coordinates are valid only when PAD(0) (or PAD(4)) is evaluated. When PAD(0) is evaluated, PAD(5) and PAD(6) are both affected, and when PAD(4), PAD(1) and PAD(2).

PLAY(<play channel>)

Returns the status of a music queue. <n> can be in the range of 0-3. If <n>=0, all 3 status are ORed and returned. If <n> is either 1,2 or 3, -1 is returned if the queue is still in operation, i.e., still playing. 0 is returned otherwise. Note that immediately after the PLAY statement is issued, the PLAY function returns -1 regardless of the actual status of the music queue.

EOF(<file number>)

Returns -1 (true) if the end of a sequential file has been reached. Otherwise, returns 0. Use EOF to test for end-of-file while INPUTing, to avoid 'Input past end' errors.

2.1.15 Special Variables

The following are special variables for MSX. When assigned, the content is changed, when evaluated, the current value is returned.

TIME (type: unsigned integer)
The system internal timer. TIME is automatically incremented by 1 everytime VDP generates interrupt (60 times per second), thus, when an interrupt is disabled (for example, when manipulating cassette), it retains the old value.

SPRITE$(<pattern number>) (type: string)
The sprite pattern.

<pattern number> must be less than 256 when size of sprites is 0 or 1, less than 64 when size of sprites is 2 or 3.

The length of this variable is fixed to 32 (bytes). So, if a string that is shorter than 32 character is assigned, the CHR$(0)s are added.

Example:

```
list
100 SCREEN 3,3
110 A$=CHR$(1)+CHR$(3)+CHR$(7)+CHR$(&HF)+CHR$(&H1F)
+CHR$(&H3F)+CHR$(&H7F)+CHR$(&HFF)
120 SPRITE$(1)=A$
130 SPRITE$(2)=A$+A$
140 SPRITE$(3)=A$+A$+A$
150 SPRITE$(4)=A$+A$+A$+A$
160 PUT SPRITE 1,(20,20),15
170 PUT SPRITE 2,(60,20),15
180 PUT SPRITE 3,(100,20),15
190 PUT SPRITE 4,(140,20),15
200 GOTO 200
Ok
run
```

```
**********************************************************************
*                          NOTE                                      *
* The following two are system variables which can be evaluated      *
* or assigned like other ordinary variables. Prepared for            *
* advanced programmers only. If you do not understand their          *
* usage fully, please do not use them.                               *
*                                                                    *
**********************************************************************
```

VDP(<n>) (type: unsigned byte)
If <n> is between 0 to 7, VDP(n) specifies the current value of the VDP write-only register. If <n> is 8, it specifies the status register of the VDP. VDP(8) is read only.

BASE(<n>) (type: integer)
Current base address for each table. The description of <n> follows next.

```
 0 - Base of name table for text mode.                          ┐
 1 - Undefined                                                  │
 2 - Base of pattern generator table for text mode.             │ 40 * 24
 3 - Undefined                                                  │
 4 - Undefined                                                  ┘

 5 - Base of name table for text mode.                          ┐
 6 - Base of color table for text mode.                         │
 7 - Base of pattern generator table for text mode.             │ 32 * 24
 8 - Base of sprite attribute table for text mode.              │
 9 - Base of sprite pattern table for text mode.                ┘

10 - Base of name table for high-resolution mode.
11 - Base of color table for high-resolution mode.
12 - Base of pattern generator table for high-resolution mode.
13 - Base of sprite attribute table for high-resolution mode.
14 - Base of sprite pattern table for high-resolution mode.

15 - Base of name table for multi-color mode.
16 - Undefined
17 - Base of pattern generator table for multi-color mode.
18 - Base of sprite attribute table for multi-color mode.
19 - Base of sprite pattern table for multi-color mode.
```

2.1.16 Machine Dependent Statements and Functions

```
**********************************************************************
*                          NOTE                                      *
* The following statements and functions access the system's         *
* I/O port directly.  Programs that use those statements and         *
* functions will thus not be compatible with MSX systems             *
* released in the future.  Programs distributed to the public        *
* should not use those statements and functions.                     *
*                                                                    *
**********************************************************************
```

OUT <port number>,<integer expression>

Sends a byte to a machine output port.

<port number> and <integer expression> are in the range 0 to 255. <integer expression> is the data byte to be transmitted.

WAIT <port number>,I[,J]

Suspends program execution while monitoring the status of a machine input port.

The WAIT statement causes execution to be suspended until a specified machine input port develops a specified bit pattern. The data read at the port is exclusive OR'ed with the integer expression J, and then is AND'ed with integer expression I. If the result is zero, BASIC loops back and reads the data at the port again. If the result is non-zero, execution continues with the next statement. If J is omitted, it is assumed as zero.

INP(<port number>I)

Returns the byte read from the port I. I must be in the range 0 to 255. INP is the complementary function to the OUT statement.

NOTE

In the above statements and functions, <port number> is handled with a 16-bit number to support the Z-80 capability to access I/O ports with the [BC] register pair, however, standard MSX systems do not support these extended I/O address spaces, and port numbers larger than 255 are undefined.

2.1.17 Summary of Error Codes and Error Messages

Code Message

1 NEXT without FOR
A variable in a NEXT statement does not correspond to any previously executed, unmatched FOR statement variable.

2 Syntax error
A line is encountered that contains some incorrect sequence of characters (such as unmatched parenthesis, misspelled command or statement, incorrect punctuation, etc.)

3 RETURN without GOSUB
A RETURN statement is encountered for which there is no previous, unmatched GOSUB statement.

4 Out of DATA
A READ statement is executed when there are no DATA statement with unread data remaining in the program.

5 Illegal function call
A parameter that is out of the range is passed to a math or string function. An FC error may also occur due to the following causes:

1. A negative or unreasonably large subscript.
2. A negative or zero argument with LOG.
3. A negative argument to SQR.
4. An improper argument to MID$, LEFT$, RIGHT$, INP, OUT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR$ or ON...GOTO.

6 Overflow
The result of a calculation is too large to be represented in BASIC's number format.

7 Out of memory
A program is too large, has too many files, has too many FOR loops or GOSUBs, too many variables, or expressions that are too complicated.

8 Undefined line number
A line reference in a GOTO, GOSUB, IF...THEN...ELSE is to a nonexistent line.

9 Subscript out of range
An array element is referenced either with a subscript that is outside the dimensions of the array, or with the wrong number of subscripts.

10 Redimensioned array
Two DIM statements are given for the same array, or DIM statement is given for an array after the default dimension of 10 has been established for that array.

11 Division by zero
A division by zero is encountered in an expression, or the operation of involution results in zero being raised to a negative power.

12 Illegal direct
A statement that is illegal in direct mode is entered as a direct mode command.

13 Type mismatch
A string variable name is assigned a numeric value or vice versa; a function that expects a numeric argument is given a string argument or vice versa.

14 Out of string space
String variables have caused BASIC to exceed the amount of free memory remaining. BASIC will allocate string space dynamically, until it runs out of memory.

15 String too long
An attempt is made to create a string more than 255 character long.

16 String formula too complex
A string expression is too long or too complex. The expression should be broken into smaller expressions.

17 Can't continue
An attempt is made to continue a program that:

1. has halted due to an error,
2. has been modified during a break in execution, or
3. does not exist.

18 Undefined user function
FN function is called before defining it with the DEF FN statement.

19 Device I/O error
An I/O error occurred on a cassette, printer, or CRT operation. It is a fatal error; i.e., BASIC cannot recover from the error.

20 Verify error
The current program is different from the program saved on the cassette.

21 No RESUME
An error trapping routine is entered but contains no RESUME statement.

22 RESUME without error
A RESUME statement is encountered before an error trapping routine is entered.

23 Unprintable error
An error message is not available for the error condition which exists. This is usually caused by an ERROR with an undefined error code.

24 Missing operand
An expression contained an operator with no operand following it.

25 Line buffer overflow
An entered line has too many characters.

26 . 49 Unprintable errors
These codes have no definitions. These are reserved for future expansion of BASIC.

50 FIELD overflow
A FIELD statement is attempting allocate more bytes than were specified for the record length of a random file in the OPEN statement. Or, the end of the FIELD buffer is encountered while doing sequential I/O(PRINT#,INPUT#) to a random file.

51 Internal error
An internal malfunction has occurred. Report to Microsoft the conditions under which the message appeared.

52 Bad file number
A statement or command references a file with a file number that is not OPEN or is out of the range of file numbers specified by MAXFILE statement.

53 File not found
A LOAD, KILL, or OPEN statement references a file that does not exist in the memory.

54 File already open
A sequential output mode OPEN is issued for a file that is already open; or a KILL is given for a file that is open.

55 Input past end
An INPUT statement is executed after all the data in the file has been INPUT, or for null (empty) file. To avoid this error, use the EOF function to detect the end of file.

56 Bad file name
An illegal form is used for the file name with LOAD, SAVE, KILL, NAME, etc.

57 Direct statement in file
A direct statement is encountered while LOADing an ASCII format file. The LOAD is terminated.

58 Sequential I/O only
A statement to random access is issued for a sequential file.

59 File not OPEN
The file specified in a PRINT#, INPUT#, etc. hasn't been OPENed.

60 Unprintable error
. These codes have no definitions. Users may
. place their own error code definitions at the
255 high end of this range.

2.1.18 MSX BASIC Reserved Words

The following is a list of reserved words used in MSX BASIC. Note that the words with asterisk (*) are reserved for future expansion only and not explained anywhere in this book.

ABS	DEFSTR	KEY	PAINT	STRING$
AND	DELETE	KILL	PDL	SWAP
ASC	DIM	LEFT$	PEEK	TAB(
*ATTR$	DRAW	LEN	PLAY	TAN
ATN	DSKF	LET	POINT	THEN
AUTO	DSKI$	LFILES	POKE	TIME
BASE	DSKO	LINE	POS	TO
BEEP	ELSE	LIST	PRESET	TROFF
BIN$	END	LLIST	PRINT	TRON
BLOAD	EOF	LOAD	PSET	USING
BSAVE	EQV	LOC	PUT	USR
CALL	ERASE	LOCATE	READ	VAL
CDBL	ERL	LOF	REM	VARPTR
CHR$	ERR	LOG	RENUM	VDP
CINT	ERROR	LPOS	RESTORE	VPEEK
CIRCLE	EXP	LPRINT	RESUME	VPOKE
CLEAR	FIELD	LSET	RETURN	WAIT
CLOAD	FILES	MAX	RIGHT$	WIDTH
CLOSE	FIX	MERGE	RND	XOR
CLS	FN	MID$	RSET	
*CMD	FOR	MKD$	RUN	
COLOR	FPOS	MKI$	SAVE	
CONT	FRE	MKS$	SCREEN	
COPY	GET	MOD	*SET	
COS	GO TO	MOTOR	SGN	
CSAVE	GOSUB	NAME	SIN	
CSNG	GOTO	NEW	SOUND	
CSRLIN	HEX$	NEXT	SPACE$	
CVD	IF	NOT	SPC(	
CVI	IMP	OCT$	SPRITE	
CVS	INKEY$	OFF	SQR	
DATA	INP	ON	STEP	
DEF	INPUT	OPEN	STICK	
DEFDBL	INSTR	OR	STOP	
DEFINT	INT	OUT	STR$	
DEFSNG	*IPL	PAD	STRIG	

2.2 Advanced Programming Guide

2.2.1 BIOS Entry List

```
   COMMENT %

  The  following Restarts (RSTs 0 through RST 5) are reserved  for
  the  BASIC interpreter,  RST 6 for inter-slot calls,  RST 7  for
  hardware interrupt.

  The following notation is used in the descriptions.

     Name             Name of function
     Function         Function to be performed
     Entry            Entry parameters
     Returns          Returned parameters
     Modifies         Registers to be modified
     Notes            (optional)

   %
   ;
   ; Name:           CHKRAM
   ; Function:       Checks RAM and sets slot for command area.
   ; Entry:          None
   ; Returns:        None
   ; Modifies:       All
   ; Note:           When done,  a jump to  INIT  must  be made for
   ;                 further initialization.
   ;
0000 DI                         ;For fail safe
     ENTR    CHKRAM
     DW      CGTABL             ;Address of character generator table
     DB      VDP.DR             ;Address of VDP data register (read)
     DB      VDP.DW             ;Address of VDP data register (write)
   ;
   ; Name:           SYNCHR
   ; Function:       Checks if  the current  character  pointed  by
   ;                 HL is  the one  desired.   If  not,  generates
   ;                 'Syntax error',  otherwise falls into  CHRGTR.
   ; Entry:          HL, character to  be checked  be placed at the
   ;                 next location to this RST.
   ; Returns:        HL points  to  next  character,   A  has   the
   ;                 character.
   ;                 Carry flag set if number,  Z flag set  if  end
   ;                 of statement.
   ; Modifies:       AF, HL
   ;
0008 ENTR    SYNCHR
     HOLE    1
   ;
   ; Name:           RDSLT
```

```
   ; Function:      Selects the appropriate slot according to the
   ;                value given through registers, and reads the
   ;                contents of memory from the slot.
   ; Entry:         A:  FxxxSSPP
   ;                    |   ||||
   ;                    |   ||└┴── Primary slot # (0-3)
   ;                    |   └┴──── Secondary slot # (0-3)
   ;                    └────────── 1 if secondary slot # specified
   ;
   ;                HL:  Address of target memory
   ; Returns:       A :  Contents of memory
   ; Modifies:      AF, BC, DE
   ; Note:          Interrupts are disabled automatically but
   ;                are never enabled by this routine.
   ;
000C ENT     RDSLT
     HOLE    1
   ;
   ; Name:          CHRGTR
   ; Function:      Gets next character (or token) from BASIC text
   ; Entry:         HL
   ; Returns:       HL points to next character, A has the
   ;                character. Carry flag set if number, Z flag
   ;                set if end of statement encountered.
   ; Modifies:      AF, HL
   ;
0010 ENTR    CHRGTR
     HOLE    1
   ;
   ; Name:          WRSLT
   ; Function:      Selects the appropriate slot according to the
   ;                value given through registers, and writes to
   ;                memory.
   ; Entry:         A:  FxxxSSPP
   ;                    |   ||||
   ;                    |   ||└┴── Primary slot # (0-3)
   ;                    |   └┴──── Secondary slot # (0-3)
   ;                    └────────── 1 if secondary slot # specified
   ;
   ;                HL:  Address of target memory
   ;                E :  Data to be written
   ; Returns:       None
   ; Modifies:      AF, BC, D
   ; Note:          Interrupts are disabled automatically but
   ;                are never enabled by this routine.
   ;
000C ENT     WRSLT
     HOLE    1
   ;
   ; Name:          OUTDO
   ; Function:      Outputs to the current device.
   ; Entry:         A, PTRFIL, PRTFLG
   ; Returns:       None
   ; Modifies:      None
   ;
```

```
0018 ENT     OUTDO
     HOLE    1
   ;
   ; Name:           CALSLT
   ; Function:       Performs inter-slot call to specified address.
   ; Entry:          IY -FxxxSSPP
   ;               (HIGH)|   ||||
   ;                     |   ||└┴── Primary slot # (0-3)
   ;                     |   └┴──── Secondary slot # (0-3)
   ;                     └──────── 1 if secondary slot # specified
   ;
   ;                   IX :  Address to call
   ; Returns:        None
   ; Modifies:       None
   ; Note:           Interrupts are disabled automatically but
   ;                 never enabled by this routine.  Arguments can
   ;                 never be passed via the alternate registers of
   ;                 the Z-80 or IX and IY.
   ;
001C ENT     CALSLT
     HOLE    1
   ;
   ; Name:           DCOMPR
   ; Function:       Compares HL with DE.
   ; Entry:          HL, DE
   ; Returns:        Flags
   ; Modifies:       AF
   ;
0020 ENTR    DCOMPR
     HOLE    1
   ;
   ; Name:           ENASLT
   ; Function:       Selects the appropriate  slot according to the
   ;                 value given through registers, and permanently
   ;                 enables the slot.
   ; Entry:          A:  FxxxSSPP
   ;                     |   ||||
   ;                     |   ||└┴── Primary slot # (0-3)
   ;                     |   └┴──── Secondary slot # (0-3)
   ;                     └──────── 1 if secondary slot # specified
   ;
   ;                   HL:  Address of target memory
   ; Returns:        None
   ; Modifies:       All
   ; Note:           Interrupts are disabled automatically but
   ;                 are never enabled by this routine.
   ;
0024 ENT     ENASLT
     HOLE    1
   ;
   ; Name:           GETYPR
   ; Function:       Returns the type of FAC.
   ; Entry:          FAC
   ; Returns:        Flags
   ; Modifies:       AF
```

```
   ;
0028 ENTR    GETYPR
   ;
   ;  The  following 5 bytes are reserved to store the MSX version
   ;  number.  The first versions hold 5 zeros.
   ;
     HOLE    5
   ;
   ; Name:           CALLF
   ; Function:       Performs far_call (i.e., inter-slot call)
   ; Entry:          None
   ; Returns:        Flags
   ; Modifies:       AF
   ; Note:           The calling sequence is as follows.
   ;
   ;                 RST     6
   ;                 DB      Destination slot
   ;                 DW      Destination address
   ;
   ;                 For a  precise description of the  parameters,
   ;                 see CALSLT.
   ;
0030 ENTR    CALLF
     HOLE    5
   ;
   ; Name:           KEYINT
   ; Function:       Performs hardware interrupt procedures.
   ; Entry:          None
   ; Returns:        None
   ; Modifies:       None
   ;
0038 ENTR    KEYINT
```

```
   COMMENT %

   The following routines are used for I/O initialization.

   %
   ;
   ; Name:           INITIO
   ; Function:       Performs device initialization.
   ; Entry:          None
   ; Returns:        None
   ; Modifies:       All
   ;
003B ENT     INITIO
   ;
   ; Name:           INIFNK
   ; Function:       Initializes function key strings.
   ; Entry:          None
   ; Returns:        None
   ; Modifies:       All
   ;
003E ENT     INIFNK
```

```
   COMMENT %

   The following routines are used to access the VDP (TI9918).

   %
   ;
   ; Name:          DISSCR
   ; Function:      Disables screen display.
   ; Entry:         None
   ; Returns:       None
   ; Modifies:      AF, BC
   ;
0041 ENT     DISSCR
   ;
   ; Name:          ENASCR
   ; Function:      Enables screen display.
   ; Entry:         None
   ; Returns:       None
   ; Modifies:      AF, BC
   ;
0044 ENT     ENASCR
   ;
   ; Name:          WRTVDP
   ; Function:      Writes to the VDP register.
   ; Entry:         Register # in [C], data in [B]
   ; Returns:       None
   ; Modifies:      AF, BC
   ;
0047 ENT     WRTVDP
   ;
   ; Name:          RDVRM
   ; Function:      Reads the VRAM addressed by [HL].
   ; Entry:         HL
   ; Returns:       A
   ; Modifies:      AF
   ;
004A ENT     RDVRM
   ;
   ; Name:          WRTVRM
   ; Function:      Writes to the VRAM addressed by [HL].
   ; Entry:         HL, A
   ; Returns:       None
   ; Modifies:      AF
   ;
004D ENT     WRTVRM
   ;
   ; Name:          SETRD
   ; Function:      Sets up the VDP for read.
   ; Entry:         HL
   ; Returns:       None
   ; Modifies:      AF
   ;
0050 ENT     SETRD
   ;
   ; Name:          SETWRT
```

```
   ; Function:       Sets up the VDP for write.
   ; Entry:          HL
   ; Returns:        None
   ; Modifies:       AF
   ;
0053 ENT     SETWRT
   ;
   ; Name:           FILVRM
   ; Function:       Fills the VRAM with the specified data.
   ; Entry:          Address in [HL], length in [BC], data in [Acc]
   ; Returns:        None
   ; Modifies:       AF, BC
   ;
0056 ENT     FILVRM
   ;
   ; Name:           LDIRMV
   ; Function:       Moves a VRAM memory block to memory.
   ; Entry:          Address of source in [HL], destination in [DE],
   ;                 length in [BC].
   ; Returns:        None
   ; Modifies:       All
   ;
0059 ENT     LDIRMV
   ;
   ; Name:           LDIRVM
   ; Function:       Moves block of memory from memory to the VRAM.
   ; Entry:          Address of source in [HL], destination in [DE],
   ;                 length in [BC].
   ; Returns:        None
   ; Modifies:       All
   ;
005C ENT     LDIRVM
   ;
   ; Name:           CHGMOD
   ; Function:       Sets the VDP mode according to SCRMOD.
   ; Entry:          SCRMOD (0..3)
   ; Returns:        None
   ; Modifies:       All
   ;
005F ENT     CHGMOD
   ;
   ; Name:           CHGCLR
   ; Function:       Changes the color of the screen.
   ; Entry:          Foreground color in FORCLR
   ;                 Background color in BAKCLR
   ;                 Border color in BDRCLR
   ; Returns:        None
   ; Modifies:       All
   ;
0062 ENT     CHGCLR
     HOLE    1
   ;
   ; Name:           NMI
   ; Function:       Performs non-maskable interrupt procedures.
   ; Entry:          None
```

```
   ; Returns:        None
   ; Modifies:       None
   ;
0066 ENT     NMI
   ;
   ; Name:           CLRSPR
   ; Function:       Initializes all sprites.
   ;                 Patterns are set to nulls, sprite names are
   ;                 set to sprite plane number, sprite colors are
   ;                 set to foreground color, vertical positions
   ;                 are set to 209.
   ; Entry:          SCRMOD
   ; Returns:        None
   ; Modifies:       All
   ;
0069 ENT     CLRSPR
   ;
   ; Name:           INITXT
   ; Function:       Initializes screen for text mode (40*24) and
   ;                 sets the VDP.
   ; Entry:          TXTNAM, TXTCGP
   ; Returns:        None
   ; Modifies:       All
   ;
006C ENT     INITXT
   ;
   ; Name:           INIT32
   ;  Function:      Initializes screen for text mode (32*24) and
   ;                 sets the VDP.
   ; Entry:          T32NAM, T32CGP, T32COL, T32ATR, T32PAT
   ; Returns:        None
   ; Modifies:       All
   ;
006F ENT     INIT32
   ;
   ; Name:           INIGRP
   ; Function:       Initializes screen for high-resolution mode
   ;                 and sets the VDP.
   ; Entry:          GRPNAM, GRPCGP, GRPCOL, GRPATR, GRPPAT
   ; Returns:        None
   ; Modifies:       All
   ;
0072 ENT     INIGRP
   ;
   ; Name:           INIMLT
   ; Function:       Initializes screen for multicolor mode and
   ;                 sets the VDP.
   ; Entry:          MLTNAM, MLTCGP, MLTCOL, MLTATR, MLTPAT
   ; Returns:        None
   ; Modifies:       All
   ;
0075 ENT     INIMLT
   ;
   ; Name:           SETTXT
   ; Function:       Sets the VDP for text (40*24) mode.
```

```
   ; Entry:          TXTNAM, TXTCGP
   ; Returns:        None
   ; Modifies:       All
   ;
0078 ENT     SETTXT
   ;
   ; Name:           SETT32
   ; Function:       Sets the VDP for text (32*24) mode.
   ; Entry:          T32NAM, T32CGP, T32COL, T32ATR, T32PAT
   ; Returns:        None
   ; Modifies:       All
   ;
007B ENT     SETT32
   ;
   ; Name:           SETGRP
   ; Function:       Sets the VDP for high-resolution mode.
   ; Entry:          GRPNAM, GRPCGP, GRPCOL, GRPATR, GRPPAT
   ; Returns:        None
   ; Modifies:       All
   ;
007E ENT     SETGRP
   ;
   ; Name:           SETMLT
   ; Function:       Sets the VDP for multicolor mode.
   ; Entry:          MLTNAM, MLTCGP, MLTCOL, MLTATR, MLTPAT
   ; Returns:        None
   ; Modifies:       All
   ;
0081 ENT     SETMLT
   ;
   ; Name:           CALPAT
   ; Function:       Returns address of sprite pattern table.
   ; Entry:          Sprite ID in [Acc]
   ; Returns:        Address in [HL]
   ; Modifies:       AF, DE, HL
   ;
0084 ENT     CALPAT
   ;
   ; Name:           CALATR
   ; Function:       Returns address of sprite attribute table.
   ; Entry:          Sprite ID in [Acc]
   ; Returns:        Address in [HL]
   ; Modifies:       AF, DE, HL
   ;
0087 ENT     CALATR
   ;
   ; Name:           GSPSIZ
   ; Function:       Returns the current sprite size.
   ; Entry:          None
   ; Returns:        Sprite size (# of bytes) in [Acc]
   ;                 Carry  set if  16*16 sprite in use,  otherwise
   ;                 reset the otherwise.
   ; Modifies:       AF
   ;
008A ENT     GSPSIZ
```

```
   ;
   ; Name:           GRPPRT
   ; Function:       Prints a character on the graphic screen.
   ; Entry:          Code to output in [Acc]
   ; Returns:        None
   ; Modifies:       None
   ;
008D ENT     GRPPRT
```

```
   COMMENT %

   The following routines are used to access the PSG.

   %
   ;
   ; Name:          GICINI
   ; Function:      Initializes PSG, and static data for PLAY
   ;                statement.
   ; Entry:         None
   ; Returns:       None
   ; Modifies:      All
   ;
0090 ENT     GICINI
   ;
   ; Name:          WRTPSG
   ; Function:      Writes data to the PSG register.
   ; Entry:         Register number in [Acc], data in [E]
   ; Returns:       None
   ; Modifies:      None
   ;
0093 ENT     WRTPSG
   ;
   ; Name:          RDPSG
   ; Function:      Reads data from the PSG register.
   ; Entry:         Register number in [Acc]
   ; Returns:       Data in [Acc]
   ; Modifies:      None
   ;
0096 ENT     RDPSG
   ;
   ; Name:          STRTMS
   ; Function:      Checks/starts background tasks for PLAY.
   ; Entry:         None
   ; Returns:       None
   ; Modifies:      All
   ;
0099 ENT     STRTMS
```

```
   COMMENT %

  The following routines are used to access the console, i.e., the
  keyboard and the CRT.

   %
   ;
   ; Name:          CHSNS
   ; Function:      Checks the status of keyboard buffer.
   ; Entry:         None
   ; Returns:       Z flag reset if any character in buffer.
   ; Modifies:      AF
   ;
009C ENT     CHSNS
   ;
   ; Name:          CHGET
   ; Function:      Waits  for  characters being input and returns
   ;                the character codes.
   ; Entry:         None
   ; Returns:       Character code in [Acc]
   ; Modifies:      AF
   ;
009F ENT     CHGET
   ;
   ; Name:          CHPUT
   ; Function:      Outputs a character to the console.
   ; Entry:         Character code to be output in [Acc]
   ; Returns:       None
   ; Modifies:      None
   ;
00A2 ENT     CHPUT
   ;
   ; Name:          LPTOUT
   ; Function:      Outputs a character to the line printer.
   ; Entry:         Character code to be output in [Acc]
   ; Returns:       Carry flag set if aborted.
   ; Modifies:      F
   ;
00A5 ENT     LPTOUT
   ;
   ; Name:          LPTSTT
   ; Function:      Checks the line printer status.
   ; Entry:         None
   ; Returns:       255 in [Acc] and Z flag reset if printer ready,
   ;                0 and Z flag set if not.
   ; Modifies:      AF
   ;
00A8 ENT     LPTSTT
   ;
   ; Name:          CNVCHR
   ; Function:      Checks graphic header byte and converts codes.
   ; Entry:         Character code in [Acc]
   ; Returns:       Cy flag reset:  graphic header byte
   ;                Cy and Z flags set, converted graphic code
   ;                Cy flag set, Z flag reset, non-converted code
```

```
   ; Modifies:      AF
   ;
00AB ENT     CNVCHR
   ;
   ; Name:          PINLIN
   ; Function:      Accepts a line from console until a CR or STOP
   ;                is typed, and stores the line in a buffer.
   ; Entry:         None
   ; Returns:       Address of buffer  top-1  in [HL],  carry flag
   ;                set if STOP is input.
   ; Modifies:      All
   ;
00AE ENT     PINLIN
   ;
   ; Name:          INLIN
   ; Function:      Same as PINLIN, except if AUTFLG is set.
   ; Entry:         None
   ; Returns:       Address of buffer top-1  in  [HL],  carry flag
   ;                set if STOP is input.
   ; Modifies:      All
   ;
00B1 ENT     INLIN
   ;
   ; Name:          QINLIN
   ; Function:      Outputs a '?' mark and a space then falls into
   ;                the INLIN routine.
   ; Entry:         None
   ; Returns:       Address of buffer  top-1  in [HL],  carry flag
   ;                set if STOP is input.
   ; Modifies:      All
   ;
00B4 ENT     QINLIN
   ;
   ; Name:          BREAKX
   ; Function:      Checks the status of the Control-STOP key.
   ; Entry:         None
   ; Returns:       Carry flag set if being pressed.
   ; Modifies:      AF
   ; Note:          This routine is  used  to  check  Control-STOP
   ;                when interrupts are disabled.
   ;
00B7 ENT     BREAKX
   ;
   ; Name:          ISCNTC
   ; Function:      Checks the status of the SHIFT-STOP key.
   ; Entry:         None
   ; Returns:       None
   ; Modifies:      None
   ;
00BA ENT     ISCNTC
   ;
   ; Name:          CKCNTC
   ; Function:      Same as ISCNTC, used by BASIC.
   ; Entry:         None
   ; Returns:       None
```

```
   ; Modifies:       None
   ;
00BD ENT     CKCNTC
   ;
   ; Name:           BEEP
   ; Function:       Sounds the buzzer.
   ; Entry:          None
   ; Returns:        None
   ; Modifies:       All
   ;
00C0 ENT     BEEP
   ;
   ; Name:           CLS
   ; Function:       Clears the screen.
   ; Entry:          None
   ; Returns:        None
   ; Modifies:       AF, BC, DE
   ;
00C3 ENT     CLS
   ;
   ; Name:           POSIT
   ; Function:       Locates the cursor at the specified position.
   ; Entry:          Column in [H], row in [L]
   ; Returns:        None
   ; Modifies:       AF
   ;
00C6 ENT     POSIT
   ;
   ; Name:           FNKSB
   ; Function:       Checks if function key display is active.   If
   ;                 it is, it displays it, otherwise does nothing.
   ; Entry:          FNKFLG
   ; Returns:        None
   ; Modifies:       All
   ;
00C9 ENT     FNKSB
   ;
   ; Name:           ERAFNK
   ; Function:       Erases the function key display.
   ; Entry:          None
   ; Returns:        None
   ; Modifies:       All
   ;
00CC ENT     ERAFNK
   ;
   ; Name:           DSPFNK
   ; Function:       Displays the function key display.
   ; Entry:          None
   ; Returns:        None
   ; Modifies:       All
   ;
00CF ENT     DSPFNK
   ;
   ; Name:           TOTEXT
   ; Function:       Forcibly places the screen in text mode.
```

```
   ; Entry:          None
   ; Returns:        None
   ; Modifies:       All
   ;
00D2 ENT      TOTEXT
```

```
   COMMENT %

   The following routines are used for game I/O access.

   %
   ;
   ; Name:          GTSTCK
   ; Function:      Returns the current joystick status.
   ; Entry:         Joystick ID in [Acc]
   ; Returns:       Direction in [Acc]
   ; Modifies:      All
   ;
00D5 ENT     GTSTCK
   ;
   ; Name:          GTTRIG
   ; Function:      Returns the current trigger button status.
   ; Entry:         Trigger button ID in [Acc]
   ; Returns:       Returns  0  in  [Acc]  if  not pressed,  255
   ;                otherwise.
   ; Modifies:      AF
   ;
00D8 ENT     GTTRIG
   ;
   ; Name:          GTPAD
   ; Function:      Checks the current touch PAD status.
   ; Entry:         ID in [Acc]
   ; Returns:       Value in [Acc]
   ; Modifies:      All
   ;
00DB ENT     GTPAD
   ;
   ; Name:          GTPDL
   ; Function:      Returns the value of the paddle.
   ; Entry:         Paddle ID in [Acc]
   ; Returns:       Value in [Acc]
   ; Modifies:      All
   ;
00DE ENT     GTPDL
```

```
   COMMENT %

   The following routines are used to access the cassette tape.

   %
   ;
   ; Name:          TAPION
   ; Function:      Sets motor on and reads header from tape.
   ; Entry:         None
   ; Returns:       Carry flag set if aborted.
   ; Modifies:      All
   ;
00E1 ENT     TAPION
   ;
   ; Name:          TAPIN
   ; Function:      Inputs from tape.
   ; Entry:         None
   ; Returns:       Data in [Acc], carry flag set if aborted.
   ; Modifies:      All
   ;
00E4 ENT     TAPIN
   ;
   ; Name:          TAPIOF
   ; Function:      Stops reading from tape.
   ; Entry:         None
   ; Returns:       None
   ; Modifies:      None
   ;
00E7 ENT     TAPIOF
   ;
   ; Name:          TAPOON
   ; Function:      Sets motor on and writes header block to
   ;                cassette.
   ; Entry:         [Acc] holds non-0 value if a long header
   ;                desired, 0 if a short header desired.
   ; Returns:       Carry flag set if aborted.
   ; Modifies:      All
   ;
00EA ENT     TAPOON
   ;
   ; Name:          TAPOUT
   ; Function:      Outputs to tape.
   ; Entry:         Data to be output in [Acc]
   ; Returns:       Carry flag set if aborted.
   ; Modifies:      All
   ;
00ED ENT     TAPOUT
   ;
   ; Name:          TAPOOF
   ; Function:      Stops writing to tape.
   ; Entry:         None
   ; Returns:       None
   ; Modifies:      None
   ;
00F0 ENT     TAPOOF
```

```
   ;
   ; Name:           STMOTR
   ; Function:       Starts the cassette motor.
   ; Entry:          0 in [Acc] to stop, 1 to start, 255 to flip.
   ; Returns:        None
   ; Modifies:       AF
   ;
00F3 ENT     STMOTR
```

```
   COMMENT %

   The following routines are used to handle queues.

   %
   ;
   ; Name:           LFTQ
   ; Function:       Returns the number of bytes left in the queue.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
00F6 ENT     LFTQ
   ;
   ; Name:           PUTQ
   ; Function:       Places a byte in the queue.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
00F9 ENT     PUTQ
```

```
   COMMENT %

  The following routines are used by the GENGRP and ADVGRP
  modules.

   %
   ;
   ; Name:          RIGHTC
   ; Function:      Moves one pixel right.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
00FC ENT     RIGHTC
   ;
   ; Name:          LEFTC
   ; Function:      Moves one pixel left.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
00FF ENT     LEFTC
   ;
   ; Name:          UPC
   ; Function:      Moves one pixel up.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
0102 ENT     UPC
   ;
   ; Name:          TUPC
   ; Function:      Moves one pixel up.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
0105 ENT     TUPC
   ;
   ; Name:          DOWNC
   ; Function:      Moves one pixel down.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
0108 ENT     DOWNC
   ;
   ; Name:          TDOWNC
   ; Function:      Moves one pixel down.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
010B ENT     TDOWNC
   ;
```

```
   ; Name:           SCALXY
   ; Function:       Scales the X-Y coordinates.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
010E ENT     SCALXY
   ;
   ; Name:           MAPXYC
   ; Function:       Maps the coordinate to the physical address.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
0111 ENT     MAPXYC
   ;
   ; Name:           FETCHC
   ; Function:       Fetches current physical address and mask
   ;                 pattern.
   ; Entry:          None
   ; Returns:        Address in [HL], mask pattern in [Acc]
   ; Modifies:       A, HL
   ;
0114 ENT     FETCHC
   ;
   ; Name:           STOREC
   ; Function:       Stores physical address and mask pattern.
   ; Entry:          Address in [HL], mask pattern in [Acc]
   ; Returns:        None
   ; Modifies:       None
   ;
0117 ENT     STOREC
   ;
   ; Name:           SETATR
   ; Function:       Sets attribute byte.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
011A ENT     SETATR
   ;
   ; Name:           READC
   ; Function:       Reads attribute of current pixel.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
011D ENT     READC
   ;
   ; Name:           SETC
   ; Function:       Sets current pixel to the specified attribute.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
```

```
0120 ENT     SETC
   ;
   ; Name:           NSETCX
   ; Function:       Sets pixels horizontally.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
0123 ENT     NSETCX
   ;
   ; Name:           GTASPC
   ; Function:       Returns the aspect ratio.
   ; Entry:          None
   ; Returns:        DE, HL
   ; Modifies:       DE, HL
   ;
0126 ENT     GTASPC
   ;
   ; Name:           PNTINI
   ; Function:       Initializes the PAINT function.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
0129 ENT     PNTINI
   ;
   ; Name:           SCANR
   ; Function:       Scans pixels to the right.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
012C ENT     SCANR
   ;
   ; Name:           SCANL
   ; Function:       Scans pixels to the left.
   ; Entry:
   ; Returns:
   ; Modifies:
   ;
012F ENT     SCANL
```

```
   COMMENT %

   The following routines are additional entries.

   %
   ;
   ; Name:          CHGCAP
   ; Function:      Changes the status of CAP lamp.
   ; Entry:         0 in [Acc] to turn off the lamp, non-0
   ;                otherwise.
   ; Returns:       None
   ; Modifies:      AF
   ;
0132 ENT     CHGCAP
   ;
   ; Name:          CHGSND
   ; Function:      Changes the status of the 1 bit sound port.
   ; Entry:         0 in [Acc] to turn off, non-0 otherwise.
   ; Returns:       None
   ; Modifies:      AF
   ;
0135 ENT     CHGSND
   ;
   ; Name:          RSLREG
   ; Function:      Reads the current output to the primary slot
   ;                register.
   ; Entry:         None
   ; Returns:       Result in [Acc]
   ; Modifies:      A
   ;
0138 ENT     RSLREG
   ;
   ; Name:          WSLREG
   ; Function:      Writes to the primary slot register.
   ; Entry:         Value in [Acc]
   ; Returns:       None
   ; Modifies:      None
   ;
013B ENT     WSLREG
   ;
   ; Name:          RDVDP
   ; Function:      Reads the VDP status register.
   ; Entry:         None
   ; Returns:       Data in [Acc]
   ; Modifies:      A
   ;
013E ENT     RDVDP
   ;
   ; Name:          SNSMAT
   ; Function:      Returns the status of a specified row of a
   ;                keyboard matrix.
   ; Entry:         Row # in [Acc]
   ; Returns:       Status in [Acc], corresponding bit is reset
   ;                to 0 if a key is being pressed.
   ; Modifies:      AF
```

```
    ;
0141 ENT     SNSMAT
    ;
    ; Name:           PHYDIO
    ; Function:       Performs operation for mass-storage devices
    ;                 such as disks.
    ; Entry:
    ; Returns:
    ; Modifies:
    ; Note:           In the minimum configuration, only a hook is
    ;                 provided.
    ;
0144 ENT     PHYDIO
    ;
    ; Name:           FORMAT
    ; Function:       Initializes mass-storage devices.
    ; Entry:
    ; Returns:
    ; Modifies:
    ; Note:           In the minimum configuration, only a hook is
    ;                 provided.
    ;
0147 ENT     FORMAT
    ;
    ; Name:           ISFLIO
    ; Function:       Checks if device I/O is being done.
    ; Entry:          None
    ; Returns:        Non-zero if so, zero otherwise.
    ; Modifies:       AF
    ;
014A ENT     ISFLIO
    ;
    ; Name:           OUTDLP
    ; Function:       Outputs to the line printer.
    ; Entry:          Code in [Acc]
    ; Returns:        None
    ; Modifies:       F
    ; Note:           This entry differs from LPTOUT in that:
    ;                  1) TABs are expanded to spaces,
    ;                  2) Hiragana and graphics are converted when
    ;                     a non-MSX printer is in use,
    ;                  3) A jump to 'Device I/O error' is made when
    ;                     aborted.
    ;
014D ENT     OUTDLP
    ;
    ; Name:           GETVCP
    ; Function:
    ; Entry:
    ; Returns:
    ; Modifies:
    ; Note:           Used only to play music in the background.
    ;
0150 ENT     GETVCP
    ;
```

```
   ; Name:           GETVC2
   ; Function:
   ; Entry:
   ; Returns:
   ; Modifies:
   ; Note:           Used only to play music in the background.
   ;
0153 ENT     GETVC2
   ;
   ; Name:           KILBUF
   ; Function:       Clears the keyboard buffer.
   ; Entry:          None
   ; Returns:        None
   ; Modifies:       HL
   ;
0156 ENT     KILBUF
   ;
   ; Name:           CALBAS
   ; Function:       Performs far_call (i.e., inter-slot call) into
   ;                 the BASIC interpreter.
   ; Entry:          Address in [IX]
   ; Returns:
   ; Modifies:
   ;
0159 ENT     CALBAS
   ;
   ; The following is a patch area for BIOS.  It is placed here to
   ; make it easier to add new entry vectors.
   ;
     HOLE    90
```

2.2.2 Work Area

```
   ;
   ;  The  following short  routines  perform the inter-slot read/
   ;  write and call functions.
   ;
      PPI.AW==&B10101000                ;A8H    Write to PPI Port A
   ;
   ;  Read primitive
   ;
F380  RMB(RDPRIM, 5)
                  OUT      PPI.AW      ;Select primary slot
                  MOV      E,M         ;Read from slot
                  JMPR     WRPRM1      ;Restore current setting
   ;
   ;  Write primitive
   ;
F385  RMB(WRPRIM, 7)
                  OUT      PPI.AW      ;Select primary slot
                  MOV      M,E         ;Write to slot
          WRPRM1: MOV      A,D         ;Load current setting
                  OUT      PPI.AW      ;Restore current setting
                  RET
   ;
   ;  Call primitive
   ;
 F38C RMB(CLPRIM, 14)
                  OUT      PPI.AW      ;Select primary slot
                  EXAF                 ;Restore [Acc] and flags
                  CALL     CLPRIM+12   ;Perform indirect call by IX
                  EXAF                 ;Save possible returned value
                  POP      PSW         ;Get old slot status
                  OUT      PPI.AW      ;Restore it
                  EXAF                 ;Restore possible    returned
                                       ;value
                  RET
                  IX
                  PCHL
 F39A RMB(USRTAB, 20)
                  DW       FCERR
                  DW       FCERR
                  DW       FCERR
                  DW       FCERR
                  DW       FCERR
                  DW       FCERR
                  DW       FCERR
                  DW       FCERR
                  DW       FCERR
                  DW       FCERR
 F3AE RMB(LINL40,1)
                  DB       39
 F3AF RMB(LINL32,1)
                  DB       LINLN
 F3B0 RMB(LINLEN, 1)
                  DB       LINLN       ;Line length
```

```
F3B1  RMB(CRTCNT, 1)
                  DB      24          ;Line count
F3B2  RMB(CLMLST, 1)
                  DB      14
   ;
   ;  Beginning of MSX-specific work area
   ;
F3B3  RMB(TXTNAM, 2)
                  DW1     &B00000000000000+$CODE  ;0000H
F3B5  RMB(TXTCOL, 2)
                  DW1     &B00000000000000+$CODE  ;       Unused
F3B7  RMB(TXTCGP, 2)
                  DW1     &B00100000000000+$CODE  ;0800H
F3B9  RMB(TXTATR, 2)
                  DW1     &B00000000000000+$CODE  ;       Unused
F3BB  RMB(TXTPAT, 2)
                  DW1     &B00000000000000+$CODE  ;       Unused
   ;
F3BD  RMB(T32NAM, 2)
                  DW1     &B01100000000000+$CODE  ;1800H
F3BF  RMB(T32COL, 2)
                  DW1     &B10000000000000+$CODE  ;2000H
F3C1  RMB(T32CGP, 2)
                  DW1     &B00000000000000+$CODE  ;0000H
F3C3  RMB(T32ATR, 2)
                  DW1     &B01101100000000+$CODE  ;1B00H
F3C5  RMB(T32PAT, 2)
                  DW1     &B11100000000000+$CODE  ;3800H
   ;
F3C7  RMB(GRPNAM, 2)
                  DW1     &B01100000000000+$CODE  ;1800H
F3C9  RMB(GRPCOL, 2)
                  DW1     &B10000000000000+$CODE  ;2000H
F3CB  RMB(GRPCGP, 2)
                  DW1     &B00000000000000+$CODE  ;0000H
F3CD  RMB(GRPATR, 2)
                  DW1     &B01101100000000+$CODE  ;1B00H
F3CF  RMB(GRPPAT, 2)
                  DW1     &B11100000000000+$CODE  ;3800H
   ;
F3D1  RMB(MLTNAM, 2)
                  DW1     &B00100000000000+$CODE  ;0800H
F3D3  RMB(MLTCOL, 2)
                  DW1     &B00000000000000+$CODE  ;       Unused
F3D5  RMB(MLTCGP, 2)
                  DW1     &B00000000000000+$CODE  ;0000H
F3D7  RMB(MLTATR, 2)
                  DW1     &B01101100000000+$CODE  ;1B00H
F3D9  RMB(MLTPAT, 2)
                  DW1     &B11100000000000+$CODE  ;3800H
   ;
F3DB  RMB(CLIKSW, 1)
                  DB      1
F3DC  RMB(CSRY,   1)
                  DB      1     ;Cursor position Y
```

```
F3DD RMB(CSRX,    1)
                  DB      1     ;Cursor position X
F3DE RMB(CNSDFG,  1)
                  DB      0     ;Function key display switch
   ;
   ;  Save area for the VDP registers
   ;
F3DF RMB(RG0SAV,  1)
                  DB      0
F3E0 RMB(RG1SAV,  1)
                  DB      &B11100000
F3E1 RMB(RG2SAV,  1)
                  DB      0
F3E2 RMB(RG3SAV,  1)
                  DB      0
F3E3 RMB(RG4SAV,  1)
                  DB      0
F3E4 RMB(RG5SAV,  1)
                  DB      0
F3E5 RMB(RG6SAV,  1)
                  DB      0
F3E6 RMB(RG7SAV,  1)
                  DB      0
F3E7 RMB(STATFL,  1)
                  DB      0
   ;
F3E8 RMB(TRGFLG,  1)
                  DB      &B11111111
F3E9 RMB(FORCLR,  1)
                  DB      15  ;Foreground color, default is white
F3EA RMB(BAKCLR,  1)
                  DB      4   ;Background color, default is blue
F3EB RMB(BDRCLR,  1)
                  DB      7   ;Screen border color
F3EC RMB(MAXUPD,  3)
                  JMP     $CODE
F3EF RMB(MINUPD,  3)
                  JMP     $CODE
F3F2 RMB(ATRBYT,  1)
                  DB      15  ;Attribute byte
   ;
F3F3 RMB(QUEUES,  2)
                  DW1   QUETAB ;Address of QUEUTL queue tables
F3F5 RMB(FRCNEW,  1)
                  DB      255
F3F6 RMB(SCNCNT,  1)
                  DB      1     ;Interval of keyscan
F3F7 RMB(REPCNT,  1)
                  DB      50
F3F8 RMB(PUTPNT,  2)
                  DW1     KEYBUF
F3FA RMB(GETPNT,  2)
                  DW1     KEYBUF
F3FC RMB(CSl20,   5*2)
         ;
```

```
        ;  Some parameters for cassette
        ;
     HEDLEN= 2000          ;Length of  header bits (mark) for short
                           ;header
        ;
        ;  The following parameters are for 1200 baud.
        ;
        INTERN  LOW01,HIGH01,LOW11,HIGH11
        LOW01=  83               ;Width of low  state for 0
        HIGH01= 92               ;Width of high state for 0
        LOW11=  38               ;Width of low  state for 1
        HIGH11= 45               ;Width of high state for 1
                DB      LOW01
                DB      HIGH01
                DB      LOW11
                DB      HIGH11
                DB      HEDLEN*2/256
        ;
        ;  The following parameters are for 2400 baud.
        ;
        INTERN  LOW02,HIGH02,LOW12,HIGH12
        LOW02=  37          ;Width  of low state  for  0  1200Hz-
                            ;416.7 usec
        HIGH02= 45          ;Width of high state for 0
        LOW12=  14          ;Width  of low state  for  1  2400Hz-
                            ;208.3 usec
        HIGH12= 22          ;Width of high state for 1
                DB      LOW02
                DB      HIGH02
                DB      LOW12
                DB      HIGH12
                DB      HEDLEN*4/256
F406 RMB(LOW,   2)
                DB      LOW01           ;Default 1200 baud
                DB      HIGH01
F408 RMB(HIGH,  2)
                DB      LOW11
                DB      HIGH11
F40A RMB(HEADER, 1)
                DB      HEDLEN*2/256    ;Default 1200 baud
F40B RMB(ASPCT1, 2)
                DW1     $CODE+256       ;256/aspect ratio
F40D RMB(ASPCT2, 2)
                DW1     $CODE+256       ;256*aspect ratio
    ;
    ;  ENDPRG must be the last one which needs initializing
    ;
F40F RMB(ENDPRG, 5)
                DB      ":"  ;Dummy program end for RESUME NEXT
    ;
    ;  End of initialized constants
    ;
    INTERN  INILEN
    INILEN= ENDPRG+1-INIRAM ;Length of initialized data
    ;
```

```
F414 RMB(ERRFLG, 1)          ;Used to save the error number
F415 RMB(LPTPOS, 1)          ;Position of printer head:  initially
                             ;0
F416 RMB(PRTFLG, 1)          ;Whether output goes to LPT
F417 RMB(NTMSXP, 1)          ;Non-0 if not 'MSX-printer'
F418 RMB(RAWPRT, 1)          ;Non-0 if printing is in 'raw-mode'
F419 RMB(VLZADR, 2)          ;Address of character replaced by VAL
F41B RMB(VLZDAT, 1)          ;Character replaced by 0 by VAL
F41C RMB(CURLIN, 2)
      ZX==    ZX+1
F41F RMB(KBUF, KBFLEN)       ;This is the crunch buffer.
F55D RMB(BUFMIN, 1)          ;Since  the data pointer always  starts
                             ;on commas or terminators, commas (pre-
                             ;load or ROM) are used by INPUTs.
F55E RMB(BUF, BUFLEN+3)      ;Type in stored here. Direct statements
                             ;execute out of here.  Remember "INPUT"
                             ;destroys BUF.   Must  be  at  a  lower
                             ;address  than  DSCTMP, or   assignment
                             ;of string values  in direct statements
                             ;won't copy into string space --  which
                             ;it must.
F660 RMB(ENDBUF, 1)          ;Place to stop big lines
F661 RMB(TTYPOS, 1)          ;Store terminal position here
F662 RMB(DIMFLG, 1)          ;In getting  a pointer  to  a  variable
                             ;it  is   important   to remember  whether
                             ;it is being done for a "DIM" or not.
                             ;DIMFLG and  VALTYP must be consecutive
                             ;locations.
F663 RMB(VALTYP, 1)          ;Type indicator
F664 RMB(OPRTYP, 0)          ;Used to store operator number  in  the
                             ;extended momentarily  before  operator
                             ;application (APPLOP)
F664 RMB(DORES,  1)          ;Whether can or  can't crunch  reserved
                             ;words turned on in the 8K when  "DATA"
                             ;is being scanned by  CRUNCH,  thus un-
                             ;quoted strings won't be crunched.
F665 RMB(DONUM,  1)          ;Flag  for   CRUNCH  =0  means  numbers
                             ;allowed, (floating, INT, DBL) 1  means
                             ;numbers allowed,  CRUNCH   by  calling
                             ;LINGET  -1   (377)    means   numbers
                             ;not allowed (scanning variable name).
F666 RMB(CONTXT, 2)          ;Saved text pointer  used  by CHRGET to
                             ;save the text pointer after a constant
                             ;has been scanned.
F668 RMB(CONSAV, 1)          ;The saved token for  a constant  after
                             ;CHRGET has been called.
F669 RMB(CONTYP, 1)          ;Saved constant VALTYPE
F66A RMB(CONLO,  8)          ;Saved constant VALUE
F672 RMB(MEMSIZ, 2)          ;Highest location in memory
F674 RMB(STKTOP, 2)          ;Top location to be used for the stack,
                             ;initially  set up  by  INIT  depending
                             ;on memory size to allow  for  50 bytes
                             ;of string space.  Changed  by  a CLEAR
                             ;command with arguments.
F676 RMB(TXTTAB, 2)          ;Pointer of beginning of text  does not
```

```
                                ;change after being set up by INIT.
F678 RMB(TEMPPT, 2)             ;Pointer at first free temporary des-
                                ;criptor initialized to point to TEMPST
F67A RMB(TEMPST, 3*NUMTMP)      ;Storage for NUMTMP temp. descriptors
F698 RMB(DSCTMP, 3)             ;String functions build answer
                                ;descriptor here must be after TEMPST
                                ;and before PARM1.

 INTERN  DSCPTR
 DSCPTR= DSCTMP+1               ;Where in DSCTMP string address stored
F69B RMB(FRETOP, 2)             ;Top of string free space
F69D RMB(TEMP3,  2)             ;Used to store the address of the end
                                ;of string arrays in garbage collection
                                ;and used momentarily by FRMEVL used
                                ;in EXTENDED by FOUT and user defined
                                ;functions and array variable handling
                                ;temporarily.
F69F RMB(TEMP8,  2)             ;7/3/79 Now used for garbage collection
                                ;not TEMP3 due to conflict
F6A1 RMB(ENDFOR, 2)             ;Saved text pointer at end of "FOR"
                                ;statement
F6A3 RMB(DATLIN, 2)             ;DATA LINE # -- remember ERRORS
F6A5 RMB(SUBFLG, 1)             ;Flag whether subscripted variable
                                ;allowed "FOR" and USR-defined function
                                ;Pointer fetching turn this on before
                                ;calling PTRGET so arrays won't be
                                ;detected.  STKINI and PTRGET clear it.
F6A6 RMB(USFLG,  0)
F6A6 RMB(FLGINP, 1)             ;Flag for INPUT or READ
F6A7 RMB(TEMP,   2)             ;Temporary for statement code.  NEWSTT
                                ;saves [H,L] here for INPUT and ^C,
                                ;"LET" saves variable pointers here,
                                ;for "FOR-NEXT" saves its text pointer
                                ;here, CLEARC saves [H,L] here.
F6A9 RMB(PTRFLG, 1)             ;=0 If no line numbers converted to
                                ;pointers, non-zero if pointers exist.
F6AA RMB(AUTFLG, 1)             ;Flag to indicate AUTO command in
                                ;progress, =0 if not, non-zero if so.
F6AB RMB(AUTLIN, 2)             ;Current line being inserted by AUTO
F6AD RMB(AUTINC, 2)             ;AUTO increment
F6AF RMB(SAVTXT, 2)             ;Place where NEWSTT saves text pointer
                                ;for "RESUME" statement
F6B1 RMB(SAVSTK, 2)             ;NEWSTT saves stack here before so
                                ;that error recovery can restore the
                                ;stack when an error occurs.
F6B3 RMB(ERRLIN, 2)             ;Line number where last error occured.
F6B5 RMB(DOT,    2)             ;Keeps current line for edit & LIST
F6B7 RMB(ERRTXT, 2)             ;Text pointer for use by "RESUME"
F6B9 RMB(ONELIN, 2)             ;Line to GOTO when an error occurs.
F6BB RMB(ONEFLG, 1)             ;ONEFLG=1 if executing an error trap
                                ;routine, otherwise 0.
F6BC RMB(TEMP2,  2)             ;Formula evaluator temp.  Must be pre-
                                ;served by operators used in EXTENDED
                                ;by FOUT and user-defined functions
                                ;array variable handler temporary
F6BE RMB(OLDLIN, 2)             ;Old line number (set up by ^C, "STOP"
```

```
                                ;or "END" in a program).
F6C0 RMB(OLDTXT, 2)             ;Old text pointer.  Points at statement
                                ;to be executed next.
F6C2 RMB(VARTAB, 2)             ;Pointer  to start of  simple  variable
                                ;space.  Updated  whenever  the size of
                                ;the program changes, set to [TXTTAB]+2
                                ;by SCRATCH ("NEW").
F6C4 RMB(ARYTAB, 2)             ;Pointer to beginning of array table.
                                ;Incremented by 6 whenever a new simple
                                ;variable is found, and set to [VARTAB]
                                ;by CLEARC.
F6C6 RMB(STREND, 2)             ;End  of  storage  in  use.    Increased
                                ;whenever  a  new  array  or  a  simple
                                ;variable   is   encountered,   set   to
                                ;[VARTAB] by CLEARC.
F6C8 RMB(DATPTR, 2)             ;Pointer to data.  Initialized to point
                                ;at the zero  in front  of  [TXTTAB]  by
                                ;"RESTORE" which is called by CLEARC,
                                ;updated by execution of a "READ"
F6CA RMB(DEFTBL, 26)            ;This  gives  the  default  VALTYP  for
                                ;each letter  of the alphabet.  It  is
                                ;set  up  by  "CLEAR" and is changed by
                                ;"DEFSTR", "DEFINT", "DEFSNG", "DEFDBL"
                                ;and used by PTRGET when  !  #  %  or
                                ;$ do not follow a variable name.
   ;
   ;  RAM storage for user-defined function parameter information
   ;
 INTERN  PRMSIZ
 PRMSIZ==^D100                  ;Number of bytes for definition block
F6E4 RMB(PRMSTK, 2)             ;Previous  definition  block  on  stack
                                ;block (for garbage collection)
F6E6 RMB(PRMLEN, 2)             ;Number of bytes in the active table
F6E8 RMB(PARM1,  PRMSIZ)        ;The active parameter definition table
F74C RMB(PRMPRV, 2)             ;Initially PRMSTK,  the  pointer at the
                                ;previous parameter block  (for garbage
                                ;collection)
F74E RMB(PRMLN2, 2)             ;Size of parameter block being built
F750 RMB(PARM2,  PRMSIZ)        ;Place to keep parameters being made
F7B4 RMB(PRMFLG, 1)             ;Used by PTRGET to flag  if  PARM1  has
                                ;been searched
F7B5 RMB(ARYTA2, 2)             ;Stopping point for simple search
                                ; (either [ARYTAB] or PARM1+[PRMLEN])
F7B7 RMB(NOFUNS, 1)             ;Zero if no functions active.   Saves
                                ;TIME in simple search
F7B8 RMB(TEMP9,  2)             ;Garbage  collection  temp.  to  chain
                                ;through parameter blocks.
F7BA RMB(FUNACT, 2)             ;Count of active functions
F7BC RMB(SWPTMP, 8)             ;Value of first  "SWAP" variable stored
                                ;here
F7C4 RMB(TRCFLG, 1)             ;Zero means no trace in progress
   ;
   ;  This is the RAM temporary area for the math package routines
   ;
F7C5 RMB(FBUFFR, 43)            ;Buffer for FOUT
```

```
F7F0 RMB(DECTMP, 2)          ;Used by decimal int to float
F7F2 RMB(DECTM2, 2)          ;Used by divide
F7F4 RMB(DECCNT, 1)          ;Used by divide
   ;
   ;  Decimal accumulator
   ;
 ZX==    ZX+1                ;Temporary sign complement
F7F6 RMB(DAC,    16)
 INTERN  FACLO
 FACLO=  DAC+2
   ;
   ;  Holding registers for decimal multiplication
   ;
F806 RMB(HOLD8,  48)         ;80*X
F836 RMB(HOLD2,  8)          ;2*X
F83E RMB(HOLD,   8)          ;1*X
   ;
   ;  Argument accumulator
   ;
 ZX==    ZX+1                ;Temporary sign complement
F847 RMB(ARG,    16)
F857 RMB(RNDX,   8)          ;Holds last random number generated
```

```
                SUBTTL  Data Area
   ;
   ;  Set up by initialization.  Unchanged by disk code.
   ;
F85F RMB(MAXFIL, 1)          ;Highest legal file number
F860 RMB(FILTAB, 2)          ;Points to adress of file data
F862 RMB(NULBUF, 2)          ;Points to file 0 buffer
   ;
   ;  Set up by file / drive selection routines.   Only PTRFIL  is
   ;  cleared elsewhere.
   ;
F864 RMB(PTRFIL, 2)          ;Points to file data of selected file
   ;
   ;  Misc.
   ;
F866 RMB(RUNFLG, 0)          ;Non-zero for RUN after LOAD
F866 RMB(FILNAM,11)          ;Holds filename for DIRSRC, from NAMSCN
F871 RMB(FILNM2,11)          ;Holds other filename for NAME
F87C RMB(NLONLY, 1)          ;Non-zero when loading program
   ;
   ;  Set up by NULOPN and BSAVE, used by BSAVE and CREATE.
   ;
F87D RMB(SAVEND, 2)          ;End of binary or memory image save
F87F RMB(FNKSTR, 16*10)      ;Function key string save area
F91F RMB(CGPNT,  3)          ;Where character pattern is held in ROM
   ;
F922 RMB(NAMBAS,2)           ;Base of current name table
F924 RMB(CGPBAS,2)           ;Base of current cgen table
F926 RMB(PATBAS,2)           ;Base of current sprite pattern table
F928 RMB(ATRBAS,2)           ;Base of current sprite attribute table
   ;
   ;  For GENGRP
   ;
F92A RMB(CLOC,  2)
F92C RMB(CMASK, 1)
F92D RMB(MINDEL,2)
F92F RMB(MAXDEL,2)
   ;
   ;  For CIRCLE
   ;
F931 RMB(ASPECT,2)           ;Aspect ratio for circle
F933 RMB(CENCNT,2)           ;End count
F935 RMB(CLINEF,1)           ;Flag to draw line to center
F936 RMB(CNPNTS,2)           ;Points to plot
F938 RMB(CPLOTF,1)           ;Plot polarity flag
F939 RMB(CPCNT, 2)           ;1/8 of number of points in circle
F93B RMB(CPCNT8,2)           ;Number of pts in circle
F93D RMB(CRCSUM,2)           ;Circle sum
F93F RMB(CSTCNT,2)           ;Start count
F941 RMB(CSCLXY,1)           ;Scaling of x and y
F942 RMB(CSAVEA,2)           ;ADVGRP C save area
F944 RMB(CSAVEM,1)           ;ADVGRP C save area
F945 RMB(CXOFF, 2)           ;X offset from center save location
F947 RMB(CYOFF, 2)           ;Y offset save location
   ;
```

```
   ;  For PAINT
   ;
F949 RMB(LOHMSK,1)          ;RAM save area for left overhang
F94A RMB(LOHDIR,1)
F94B RMB(LOHADR,2)
F94D RMB(LOHCNT,2)
F94F RMB(SKPCNT,2)          ;Skip count
F951 RMB(MOVCNT,2)          ;Move count
F953 RMB(PDIREC,1)          ;Paint direction
F954 RMB(LFPROG,1)
F955 RMB(RTPROG,1)
   ;
   ;  For MACLNG
   ;
F956 RMB(MCLTAB,2)
F958 RMB(MCLFLG,1)          ;Indicates PLAY/DRAW
   ;
   ;  QUEUES for PLAY statement
   ;
F959 RMB(QUETAB,^D24)       ;4 queues (6 bytes each)
F971 RMB(QUEBAK,^D4)        ;For BCKQ
 MUSQLN=:^D128              ;Size of voice queues
 RSIQLN=:^D64
F975 RMB(VOICAQ,MUSQLN)     ;Voice a queue
F9F5 RMB(VOICBQ,MUSQLN)     ;Voice b queue
FA75 RMB(VOICCQ,MUSQLN)     ;Voice c queue
FAF5 RMB(RS2IQ, RSIQLN)     ;RS232 input queue
   ;
   ;  Music stuff
   ;
FB35 RMB(PRSCNT,1)          ;D1-D0 = number of strings parsed
                            ;D7=0 if first pass, 1 if not
FB36 RMB(SAVSP, 2)          ;Save main stack pointer During play
FB38 RMB(VOICEN,1)          ;Set current voice being parsed
FB39 RMB(SAVVOL,2)          ;Save volume for pause
FB3B RMB(MCLLEN,1)
FB3C RMB(MCLPTR,2)
FB3E RMB(QUEUEN,1)          ;Used by intime-action-dequeue
   ;
FB3F RMB(MUSICF,1)          ;Music interrupt flag
FB40 RMB(PLYCNT,1)          ;Number of play statements queued for
                            ;background task
   ;
   ;  Per Voice Static Data Area Displacement Definitions
   ;
 METREX=:0                  ;Timer countdown
 VCXLEN=:METREX+2           ;MCLLEN for this voice
 VCXPTR=:VCXLEN+1           ;MCLPTR for this voice
 VCXSTP=:VCXPTR+2           ;Save top of stack pointer
 QLENGX=:VCXSTP+2           ;Number of bytes to be queued
 NTICSX=:QLENGX+1           ;New countdown
 TONPRX=:NTICSX+2           ;Tone period
 AMPLTX=:TONPRX+2           ;Amplitude/shape
 ENVPRX=:AMPLTX+1           ;Envelope period
 OCTAVX=:ENVPRX+2           ;Octave
```

```
 NOTELX=:OCTAVX+1           ;Note length
 TEMPOX=:NOTELX+1           ;Tempo
 VOLUMX=:TEMPOX+1           ;Volume
 ENVLPX=:VOLUMX+1           ;Envelope shape
 MCLSTX=:ENVLPX+^D14        ;Stack save area
 MCLSEX=:MCLSTX+3           ;Initial stack
 VCBSIZ=:MCLSEX-METREX+1    ;Voice static buffer size
FB41 RMB(VCBA,  VCBSIZ)     ;Static data for voice 0
FB66 RMB(VCBB,  VCBSIZ)     ;Static data for voice 1
FB8B RMB(VCBC,  VCBSIZ)     ;Static data for voice 2
   ;
   ;  Area between here and MUSICF  is cleared  everytime a IGICIN
   ;  is called.
   ;
FBB0 RMB(ENSTOP,1)          ;Non-zero if warm start enabled
FBB1 RMB(BASROM,1)          ;Non-zero if BASIC text is in ROM
FBB2 RMB(LINTTB,24)         ;Line terminator table
FBCA RMB(FSTPOS,2)          ;First position when entered INLIN
FBCC RMB(CODSAV,1)          ;Code save area for cursor
FBCD RMB(FNKSWI,1)          ;Indicates which function key is
                            ;dislayed
FBCE RMB(FNKFLG,10)         ;Indicates key is assigned to event
                            ;device
FBD8 RMB(ONGSBF,1)          ;Global event flag
FBD9 RMB(CLIKFL,1)
FBDA RMB(OLDKEY,11)         ;Old key status
FBE5 RMB(NEWKEY,11)         ;New key status
 INTERN  SFTKEY
 SFTKEY= NEWKEY+6           ;GR,CTRL,SHIFT status
FBF0 RMB(KEYBUF,40)         ;Key code buffer
FC18 RMB(BUFEND,0)          ;End of KEYBUF
FC18 RMB(LINWRK,40)         ;Scratch area for screen handler
FC40 RMB(PATWRK,8)          ;Scratch area for pattern converter
FC48 RMB(BOTTOM,2)          ;Bottom of equipped RAM
FC4A RMB(HIMEM, 2)          ;Highest available memory
FC4C RMB(TRPTBL,3*NUMTRP)   ;Trap table
FC9A RMB(RTYCNT,1)
FC9B RMB(INTFLG,1)
FC9C RMB(PADY,  1)
FC9D RMB(PADX,  1)
FC9E RMB(JIFFY, 2)
FCA0 RMB(INTVAL,2)
FCA2 RMB(INTCNT,2)
FCA4 RMB(LOWLIM,1)          ;Used when reading cassette
FCA5 RMB(WINWID,1)          ;Used when reading cassette
FCA6 RMB(GRPHED,1)          ;Flag for graphic character output
FCA7 RMB(ESCCNT,1)          ;Escape sequence counter
FCA8 RMB(INSFLG,1)          ;Insert mode flag
FCA9 RMB(CSRSW, 1)          ;Cursor display switch
FCAA RMB(CSTYLE,1)          ;Cursor style
FCAB RMB(CAPST, 1)          ;Capital status
FCAC RMB(KANAST,1)          ;Kana lock status
FCAD RMB(KANAMD,1)          ;Non-0 if JIS
FCAE RMB(FLBMEM,1)          ;0 if loading BASIC program
FCAF RMB(SCRMOD,1)          ;Screen mode
```

```
                                ;(0-ext,1-text,2-hires,2-multi)
FCB0 RMB(OLDSCR,1)              ;Screen mode save area
FCB1 RMB(CASPRV,1)              ;Previous character save area for CAS:
FCB2 RMB(BRDATR,1)              ;Border color for PAINT
FCB3 RMB(GXPOS, 2)
FCB5 RMB(GYPOS, 2)
FCB7 RMB(GRPACX,2)              ;Graphic accumulater
FCB9 RMB(GRPACY,2)
FCBB RMB(DRWFLG,1)
FCBC RMB(DRWSCL,1)              ;Draw scale factor - 0 means no scaling
FCBD RMB(DRWANG,1)              ;Draw angle (0-3)
   ;
   ;  For BLOAD and BSAVE
   ;
FCBE RMB(RUNBNF,1)              ;Doing BLOAD, BSAVE or not
FCBF RMB(SAVENT,2)              ;Start address for BSAVE
   ;
   ;  Information save area for slots
   ;
FCC1 RMB(EXPTBL, 4)             ;Flag table for expanded slot
                                ;Holds 255 if expanded
FCC5 RMB(SLTTBL, 4)             ;Current setting for each expanded
                                ;slot register
FCC9 RMB(SLTATR, 64)            ;Holds attributes for each slot
FD09 RMB(SLTWRK, 128)           ;Holds work area specific for each slot
   ;
   ;  For CALL statement and device expander
   ;
FD89 RMB(PROCNM,16)             ;Name of expanded  statement terminated
                                ;by 0
FD99 RMB(DEVICE, 1)             ;The device ID for a cartridge (0 to 3)
```

```
   COMMENT %

   The following are definitions of hooks and their functions:

            Name          - Name of hook
            Location      - Location in module it is used
            Purpose       - Use

   %

   GSX==   ZX
FD9A RMB(HOKJMP,0)
   ;
   ;  Name:            H.KEYI
   ;  Location:        MSXIO, at the beginning of interrupt handler
   ;  Purpose:         Does additional interrupt handling such as
   ;                   RS-232C.
   ;
FD9A RMB(H.KEYI,5)
   ;
   ;  Name:            H.TIMI
   ;  Location:        MSXIO, in timer interrupt handler
   ;  Purpose:         Allows other interrupt handling invoked by
   ;                   timer.
   ;
FD9F RMB(H.TIMI,5)
   ;
   ;  Name:            H.CHPU
   ;  Location:        MSXIO, at the beginning of CHPUT (CHaracter
   ;                   outPUT) routine.
   ;  Purpose:         Allows for other console output devices.
   ;
FDA4 RMB(H.CHPU,5)
   ;
   ;  Name:            H.DSPC
   ;  Location:        MSXIO, at the beginning of DSPCSR (DiSPlay
   ;                   CurSoR) routine.
   ;  Purpose:         Allows for other console output devices.
   ;
FDA9 RMB(H.DSPC,5)
   ;
   ;  Name:            H.ERAC
   ;  Location:        MSXIO, at the beginning of ERACSR (ERAse
   ;                   CurSoR) routine.
   ;  Purpose:         Allows for other console output devices.
   ;
FDAE RMB(H.ERAC,5)
   ;
   ;  Name:            H.DSPF
   ;  Location:        MSXIO, at the beginning of DSPFNK (DiSPlay
   ;                   FuNction Key) routine.
   ;  Purpose:         Allows for other console output devices.
   ;
FDB3 RMB(H.DSPF,5)
   ;
```

```
    ;  Name:            H.ERAF
    ;  Location:        MSXIO,   at   the beginning of ERAFNK (ERAse
    ;                   FuNction Key) routine.
    ;  Purpose:         Allows for other console output devices.
    ;
FDB8 RMB(H.ERAF,5)
    ;
    ;  Name:            H.TOTE
    ;  Location:        MSXIO,  at  the  beginning  of  TOTEXT (force
    ;                   screen TO TEXT mode) routine.
    ;  Purpose:         Allows for other console output devices.
    ;
FDBD RMB(H.TOTE,5)
    ;
    ;  Name:            H.CHGE
    ;  Location:        MSXIO, at  the beginning of CHGET (CHaracter
    ;                   GET) routine.
    ;  Purpose:         Allows for other console input devices.
    ;
FDC2 RMB(H.CHGE,5)
    ;
    ;  Name:            H.INIP
    ;  Location:        MSXIO,at the beginning of INIPAT (INItialize
    ;                   PATtern) routine.
    ;  Purpose:         Allows for other character sets.
    ;
FDC7 RMB(H.INIP,5)
    ;
    ;  Name:            H.KEYC
    ;  Location:        MSXIO,  at  the  beginning  of  KEYCOD  (KEY
    ;                   CODer) routine.
    ;  Purpose:         Allows for other key assignments.
    ;
FDCC RMB(H.KEYC,5)
    ;
    ;  Name:            H.KYEA
    ;  Location:        MSXIO, at the beginning of KYEASY (KeY EASY)
    ;                   routine.
    ;  Purpose:         Allows for other key assignments.
    ;
FDD1 RMB(H.KYEA,5)
    ;
    ;  Name:            H.NMI
    ;  Location:        MSXIO, at the beginning of NMI (Non Maskable
    ;                   Interrupt) routine.
    ;  Purpose:         Allows for NMI handling.
    ;
FDD6 RMB(H.NMI, 5)
    ;
    ;  Name:            H.PINL
    ;  Location:        MSXINL, at the beginning of PINLIN  (Program
    ;                   INput LINe) routine.
    ;  Purpose:         Allows other console input devices or  other
    ;                   input designs to be used.
    ;
```

```
FDDB  RMB(H.PINL,5)
   ;
   ;  Name:            H.QINL
   ;  Location:        MSXINL, at the beginning of QINLIN (Question
   ;                   mark and INput LINe) routine.
   ;  Purpose:         Allows other console input devices  or other
   ;                   input designs to be used.
   ;
FDE0  RMB(H.QINL,5)
   ;
   ;  Name:            H.INLI
   ;  Location:        MSXINL,  at  the  beginning  of  INLIN (INput
   ;                   LINe) routine.
   ;  Purpose:         Allows other console input  devices or other
   ;                   input designs to be used.
   ;
FDE5  RMB(H.INLI,5)
   ;
   ;  Name:            H.ONGO
   ;  Location:        MSXSTS, at the beginning of ONGOTP (ON  GOTo
   ;                   Procedure) routine.
   ;  Purpose:         Allows for other console input devices to be
   ;                   used.
FDEA  RMB(H.ONGO,5)
   ;
   ;  Name:            H.DSKO
   ;  Location:        MSXSTS,  at  the  beginning  of  DSKO$ (DiSK
   ;                   Output) routine.
   ;  Purpose:         Installs the disk driver.
   ;
FDEF  RMB(H.DSKO,5)
   ;
   ;  Name:            H.SETS
   ;  Location:        MSXSTS,  at   the  beginning  of  SETS  (SET
   ;                   attributeS) routine.
   ;  Purpose:         Installs the disk driver.
   ;
FDF4  RMB(H.SETS,5)
   ;
   ;  Name:            H.NAME
   ;  Location:        MSXSTS, at the NAME (reNAME) routine.
   ;  Purpose:         Installs the disk driver.
   ;
FDF9  RMB(H.NAME,5)
   ;
   ;  Name:            H.KILL
   ;  Location:        MSXSTS,  at  the  beginning  of  KILL  (KILL
   ;                   file) routine.
   ;  Purpose:         Installs the disk driver.
   ;
FDFE  RMB(H.KILL,5)
   ;
   ;  Name:            H.IPL
   ;  Location:        MSXSTS,  at the beginning of  IPL (Initial
   ;                   Program Load) routine.
```

```
   ;  Purpose:         Installs the disk driver.
   ;
FE03 RMB(H.IPL, 5)
   ;
   ;  Name:            H.COPY
   ;  Location:        MSXSTS,  at  the  beginning  of  COPY (COPY
   ;                   file) routine.
   ;  Purpose:         Installs the disk driver.
   ;
FE08 RMB(H.COPY,5)
   ;
   ;  Name:            H.CMD
   ;  Location:        MSXSTS, at the beginning  of  CMD  (CoMmanD)
   ;                   routine.
   ;  Purpose:         Installs the disk driver.
   ;
FE0D RMB(H.CMD, 5)
   ;
   ;  Name:            H.DSKF
   ;  Location:        MSXSTS, at the beginning of DSKF (DiSK Free)
   ;                   routine.
   ;  Purpose:         Installs the disk driver.
   ;
FE12 RMB(H.DSKF,5)
   ;
   ;  Name:            H.DSKI
   ;  Location:        MSXSTS,  at  the  beginning  of  DSKI  (DiSK
   ;                   Input) routine.
   ;  Purpose:         Installs the disk driver.
   ;
FE17 RMB(H.DSKI,5)
   ;
   ;  Name:            H.ATTR
   ;  Location:        MSXSTS,at the beginning of ATTR$ (ATTRibute)
   ;                   routine.
   ;  Purpose:         Installs the disk driver.
   ;
FE1C RMB(H.ATTR,5)
   ;
   ;  Name:            H.LSET
   ;  Location:        MSXSTS, at the beginning of  LSET (Left SET)
   ;                   routine.
   ;  Purpose:         Installs the disk driver.
   ;
FE21 RMB(H.LSET,5)
   ;
   ;  Name:            H.RSET
   ;  Location:        MSXSTS, at the beginning of RSET (Right SET)
   ;                   routine.
   ;  Purpose:         Installs the disk driver.
   ;
FE26 RMB(H.RSET,5)
   ;
   ;  Name:            H.FIEL
   ;  Location:        MSXSTS,  at the beginning  of  FIELD (FIELD)
```

```
    ;                   routine.
    ;  Purpose:         Installs the disk driver.
    ;
FE2B RMB(H.FIEL,5)
    ;
    ;  Name:            H.MKI$
    ;  Location:        MSXSTS, at the beginning of  MKI$ (MaKe Int)
    ;                   routine.
    ;  Purpose:         Installs the disk driver.
    ;
FE30 RMB(H.MKI$,5)
    ;
    ;  Name:            H.MKS$
    ;  Location:        MSXSTS,  at  the  beginning  of  MKS$  (Make
    ;                   Single) routine.
    ;  Purpose:         Installs the disk driver.
    ;
FE35 RMB(H.MKS$,5)
    ;
    ;  Name:            H.MKD$
    ;  Location:        MSXSTS,  at  the  beginning  of  MKD$  (Make
    ;                   Double) routine.
    ;  Purpose:         Installs the disk driver.
    ;
FE3A RMB(H.MKD$,5)
    ;
    ;  Name:            H.CVI
    ;  Location:        MSXSTS,  at  the  beginning  of CVI (Convert
    ;                   Int) routine.
    ;  Purpose:         Installs the disk driver.
    ;
FE3F RMB(H.CVI,5)
    ;
    ;  Name:            H.CVS
    ;  Location:        MSXSTS,  at  the  beginning  of CVS (Convert
    ;                   Sng) routine.
    ;  Purpose:         Installs the disk driver.
    ;
FE44 RMB(H.CVS,5)
    ;
    ;  Name:            H.CVD
    ;  Location:        MSXSTS,  at  the  beginning  of CVD (Convert
    ;                   Dbl) routine.
    ;  Purpose:         Installs the disk driver.
    ;
FE49 RMB(H.CVD,5)
```

```
    ;  Name:            H.GETP
    ;  Location:        SPCDSK, at the GETPTR (GET file PoinTeR).
    ;  Purpose:         Installs the disk driver.
    ;
FE4E RMB(H.GETP,5)
    ;
    ;  Name:            H.SETF
    ;  Location:        SPCDSK, at the SETFIL (SET FILe pointer).
    ;  Purpose:         Installs the disk driver.
    ;
FE53 RMB(H.SETF,5)
    ;
    ;  Name:            H.NOFO
    ;  Location:        SPCDSK, at the NOFOR (NO FOR clause) routine.
    ;  Purpose:         Installs the disk driver.
    ;
FE58 RMB(H.NOFO,5)
    ;
    ;  Name:            H.NULO
    ;  Location:        SPCDSK,at the NULOPN (NULl file OPeN)routine.
    ;  Purpose:         Installs the disk driver.
    ;
FE5D RMB(H.NULO,5)
    ;
    ;  Name:            H.NTFL
    ;  Location:        SPCDSK, at the NTFL0 (NoT FiLe number 0).
    ;  Purpose:         Installs the disk driver.
    ;
FE62 RMB(H.NTFL,5)
    ;
    ;  Name:            H.MERG
    ;  Location:        SPCDSK, at  the MERGE (MERGE  program files)
    ;                   routine.
    ;  Purpose:         Installs the disk driver.
    ;
FE67 RMB(H.MERG,5)
    ;
    ;  Name:            H.SAVE
    ;  Location:        SPCDSK, at the SAVE routine.
    ;  Purpose:         Installs the disk driver.
    ;
FE6C RMB(H.SAVE,5)
    ;
    ;  Name:            H.BINS
    ;  Location:        SPCDSK, at the BINSAV (BINary SAVe) routine.
    ;  Purpose:         Installs the disk driver.
    ;
FE71 RMB(H.BINS,5)
    ;
    ;  Name:            H.BINL
    ;  Location:        SPCDSK, at the BINLOD (BINary LOaD) routine.
    ;  Purpose:         Installs the disk driver.
    ;
FE76 RMB(H.BINL,5)
    ;
```

```
   ;  Name:           H.FILE
   ;  Location:       SPCDSK, at the FILES command.
   ;  Purpose:        Installs the disk driver.
   ;
FE7B RMB(H.FILE,5)
   ;
   ;  Name:           H.DGET
   ;  Location:       SPCDSK, at the DGET (Disk GET) routine.
   ;  Purpose:        Installs the disk driver.
   ;
FE80 RMB(H.DGET,5)
   ;
   ;  Name:           H.FILO
   ;  Location:       SPCDSK, at the FILOUl (FILe OUt 1) routine.
   ;  Purpose:        Installs the disk driver.
   ;
FE85 RMB(H.FILO,5)
   ;
   ;  Name:           H.INDS
   ;  Location:       SPCDSK, at the INDSKC (INput DiSK Character)
   ;                  routine.
   ;  Purpose:        Installs the disk driver.
   ;
FE8A RMB(H.INDS,5)
   ;
   ;  Name:           H.RSLF
   ;  Location:       SPCDSK, to re-select the old drive.
   ;  Purpose:        Installs the disk driver.
   ;
FE8F RMB(H.RSLF,5)
   ;
   ;  Name:           H.SAVD
   ;  Location:       SPCDSK, to save the current drive.
   ;  Purpose:        Installs the disk driver.
   ;
FE94 RMB(H.SAVD,5)
   ;
   ;  Name:           H.LOC
   ;  Location:       SPCDSK, at the LOC (LOCation) function.
   ;  Purpose:        Installs the disk driver.
   ;
FE99 RMB(H.LOC, 5)
   ;
   ;  Name:           H.LOF
   ;  Location:       SPCDSK, at the LOF (Length Of File)function.
   ;  Purpose:        Installs the disk driver.
   ;
FE9E RMB(H.LOF, 5)
   ;
   ;  Name:           H.EOF
   ;  Location:       SPCDSK, at the EOF (End Of File) function.
   ;  Purpose:        Installs the disk driver.
   ;
FEA3 RMB(H.EOF, 5)
   ;
```

```
    ;  Name:            H.FPOS
    ;  Location:        SPCDSK, at FPOS (File POSition) function.
    ;  Purpose:         Installs the disk driver.
    ;
FEA8 RMB(H.FPOS,5)
    ;
    ;  Name:            H.BAKU
    ;  Location:        SPCDSK, at the BAKUPT (BAcK UP) routine.
    ;  Purpose:         Installs the disk driver.
    ;
FEAD RMB(H.BAKU,5)
    ;
    ;  Name:            H.PARD
    ;  Location:        SPCDEV,  at the PARDEV  (PARse DEVice  name)
    ;                   routine.
    ;  Purpose:         Epands logical device names.
    ;
FEB2 RMB(H.PARD,5)
    ;
    ;  Name:            H.NODE
    ;  Location:        SPCDEV, at the NODEVN(NO DEVice Name)routine.
    ;  Purpose:         Sets other default devices.
    ;
FEB7 RMB(H.NODE,5)
    ;
    ;  Name:            H.POSD
    ;  Location:        SPCDEV, at the POSDSK (POSsibly DiSK)routine.
    ;  Purpose:         Installs the disk driver.
    ;
FEBC RMB(H.POSD,5)
    ;
    ;  Name:            H.DEVN
    ;  Location:        SPCDEV, at the DEVNAM (DEVice NAMe) routine.
    ;  Purpose:         Expands logical device names.
    ;
FEC1 RMB(H.DEVN,5)
    ;
    ;  Name:            H.GEND
    ;  Location:        SPCDEV,   at   the   GENDSP   (GENeral device
    ;                   DiSPatcher).
    ;  Purpose:         Expands logical device names.
    ;
FEC6 RMB(H.GEND,5)
    ;
    ;  Name:            H.RUNC
    ;  Location:        BIMISC, at the RUNC (RUN Clear) routine.
    ;  Purpose:
    ;
FECB RMB(H.RUNC,5)
    ;
    ;  Name:            H.CLEA
    ;  Location:        BIMISC, at the CLEARC (CLEAR Clear) routine.
    ;  Purpose:
    ;
FED0 RMB(H.CLEA,5)
```

```
    ;
    ;  Name:            H.LOPD
    ;  Location:        BIMISC, at the LOPDFT (LOop and set DeFaulT)
    ;                   routine.
    ;  Purpose:         Uses other defaults for variables.
    ;
FED5 RMB(H.LOPD,5)
    ;
    ;  Name:            H.STKE
    ;  Location:        BIMISC, at the STKERR (STacK ERRor) routine.
    ;  Purpose:
    ;
FEDA RMB(H.STKE,5)
    ;
    ;  Name:            H.ISFL
    ;  Location:        BIMISC, at the ISFLIO (IS FiLe I/O) routine.
    ;  Purpose:
    ;
FEDF RMB(H.ISFL,5)
    ;
    ;  Name:            H.OUTD
    ;  Location:        BIO, at the OUTDO (OUT DO) routine.
    ;  Purpose:
    ;
FEE4 RMB(H.OUTD,5)
    ;
    ;  Name:            H.CRDO
    ;  Location:        BIO, at the CRDO (CRlf DO) routine.
    ;  Purpose:
    ;
FEE9 RMB(H.CRDO,5)
    ;
    ;  Name:            H.DSKC
    ;  Location:        BIO,  at  the  DSKCHI (DiSK CHaracter Input)
    ;                   routine.
    ;  Purpose:
    ;
FEEE RMB(H.DSKC,5)
    ;
    ;  Name:            H.DOGR
    ;  Location:        GENGRP, at the DOGRPH (DO GRaPH) routine.
    ;  Purpose:
    ;
FEF3 RMB(H.DOGR,5)
    ;
    ;  Name:            H.PRGE
    ;  Location:        BINTRP, at the PRGEND (PRoGram END) routine.
    ;  Purpose:
    ;
FEF8 RMB(H.PRGE,5)
    ;
    ;  Name:            H.ERRP
    ;  Location:        BINTRP, at the ERRPRT (ERRor PRinT) routine.
    ;  Purpose:
    ;
```

```
FEFD RMB(H.ERRP,5)
   ;
   ;  Name:
   ;  Location:        BINTRP
   ;  Purpose:
   ;
FF02 RMB(H.ERRF,5)
   ;
   ;  Name:            H.READ
   ;  Location:        BINTRP, at the READY entry.
   ;  Purpose:
   ;
FF07 RMB(H.READ,5)
   ;
   ;  Name:            H.MAIN
   ;  Location:        BINTRP, at the MAIN entry.
   ;  Purpose:
   ;
FF0C RMB(H.MAIN,5)
   ;
   ;  Name:            H.DIRD
   ;  Location:        BINTRP, at the DIRDO (DIRect DO) entry.
   ;  Purpose:
   ;
FF11 RMB(H.DIRD,5)
   ;
   ;  Name:
   ;  Location:        BINTRP
   ;  Purpose:
   ;
FF16 RMB(H.FINI,5)
   ;
   ;  Name:
   ;  Location:        BINTRP
   ;  Purpose:
   ;
FF1B RMB(H.FINE,5)
   ;
   ;  Name:
   ;  Location:        BINTRP
   ;  Purpose:
   ;
FF20 RMB(H.CRUN,5)
   ;
   ;  Name:
   ;  Location:        BINTRP
   ;  Purpose:
   ;
FF25 RMB(H.CRUS,5)
   ;
   ;  Name:
   ;  Location:        BINTRP
   ;  Purpose:
   ;
FF2A RMB(H.ISRE,5)
```

```
    ;
    ;  Name:
    ;  Location:        BINTRP
    ;  Purpose:
    ;
FF2F RMB(H.NTFN,5)
    ;
    ;  Name:
    ;  Location:        BINTRP
    ;  Purpose:
    ;
FF34 RMB(H.NOTR,5)
    ;
    ;  Name:
    ;  Location:        BINTRP
    ;  Purpose:
    ;
FF39 RMB(H.SNGF,5)
    ;
    ;  Name:
    ;  Location:        BINTRP
    ;  Purpose:
    ;
FF3E RMB(H.NEWS,5)
    ;
    ;  Name:
    ;  Location:        BINTRP
    ;  Purpose:
    ;
FF43 RMB(H.GONE,5)
    ;
    ;  Name:
    ;  Location:        BINTRP
    ;  Purpose:
    ;
FF48 RMB(H.CHRG,5)
    ;
    ;  Name:
    ;  Location:        BINTRP
    ;  Purpose:
    ;
FF4D RMB(H.RETU,5)
    ;
    ;  Name:
    ;  Location:        BINTRP
    ;  Purpose:
    ;
FF52 RMB(H.PRTF,5)
    ;
    ;  Name:
    ;  Location:        BINTRP
    ;  Purpose:
    ;
FF57 RMB(H.COMP,5)
    ;
```

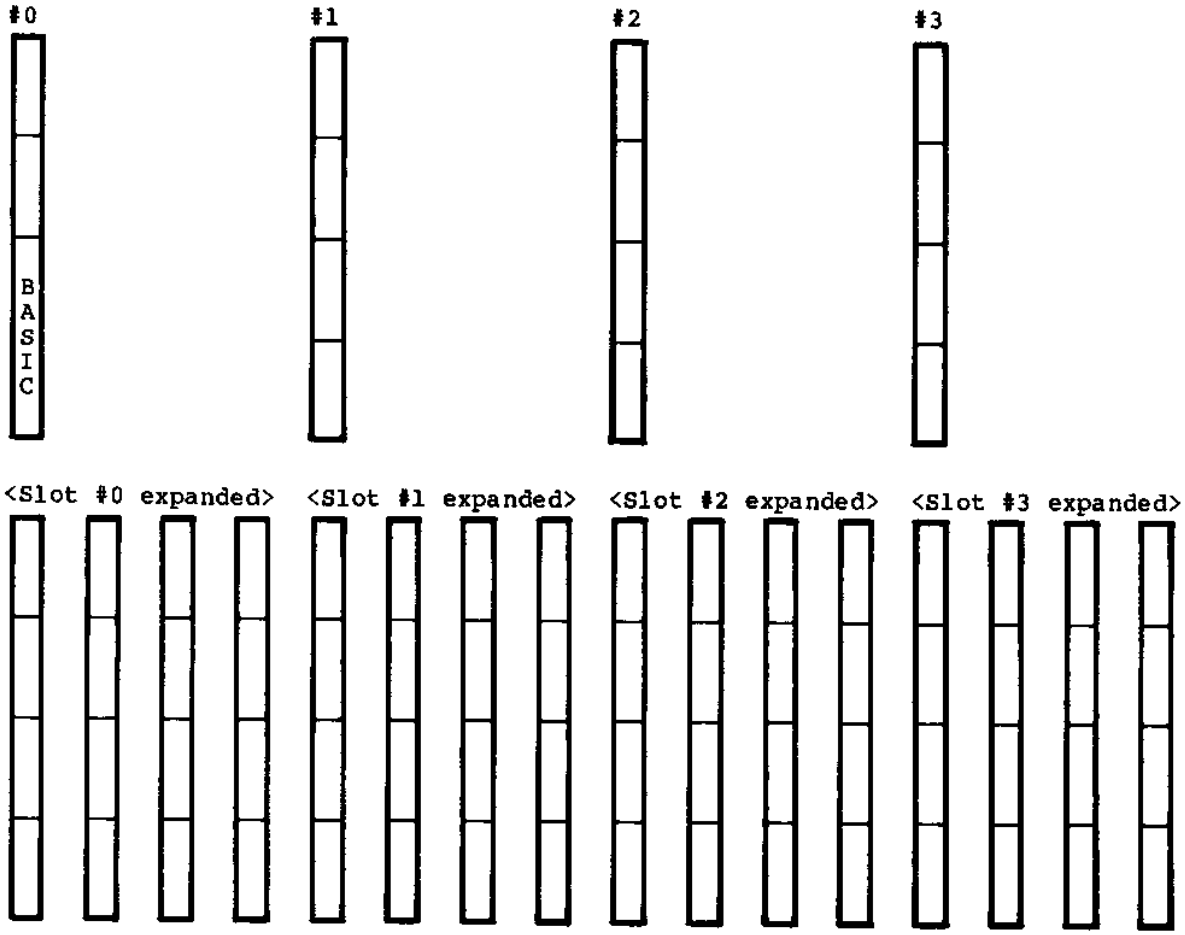
```
    ;  Name:
    ;  Location:        BINTRP
    ;  Purpose:
    ;
FF5C RMB(H.FINP,5)
    ;
    ;  Name:
    ;  Location:        BINTRP
    ;  Purpose:
    ;
FF61 RMB(H.TRMN,5)
    ;
    ;  Name:
    ;  Location:        BINTRP
    ;  Purpose:
    ;
FF66 RMB(H.FRME,5)
    ;
    ;  Name:
    ;  Location:        BINTRP
    ;  Purpose:
    ;
FF6B RMB(H.NTPL,5)
    ;
    ;  Name:
    ;  Location:        BINTRP
    ;  Purpose:
    ;
FF70 RMB(H.EVAL,5)
    ;
    ;  Name:
    ;  Location:        BINTRP
    ;  Purpose:
    ;
FF75 RMB(H.OKNO,5)
    ;
    ;  Name:
    ;  Location:        BINTRP
    ;  Purpose:
    ;
FF7A RMB(H.FING,5)
    ;
    ;  Name:            H.ISMI
    ;  Location:        BINTRP, at the ISMID$ (IS MID$) routine.
    ;  Purpose:
    ;
FF7F RMB(H.ISMI,5)
    ;
    ;  Name:            H.WIDT
    ;  Location:        BINTRP, at the WIDTHS (WIDTH) routine.
    ;  Purpose:
    ;
FF84 RMB(H.WIDT,5)
    ;
    ;  Name:            H.LIST
```

```
   ;  Location:        BINTRP, at the LIST routine.
   ;  Purpose:
   ;
FF89 RMB(H.LIST,5)
   ;
   ;  Name:            H.BUFL
   ;  Location:        BINTRP, at the BUFLIN (BUFfer LINe) routine.
   ;  Purpose:
   ;
FF8E RMB(H.BUFL,5)
   ;
   ;  Name:            H.FRQI
   ;  Location:        BINTRP, at the FRQINT routine.
   ;  Purpose:
   ;
FF93 RMB(H.FRQI,5)
   ;
   ;  Name:
   ;  Location:        BINTRP
   ;  Purpose:
   ;
FF98 RMB(H.SCNE,5)
   ;
   ;  Name:            H.FRET
   ;  Location:        BISTRS, at the FRETMP (FREe  up TeMPoraries)
   ;                   routine.
   ;  Purpose:
   ;
FF9D RMB(H.FRET,5)
   ;
   ;  Name:            H.PTRG
   ;  Location:        BIPTRG, at the PTRGET (PoinTeR GET) routine.
   ;  Purpose:         Uses other variable names than default.
   ;
FFA2 RMB(H.PTRG,5)
   ;
   ;  Name:            H.PHYD
   ;  Location:        MSXIO, at the PHYDIO (PHYsical Disk I/O).
   ;  Purpose:         Installs the disk driver.
   ;
FFA7 RMB(H.PHYD,5)
   ;
   ;  Name:            H.FORM
   ;  Location:        MSXIO, at the FORMAT (disk FORMATter)routine.
   ;  Purpose:         Installs the disk driver.
   ;
FFAC RMB(H.FORM,5)
   ;
   ;  Name:            H.ERRO
   ;  Location:        BINTRP, at the ERROR routine.
   ;  Purpose:         Traps errors from application programs.
   ;
FFB1 RMB(H.ERRO,5)
   ;
   ;  Name:            H.LPTO
```

```
   ;  Location:        MSXIO, at the LPTOUT (Line  PrinTer  OUTput)
   ;                   routine.
   ;  Purpose:         Uses a non-default printer.
   ;
FFB6  RMB(H.LPTO,5)
   ;
   ;  Name:            H.LPTS
   ;  Location:        MSXIO, at the  LPTSTT (Line  PrinTer STaTus)
   ;                   routine.
   ;  Purpose:         Uses a non-default printer.
   ;
FFBB  RMB(H.LPTS,5)
   ;
   ;  Name:            H.SCRE
   ;  Location:        MSXSTS, at the entry to SCREEN statement.
   ;  Purpose:         Expands the SCREEN statement.
   ;
FFC0  RMB(H.SCRE,5)
   ;
   ;  Name:            H.PLAY
   ;  Location:        MSXSTS, at the entry to PLAY statement.
   ;  Purpose:         Expands the PLAY statement.
   ;
FFC5  RMB(H.PLAY,5)
   ;
FFCA  RMB(ENDWRK,0)      ;End of work area.
```

2.2.3 Slot Control

[Memory structure of MSX]



Total: 1024K bytes (16*64K bytes)

Terminology:

Primary slot:	Slot enabled by the slot select register in the 8255 PPI.
Secondary slot:	Slot enabled by the expansion slot register at 0FFFFH.
Page:	Memory block (maximum 16K) in each slot. The slots are divided into four pages (0000H to 3FFFH, 4000H to 7FFFH, 8000H to 0BFFFH, and 0C000H to 0FFFFH).

o Minimum configuration

a) Microsoft MSX-BASIC interpreter at slot #0 from 0000H to 7FFFH.
b) Minimum of 8K RAM from 0E000H to 0FFFFH in any slot (including the secondary slot)

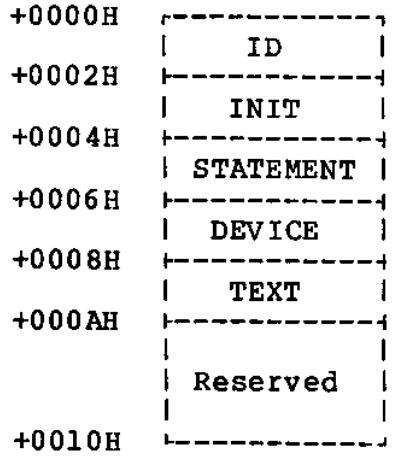
o RAM search procedure

MSX-BASIC first searches for available RAM from 0BFFFH down to 08000H (including the secondary slots), then enables the page with the largest available RAM. If there are more than one such pages, MSX-BASIC selects the leftmost page in the figure above. MSX-BASIC next searches for RAM from 0FFFFH down to 0C000H, and does the same procedure. Finally, MSX-BASIC searches for a continuous RAM block from 0FFFFH to 8000H and sets the system variable 'BOTTOM'.

o PROGRAM CARTRIGE search procedure

MSX-BASIC scans all slots (including secondary slots) from 4000H to 0BFFFH for a valid ID at the beginning of each page, collects information, and passes control to each page. The scan order is from left to right in the figure above. The format of the ID and other information are as follows.

Offset from top



- The ID is a two-byte code used to distinguish the ROM cartridges from the empty pages by using 'AB' (41H,42H).

- INIT holds the address of the initialization procedure specific to this cartridge. The default is 0 when no such procedure is necessary. Programs that need to interact with the BASIC interpreter should return control to it with a Z-80 'RET' instruction (all registers except [SP] may be destroyed). Note, however, that other programs (such as games) do not need to do this.

- STATEMENT holds an address of the expanded statement handler when contained in the cartridge; the address is 0 if no handler is contained. If BASIC encounters a 'CALL' statement, it calls this address, with the statement name in the system area. Note the following points. (In the notes below, the [HL] register pair is called a 'text pointer'.)

  1) The cartridge must be placed at 4000H to 7FFFH.

  2) The syntax for the expanded statement is as follows.

     CALL <statement_name> [(<arg> [,<arg>]..)]

     The keyword "CALL" can be replaced by an underscore (_).

  3) The statement name is stored in the system area, terminated by a 0. Since the buffer for statement name is of a fixed length (16 bytes), the statement name cannot be longer than 15 characters.

  4) If the handler for the statement is not contained within the cartridge, set the carry flag and return. Note that the text pointer must be returned unchanged.

  5) If the handler for that statement is contained within the cartridge, it should handle the specified function, update the text pointer to the end of the statement (Normally it would point to 0, indicating the end of the line, or to ': ' to indicate the end of the statement), and return with carry flag reset (all registers except [SP] may be destroyed). At the entry to the expanded statement handler, the text pointer should point to the first non-blank character after the statement name.

- DEVICE holds the address of the expanded device handler if it is contained in this cartridge. The default is 0 if no handler is contained. BASIC calls this address with the device name in the system area. Note the following points.

  1) The cartridge must be placed at 4000H to 7FFFH.

  2) The device name is stored in the system area terminated by 0. Since the length of the statement name buffer is fixed (16 bytes), the device name cannot be longer than 15 characters.

  3) Each cartridge (16K) can have up to 4 logical devices.

  4) When BASIC encounters an unidentifiable device name, it it calls the DEVICE entry with 0FFH in [Acc]. If the specified device handler is not contained within the cartridge, the carry flag should be set upon return. If the specified device handler is contained inside, the device ID (0 to 3) should be returned in [Acc], and the

carry should be reset. All registers may be destroyed.

5) Real I/O operations take place when a DEVICE entry is entered with one of the following values in [Acc].

0	Open
2	Close
4	Random I/O
6	Sequential output
8	Sequential input
10	LOC function
12	LOF function
14	EOF function
16	FPOS function
18	Back up a character

Device ID is passed in the system variable 'DEVICE'.

- TEXT holds the beginning address of the (tokenized) BASIC text contained in the cartridge. The default is 0 when no such text is inside. BASIC regards this as the beginning address of BASIC text, sets pointer there, and begins execution of the program. Note the following points.

1) When there is more than one such slot, only the leftmost one (in the figure above) is enabled and executed.

2) The cartridge must be placed at 8000H to 0BFFFH, thus the maximum length of BASIC text cannot exceed 16K bytes.

3) Even if there is a RAM block at 8000H to 0BFFFH, it cannot be used.

4) The address pointed to by the TEXT entry must contain a zero.

5) The line numbers (for statements which reference line numbers, such as GOTO and GOSUB) should be translated to pointers in advance because they are not converted to pointers during execution. Note that while they CAN be line numbers, the execution would be slower.

Note: INIT, STATEMENT, DEVICE and TEXT are placed with the low order byte first.

o How slot information is kept in the system area

EXPTBL - Indicates which slot is expanded.

```
EXPTBL: DS      1       ;for slot #0
        DS      1       ;for slot #1
        DS      1       ;for slot #2
        DS      1       ;for slot #3
```

Each entry in the EXPTBL holds 80H if expanded, 0 if not expanded.

SLTTBL - Indicates the value currently output to the expansion slot register. Valid only when corresponding EXPTBL holds 80H.

```
SLTTBL: DS      1       ;for slot #0
        DS      1       ;for slot #1
        DS      1       ;for slot #2
        DS      1       ;for slot #3
```

SLTATR - Holds attributes for each page.

```
SLTATR: DS      64
```

Each byte in the SLTATR table corresponds to each page. Bits are assigned as follows.

```
XXXXXXXX
||||||||
|||||||└-Unused
||||||└--Unused
|||||└---Unused
||||└----Unused
|||└-----Unused
||└------Statement expander inside
|└-------Device expander inside
└--------BASIC text inside
```

SLTWRK - Holds working storage for each page.

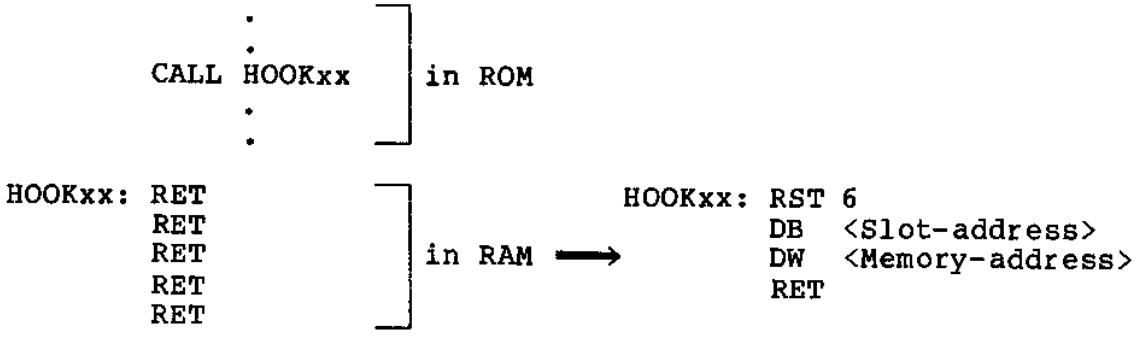
```
SLTWRK: DS      128
```

Each word in the SLTWRK table can be exclusively used by each page. The use of this work area depends entirely on the page.

o Usage of hooks

Hooks are one of the methods in which MSX-BASIC can be expanded. Some procedures (such as 'console input', 'console output') have a Z-80 'CALL' instruction directed to the common RAM area. The areas consist of a five-byte storage area per hook, and are initialized with five Z-80 'RET' instructions upon cold start. Expansion is done by redirecting this entry elsewhere.

Example:



RST 6 performs an inter-slot call to a different slot. Refer to BIOENT.MAC for further details of the interslot call facility.

To connect the hook to the desired routine, the routine must determine its location (slot). This is important because the routine's slot location is unpredictable. This is done by the following procedure.

```
RSLREG  EQU     138H
EXPTBL  EQU     0FCC1H
B8000   EQU     1               ;Set this true if the
                                ;program resides at
                                ;8000..0BFFFH
        CALL    RSLREG          ;Read primary slot #
        RRC                     ;Move it to bit 0,1
        RRC                     ;of [Acc]
IF      B8000
        RRC
        RRC
ENDIF
        ANI     11B
        MOV     C,A
        MVI     B,0
        LXI     H,EXPTBL        ;See if this slot is
        DAD     B               ;expanded or not
        ORA     M               ;Set MSB if so
        MOV     C,A
        INX     H               ;Point to SLTTBL entry
        INX     H
```

```
                INX     H
                INX     H
                MOV     A,M             ;Get what is currently
                                        ;output to expansion
                                        ;slot register
        IF      B8000
                RRC                     ;Move it to bit 2,3
                RRC                     ;of [Acc]
        ENDIF
                ANI     1100B
                ORA     C               ;Finally form slot
                RET                     ;address
```

< CAUTION >

A machine language program in a cartridge must be able to run in any slot (including secondary slots). The slot for running the cartridge is unpredictable.

o Usage of USR function

There are 10 USR functions, USR0 through USR9. USR0 can be abbreviated as USR. The address for a USR function jump is defined as follows.

```
DEFUSR0=&HE000  (This can be DEFUSR=&HE000)
DEFUSR3=&HE023
```

The USR functions can be invoked as follows.

```
A=USR0(12)      (This can also be A=USR(12))
PRINT USR("ABCD")+" This is a test"
```

The USR function parameters are passed to the machine language programs in the following manner.

Integer

When USR is called as an integer function, the address 0F663H contains 2, and its value is located at 0F7F8H and 0F7F9H, with the lower byte first.

String

When USR is called as a string function, the address 0F663H contains 3, and its string descriptor is located at 0F7F8H and 0F7F9H. String descriptors consist of three bytes, the first byte is the length of string, the second and third are the address of the string.

Single-precision

When USR is called as a single-precision function, the address 0F663H contains 4, and its value is located at 0F7F6H to 0F7F9H.

Double-precision

When USR is called as a double-precision function, the address 0F663H contains 8, and its value is located at 0F7F6H to 0F7FDH.

The value from a USR function can be returned to BASIC in the following manner.

Integer

The data at the address 0F663H should be set to 2. The value should be placed in 0F7F8H and 0F7F9H, with the lower byte first.

String

The data at the address 0F663H should be set to 3. The address of the string descriptor should be placed in 0F7F8H and 0F7F9H. String descriptors consist of three bytes, the first byte is set to the string length, the second and third bytes indicate the string address.

Single-precision

The data at the address 0F663H should be set to 4. The value should be placed in 0F7F6H through 0F7F9H.

Double-precision

The data at the address 0F663H should be set to 8. The value should be placed in 0F7F6H through 0F7FDH.

o How to allocate work area for cartridges

If the program is stand-alone (i.e., does not need to run with other programs in other cartridges), all RAM area below the fixed work area for BIOS (i.e., below 0F380H) is free. However, if the program must run with the BASIC interpreter and programs in other cartridges, the RAM usage is restricted.

There are three ways to allocate RAM to be used exclusively by each cartridge.

1) Put RAM on the cartridge. (Easiest and best)

2) If the work area is less than 3 bytes, use SLTWRK.

3) If the work area is greater than 2 bytes, make SLTWRK point to the system variable BOTTOM (0FC48H), then update it by the amount of memory required. BOTTOM is set by the initialization code to point to the bottom of the RAM.

Example:
Program is at 4000H to 7FFFH

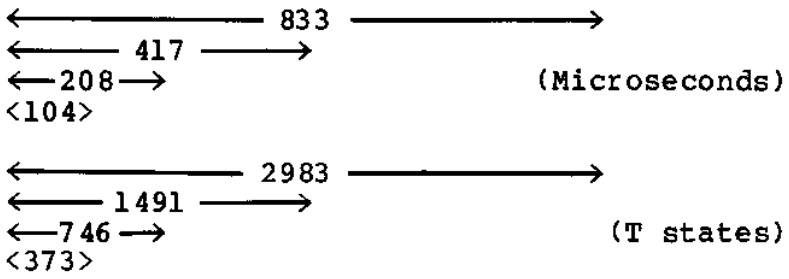
```
SIZE    EQU     ???             ;Size of memory required
RSLREG  EQU     138H
EXPTBL  EQU     0FCC1H
BOTTOM  EQU     0FC48H
;
        CALL    RSLREG          ;Read primary slot #
        RRC                     ;Move it to bit 0,1
        RRC                     ;of [Acc]
        ANI     00000011B
        MOV     C,A
        MVI     B,0
        LXI     H,EXPTBL        ;See if this slot is
        DAD     B               ;expanded or not
        ADD     A
        ADD     A
        ADD     A
        ADD     A
        MOV     C,A
        MOV     A,M
        ADD     A
        SBB     A               ;Form mask pattern
        ANI     00001100B
        INX     H               ;Point to SLTTBL entry
        INX     H
        INX     H
        INX     H
        ANA     M               ;Get what is currently
                                ;output to expansion
                                ;slot register
        ORA     C
```

```
          ORI     00000001B
;
;       Now, we have the sequence number for this
;       cartridge as follows.
;
;       00PPSSBB
;         ||||||
;         ||||└┴-- Higher 2 bits of memory address
;         ||└┴---- Secondary slot # (0..3)
;         └┴------ Primary slot # (0..3)
;
          ADD     A               ;Double since word table
          MOV     C,A
          MVI     B,0
          LXI     H,SLTWRK        ;Point to entry in
          DAD     B               ;SLTWRK table
          LBCD    BOTTOM          ;Get current RAM bottom
          MOV     M,C             ;Register this
          INX     H
          MOV     M,B
          LXI     H,SIZE
          DAD     B
          MOV     A,H             ;Beyond 0EFFFH?
          CPI     0F0H            ;Too much RAM required?
          JRNC    NOROOM          ;Yes, cannot allocate
          SHLD    BOTTOM
          RET
;
;       BOTTOM became greater than 0EFFFH, there is
;       no RAM left to be allocated.
;
NOROOM:                           ;Print messages or
                                  ;something like that
```

2.2.4 Cassette I/O Mechanism

o Physical Format

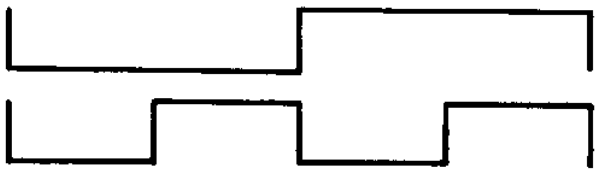
A. Pulse Width



[1200 baud]

0 - 1200 Hz

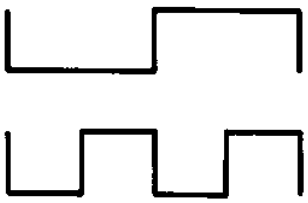
1 - 2400 Hz



[2400 baud]

0 - 2400 Hz

1 - 4800 Hz



Note that a pulse begins in the low state when it is being written.

B. Header

There are two kinds of headers; long headers and short headers. The long header is used for the file header, and the short header is used for the body of the file.

[1200 baud]

Long header	16000 x 2400 Hz
Short header	4000 x 2400 Hz

[2400 baud]

Long header	32000 x 4800 Hz
Short header	8000 x 4800 Hz

The baud rate is determined when reading the header.

C. Data

Data is composed of one '0' (Start bit) followed by an 8-bit data stream, and is followed by two '1's (Stop bits). The sequence of the data is from the least significant bit (LSB) to the most significant bit (MSB). When reading from cassette, the software measures the number of transitions during 3/4 of the baud rate. The result should be a 1 when reading a space, or 2 or 3 when reading a mark.

o Logical Format

There three file types (also called file attributes) supported in MSX-BASIC. These file types, or attributes, are: BASIC text files, ASCII text files, and machine language files.

A. BASIC Text File Format

[File header]

```
Long header
10 x 0D3H
File name (6 bytes)
```

[File body]

```
Short header
Tokenized BASIC text
7 x 00H
```

B. ASCII Text File Format

```
Long header
10 x 0EAH
File name (6 bytes)
```

[File body]

```
Short header
256 x data
Short header
256 x data
Short header
256 x data
    .
    .
Short header
256 x data (includes Control-Z)
```

C. Machine Language File Format

[File header]

Long header
10 x 0D0H
File name (6 bytes long)

[File body]

Short header
Load start address (1 word)
Load end address (1 word)
Execution start address (1 word)
Machine language program

o Related BIOS Entries

Name:	TAPION (00E1H)
Function:	Sets the cassette motor on and reads tape header
Entry:	None
Returns:	Carry flag is set if aborted
Modifies:	All

Name:	TAPIN (00E4H)
Function:	Reads data from tape
Entry:	None
Returns:	Data in [ACC], carry flag is set if aborted
Modifies:	All

Name:	TAPIOF (00E7H)
Function:	Stops reading from tape
Entry:	None
Returns:	None
Modifies:	None

Name:	TAPOON (00EAH)
Function:	Sets the motor on and writes the tape header block onto cassette
Entry:	[ACC] will contain a non-zero value if a long header is desired, zero if a short header is desired
Returns:	Carry flag is set if aborted
Modifies:	All

Name:	TAPOUT (00EDH)
Function:	Writes data to tape
Entry:	Data to be output in [ACC]
Returns:	Carry flag is set if aborted
Modifies:	All

Name:	TAPOOF (00F0H)
Function:	Stops writing to tape
Entry:	None
Returns:	None
Modifies:	None

[NOTES]

All of the above routines must be entered with the interrupts disabled.

Because the above pulses are software-generated, all of the above routines must be called using the same time intervals as when using BASIC.

2.2.5 MSX Printer Specifications

This document summarizes the requirements for the dot matrix printers connected to MSX computers.

A. Character Set

The MSX printer should have the same character set that the MSX computer has. This is a character set with codes of 00 to FE. The graphics characters (codes between 00H and 1FH) are represented by two-byte code sequence, preceded by 01H, then followed by the code itself, added to an offset of 40H.

Example:

To print a character with the code 02H, first send 01H, the graphic header, then send 42H, the sum of the code (02H) and the offset (40H).

This rule is the same as when sending characters to the screen.

B. Control Codes

The MSX printer supports codes in the format of the NEC PC-8023 printer. The minimum requirements for the MSX printer are as follows:

0A - Line feed
0C - Form feed (Recommended page length: 66 lines/page)
0D - Carriage return

ESC+"A" - 1/6" line spacing for 8-pin printers, or place a space between lines.
ESC+"B" - 1/8' line spacing for 8-pin printers, or place no space between lines.
ESC+"Snnnn" - Dot image print. <nnnn> represents the number of to follow, in ASCII decimal characters.

If the printer has a line buffer, the following control character initiates printing of the contents of the line buffer.

0D - Carriage return / Print contents of buffer

C. Non-MSX Printers

MSX-BASIC has a switch in the 5th parameter of the SCREEN statement. When this is set to 1, MSX-BASIC assumes that the printer connected to the system has no such capabilities as described above. In this mode, MSX-BASIC converts those characters with codes between 00 to 1FH to blanks. The default value of this switch is 00, meaning that the MSX printer is connected.

D. Control Functions for the PC-8023 Printer

Control Code	(Hex)	Function
8	8	Back space
9	9	Horizontal TAB
10	A	Line feed
11	B	Vertical TAB
12	C	Form feed
13	D	Carriage return
14	E	Double width
15	F	Normal width
27	1B	Escape character
29	1D	Vertical form control setting start
30	1E	Vertical form control setting end
31	1F	+chr$(16+n) 1<=n<=15 N lines feed
		+chr$(n) 2<=n<=6 Vertical tab channel select

ESC + Control Code	Function
1~6	Dot spacing
!	Enhanced print
"	Cancel enhanced mode
&	Alphanumerics/Hiragana
$	Alphanumerics/Katakana
A	1/6" feed
B	1/8" feed
T+"nn"	n/144" feed
N	Normal spacing (10 CPI)
P	Proportional spacing (20 CPI)
	Double density dot spacing in graphic print
E	Elite spacing (12 CPI)
Q	Condensed spacing, 136 characters/line
L+"nnn"	Set left margin
S+"nnnn"	Bit image print (nnnn:number of dots follow)
X	Start under line
Y	End under line
r	Reverse feed
f	Forward feed
[	Incremental printing.BS erases last character sent
]	Logical seeking bidirectional print. A chr$(24) cancels the line sent.
(+"nnn",,,.	Set horizontal tab
)+"nnn",,,.	Clear horizontal tab (specified position only)
2	Clear all the horizontal tab position

PART C

EXPANDED MSX SYSTEM SOFTWARE

3. MSX-DOS

MSX-DOS is a disk operating system for MSX computers. The system with its compatibility to other versions of MS-DOS will surely provide you a comfortable environment around. All Microsoft languages (BASIC Interpreter, BASIC Compiler, FORTRAN, COBOL, Pascal) will be available under MSX-DOS. Users of MSX-DOS are assured that their operating system will be the first that Microsoft will support when any new products or major releases are announced.

3.1 MSX-DOS User's Guide

3.1.1 System Requirements

The MSX-DOS operating system requires a MSX microcomputer system with 64k bytes of memory (RAM) and at least one disk drive.

The MSX-DOS disk contains the following files:

File Name	Function of File
COMMAND.COM	MSX-DOS command processor
MSXDOS.SYS	MSX-DOS operating system

3.1.2 Getting Started

Once MSX-DOS has been loaded, the system searches the MSX-DOS disk for the COMMAND.COM file and loads it into memory. The COMMAND.COM file is a program that processes the commands you enter and then runs the appropriate programs. It is also called the command processor.

When the command processor is loaded, you will see the following display on your screen (the underscore represents the cursor):

```
MSX-DOS Version 1.00
Copyright 1984 by Microsoft

Command version 1.00

Current date is Sun  1-01-1984
Enter new date: _
```

NOTE

The date format (mm-dd-yy) may be changed depending on versions. For example, it is "yy-mm-dd" in Japanese version.

Any date is acceptable in answer to the new date prompt as long as it follows the above format. Separators between the numbers can be hyphens (-) or slashes (/).

After you have answered the new time prompt, the MSX-DOS

```
A>_
```

will be displayed.
It tells you that MSX-DOS is ready to accept commands. If you have inserted the MSX-DOS disk into a drive other than A, the command processor prompt will reflect that drive (for example, B>). However, usually you will load MSX-DOS in drive A.

The A in the previous prompt represents the default disk drive. This means that MSX-DOS will search only the disk in drive A for any filenames you may enter and will write files to that disk unless you specify a different drive. You can ask MSX-DOS to search the disk in drive B by changing the drive designation or by specifying B: in a command. To change the disk drive designation, enter the new drive letter followed by a colon. For example:

```
A>      (MSX-DOS prompt)
A>B:    (you have typed B: in response to
        the prompt)
B>      (system responds with B> and drive B
        is now the default drive)
```

The system prompt B> will appear and MSX-DOS will search only the disk in drive B until you specify a different default drive.

If you have only one disk drive attached to your computer, turn to 3.1.14 'Instructions for Users with Single-Drive Systems', for important information.

A filename can be from 1 to 8 characters long. The filename extension can be three or fewer characters. You can type any filename in small or capital letters and MSX-DOS will translate these letters into uppercase characters.

In addition to the filename and the filename extension, the name of your file may include a drive designation. A drive designation tells MSX-DOS to look on the disk in the designated drive to find the filename typed.

The following characters are allowed for file names and their extensions.

```
A-Z   0-9   $   &   #

%   '   (   )   -   @

¥   ^   {   }   ~   `   !
```

(A backslash instead of Yen sign in international versions.)

The term file specification (or filespec) will be used in this book to indicate the following filename format:

[<drive designation:>]<filename>[<.filename extension>]

3.1.3 Wild Cards

Two special characters (called wild cards) can be used in filenames and extensions: the asterisk (*) and the question mark (?). These special characters give you greater flexibility when using filenames in MSX-DOS commands.

o The ? Wild Card

A question mark (?) in a filename or filename extension indicates that any character can occupy that position. For example, the MSX-DOS command

```
DIR TEST?RUN.COM
```

will list all directory entries on the default drive that have 8 characters, begin with TEST, have any next character, end with the letters RUN, and have a filename extension of .COM.

o The * Wild Card

An asterisk (*) in a filename or filename extension indicates that any character can occupy that position or any of the remaining positions in the filename or extension.
For example:

```
DIR TEST*.COM
```

will list all directory entries on the default drive with filenames that begin with the characters TEST and have an extension of .COM.

The wild card designation *.* refers to all files on the disk. Note that this can be very powerful and destructive when used in MSX-DOS commands. For example, the command DEL *.* deletes all files on the default drive, regardless of filename or extension.

3.1.4 Illegal File Names

MSX-DOS treats some device names specially, and certain 3-letter names are reserved for the names of these devices. These 3-letter names cannot be used as filenames or extensions. You must not name your files any of the following:

AUX — Used when referring to input from or output to an auxiliary device (such as a printer or disk drive).

CON — Used when referring to keyboard input or to output to the terminal console (screen).

LST or PRN — Used when referring to the printer device.

NUL — Used when you do not want to create a particular file, but the command requires an input or output filename.

Even if you add device designations or filename extensions to these filenames, they remain associated with the devices listed above. For example, A:CON.XXX still refers to the console and is not the name of a disk file.

3.1.5 Directories

The directory also contains information on the size of the files, their locations on the disk, and the dates that they were created and updated.

3.1.6 Types of MSX-DOS Commands

There are two types of MSX-DOS commands:

Internal commands

External commands

Internal commands are the simplest, most commonly used commands. You cannot see these commands when you do a directory listing on your MSX-DOS disk; they are part of the command processor. When you type these commands, they execute immediately. The following internal commands are described in 3.2.

BASIC	DIR	REM
COPY	FORMAT	REN (RENAME)
DATE	MODE	TIME
DEL (ERASE)	PAUSE	TYPE
		VERIFY

External commands reside on disks as program files. They must be read from disk before they can execute. If the disk containing the command is not in the drive, MSX-DOS will not be able to find and execute the command.

Any filename with a filename extension of .COM or .BAT is considered an external command. For example, programs such as FILCON.COM and COMP.COM are external commands. Because all external commands reside on disk, you can create commands and add them to the system. Programs that you create with most languages (including assembly language) will be .COM (executable) files.

When you enter an external command, do not include its filename extension.

3.1.7 Command Options

Options can be included in your MSX-DOS commands to specify additional information to the system. If you do not include some options, MSX-DOS provides a default value.

The following is the format of all MSX-DOS commands:

```
Command [options...]
```

where:

switches	Switches are options that control MSX-DOS commands. They are preceded by a slash (for example, /P).
arguments	Provide more information to MSX-DOS commands. You usually choose between arguments; for example, ON or OFF.
filespec	Refers to an optional drive designation, a filename, and an optional three letter filename extension in the following format: [<d:>]<filename>[<.ext>]
d:	Refers to a disk drive designation.
filename	Refers to any valid name for a disk file, including an optional filename extension. The filename option does not refer to a device or to a disk drive designation.
.ext	Refers to an optional filename extension consisting of a period and 1-3 characters. When used, filename extensions immediately follow filenames.

3.1.8 Information Common to All MSX-DOS Commands

The following information applies to all MSX-DOS commands:

- Commands are usually followed by one or more options.
- Commands and options may be entered in uppercase or lowercase, or a combination of keys.
- Commands and options must be separated by delimiters. Because they are easiest, you will usually use the space and comma as delimiters. For example:

```
DEL MYFILE.OLD NEWFILE.TXT
RENAME,THISFILE THATFILE
```

  You can also use the semicolon (;), the equal sign (=), or the tab key as delimiters in MSX-DOS commands.
- Do not separate a file specification with delimiters, since the colon and the period already serve as delimiters.
- When instructions say "Strike a key when ready", you can press any key except <CONTROL-C>.
- You must include the filename extension when referring to a file that already has a filename extension.
- You can abort commands when they are running by pressing <CONTROL-C>.
- Commands take effect only after you have pressed the <RETURN> key.
- Wild cards (global filename characters) and device names (for example, PRN or CON) are not allowed in the names of any commands.
- When commands produce a large amount of output on the screen, the display will automatically scroll to the next screen. You can press <CONTROL-S> to suspend the display. Press any key to resume the display on the screen.
- MSX-DOS editing and function keys can be used when entering commands. Refer to 3.1.13 MSX-DOS Editing and Function Keys, for a complete description of these keys.

- The prompt from the command processor is the default drive designation plus a right angle bracket (>); for example, A>.

- Disk drives will be referred to as source drives and destination drives. A source drive is the drive you will be transferring information from. A destination drive is the drive you will be transferring information to.

3.1.9 Batch Processing

With MSX-DOS, you can put the command sequence into a special file called a batch file, and execute the entire sequence simply by typing the name of the batch file. "Batches" of your commands in such files are processed as if they were typed at a terminal. Each batch file must be named with the .BAT extension, and is executed by typing the filename without its extension.

Two MSX-DOS commands are available for use expressly in batch files: REM and PAUSE. REM permits you to include remarks and comments in your batch files without these remarks being executed as commands. PAUSE prompts you with an optional message and permits you to either continue or abort the batch process at a given point.

The following list contains information that you should read before you execute a batch process with MSX-DOS:

- Do not enter the filename BATCH (unless the name of the file you want to execute is BATCH.BAT).
- Only the filename should be entered to execute the batch file. Do not enter the filename extension.
- The commands in the file named <filename>.BAT are executed.
- If you press <CONTROL-C> while in batch mode, this prompt appears:

  Terminate batch job (Y/N)?

  If you press Y, the remainder of the commands in the batch file are ignored and the system prompt appears.

  If you press N, only the current command ends and batch processing continues with the next command in the file.
- If you remove the disk containing a batch file being executed, MSX-DOS prompts you to insert it again before the next command can be read.
- The last command in a batch file may be the name of another batch file. This allows you to call one batch file from another when the first is finished.

3.1.10 The AUTOEXEC.BAT File

When you start MSX-DOS, the command processor searches the MSX-DOS disk for a file named AUTOEXEC.BAT. The AUTOEXEC.BAT file is a batch file that is automatically executed each time you start the system.

If MSX-DOS finds the AUTOEXEC.BAT file, the file is immediately executed by the command processor and the date prompts are bypassed.

If MSX-DOS does not find an AUTOEXEC.BAT file when you first load the MSX-DOS disk, then the date and time prompts will be issued.

3.1.11 How To Create a Batch File

If, for example, you wanted to automatically load BASIC and run a program called MENU each time you started MSX-DOS, you could create an AUTOEXEC.BAT file as follows:

1. Type:

   COPY CON: AUTOEXEC.BAT

   This statement tells MSX-DOS to copy the information from the console (keyboard) into the AUTOEXEC.BAT file. Note that the AUTOEXEC.BAT file must be created in the root directory of your MSX-DOS disk.

2. Now type:

   BASIC MENU

   This statement goes into the AUTOEXEC.BAT file. It tells MSX-DOS to load BASIC and run the MENU program whenever MSX-DOS is started.

3. Press the <CONTROL-Z> key; then press the <RETURN> key to put the command BASIC MENU in the AUTOEXEC.BAT file.

4. The MENU program will now run automatically whenever you start MSX-DOS.

To run your own BASIC program, enter the name of your program in place of MENU in the second line of the example. You can enter any MSX-DOS command or series of commands in the AUTOEXEC.BAT file.

NOTE

Remember that if you use an AUTOEXEC.BAT file, MSX-DOS will not prompt you for a current date unless you include the DATE command in the AUTOEXEC.BAT file. It is strongly recommended that you include this command in your AUTOEXEC.BAT file, since MSX-DOS uses this information to keep your directory current.

3.1.12 Replaceable Parameters in .BAT Files.

There may be times when you want to create an application program and run it with different sets of data. These data may be stored in various MSX-DOS files.

When used in MSX-DOS commands, a parameter is an option that you define. With MSX-DOS, you can create a batch (.BAT) file with dummy (replaceable) parameters. These parameters, named %0-%9, can be replaced by values supplied when the batch file executes.

For example, when you type the command line COPY CON MYFILE.BAT, the next lines you type are copied from the console to a file named MYFILE.BAT on the default drive:

```
A>COPY CON MYFILE.BAT
COPY %1.MAC %2.MAC
TYPE %2.PRN
TYPE %0.BAT
```

Now, press <CONTROL-Z> and then press <RETURN>. MSX-DOS responds with this message:

```
1 File(s) copied
A>_
```

The file MYFILE.BAT, which consists of three commands, now resides on the disk in the default drive.

The dummy parameters %1 and %2 are replaced sequentially by the parameters you supply when you execute the file. The dummy parameter %0 is always replaced by the drive designator, if specified, and the filename of the batch file (for example, MYFILE).

NOTES:

1. Up to 10 dummy parameters (%0-%9) can be specified.
2. If you use the percent sign as part of a filename within a batch file, you must type it twice. For example, to specify the file ABC%.COM, you must type it as ABC%%.COM in the batch file.

To execute the batch file MYFILE.BAT and to specify the parameters that will replace the dummy parameters, you must enter the batch filename (without its extension) followed by the parameters you want MSX-DOS to substitute for %1, %2, etc.

Remember that the file MYFILE.BAT consists of 3 lines:

```
COPY %1.MAC %2.MAC
TYPE %2.PRN
TYPE %0.BAT
```

To execute the MYFILE batch process, type:

```
MYFILE A:PROG1 B:PROG2
```

MYFILE is substituted for %0, A:PROG1 for %1, and B:PROG2 for %2.

The result is the same as if you had typed each of the commands in MYFILE with their parameters, as follows:

```
COPY A:PROG1.MAC B:PROG2.MAC
TYPE B:PROG2.PRN
TYPE MYFILE.BAT
```

The following table illustrates how MSX-DOS replaces each of the above parameters:

BATCH FILENAME	PARAMETER1 (%0) (MYFILE)	PARAMETER2 (%1) (PROG1)	PARAMETER3 (%2) (PROG2)
MYFILE	MYFILE.BAT	PROG1.MAC	PROG2.MAC PROG2.PRN

Remember that the dummy parameter %0 is always replaced by the drive designator (if specified) and the filename of the batch file.

3.1.13 MSX-DOS Editing and Function Keys

Special MSX-DOS Editing Keys

Control Character Functions

3.1.13.1 Special MSX-DOS Editing Keys

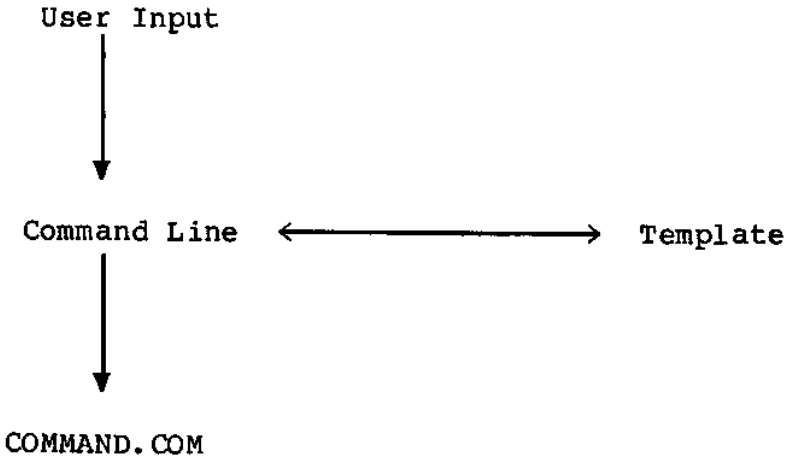
The special editing keys deserve particular emphasis because they depart from the way in which most operating systems handle command input. You do not have to type the same sequences of keys repeatedly, because the last command line is automatically placed in a special storage area called the template.

By using the template and the special editing keys, you can take advantage of the following MSX-DOS features:

- A command line can be instantly repeated by pressing two keys.
- If you make a mistake in the command line, you can edit it and retry without having to retype the entire command line.
- A command line that is similar to a preceding command line can be edited and executed with a minimum of typing by pressing special editing keys.

When you type a line to the system call 0AH (buffered line input) and press the RETURN key, the line is returned to the caller of the system call. This line is copied to the new template. You can now recall the last line or modify it with MSX-DOS special editing keys.

The relationship between the command line and the template is shown in the next figure.



Command Line and Template

NAME	KEY	FUNCTION
COPY1	RIGHT ARROW ^¥ (*)	Copies one character from the template to the new line.
COPYUP	SELECT ^X	Copies all characters from the template to the new line, up to the character specified.
COPYALL	DOWN ARROW ^_	Copies all remaining characters in the template to the new line.
SKIP1	DEL	Skips over (does not copy) a character in the template.
SKIPUP	CLS ^L	Skips over (does not copy) the characters in the template, up to the character specified.
VOID	UP ARROW ESCAPE ^^ ^U ^[	Voids the current input. Leaves the template unchanged.
BS	LEFT ARROW BS ^H ^]	Deletes the last character typed.
INSERT	insert ^R	Enters/exits insert mode.
NEWLINE	home ^K	Makes the current line the new template.

* Japanese. ^\ in all other versions.

Example:

If you type the following command

```
DIR PROG.COM
```

MSX-DOS displays information about the file PROG.COM on your screen. The command line is also saved in the template. To repeat the command, just press two keys: <COPYALL> and <RETURN>.

The repeated command is displayed on the screen as you type, as shown below:

```
<COPYALL>DIR PROG.COM<RETURN>
```

Notice that pressing the <COPYALL> key causes the contents of the template to be copied to the command line; pressing <RETURN> causes the command line to be sent to the command processor for execution.

If you want to display information about a file named PROG.ASM, you can use the contents of the template and type:

```
<COPYUP>C
```

Typing <COPYUP>C copies all characters from the template to the command line, up to but not including "C". MSX-DOS displays:

```
DIR PROG._
```

Note that the underline is your cursor. Now type:

```
.ASM
```

The result is:

```
DIR PROG.ASM_
```

The command line "DIR PROG.ASM" is now in the template and ready to be sent to the command processor for execution. To do this, press <RETURN>.

Now assume that you want to execute the following command:

```
TYPE PROG.ASM
```

To do this, type:

```
TYPE<INSERT> <COPYALL><RETURN>
```

Notice that when you are typing, the characters are entered directly into the command line and overwrite corresponding characters in the template. This automatic replacement feature is turned off when you press the insert key. Thus, the characters "TYPE" replace the characters "DIR " in the template. To insert

a space between "TYPE" and "PROG.ASM", you press <INSERT> and then the space bar. Finally, to copy the rest of the template to the command line, you press <COPYALL> and then <RETURN>. The command "TYPE PROG.ASM" will be processed by MSX-DOS, and the template becomes "TYPE PROG.ASM".

If you had misspelled "TYPE" as "BYTE", a command error would have occurred. Still, instead of throwing away the whole command, you could save the misspelled line before you press <RETURN> by creating a new template with the <NEWLINE> key:

```
BYTE PROG.ASM<NEWLINE>
```

You could then edit this erroneous command by typing:

```
T<COPY1>P<COPYALL>
```

The <COPY1> key copies a single character from the template to the command line. The resulting command line is then the command that you want:

```
TYPE PROG.ASM
```

As an alternative, you can use the same template containing BYTE PROG.ASM and then use the <SKIP1> and <INSERT> keys to achieve the same result:

```
<SKIP1><SKIP1><COPY1><INSERT>YP<COPYALL>
```

To illustrate how the command line is affected as you type, examine the keys typed on the left; their effect on the command line is shown on the right:

<SKIP1>	_	Skips over 1st template character
<SKIP1>	_	Skips over 2nd template character
<COPY1>	T	Copies 3rd template character
<INSERT>YP	TYP	Inserts two characters
<COPYALL>	TYPE PROG.ASM	Copies rest of template

Notice that <SKIP1> does not affect the command line. It affects the template by deleting the first character. Similarly, <SKIPUP> deletes characters in the template, up to but not including a given character.

These special editing keys can add to your effectiveness at the keyboard. The next section describes control character functions that can also help when you are typing commands.

3.1.13.2 Control Character Functions

A control character function is a function that affects the command line. You have already learned about <CONTROL-C> and <CONTROL-S>. Other control character functions are described below.

Remember that when you type a control character, such as <CONTROL-C>, you must hold down the control key and then press the "C" key.

Table of Control Character Functions

Control Character	Function
<CONTROL-N>	Cancels echoing of output to line printer.
<CONTROL-C>	Aborts current command.
<CONTROL-H>	Removes last character from command line, and erases character from terminal screen.
<CONTROL-J>	Inserts physical end-of-line, but does not empty command line. Use the <LINE FEED> key to extend the current logical line beyond the physical limits of one terminal screen.
<CONTROL-P>	Echoes terminal output to the line printer.
<CONTROL-S>	Suspends display of output to terminal screen. Press any key to resume.

3.1.14 Instructions for Users with Single-drive Systems

On a single-drive system, you enter the commands as you would on a multi-drive system.

You should think of the single-drive system as having two drives (drive A and drive B). But instead of A and B representing two physical drives as on the multi-drive system, the A and B represent disks.

If you specify drive B when the "drive A disk" was last used, you are prompted to insert the disk for drive B. For example:

```
A> COPY COMMAND.COM B:
Insert diskette for drive B:
and strike a key when ready
  1 File(s) copied
A>_
```

If you specify drive A when the "drive B disk" was last used, you are prompted again to change disks. This time, MSX-DOS prompts you to insert the "drive A disk."

The same procedure is used if a command is executed from a batch file. MSX-DOS waits for you to insert the appropriate disk and to press any key before it continues. You will be prompted to do this.

NOTE

The letter displayed in the system prompt represents the default drive where MSX-DOS looks to find a file whose name is entered without a drive specifier. The letter in the system prompt does not represent the last disk used.

For example, assume that A is the default drive. If the last operation performed was DIR B:, MSX-DOS believes the "drive B disk" is still in the drive. However, the system prompt is still A>, because A is still the default drive. If you type DIR, MSX-DOS prompts you for the "drive A disk" because drive A is the default drive, and you did not specify another drive in the DIR command.

3.1.15 Disk Errors

If a disk error occurs at any time during a command or program, MSX-DOS retries the operation three times. If the operation cannot be completed successfully, MSX-DOS returns an error message in the following format:

```
<yyy> error <I/O action> drive x
Abort, Retry, Ignore? _
```

In this message,<yyy> may be one of the following:

```
Write protect
Not ready
Disk
```

The <I/O-action> may be either of the following:

```
reading
writing
```

The drive <x> indicates the drive in which the error has occurred.

MSX-DOS waits for you to enter one of the following responses:

A	Abort. Terminate the program requesting the disk read or write.
I	Ignore. Ignore the bad sector and pretend the error did not occur.
R	Retry. Repeat the operation. This response is to be used when the operator has corrected the error.

Usually, you will want to attempt recovery by entering responses in this order:

R	(to try again)
A	(to terminate program and try a new disk)

One other error message might be related to faulty disk read or write:

```
Bad FAT
```

This message means that the copy in memory of one of the allocation tables has pointers to nonexistent blocks. Possibly the disk was incorrectly formatted or not formatted before use. If this error persists, the disk is currently unusable and must be formatted prior to use.

3.2 MSX-DOS Command Guide

NOTE

Users of single-drive systems should refer to 3.1.14 for the additional procedures required when executing many of the following commands.

The following MSX-DOS commands are described here. Note that synonyms for commands are enclosed in parentheses.

BASIC	Goto MSX-BASIC
COPY	Copies file(s) specified
DATE	Displays and sets date
DEL	Deletes file(s) specified (ERASE)
DIR	Lists requested directory entries
FORMAT	Formats a disk to receive MSX-DOS file
MODE	Sets display screen mode
PAUSE	Pauses for input in a batch file
REM	Displays a comment in a batch file
REN	Renames first file as second file (RENAME)
TIME	Displays and sets time
TYPE	Displays the contents of file specified
VERIFY	Sets/Resets verify mode

BASIC

SYNTAX: BASIC [<filespec>]

PURPOSE: Boots MSX-BASIC

COMMENTS: This command boots the MSX Disk BASIC from the MSX-DOS.

If a BASIC program file is designated by the <filespec>, the program is automatically loaded and run after BASIC starts.

This command changes the slot to make the BASIC ROM effective. So the memory map is different between the MSX-DOS and MSX-Disk-BASIC.

Use "CALL SYSTEM" statement to return to the MSX-DOS from the BASIC.

COPY

SYNTAX: COPY <filespec> [<filespec>]

PURPOSE: Copies one or more files to another disk. If you prefer, you can give the copies different names. This command can also copy files on the same disk.

COMMENTS: If the second filespec option is not given, the copy will be on the default drive and will have the same name as the original file (first filespec option). If the first filespec is on the default drive and the second filespec is not specified, the COPY will be aborted. (Copying files to themselves is not allowed.) MSX-DOS will return the error message:

```
File cannot be copied onto itself
0 files copied
```

The second option may take three forms:

1. If the second option is a drive designation (d:) only, the original file is copied with the original filename to the designated drive.

2. If the second option is a filename only, the original file is copied to a file on the default drive with the filename specified.

3. If the second option is a full filespec, the original file is copied to a file on the default drive with the filename specified.

The COPY command also allows file concatenation (joining) while copying. Concatenation is accomplished by simply listing any number of files as options to COPY, separated by "+".

For example,

```
COPY  A.XYZ + B.COM + B:C.TXT BIGFILE.CRP
```

This command concatenates files named A.XYZ, B.COM, and B:C.TXT and places them in the file on the default drive called BIGFILE.CRP.

To combine several files using wild cards into one file, you could type:

```
COPY *.LST COMBIN.PRN
```

This command would take all files with a filename extension of .LST and combine them into a file named COMBIN.PRN.

In the following example, for each file found matching *.LST, that file is combined with the corresponding .REF file. The result is a file with the same filename but with the extension .PRN. Thus, FILE1.LST will be combined with FILE1.REF to form FILE1.PRN; then XYZ.LST with XYZ.REF to form XYZ.PRN; and so on.

```
COPY *.LST + *.REF *.PRN
```

The following COPY command combines all files matching *.LST, then all files matching *.REF, into one file named COMBIN.PRN:

```
COPY *.LST + *.REF COMBIN.PRN
```

Do not enter a concatenation COPY command where one of the source filenames has the same extension as the destination. For example, the following command is an error if ALL.LST already exists:

```
COPY  *.LST  ALL.LST
```

The error would not be detected, however, until ALL.LST is appended. At this point it could have already been destroyed.

COPY compares the filename of the input file with the filename of the destination. If they are the same, that one input file is skipped, and the error message "Content of destination lost before copy" is printed. Further concatenation proceeds normally. This allows "summing" files, as in this example:

```
COPY  ALL.LST + *.LST
```

This command appends all *.LST files, except ALL.LST itself, to ALL.LST. This command will not produce an error message and is the correct way to append files using the COPY command.

Because ASCII files are usually concatenated, this command interprets a CTRL+Z (1AH) as a end of file mark in a file. So there is a need of a "/B" switch to use a physical end of file (length of file displayed by the DIR command), when binary files shall be concatenated.

```
COPY/B A.COM+B.COM
```

In this example, the B.COM is appended after the A.COM, and the destination file name is still A.COM.

Any files can be concatenated by using "/B" switch for binary file and "/A" for ASCII file. A switch is effective for the switched file and the after until a other switch appears.

Whether a CTRL+Z is appended at the end of the destination file or not is decided by a switch of the destination file. There is no CTRL+Z in the source file which is read in effect of "/A". Only one CTRL+Z is written when a file is written in effect of "/A". Therefore more CTRL+Z are appended as follows.

```
COPY A.ASM/B B.ASM/A
```

In this example, "/B" avoids removing CTRL+Z and "/A" appends a CTRL+Z.

When there is no concatenation, "/A" and "/B" switchs are valid, and the default file type is binary. "/A" switch terminates the copy at the first CTRL+Z.

DATE

SYNTAX: DATE [<mm>-<dd>-<yy>]

PURPOSE: Enter or change the date known to the system. This date will be recorded in the directory for any files you create or alter.

You can change the date from your terminal or from a batch file. (MSX-DOS does not display a prompt for the date if you use an AUTOEXEC.BAT file, so you may want to include a DATE command in that file.)

COMMENTS: If you type DATE, DATE will respond with the message:

```
Current date is <day>-<mm>-<dd>-<yy>
Enter new date:_
```

Press <RETURN> if you do not want to change the date shown.

You can also type a particular date after the DATE command, as in:

```
DATE 3-9-81
```

In this case, you do not have to answer the "Enter new date:" prompt.

The new date must be entered using numerals only; letters are not permitted. The allowed options are:

```
<mm> = 1-12
<dd> = 1-31
<yy> = 0-79, 80-99 or 1980-2099
```

The date, month, and year entries may be separated by hyphens (-), slashes (/) or periods (.). MSX-DOS is programmed to change months and years correctly, whether the month has 31, 30, 29, or 28 days. MSX-DOS handles leap years, too.

<yy> is a two-digit number from 80-99 (the 19 is assumed), or a two-digit number from 00-79 (the 20 is assumed), or a four-digit number from 1980-2099 (representing year.)

If the options or separators are not valid, DATE displays the message:

```
Invalid date
Enter new date:_
```

DATE then waits for you to enter a valid date.

NOTE

The date format (mm-dd-yy) may be changed depending on versions. For example, it is "yy-mm-dd" in Japanese version.

DEL

SYNONYM: DELETE
ERASE

SYNTAX: DEL [filespec]

PURPOSE: Deletes all files with the designated filespec.

COMMENTS: If the filespec is *.*, the prompt "Are you sure?" appears. If a "Y" or "y" or <RETURN> is typed as a response, then all files are deleted as requested. You can also type ERASE for the DELETE command.

DIR

SYNTAX: DIR [filespec] [/P][/W]

PURPOSE: Lists the files in a directory.

COMMENTS: If you just type DIR, all directory entries on the default drive are listed. If only the drive specification is given (DIR d:), all entries on the disk in the specified drive are listed. If only a filename is entered with no extension (DIR filename), then all files with the designated filename on the disk in the default drive are listed. If you designate a file specification (for example, DIR d:filename.ext), all files with the filename specified on the disk in the drive specified are listed. In all cases, files are listed with their size in bytes and with the time and date of their last modification.

The wild card characters ? and * (question mark and asterisk) may be used in the filename option. Note that for your convenience the following DIR commands are equivalent:

COMMAND	EQUIVALENT
DIR	DIR *.*
DIR FILENAME	DIR FILENAME.*
DIR .EXT	DIR *.EXT
DIR .	DIR *.

Two switches may be specified with DIR. The /P switch selects Page Mode. With /P, display of the directory pauses after the screen is filled. To resume display of output, press any key.

The /W switch selects Wide Display. With /W, only filenames are displayed, without other file information. Files are displayed as much as possible per line.

FORMAT

SYNTAX: FORMAT

PURPOSE: Formats the disk in the specified drive to accept MSX-DOS files.

COMMENTS: This command initializes the directory and file allocation tables. A new disk must be formatted before use. If a used disk is formatted, all files in the disk are destroyed.

MSX-DOS issues the following message:

```
Drive name? (A,B) _
```

Select a drive name carefully. After you enter the drive name, the following message is displayed.

```
Strike a key when ready_
```

After you insert the new disk in the drive and press any key on the keyboard.

When the formatting finish , MSX-DOS will issue a following message.

```
Format complete
```

NOTE

The format procedure may be different with this description. For example, you can choose disk format from single side or double side with some disk driver. See your disk driver's manual.

MODE

SYNTAX: MODE <width>

PURPOSE: Sets the width of the display.

COMMENTS: <width> is the maximum number of characters per line on display.

<width> must be between 1 and 40. If it is 32 or less, screen mode 1 is selected , else mode 0 is selected.

The default screen mode and width of international MSX versions are as follows.

<table>
<tr><th>Version</th><th>Default screen mode</th><th>Default screen width</th></tr>
<tr><td>Japan</td><td>1</td><td>29</td></tr>
<tr><td>USA</td><td rowspan="5">0</td><td>39</td></tr>
<tr><td>UK</td><td rowspan="4">37</td></tr>
<tr><td>DIN</td></tr>
<tr><td>French</td></tr>
<tr><td>INT</td></tr>
</table>

PAUSE

SYNTAX: PAUSE [comment]

PURPOSE: Suspends execution of the batch file.

COMMENTS: During the execution of a batch file, you may need to change disks or perform some other action. PAUSE suspends execution until you press any key, except <CONTROL-C>.

When the command processor encounters PAUSE, it prints:

Strike a key when ready . . .

If you press <CONTROL-C>, another prompt will be displayed:

Terminate batch file (Y/N)?

If you type "Y" in response to this prompt, execution of the remainder of the batch command file will be aborted and control will be returned to the operating system command level. Therefore, PAUSE can be used to break a batch file into pieces, allowing you to end the batch command file at an intermediate point.

The comment is optional and may be entered on the same line as PAUSE. You may also want to prompt the user of the batch file with some meaningful message when the batch file pauses. For example, you may want to change disks in one of the drives. An optional prompt message may be given in such cases. The comment prompt will be displayed before the "Strike a key" message.

REM

SYNTAX: REM [comment]

PURPOSE: Displays remarks which are on the same line as the REM command in a batch file during execution of that batch file.

COMMENTS: The only separators allowed in the comment are the space, tab, and comma.

REN

SYNONYM: RENAME

SYNTAX: REN <filespec> <filename>

PURPOSE: Changes the name of the first option (filespec) to the second option (filename).

COMMENTS: The first option (filespec) must be given a drive designation if the disk resides in a drive other than the default drive. Any drive designation for the second option (filename) is ignored. The file will remain on the disk where it currently resides.

The wild card characters may be used in either option. All files matching the first filespec are renamed. If wild card characters appear in the second filename, corresponding character positions will not be changed.

For example, the following command changes the names of all files with the .LST extension to similar names with the .PRN extension:

```
REN *.LST *.PRN
```

In the next example, REN renames the file ABODE on drive B to ADOBE:

```
REN B:ABODE ?D?B?
```

The file remains on drive B.

An attempt to rename a filespec to a name already present in the directory will result in the error message "Rename error"

TIME

SYNTAX: TIME [<hh>[:<mm>[:<ss>]]]

PURPOSE: Displays and sets the time.

COMMENTS: If the TIME command is entered without any arguments, the following message is displayed:

```
Current time is <hh>:<mm>:<ss>.<cc>
Enter new time:_
```

Press the <RETURN> key if you do not want to change the time shown. A new time may be given as an option to the TIME command as in:

```
TIME 8:20
```

The new time must be entered using numerals only; letters are not allowed. The allowed options are:

```
<hh> = 00-24
<mm> = 00-59
<ss> = 00-59
```

The hour and minute entries must be separated by colons. You do not have to type the <ss> (seconds) or <cc> (hundredths of seconds) options.

MSX-DOS uses the time entered as the new time if the options and separators are valid. If the options or separators are not valid, MSX-DOS displays the message:

```
Invalid time
Enter new time:_
```

MSX-DOS then waits for you to type a valid time.

NOTE

If your computer does not have a clock, this command is nonsense.

TYPE

SYNTAX: TYPE <filespec>

PURPOSE: Displays the contents of the file on the console screen.

COMMENTS: Use this command to examine a file without modifying it. (Use DIR to find the name of a file.) The only formatting performed by TYPE is that tabs are expanded to spaces consistent with tab stops every eighth column. Note that a display of binary files causes control characters (such as CONTROL-Z) to be sent to your computer, including bells, form feeds, and escape sequences.

VERIFY

SYNTAX: VERIFY { ON | OFF }

PURPOSE: Set/reset verify (read after write) mode.

COMMENTS: The VERIFY ON command sets verify mode. Whenever some data are written into disk, that data are read from disk and verified. If the verified data is not correct, "DISK I/O error" occurs.

The VERIFY OFF command resets verify mode.

Default mode is VERIFY OFF.

Writing is more reliable but needs longer time in verify mode.

3.3 MSX Disk BASIC Reference Guide

Microsoft(TM) BASIC is the most extensive implementation of BASIC available for microprocessors. Microsoft BASIC meets the ANSI qualifications for BASIC, as set forth in document BSRX3.60-1978. Each release of Microsoft BASIC is compatible with previous versions.

MSX(TM) disk BASIC is a release of Microsoft BASIC for the MSX computer and its flexible disk system.

3.3.1 Commands and Statements

BLOAD
BSAVE
CLOSE
COPY
DSKO
FIELD
FILES and LFILES
FORMAT
GET
INPUT#
KILL
LINE INPUT#
LOAD
LSET and RSET
MAXFILES
MERGE
NAME
OPEN
PRINT# and PRINT# USING
PUT
RUN
SAVE
SYSTEM
VERIFY

BLOAD

SYNTAX: BLOAD "<filespec>"{[,R]|[,S]}[,offset]

PURPOSE: Loads a machine language program or an array from disk or cassette tape into memory.

COMMENTS: The file name can be omitted only for the file in the cassette tape, not for the disk.

If no <offset> is specified, the program is loaded from the address designated by the BSAVE command. If an <offset> is specified, the program is loaded from the address added <offset> to the saved address. Programs to be loaded with the offset must be relocatable.

The R option automatically runs the program after it has been loaded.

The S option loads the screen image saved by the "BSAVE ,S" statement to video RAM.

If no drive name is specified, the program in the current drive is loaded.

See also "BSAVE,".

EXAMPLE: BLOAD "MAX2"

Loads file "MAX2" into memory.

BSAVE

SYNTAX: BSAVE "<file spec>",<start address>,<end address> {[,<execute address>]|[,S]}

PURPOSE: Saves the machine language program currently in memory on disk or cassette tape.

COMMENTS: The program from <start address> to <end address> in memory is saved on disk or cassette tape.

If no drive name is specified, the program is saved on the current drive.

<start address> defines the default execution address.

The S option saves the content of video RAM to the file.

See also "BLOAD,".

EXAMPLE: BSAVE "TIMER",&HC000,&HCFFF

Saves the program currently in memory from &HC000 to &HCFFF on current drive under filename "TIMER".

CLOSE

SYNTAX: CLOSE [[#]<file number>[,[#]<file number...>]]

PURPOSE: Concludes I/O to a disk file.

COMMENTS: <file number> is the number under which the file was OPENed. A CLOSE with no arguments closes all open files.

The association between a particular file and file number terminates upon execution of a CLOSE statement. The file may then be reOPENed using the same or a different file number; likewise, that file number may now be reused to OPEN any file.

A CLOSE for a sequential output file writes the final buffer of output.

The END, CLEAR statements and the NEW command always CLOSE all disk files automatically.
(STOP does not close disk files.)

EXAMPLE: CLOSE #1

COPY

SYNTAX: COPY "<file spec>" TO "<file spec>"

PURPOSE: Copies one or more files to another disk. If you prefer, you can give the copies different names. This command can also copy files on the same disk.

COMMENTS: The second option may take three forms:

1. If the second option is a drive designation (d:) only, the original file is copied with the original filename to the designated drive.

2. If the second option is a filename only, the original file is copied to a file on the default drive with the filename specified.

3. If the second option is a full filespec, the original file is copied to a file on the default drive with the filename specified.

On a single-drive system, you enter the commands as you would on a multi-drive system.

If you specify drive B when the "drive A disk" was last used, you are prompted to insert the disk for drive B. For example:

```
COPY "A:TEST.ASC" TO "B:"
```

After the file is loaded from "drive A disk" to memory, you are prompted as follows.

```
Insert diskette for drive B:
and strike a key when ready
```

You remove "A disk" and insert "B disk". Then strike any key (except CONTROL-STOP). If the file is small, copy is completed.

But, if the file is big, you must exchange two disks following the prompted instructions until copy is completed. Because parts of the file are loaded and saved one after another.

If you specify drive A when the "drive B disk" was last used, you are prompted again to change disks. This time, BASIC prompts you to insert the "drive A disk". See also section 3.1.14.

DSKO

SYNTAX: DSKO <drive_number>,<logical_sector_number>

COMMENTS: Writes to the specified sector from memory pointed to by the content of (0F351H,0F352H).

<drive_number> is 0 for default drive, 1 for drive A, 2 for drive B, and so on.

<logical_sector_number> is a 0 based number. No check for the valid sector number is made.

NOTE: This memory area is destroyed when any disk statements (ex. FILES, OPEN, CLOSE, PRINT#, etc.) are executed.

FIELD

SYNTAX: FIELD [#]<file number>,<field width>
AS <string variable>...

PURPOSE: Allocates space for variables in a random file buffer.

COMMENTS: Before a GET statement or PUT statement can be executed, a FIELD statement must be executed to format the random file buffer.

<file number> is the number under which the file was OPENed. <field width> is the number of characters to be allocated to <string variable>.
For example,

```
FIELD 1,20 AS N$,10 AS ID$,40 AS ADD$
```

allocates the first 20 positions (bytes) in the random file buffer to the string variable N$, the next 10 positions to ID$, and the next 40 positions to ADD$. FIELD does NOT place any data in the random file buffer. (See "LSET/RSET," and "GET,".)

The total number of bytes allocated in a FIELD statement must not exceed the record length that was specified when the file was OPENed. Otherwise, a "Field overflow" error occurs.
(The default record length is 256 bytes.)

Any number of FIELD statements may be executed for the same file. All FIELD statements that have been executed will remain in effect at the same time.

NOTE: Do not use a FIELDed variable name in an INPUT or LET statement. Once a variable name is FIELDed, it points to the correct place in the random file buffer. If a subsequent INPUT or LET statement with that variable name is executed, the variable's pointer is moved to string space.

EXAMPLE 1:

```
10 OPEN "A:PHONELST" AS #1 LEN=35
15 FIELD #1,2 AS RECNBR$,33 AS DUMMY$
20 FIELD #1,25 AS NAMES,10 AS PHONENBR$
25 GET #1
30 TOTAL=CVI(RECNBR)$
35 FOR I=2 TO TOTAL
40 GET #1, I
45 PRINT NAMES, PHONENBR$
50 NEXT I
```

Illustrates a multiple defined FIELD statement. In statement 15, the 35 byte field is defined for the first record to keep track of the number of records

in the file. In the next loop of statements (35-50), statement 20 defines the field for individual names and phone numbers.

EXAMPLE 2:

```
10 FOR LOOP%=0 TO 7
20 FIELD #1,(LOOP%*16) AS OFFSETS,16 AS A$(LOOP%)
30 NEXT LOOP%
```

Shows the construction of a FIELD statement using an array of elements of equal size. The result is equivalent to the single declaration:

```
FIELD #1,16 AS A$(0),16 AS A$(1),...,16 AS A$(6)
,16 AS A$(7)
```

EXAMPLE 3:

```
10 DIM SIZE% (NUMB%): REM ARRAY OF FIELD SIZES
20 FOR LOOP%=0 TO NUMB%:READ SIZE%(LOOP%): NEXT LOOP%
30 DATA 9,10,12,21,41
 .
 .
 .
120 DIM A$(NUMB%): REM ARRAY OF FIELDED VARIABLES
130 OFFSET%=0
140 FOR LOOP%=0 TO NUMB%
150 FIELD #1,OFFSET% AS OFFSET$,SIZE%(LOOP%)
AS A$(LOOP%)
160 OFFSET%=OFFSET%+SIZE%(LOOP%)
170 NEXT LOOP%
```

Creates a field in the same manner as Example 2. However, the element size varies with each element. The equivalent declaration is:

```
FIELD #1,SIZE%(0) AS A$(0),SIZE%(1) AS A$(1),...
SIZE%(NUMB%) AS A$(NUMB%)
```

FILES and LFILES

SYNTAX: FILES ["<file spec>"]
LFILES ["<file spec>"]

PURPOSE: Displays or prints file names of disk files.

COMMENTS: The file names designated by the <file spec> are displayed. If the designated file does not exists, "File not found" error is occurs.

If no <file spec> is specified, all file names in the current drive are displayed.

There can be question mark (?) in the file name to substitute for a character in the file name or extension. And, there can be asterisk (*) to substitute for any file name or extension.

If the drive name is designated, the file names in that drive is displayed, else in current drive.

The LFILES command outputs file names not to display but to printer.

EXAMPLE: FILES "B:*.BAS"

FORMAT

SYNTAX:

```
CALL FORMAT
   or
_FORMAT
```

PURPOSE: Initializes a disk.

COMMENTS: Menu is displayed as follows.

```
Drive name? (A,B) _
```

Select a drive name carefully. After you enter the drive name, the following message is displayed.

```
Strike a key when ready_
```

After you insert the new disk in the drive and press any key on the keyboard.

When the formatting is finished, BASIC will issue the following message.

```
Format complete
```

NOTE: If a used disk is formatted, all files in that disk is destroyed.

New disks must be formatted before use.

The format procedure may be different with this description. For example, you can choose disk format from single side or double side with some disk driver. See your disk driver's manual.

GET

SYNTAX: GET [#]<file number>[,<record number>]

PURPOSE: Reads a record from a random disk file into a random buffer.

COMMENTS: <file number> is the number under which the file was OPENed. If <record number> is omitted, the next record (after the last GET) is read into the buffer. The largest possible record number is 4,294,967,295.

EXAMPLE:
```
10 OPEN "SAMPLE.DAT" AS #1
20 FIELD #1, 2 AS A$, 10 AS B$
30 FOR I%=1 TO 10
40   GET #1, I%
50   PRINT CVI(A$); B$
60   NEXT
70 CLOSE #1
80 END
```

NOTE: After an execution of a GET statement, INPUT# and LINE INPUT# may be executed to read characters from the random file buffer.

INPUT#

SYNTAX: INPUT#<file number>,<variable list>

PURPOSE: Reads data items from a sequential disk file and assigns them to program variables.

COMMENTS: <file number> is the number used when the file was OPENed for input. <variable list> contains the variable names that will be assigned to the items in the file. (The variable type must match the type specified by the variable name.)
With INPUT#, no question mark is printed, as with INPUT.

The data items in the file should appear just as they would if data were being typed in response to an INPUT statement. With numeric values, leading spaces, carriage returns, and line feeds are ignored. The first character encountered that is not a space, carriage return, or line feed is assumed to be the start of a number. The number terminates on a space, carriage return, line feed, or comma.

If MSX BASIC is scanning the sequential data file for a string item, leading spaces, carriage returns, and line feeds are also ignored. The first character encountered that is not a space, carriage return, or line feed is assumed to be the start of a string item. If this first character is a quotation mark ("), the string item will consist of all characters read between the first quotation mark and the second. Thus, a quoted string may not contain a quotation mark as a character. If the first character of the string is not a quotation mark, the string is an unquoted string, and will terminate on a comma, a carriage return, or a line feed (or after 255 characters have been read). If end-of-file is reached when a numeric or string item is being INPUT, the item is terminated.

EXAMPLE:

```
10 OPEN "SAMPLE2.DAT" FOR INPUT AS #1
20 INPUT #1, A$
30 PRINT A$
40 IF EOF(1)=0 THEN 20
50 CLOSE #1
60 END
```

KILL

SYNTAX: KILL "<file spec>"

PURPOSE: Deletes a file from disk.

COMMENTS: If a KILL statement is given for a file that is currently OPEN, a "File already open" error occurs.

KILL is used for all types of disk files: program files, random data files, and sequential data files.

EXAMPLE:
```
200 KILL "DATA1.DAT"
```

LINE INPUT#

SYNTAX: LINE INPUT#<file number>,<string variable>

PURPOSE: Reads an entire line (up to 254 characters), without delimiters, from a sequential disk data file to a string variable.

COMMENTS: <file number> is the number under which the file was OPENed. <string variable> is the variable name to which the line will be assigned. LINE INPUT# reads all characters in the sequential file up to a carriage return. It then skips over the carriage return/line feed sequence. The next LINE INPUT# reads all characters up to the next carriage return. (If a line feed/carriage return sequence is encountered, it is understood as a string ending with a line feed character.)

LINE INPUT# is especially useful if each line of a data file has been broken into fields, or if an MSX BASIC program saved in ASCII format is being read as data by another program. (See "SAVE,".)

EXAMPLE:

```
10 OPEN "LIST" FOR OUTPUT AS #1
20 LINE INPUT "CUSTOMER INFORMATION? ";C$
30 PRINT #1, C$
40 CLOSE 1
50 OPEN "LIST" FOR INPUT AS #1
60 LINE INPUT #1, C$
70 PRINT C$
80 CLOSE 1
RUN
CUSTOMER INFORMATION? LINDA JONES     234,4 MEMPHIS
LINDA JONES     234,4 MEMPHIS
Ok
```

LOAD

SYNTAX: LOAD <filename>[,R]

PURPOSE: Loads a file from disk into memory.

COMMENTS: <filename> is the name that was used when the file was SAVEd.

The R option automatically runs the program after it has been loaded.

LOAD closes all open files and deletes all variables and program lines currently residing in memory before it loads the designated program. However, if the R option is used with LOAD, the program is RUN after it is LOADed, and all open data files are kept open. Thus, LOAD with the R option may be used to chain several programs (or segments of the same program). Information may be passed between the programs using their disk data files.

Until the designated file is found and started being loaded, the program in memory is kept.

EXAMPLE: LOAD "STRTRK",R

LOAD "B:MYPROG"

LSET and RSET

SYNTAX: LSET <string variable>=<string expression>
RSET <string variable>=<string expression>

PURPOSE: Moves data from memory to a random file buffer (in preparation for a PUT statement).

COMMENTS: If <string expression> requires fewer bytes than were FIELDed to <string variable>, LSET left-justifies the string in the field, and RSET right-justifies the string. (Spaces are used to pad the extra positions.) If the string is too long for the field, characters are dropped from the right. Numeric values must be converted to strings before they are LSET or RSET. (See "MKI$, MKS$, MKD$,".)

EXAMPLE:
```
150 LSET A$=MKS$(AMT)
160 LSET D$=DESC($)
```

NOTE: LSET or RSET may also be used with a nonfielded string variable to left-justify or right-justify a string in a given field. For example, the program lines

```
110 A$=SPACE$(20)
120 RSET A$=N$
```

right-justify the string N$ in a 20-character field. This can be very handy for formatting printed output.

MAXFILES

SYNTAX: MAXFILES=<expression>

PURPOSE: Specifies the maximum number of files opened at a time.

COMMENTS: <expression> can be in the range of 0 to 15. When 'MAXFILES=0' is executed, only SAVE and LOAD can be performed.

MERGE

SYNTAX: MERGE <filename>

PURPOSE: Merges a specified disk file into the program currently in memory.

COMMENTS: <filename> is the name used when the file was SAVEd. The file must have been SAVEd in ASCII format. (If not, a "Bad file mode" error occurs.)

If any lines in the disk file have the same line numbers as lines in the program in memory, the lines from the file on disk will replace the corresponding lines in memory. (MERGEing may be thought of as "inserting" the program lines on disk into the program in memory.)

MSX BASIC always returns to command level after executing a MERGE command.

EXAMPLE: MERGE "NUMBRS"

NAME

SYNTAX: NAME <old filespec> AS <new filename>

PURPOSE: Changes the name of a disk file.

COMMENTS: <old filespec> must exist and <new filename> must not exist; otherwise, an error will result. After a NAME command, the file exists on the same disk, in the same area of disk space, with the new name.

If no drive name is specified, the current drive is selected.

EXAMPLE: NAME "ACCTS" AS "LEDGER"

In this example, the file that was formerly named ACCTS will now be named LEDGER.

OPEN

SYNTAX: OPEN "<filespec>" [FOR<mode>] AS [#] <file number> [LEN=<reclen>]

PURPOSE: Allows I/O to a disk file.

COMMENTS: A disk file must be OPENed before any disk I/O operation can be performed on that file. OPEN allocates a buffer for I/O to the file and determines the mode of access that will be used with the buffer.

<mode> is one of the following:

FOR OUTPUT	Specifies sequential output mode.
FOR INPUT	Specifies sequential input mode.
FOR APPEND	Specifies sequential append mode after end of an existent file.
default	Specifies random input/output mode.

<file number> is an integer expression whose value is between one and the maximum number of files specified in a MAXFILES statement.The number is then associated with the file as long as it is OPEN and is used to refer to other disk I/O statements to the file.

<filename> is a string expression containing a name that conforms to your operating system's rules for disk filenames.

<reclen> is an integer expression which, if included, sets the record length for random files. The default record length is 256 bytes. The largest possible record length is 256. The smallest is 1.

NOTE: If sequential input or append mode is used for non-existent file, "File not found" error occurs. If sequential output mode is used for existent file, the old file is deleted.

A file can be OPENed for sequential input or random access on more than one file number at a time. A file may be OPENed for output, however, on only one file number at a time.

EXAMPLE:
```
10 OPEN "INVEN" FOR INPUT AS #1
```

PRINT# and PRINT# USING

SYNTAX: PRINT#<file number>,[USING <string exp>;] <list of expressions>

PURPOSE: Writes data to a sequential disk file.

COMMENTS: <file number> is the number used when the file was OPENed for output. <string exp> consists of formatting characters as described in "PRINT USING." The expressions in <list of expressions> are the numeric and/or string expressions that will be written to the file.

PRINT# does not compress data on the disk. An image of the data is written to the disk, just as it would be displayed on the terminal screen with a PRINT statement. For this reason, care should be taken to delimit the data on the disk, so that it will be input correctly from the disk.

In the list of expressions, numeric expressions should be delimited by semicolons. For example:

```
PRINT#1,A;B;C;X;Y;Z
```

(If commas are used as delimiters, the extra blanks that are inserted between print fields will also be written to the disk.)

String expressions must be separated by semicolons in the list. To format the string expressions correctly on the disk, use explicit delimiters in the list of expressions.

For example, let A$="CAMERA" and B$="93604-1". The statement

```
PRINT#1,A$;B$
```

would write CAMERA93604-1 to the disk. Because there are no delimiters, this could not be input as two separate strings. To correct the problem, insert explicit delimiters into the PRINT# statement as follows:

```
PRINT#1,A$;",";B$
```

The image written to disk is

```
CAMERA,93604-1
```

which can be read back into two string variables.

If the strings themselves contain commas, semicolons, significant leading blanks, carriage returns, or line feeds, write them to disk surrounded by explicit quotation marks, CHR$(34).

For example, let A$="CAMERA, AUTOMATIC" and B$=" 93604-1". The statement

```
PRINT#1,A$;B$
```

would write the following image to disk:

```
CAMERA, AUTOMATIC   93604-1
```

And the statement

```
INPUT#1,A$,B$
```

would input "CAMERA" to A$ and "AUTOMATIC 93604-1" to B$. To separate these strings properly on the disk, write double quotation marks to the disk image using CHR$(34). The statement

```
PRINT#1,CHR$(34);A$;CHR$(34);CHR$(34);B$;CHR$(34)
```

writes the following image to disk:

```
"CAMERA, AUTOMATIC""   93604-1"
```

And the statement

```
INPUT#1,A$,B$
```

would input "CAMERA, AUTOMATIC" to A$ and " 93604-1" to B$.

The PRINT# statement may also be used with the USING option to control the format of the disk file. For example:

```
PRINT#1,USING"¥¥###.##,";J;K;L
```

(Japanase. Refer to 5.4 for other versions.)

PUT

SYNTAX: PUT [#]<file number>[,<record number>]

PURPOSE: Writes a record from a random buffer to a random disk file.

COMMENTS: <file number> is the number under which the file was OPENed. If <record number> is omitted, the record will assume the next available record number (after the last PUT). The largest possible record number is 4,294,967,295. The smallest record number is 1.

EXAMPLE:

```
10 OPEN "SAMPLE.DAT" AS #1
20 FIELD #1, 2 AS A$, 10 AS B$
30 FOR I%=1 TO 10
40   INPUT N%, S$
50   LSET A$=MKI$(N%)
60   LSET B$=S$
70   PUT #1, I%
80   NEXT
90 CLOSE #1
100 END
```

NOTE: LSET or RSET statement must be used to put characters in the random file buffer before executing a PUT statement.

Any attempt to read or write past the end of the buffer causes a "Field overflow" error.

RUN

SYNTAX: RUN <filename>[,R]

PURPOSE: Loads a file from disk into memory and runs it.

COMMENTS: <filename> is the name used when the file was SAVEd.

RUN closes all open files and deletes the current contents of memory before loading the designated program. However, with the "R" option, all data files remain OPEN.

EXAMPLE: RUN "NEWFIL",R

SAVE

SYNTAX: SAVE <filespec>[,A]

PURPOSE: Saves a program file on disk.

COMMENTS: <filespec> is a quoted string that conforms to MSX-DOS's requirements for filenames. If <filespec> already exists, the file will be written over.

Use the A option to save the file in ASCII format. Otherwise, MSX BASIC saves the file in a compressed binary format. ASCII format takes more space on the disk, but some disk access requires that files be in ASCII format. For instance, the MERGE command requires an ASCII format file, and some operating system commands such as LIST may require an ASCII format file.

NOTE: "CSAVE" and "SAVE" are used for binary and ASCII save of cassete tape file. But "SAVE" and "SAVE ... ,A" are used for that cases of disk file.

EXAMPLE: SAVE "COM2",A

SYSTEM

SYNTAX: CALL SYSTEM
or
_SYSTEM

PURPOSE: Exits from disk BASIC and returns to MSX-DOS.

COMMENTS: This command is valid only when BASIC has been booted from MSX-DOS.

By this command all files are closed and the program and the data in memory are destroyed.

VERIFY

SYNTAX: CALL VERIFY { ON | OFF }
or
_VERIFY { ON | OFF }

PURPOSE: Sets/resets verify (read after write) mode.

COMMENTS: The VERIFY ON command sets verify mode. Whenever some data are written into disk, that data are read from disk and verified. If the verified data is not correct, "DISK I/O error" occurs.

The VERIFY OFF command resets verify mode.

Default mode is VERIFY OFF.

NOTE: Writing is more reliable but needs longer time in verify mode.

3.3.2 Functions

CVI,CVS,CVD
DSKF
DSKI$
EOF
INPUT$
LOC
LOF
MKI$,MKS$,MKD$
VARPTR

CVI, CVS, CVD

SYNTAX: CVI(<2-byte string>)
CVS(<4-byte string>)
CVD(<8-byte string>)

PURPOSE: To convert string values to numeric values. Numeric values that are read in from a random disk file must be converted from strings back into numbers. CVI converts a 2-byte string to an integer. CVS converts a 4-byte string to a single precision number. CVD converts an 8-byte string to a double-precision number.

EXAMPLE:

```
.
.
.
70 FIELD #1,4 AS N$, 12 AS B$, ...
80 GET #1
90 Y=CVS(N$)
.
.
.
```

See also "MKI$, MKS$, MKD$,".

DSKF

SYNTAX: DSKF(<drive number>)

PURPOSE: To know free area size of specified disk by K byte.

The drive number corresponds to the drive name as follows.

0 default drive
1 drive A:
2 drive B:
and so on

EXAMPLE: PRINT DSKF(1)

DSKI$

SYNTAX: DSKI$(<drive_number>,<logical_sector_number>)

PURPOSE: To read the specified sector to memory pointed to by the content of (0F351H,0F352H).

<drive_number> is 0 for default drive, 1 for drive A, 2 for drive B, and so on.

<logical_sector_number> is a 0 based number. No check for the valid sector number is made.

NOTE: This memory area is destroyed when any disk statements (ex. FILES, OPEN, CLOSE, PRINT#, etc.) are executed.

EOF

SYNTAX: EOF(<file number>)

PURPOSE: To know if the end of a sequential file has been reached. Returns -1 (true) if so. Use EOF to test for end-of-file while INPUTting, to avoid "Input past end" errors.

The file specified by the file number must be opened as sequential input mode.

EXAMPLE:
```
10 OPEN "DATA" FOR INPUT AS #1
20 C=0
30 IF EOF(1) THEN 100
40 INPUT #1,M(C)
50 C=C+1:GOTO 30
 .
 .
 .
```

INPUT$

SYNTAX: INPUT$(X[,[#]Y])

PURPOSE: To read data from the terminal or from file number Y. Returns a string of X characters, If the terminal is used for input, no characters will be echoed. All control characters are passed through except Control-STOP, which is used to interrupt the execution of the INPUT$ function.

EXAMPLE:
```
5 'LIST THE CONTENTS OF A SEQUENTIAL FILE IN
HEXADECIMAL
10 OPEN "DATA" FOR INPUT AS #1
20 IF EOF(1) THEN 50
30 PRINT HEX$(ASC(INPUT$(1,#1)));
40 GOTO 20
50 PRINT
60 END
```

LOC

SYNTAX: LOC(<file number>)

where <file number> is the number under which the file was OPENed.

PURPOSE: With random disk files, LOC returns the record number just read or written from a GET or PUT statement. If the file was opened but no disk I/O has been performed yet, LOC returns a 0. With sequential files, LOC returns the number of records read from or written to the file since it was OPENed. When no record is read from the sequential input file since it was opened, LOC returns 1, because SYSTEM has read the first sector.

EXAMPLE:
```
200 IF LOC(1)>50 THEN STOP
```

LOF

SYNTAX: LOF(<file number>)

PURPOSE: LOF returns the size of the specified file by byte.

EXAMPLE:
```
IF NUM%>LOF(1) THEN PRINT "INVALID"
```

MKI$, MKS$, MKD$

SYNTAX:
```
MKI$(<integer expression>)
MKS$(<single precision expression>)
MKD$(<double precision expression>)
```

PURPOSE: To convert numeric values to string values. Any numeric value that is placed in a random file buffer with an LSET or RSET statement must be converted to a string. MKI$ converts an integer to a 2-byte string. MKS$ converts a single precision number to a 4-byte string. MKD$ converts a double precision number to an 8-byte string.

EXAMPLE:
```
90 AMT=(K+T)
100 FIELD #1,8 AS D$,20 AS N$
110 LSET D$=MKS$(AMT)
120 LSET N$=A$
130 PUT #1
 .
 .
```

See also "CVI, CVS, CVD,".

VARPTR

SYNTAX: VARPTR(#<file number>)

PURPOSE: VARPTR returns the address of the file control block assigned to <file number>.

EXAMPLE:
```
100 X=USR(VARPTR(#1))
```

3.3.3 Error Codes and Error Messages

Code Number Disk Errors Message

50 Field overflow

A FIELD statement is attempting to allocate more bytes than were specified for the record length of a random file.

51 Internal error

An internal malfunction has occurred in MSX BASIC. Report to Microsoft the conditions under which the message appeared.

52 Bad file number

A statement or command references a file with a file number that is not OPEN or is out of the range of file numbers specified at initialization.

53 File not found

A LOAD, KILL, or OPEN statement references a file that does not exist on the current disk.

54 File already open

A sequential output mode OPEN statement is issued for a file that is already open; or a KILL statement is given for a file that is open.

55 Input past end

An INPUT statement is executed after all the data in the file has been INPUT, or for a null (empty) file. To avoid this error, use the EOF function to detect the end-of-file.

56 Bad file name

An illegal form is used for the filename with a LOAD, SAVE, KILL, or OPEN statement (e.g., a filename with too many characters).

57 Direct statement in file

A direct statement is encountered while LOADing an ASCII-format file. The LOAD is terminated.

58 Sequential I/O only

A GET or PUT statement is used on a sequential file.

59 File not open

An input or output statement is executed on a not opened file.

60 Bad allocation table

The disk is not initialized.

61 Bad file mode

An attempt is made to use PUT, GET, or LOF with a sequential file, to LOAD a random file, or to execute an OPEN statement with a file mode other than "FOR INPUT", "FOR OUTPUT" , "FOR APPEND" or default (random).

62 Bad drive name

A invalid drive name is specified.

64 File still open

The file is not closed.

65 File already exists

The filename specified in a NAME statement is identical to a filename already in use on the disk.

66 Disk full

All disk storage space is in use.

67 Too many files

An attempt is made to create a new file (using SAVE or OPEN) when all 255 directory entries are full.

68 Disk write protected

A PUT or PRINT# statement is executed on a write protected disk.

69 Disk I/O error

An I/O error occurred on a disk I/O

operation. It is a fatal error; i.e., the operating system cannot recover from the error.

70 Disk offline

There is no disk in the specified drive.

71 Rename across disk

A RENAME statement is executed, across one drive to another.

3.4 MSX-DOS Boot Procedure

1) Boot Procedure

When all the buffers for the disk system are successfully allocated, the disk ROM checks the contents of address 0FEDAH to see if a ROM cartridge has set the hook (H.STKE) to gain control of the disk system. If the contents is not a 'RET' instruction (0C9H), the disk ROM sets up environments for disk BASIC and jumps to this hook.

The disk ROM next checks if there is an existing cartridge which has a TEXT entry in the cartridge header. If such a cartridge is found, the disk ROM sets up environments for disk BASIC and executes the BASIC program from the cartridge.

Next, the first sector of a first track (logical sector number 0) is read and transferred to 0C000H to 0C0FFH. If this read routine fails because of a drive not ready, a read error, or if the first byte of the boot sector is not 0EBH nor 0E9H, disk BASIC starts up.

Next, address 0C01EH is called with the carry flag set. This routine is provided so as to make game or other application programs take control of the disk system. The standard boot sector (provided) will just execute a 'RET' instruction if the carry flag is reset.

The ROM program next does a non-destructive memory check. If a 64K-byte RAM is not available, the program transfers control to disk BASIC.

Next the environments for MSXDOS are set up, and the routine jumps to 0C01EH with the carry flag set. Our standard boot sector loads MSXDOS.SYS at 100H and jumps to it. If MSXDOS.SYS not present, disk BASIC is invoked.

MSXDOS.SYS loads COMMAND.COM at 100H and jumps to it. If COMMAND.COM is not present, the routine prompts the user to insert a diskette with COMMAND.COM in it.

2) AUTOEXEC.BAT

When MSXDOS is first booted, it searches for a file named AUTOEXEC.BAT and executes it as a batch file.

3) AUTOEXEC.BAS

When MSX disk BASIC is first invoked, it looks for a file named AUTOEXEC.BAS and executes it as a BASIC program.

3.5 MSX-DOS and MSX Disk BASIC Disk Driver

The following values must be defined and declared as PUBLIC by the person or organization doing the interfacing.

MYSIZE
: Byte size of the work area used by the driver.

SECLEN
: The maximum sector size for the media supported by the driver.

DEFDPB
: The base address of the DPB (which consists of 18 bytes) for the media having the largest value for FATSIZ*SECSIZ.

The following subroutines must be provided and declared as PUBLIC by the person or organization doing the interfacing.

INIHRD	Initialize hardware
DRIVES	Return number of drives in system
INIENV	Initialize work area
DSKIO	Read/Write to disk
DSKCHG	Get disk change status
GETDPB	Get drive parameter block
CHOICE	Return character string for disk formatting
DSKFMT	Format disk
OEMSTATEMENT	(Entry point for use in system expansion)

The following is a detailed description the above routines.

INIHRD

Inputs:
: None

Outputs:
: None

Registers:
: AF, BC, DE, HL, IX, IY may be affected.

This routine initializes the hardware as soon as the control passes to the cartridge. Note that no work area is assigned when this routine is initiated.

DRIVES

Inputs:
[F] = The zero flag is reset in case one physical drive must act as two logical drives.

Outputs:
[L] = Number of drives connected

Registers:
F, HL, IX, IY may be affected.

Before any other processing can be done, the number of drives connected to the cartridge must be counted. If more than one drive is detected, or if the zero flag passed from the calling routine is set, the number of drives is returned (unmodified).

If only one drive has been detected and the zero flag passed is reset, a '2' must be returned as the number of drives, and the DSKIO and DSKFMT routines must logically support two drives. Use the PROMPT routine (described below) when switching drives.

When this routine is entered, the work area for the driver is already allocated.

INIENV

Inputs:
None

Outputs:
None

Registers:
AF, BC, DE, HL, IX, IY may be affected.

This entry initializes the work area (environment).

```
==========================================================
=       INIHRD, DRIVES and INIENV are called only        =
=       once during initialization, in the above         =
=       order.                                           =
==========================================================
```

DSKIO

Inputs:

[F]	=	Carry flag reset for read, set for write
[A]	=	Drive number (starts at 0)
[B]	=	Number of sectors to read/write
[C]	=	Media descriptor
[DE]	=	Logical sector number (starts at 0)
[HL]	=	Transfer address

Outputs:

If successful, carry flag cleared.
Otherwise, carry flag set,
error code is placed in [A],
number of remaining sectors in [B].

Registers:

AF, BC, DE, HL, IX, IY may be affected.

The drive number and media descriptor come from the drive parameter block. The number of sectors may range from 1 to 255. The logical sector numbers start at zero and is incremented in ones, so the I/O system must map these the logical sector numbers into tracks and sectors. The logical sector 0 corresponds to track 0, sector 1.

The error codes are defined as follows:

0	Write protected
2	Not ready
4	Data (CRC) error
6	Seek error
8	Record not found
10	Write fault
12	Other errors

DSKCHG

Inputs:

```
[A]  =  Drive number
[B]  =  0
[C]  =  Media descriptor
[HL] =  Base address of DPB
```

Outputs:

```
If successful:
    Carry flag reset,
    [B]  = Disk change status
              1      Disk unchanged
              0      Unknown
             -1      Disk changed

ELSE:
    Carry flag set,
    Error code in [A] (same as DSKIO above)
```

[NOTE]

If the disk has been changed or may have been changed (Unknown), read the boot sector or the first byte of the FAT of the currently inserted disk and transfer a new DPB as with the GETDPB call described below.

Registers:

AF, BC, DE, HL, IX, IY may be affected.

GETDPB

Inputs:

[A]	=	Drive number
[B]	=	First byte of FAT
[C]	=	Media descriptor
[HL]	=	Base address of DPB

Outputs:

[HL+1] .. [HL+18] = DPB for the specified drive

The Drive Descriptor Block (DPB) is defined as follows:

MEDIA	Byte	Media type
SECSIZ	Word	Sector size (Must be 2^n)
DIRMSK	Byte	(SECSIZ/32)-1
DIRSHFT	Byte	Number of one bits in DIRMSK
CLUSMSK	Byte	(Sectors per cluster)-1
CLUSSHFT	Byte	(Number of one bits in CLUSMSK)+1
FIRFAT	Word	Logical sector number of first FAT
FATCNT	Byte	Number of FATs
MAXENT	Byte	Number of directory entries (Max=254)
FIRREC	Word	Logical sector number of where the data area starts
MAXCLUS	Word	(Number of clusters on drive [not including reserved sectors, FAT sectors, or directory sectors])+1
FATSIZ	Byte	Number of sectors used
FIRDIR	Word	FAT logical sector number of start of directory

Note that the logical sector number always begins at zero.

CHOICE

Returns in [HL] the pointer to the character string (terminated by a zero) that is used as a user prompt in menu form by the main code. The simplest form of the routine be as follows.

```
CHOISE: LD      HL,CHOMSG
        RET
;
CHOMSG: DEFB    '1 - Single sided, 8 sectors',CR,LF
        DEFB    '2 - Single sided, 9 sectors',CR,LF
        DEFB    '3 - Double sided, 8 sectors',CR,LF
        DEFB    '4 - Double sided, 9 sectors',CR,LF
        DEFB    0
```

If there is no choice (i.e., only one format is supported), return with 0 in [HL] register.

All registers except SP may be affected.

DSKFMT

Formats a disk, both physically and logically. The input parameters are as follows.

[A] Choice specified by the user (1 to 9). Meaningless unless there is a choice.
[D] Drive number, beginning at zero
[HL] Beginning address of the work area which can be used by the format process.
[BC] Length of the work area described above.

All registers except SP may be affected.

This routine formats all of the disk's tracks physically, writing the boot sector, and clearing FATs and directory entries.

'Clearing FATs' means:

Writing the media descriptor byte at the first byte, writing 0FFH at the second and the third byte, and filling the remainder with 0's

'Clearing directory entries' means:

Filling all bytes with 0's

If the format ends successfully, return with carry flag reset, otherwise return with carry flag set. The error codes are defined as follows:

0	Write protected
2	Not ready
4	Data (CRC) error
6	Seek error
8	Record not found
10	Write fault
12	Bad parameter
14	Insufficient memory
16	Other errors

[NOTE]

No prompting messages should be generated by this routine.

OEMSTATEMENT

Statement for system expansion for use by OEMs. After disk BASIC scans its own expanded statements, control is passed to this entry. The calling sequence is identical to using a general-purpose expansion statement handler. If your ROM does not have expansion statements, set the carry flag and do a Z80 'RET' instruction.

```
****************************************
*                                      *
*    Some useful external routines     *
*                                      *
****************************************
```

PROMPT

Prints a message as follows and waits for the user to enter a key from the keyboard.

```
'Insert diskette for drive X:
 and strike a key when ready'
```

The 'X' is the drive name of the current target drive of your cartridge.

SETINT

This routine saves a previously set interrupt hook to a location specific to your cartridge, and sets the new interrupt hook. The address of the interrupt routine should be passed via the [HL] register. See DSKDRV.Z80 for details.

PRVINT

This routine jumps to the interrupt hook that you might have overwritten. Requires no argument. See DSKDRV.Z80 for details.

GETSLOT

Gets the slot address (i.e., where I am) in [A].
Preserves DE, IX, IY

GETWRK

Gets the base of the work area in [IX] and [HL].
Preserves DE, IY

DIV16

[BC]=[BC]/[DE], remainder in [HL].
Preserves DE, IX, IY

ENASLT

Enables a slot at an address specified by [A] and [HL], respectively. Destroys all registers.

XFER

Moves [BC] bytes from [HL] to [DE] (i.e., LDIR)
Preserves AF, IX, IY
BC is set to 0, HL, and DE pointing to the next location of source and destination, respectively.

Use this routine when a read/write operation is requested to 4000H..7FFFH, and your hardware does not have any special mechanism to transfer directly to these areas.

```
****************************************
*                                      *
*          External variables          *
*                                      *
****************************************
```

$SECBUF

Pointer to a temporary storage which is at least SECLEN byte long. Prepared for use combined with the XFER subroutine described above, but can be used TEMPORARILY for any purpose.

RAMAD0, RAMAD1, RAMAD2, RAMAD3

Slot address of RAM (if present) at 0000H..3FFFH, 4000H..7FFFH, 8000H..BFFFH, C000H.FFFFH respectively.

RAWFLG

Read-After-Write flag. When this byte contains non-0 value, the disk driver should do a read-after-write check. However, it is completely up to the driver whether to do the check or not.

How to determine media types

a) Read the boot sector (track 0, sector 1) of the target drive.

b) Check if the first byte is either 0E9H or 0EBH (the JMP instruction on the 8086)

c) If step b) fails, the disk is a version prior to MS-DOS 2.0; therefore, use the first byte of FAT passed from the caller and make sure it is between 0F8H and 0FFH.

   If step c) is successful, use this as a media descriptor.
   If step c) fails, then this disk cannot be read.

d) If step b) succeeds, read bytes # 0B to # 1D. This is the DPB for MS-DOS, Version 2.0 and above. The DPB for MSXDOS can be obtained as follows.

Contents of MS-DOS boot sector

+00	0E9H,XX,XX or 0EBH,XX,XX	
+03	ASCII string of OEM name	
+0B	Bytes per sector	(low)
+0C		(high)
+0D	Sectors per cluster	
+0E	Number of reserved sectors	(low)
+0F		(high)
+10	Number of FATs	
+11	Number of directory entries	(low)
+12		(high)
+13	Total number of sectors in the media	(low)
+14		(high)
+15	Media descriptor	
+16	Number of sectors per FAT	(low)
+17		(high)
+18	Sectors per track	(low)
+19		(high)
+1A	Number of heads	(low)
+1B		(high)
+1C	Number of hidden sectors	(low)
+1D		(high)

MS-DOS Disk formats

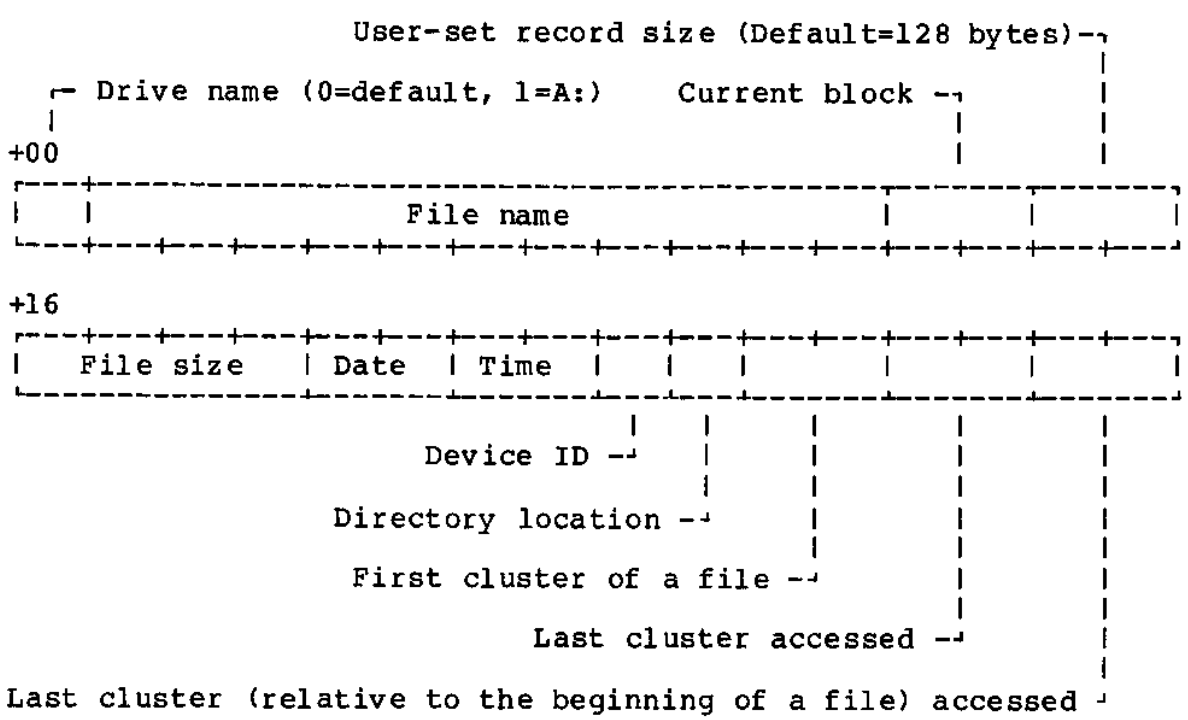
For 3, 3.5, and 5 inch disks (IBM PC format)

First digit : track number 8=80, 4=40
Second digit: Sector count 8 or 9
Third digit : Head count 1 or 2

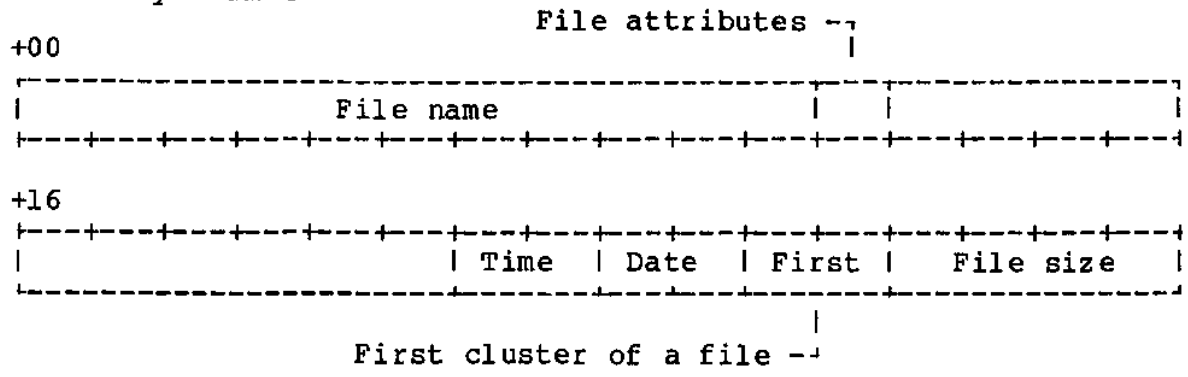
	891	892	881	882	491	492	481	482
Root directory entry	112	112	112	112	64	112	64	112
Media descriptor byte (FATID)	0F8H	0F9H	0FAH	0FBH	0FCH	0FDH	0FEH	0FFH
Sectors per FAT	2	3	1	2	2	2	1	1
Sectors/track	9	9	8	8	9	9	8	8
No. of sides	1	2	1	2	1	2	1	2
Tracks/side	80	80	80	80	40	40	40	40
Bytes/sector	512	512	512	512	512	512	512	512
No. of FATs	2	2	2	2	2	2	2	2
Sectors/cluster	2	2	2	2	1	2	1	2

3.6 MSX-DOS System Calls

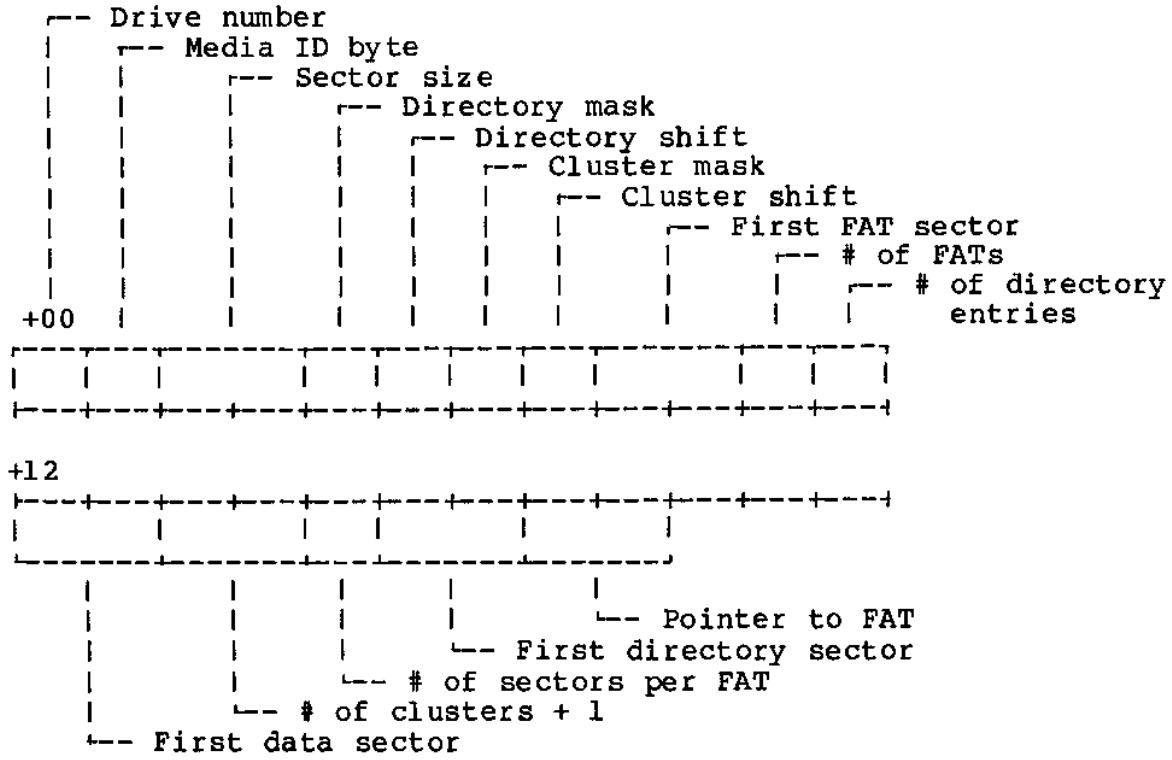
1) File Control Block (FCB) and Directory Entry



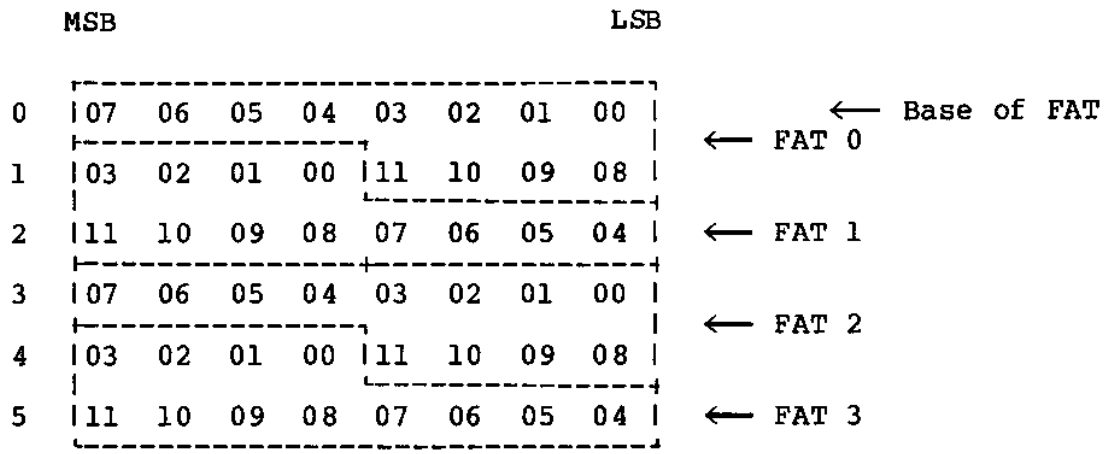
Directory format



2) Drive Parameter Block (DBP)



3) File Allocation Table (FAT)



4) System call entry

F37DH - MSX disk BASIC
0005H - MSX-DOS

To invoke a system call, call this entry with C register containing the function number.

5) System call specification

[Notes]

1) 'Compatibility' means 'compatibility with CP/M'. CP/M is a registered trademark of Digital Research, Inc.

2) Function calls entitled 'no function' will only return a 0 in the A register.

00 SYSTEM RESET

Parameters:	None
Returns:	None
Function:	If MSX-DOS Jumps to 0000H. Else Jumps to warm start of disk BASIC.
Compatibility:	Yes

01 CONSOLE INPUT

Parameters:	None
Returns:	A
Function:	Inputs a character from the console. Checks control-C and does function 00. Checks control-P and begins echoing to the printer. Checks control-N and stops echoing to the printer. Echoes the input character.
Compatibility:	Yes

02 CONSOLE OUTPUT

Parameters:	E
Returns:	None
Function:	Outputs character in E to the console.
Compatibility:	Yes

03 AUX INPUT

Parameters:	None
Returns:	A
Function:	Inputs character from an AUX device.
Compatibility:	Yes

04 AUX OUTPUT

Parameters:	E
Returns:	None
Function:	Outputs character in E to an AUX device.
Compatibility:	Yes

05 LST OUTPUT

Parameters:	E
Returns:	None
Function:	Outputs character in E to the printer.
Compatibility:	Yes

06 DIRECT CONSOLE I/O

Parameters:	E
Returns:	A

Function:	If E is 0FFH
	  If no input from the console
	  Returns 0.
	  Else
	  Returns the code.
	  No check.
	  No echo.
	Else
	  Outputs character in E to the console.

Compatibility:	Yes

07 DIRECT INPUT

Parameters:	None
Returns:	A
Function:	Inputs character from the console. No check. No echo.
Compatibility:	No (get I/O byte)

08 DIRECT INPUT

Parameters:	None
Returns:	A
Function:	Inputs character from the console. Checks for control-C. Checks for control-P. Checks for control-N. No echo.
Compatibility:	No (set I/O byte)

09 STRING OUTPUT

Parameters:	DE
Returns:	None
Function:	Outputs the string pointed to by DE to the console until a '$' is encountered in the given string.
Compatibility:	Yes

0A BUFFERED INPUT

Parameters:	DE
Returns:	None
Function:	Inputs the string from console beginning at [DE+2] until carriage return is input. [DE+1] is set to the length of input string, not including the terminator. The maximum length of the string is passed via [DE].
Compatibility:	Yes

0B CONSOLE STATUS

Parameters:	None
Returns:	A
Function:	If no input from the console Returns 0. Else Returns 0FFH.
Compatibility:	Yes

0C GET VERSION NUMBER

Parameters:	None
Returns:	H, L
Function:	Sets 0 in H register, 22H in L register.
Compatibility:	Yes

0D DISK RESET

Parameters:	None
Returns:	None
Function:	Sets default drive to (A:). Sets transfer address to 80H. Flushes out all sectors which have been changed but have not been written to disk.
Compatibility:	Yes

0E SELECT DISK

Parameters:	E
Returns:	None
Function:	Sets the default drive, (with a 0 corresponding to A:)
Compatibility:	Yes

0F OPEN FILE

Parameters:	DE
Returns:	A
Function:	Opens a file specified by an FCB pointed to by DE. The record size field, the current block field, the current record field, and the random record field should be set after this function is executed. The file size field, the date and time fields, the device ID field, the directory location field, the first cluster field, the last cluster field, and the last accessed cluster field is copied from the directory. If successful Returns 0. Else Returns 0FFH.
Compatibility:	Yes

10 CLOSE FILE

Parameters:	DE
Returns:	A
Function:	Closes a file specified by an FCB pointed to by DE. If successful Returns 0. Else Returns 0FFH.
Compatibility:	Yes

11 SEARCH FIRST

Parameters:	DE
Returns:	A
Function:	Searches for the first occurrence of a file specified by an FCB pointed to by DE. If found The directory entry (32 bytes long) is copied to the transfer address. Returns 0. Else Returns 0FFH. [Note] Wild card characters such as (* and ?) are permitted in the file name.
Compatibility:	Yes

12 SEARCH NEXT

Parameters:	None
Returns:	A
Function:	Searches for the next occurrence of a file specified by the last 'search first' function call. If found The directory entry (32 bytes long) is copied to the transfer address. Returns 0. Else Returns 0FFH. [Note] Wild card characters such as (* and ?) are permitted in the file name.
Compatibility:	Yes

13 DELETE FILE

Parameters: DE
Returns: A
Function: Deletes a file specified by an FCB pointed to by DE.
If successful
    Returns 0.
Else
    Returns 0FFH.
[Note]
    Wild card characters such as (* and ?) are permitted in the file name.
Compatibility: Yes

14 SEQUENTIAL READ

Parameters: DE
Returns: A
Function: Reads a record of a file specified by the FCB pointed to by DE and transfers the record to the transfer address.
The record is determined by the current block field and the current record field.
The current block field and the current record field are automatically incremented upon return.
The record size is always 128 bytes.
If successful
    Returns 0.
Else
    Returns 1.

[NOTE]
This system call is prepared to maintain compatibility with CP/M. The use of the 'random block read' function is strongly recommended.

Compatibility: Yes

15 SEQUENTIAL WRITE

Parameters: DE
Returns: A
Function: Writes a record to a file specified by the FCB pointed to by DE from the transfer address. The record is determined by the current block field and the current record field.
The current block field and the current record field are automatically incremented upon return.
The record size is always 128 bytes.
If successful
    Returns 0.
Else
    Returns 1.

[NOTE]
This system call is prepared to maintain compatibility with CP/M. The use of the 'random block write' function is strongly recommended.

Compatibility: Yes

16 CREATE FILE

Parameters: DE
Returns: A
Function: Creates a file specified by an FCB pointed to by DE. If the specified file already exists, it is overwritten.
The record size field, the current block field, the current record field, and the random record field should be set after this function is executed.
If successful
    Returns 0.
Else
    Returns 0FFH.
Compatibility: Yes

17 RENAME FILE

Parameters:	DE
Returns:	A
Function:	Renames a file name specified by an FCB pointed to by DE to a file name specified by an FCB pointed to by DE+16. If successful Returns 0. Else Returns 0FFH. [Note] Wild card characters such as (* and ?) are permitted in the file name.
Compatibility:	Yes

18 GET LOGIN VECTOR

Parameters:	None
Returns:	HL
Function:	Returns a bit table for on-line drives. Unlike CP/M, all system drives are on-line.
Compatibility:	Yes

19 GET DEFAULT DRIVE NAME

Parameters:	None
Returns:	A
Function:	Gets the default drive name.
Compatibility:	Yes

1A SET DMA ADDRESS

Parameters:	DE
Returns:	None
Function:	Sets transfer address.
Compatibility:	Yes

1B GET ALLOCATION

Parameters:	E
Returns:	A, BC, DE, HL, IY
Function:	Returns information of a drive specified by E. If drive name is valid A = Number of sectors/cluster BC = Sector size DE = Number of clusters on disk HL = Number of free clusters IX = Pointer to DPB IY = Pointer to FAT Else A = 0FFH
Compatibility:	No (Get allocation address)

System calls for CP/M version 2.0 or later

1C NO FUNCTION

Compatibility: No (Set write protect vector)

1D NO FUNCTION

Compatibility: No (Get write protect vector)

1E NO FUNCTION

Compatibility: No (Set file attributes)

1F NO FUNCTION

Compatibility: No (Get disk parameter address)

20 NO FUNCTION

Compatibility: No (Set/Get user code)

21 RANDOM READ

Parameters: DE
Returns: A
Function: Reads a record of a file specified by the FCB pointed to by DE and transfers the record to the transfer address.
The record is determined by the random block field. The random block field is not affected by this function.
The record size is always 128 bytes.
If successful
    Returns 0.
Else
    Returns 1.

[NOTE]
This system call is prepared to maintain compatibility with CP/M. The use of the 'random block read' function is strongly recommended.

Compatibility: Yes

22 RANDOM WRITE

Parameters: DE
Returns: A
Function: Writes a record to a file specified by the FCB pointed to by DE from the transfer address. The record is determined by the random block field. The random block field is not affected by this function.
The record size is always 128 bytes.
If successful
    Returns 0.
Else
    Returns 1.

[NOTE]
This system call is prepared to maintain compatibility with CP/M. The use of the 'random block write' function is strongly recommended.

Compatibility: Yes

23 GET FILE SIZE

Parameters: DE
Returns: A
Function: Calculates the file size (a multiple of 128) of the file specified by the FCB pointed to by DE, and sets the file size to the random record field of the given FCB.
If successful
    Returns 0.
Else
    Returns 0FFH.
Compatibility: Yes

24 SET RANDOM RECORD

Parameters: DE
Returns: None
Function: Calculates the current record position from the current block field and the current record field of the given FCB pointed to by DE, and sets the record position to the random record field of the given FCB.
Compatibility: Yes

System calls for CP/M version 2.2 or later

25 NO FUNCTION

Compatibility: No (Resets disk drive)

26 RANDOM BLOCK WRITE

Parameters: DE, HL
Returns: A
Function: Writes records to a file specified by the FCB pointed to by DE from the transfer address. The record is determined by the random block field. The current random record field is automatically incremented upon successful return. The record size is determined by the record size field.
The number of records to write is passed via HL.
If successful
    Returns 0.
Else
    Returns 1.
Compatibility: No (No function)

27 RANDOM BLOCK READ

Parameters: DE, HL
Returns: A,HL
Function: Reads records of a file specified by the FCB pointed to by DE and transfers the record to the transfer address. The record is determined by the random block field. The current random record field is automatically incremented upon successful return.
The record size is determined by the record size field.
The number of records to read is passed via HL.
The number of records actually read is returned in HL.
If successful
    Returns 0 in A.
Else
    Returns 1 in A.
Compatibility: No (No function)

28 RANDOM WRITE WITH ZERO FILL

Parameters:	DE
Returns:	A
Function:	Writes a record to a file specified by the FCB pointed to by DE from the transfer address. The record is determined by the random block field. The random block field is not affected by this function. The record size is always 128 bytes. When extending a file, all records that are not written are filled with 0's. If successful Returns 0. Else Returns 1.
Compatibility:	Yes

System calls for MSX-DOS only

29 NO FUNCTION

Compatibility: No

2A GET DATE

Parameters: None
Returns: HL, DE, A
Function: HL = year
D = month
E = day
A = day of the week
Compatibility: No

2B SET DATE

Parameters: HL, DE
Returns: A
Function: Sets current date to the date passed via registers.
The registers are as for 'get date'.
If successful
    Returns 0.
Else
    Returns 0FFH.
Compatibility: No

2C GET TIME

Parameters: None
Returns: H, L, D, E
Function: H = hours
L = minutes
D = seconds
E = 1/100 seconds
Compatibility: No

2D SET TIME

Parameters:	H, L, D, E
Returns:	A
Function:	Sets current time to the date passed via registers. The registers are as for 'get time'. If successful Returns 0. Else Returns 0FFH.
Compatibility:	No

2E SET/RESET VERIFY FLAG

Parameters:	E
Returns:	None
Function:	If E is 0 Reset verify flag. Else Set verify flag.
Compatibility:	No

2F ABSOLUTE DISK READ

Parameters:	DE, H, L
Returns:	None
Function:	Read H sectors from logical sector number DE on the drive specified by L to the transfer address.
Compatibility:	No

30 ABSOLUTE DISK WRITE

Parameters:	DE, H, L
Returns:	None
Function:	Write H sectors to logical sector number DE on the drive specified by L from the transfer address.
Compatibility:	No

6) Direct BIOS access of MSX-DOS

On many CP/M application programs, the BIOS jump table is directly referenced by adding offsets to the contents of addresses 1 and 2. To make the above programs work, MSX-DOS creates a CP/M-style BIOS front end, vectored by the contents of addresses 1 and 2. Due to the differences in file handling between MSX-DOS and CP/M, only the following entries are guaranteed.

BOOT	Cold boot
WBOOT	Warm boot
CONST	Console status
CONIN	Console input
CONOUT	Console output

7) Zero page usage and memory map of MSX-DOS

00	JP	WBOOT	(Used)
01			(Used)
02			(Used)
03			
04			
05	JP	BDOS	(Used)
06			(Used)
07			(Used)
08			
09			
0A			
0B			
0C	JP	RDSLT	(Used)
0D			(Used)
0E			(Used)
0F			
10			
11			
12			
13			
14	JP	WRSLT	(Used)
15			(Used)
16			(Used)
17			
18			
19			
1A			
1B			
1C	JP	CALSLT	(Used)
1D			(Used)
1E			(Used)
1F			
20			
21			
22			
23			
24	JP	ENASLT	(Used)
25			(Used)
26			(Used)
27			
28			
29			
2A			
2B			
2C			
2D			
2E			
2F			

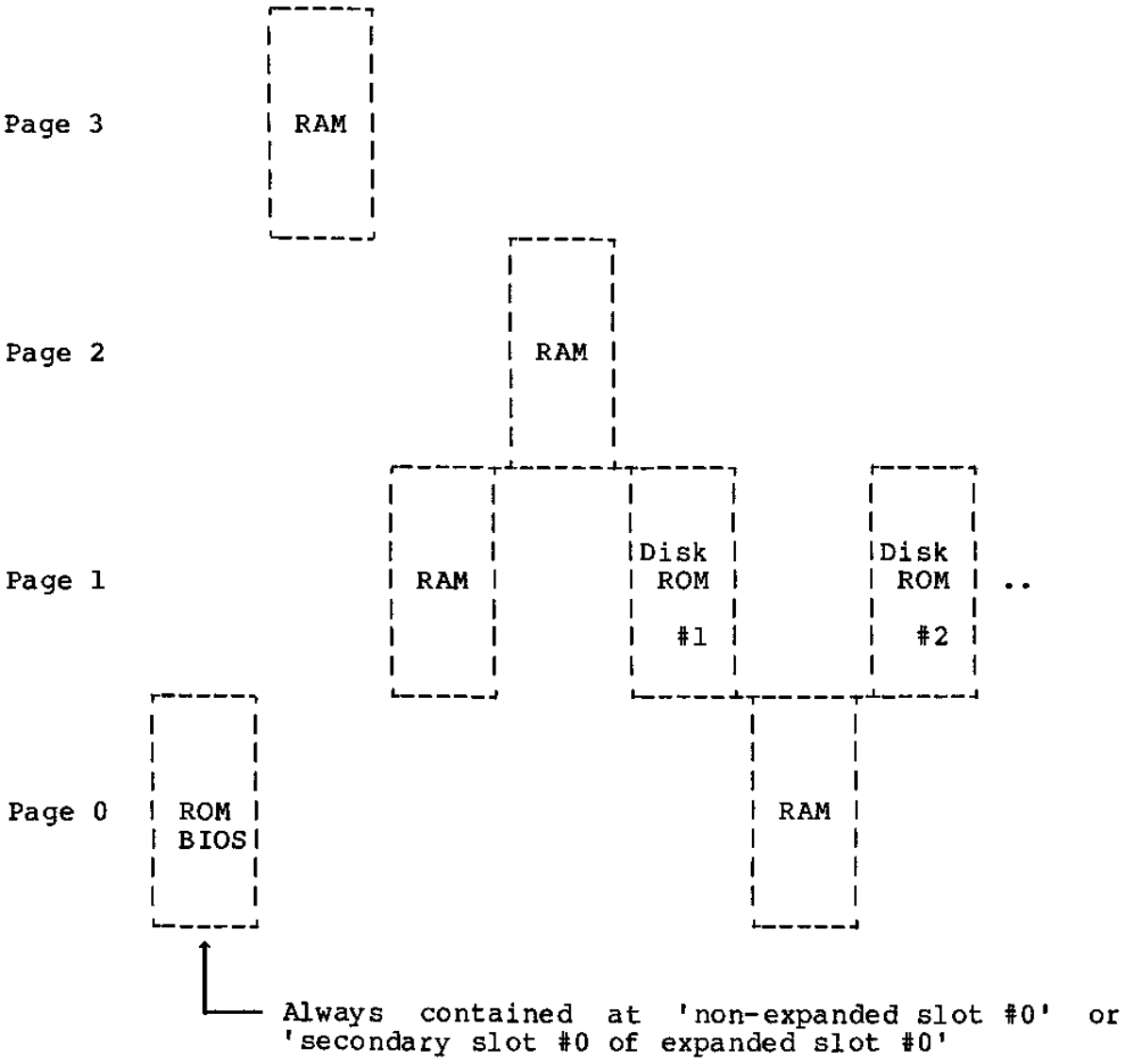
```
30  JP      CALLF                       (Used)
31                                      (Used)
32                                      (Used)
33
34
35
36
37
38  JP      INTRPT                      (Used)
39                                      (Used)
3A                                      (Used)
3B  ---┐
3C     |
3D     |
3E     |
3F     |
40     |
41     |
42     |
43     |
44     |
45     |
46     |
47     |
48     |
49     |
4A     |
4B     |-- Routine to switch secondary slots
4C     |
4D     |                                (Used)
4E     |
4F     |
50     |
51     |
52     |
53     |
54     |
55     |
56     |
57     |
58     |
59     |
5A     |
5B  ---┘
```

5C	FCB for first argument	(Used)
5D		(Used)
5E		(Used)
5F		(Used)
60		(Used)
61		(Used)
62		(Used)
63		(Used)
64		(Used)
65		(Used)
66		(Used)
67		(Used)
68		(Used)
69		(Used)
6A		(Used)
6B		(Used)
6C	FCB for second argument	(Used)
6D		(Used)
6E		(Used)
6F		(Used)
70		(Used)
71		(Used)
72		(Used)
73		(Used)
74		(Used)
75		(Used)
76		(Used)
77		(Used)
78		(Used)
79		(Used)
7A		(Used)
7B		(Used)
7C		(Used)
7D		(Used)
7E		(Used)
7F		(Used)
80	Default DMA address	(Used)
..		(Used)
..		(Used)
FF		(Used)
100	TPA	

The word at addresses 6 and 7 contains the 'highest available memory + 1' for the TPA.

The entry addresses for RDSLT, WRSLT, ENASLT, CALSLT, and CALLF are identical to the ROM BIOS. However, pay GREAT attention when using these entries directly. You must make sure that the stack area is guaranteed when changing slots. For example, when calling the ROM BIOS routines from MSX-DOS through CALSLT, page 0 is set to ROM, and when an interrupt occurs when the ROM BIOS is active, Page 1 may be set to ROM (i.e., the disk ROM), because some manufacturers are using the timer interrupt hook to stop the motor.

[Memory Map]



FCB organization (for disk BASIC)

NOTE

The following information is only for use by advanced programmers. Please ignore it if you do not understand it.

The FCB holds information about file channels. Each channel is allocated 265 bytes, 9 bytes of which are used by the BASIC interpreter, and the other 256 bytes for buffering.

Offset	Label (For SPCDSK)	Meaning (For MSX Disk BASIC)
+0	FL.MOD	Mode which the file was opened for
+1	FL.FCA	Pointer to FCB for BDOS (low)
+2	FL.LCA	Pointer to FCB for BDOS (high)
+3	FL.LSA	Back up character
+4	FL.DSK	Device number
+5	FL.SLB	
+6	FL.BPS	Position in FL.BUF
+7	FL.FLG	Holds various information
+8	FL.OPS	Pseudo head position
+9..	FL.BUF	256-byte file buffer

4. Other Expansions

4.1 MSX RS-232C Support
4.1.1 Extended BASIC for RS-232C Communication
1) Set up Communication Parameters
2) Open and Close Communication Channels
3) Sequential Input and Output
4) Program Load/Save Statements
5) Event Trap Control Statements
6) Miscellaneous Control Statements
7) Functions
8) Terminal Mode
9) Help Function (Optional)
10) Behavior of Control Signals
11) Handling of EOF
4.1.2 Extended BIOS Calls Handling RS-232C
Build a Slot Address Table Entry to the Jump Table
Return Number of Channels
Description of each Extended BIOS Call
4.2 Other MSX Extended BIOS Calls
4.2.1 Extended BIOS Calls
1) Broad Cast Command
2) System Exclusive Extended BIOS Call
3) Summary of Extended BIOS Calls
4.2.2 Extended BIOS Maker ID Number
4.3 Tenkey Support on MSX

4.1 MSX RS-232C Support

This section describes the specifications of the support for the RS-232C communication interface on MSX computers.

4.1.1 Extended BASIC for RS-232C Communication

1) Set up Communication Parameters

CALL COMINI [([<string exp>][,[<Rx baud rate>][,[<Tx baud rate>]
[,[<time out>]]]])]

Initializes an RS-232C port with the specified parameters. The <string exp> is a string that specifies the channel control parameters. See the detailed desciption below.

BAUD RATE
It is possible to set a different baud rate for transmitter and receiver. The possible baud rates are as follows:

50	110	600	1800	2400	4800	9600
75	300	1200	2000	3600	7200	19200

When only the receiver's baud rate is specified, the baud rate for the transmitter assumes the same speed as the receiver. When only the transmitter's baud rate is specified, the baud rate for the receiver is set to the default value.

If a negative value is specified, its absolute value is written to i8253 Timer/Counter directly.

TIME OUT
The RS-232C driver waits for the CTS (Clear To Send) to turn on or/and XON is received when the character is sent. The driver generates a time out error if the specified time has elapsed. The value for the time out error is specified in seconds. If zero (0) is specified, the driver does not generate a time out error, and the driver waits indefinitely.

STRING FIELDS

```
"[0:][8[N[1[X[H[N[N[N]]]]]]]]"
  |   | | | | | | | |
  |   | | | | | | | |
  |   | | | | | | | |
```

2) Open and Close Communication Channels

OPEN "COM[n]:" [FOR <mode>] AS [#] <file number>

This statement opens the RS-232C channel for further processing. That is, a I/O buffer is allocated and the mode that will be used with the buffer is set. The RTS signal is also activated.

The <mode> is one of the following:

OUTPUT:	Specifies sequential output mode
INPUT :	Specifies sequential input mode

If the <mode> clause is not specified, the channel can be accessed for both input and output and no EOF character handling is done.

The <file number> is an integer expression whose value is between one and the maximum number of files specified in a MAXFILES= statement.

The <file number> is the number that is associated with the file for as long as it is OPEN and is used by other I/O statements reffering to the file.

An OPEN statement must be executed before I/O may be done to the file using any of the following statements. The OPEN statement must be executed before any statement or function requiring a file number:

```
PRINT #, PRINT # USING
INPUT #, LINE INPUT #
INPUT$
```

Example:

```
OPEN "COM0:" AS #1
```

NOTE

Random access to RS-232C channel is not possible. Logically, only sequential accesses are permitted.

CLOSE [[#]<file number>[,<file number>]]

Closes the channel and releases the associated buffer. If no <file number>s are specified, all open channels are closed.

If the channel was opened in output mode, the EOF character is sent.

3) Sequential Input and Output

After the channel is opened in input mode or file mode (open without <mode> clause), characters from communication channel can be sequentially input by one of the following statements.

```
INPUT #n
LINE INPUT #n
INPUT$(#n,m)
```

After the channel is opened in output mode or file mode (open without <mode> clause),characters can be sequentially output to the communication channel by one of the following statements.

```
PRINT #n
PRINT #n USING
```

Refer to the reference manuals for the language for details on the statements.

4) Program Load/Save Statements

SAVE "COM[<n>:]"[,A]

Sends a BASIC program to the communication channel. A Control-Z is treated as the end-of-file character. The program is sent in ASCII format, whether the optional parameter, "A", is specified or not. No file name is allowed.

LOAD "COM[<n>:]"

Loads a BASIC program from the channel. A LOAD statement closes all open files and deletes the current program from memory. If the "R" option is specified, however, all data files remain OPEN and the program that is loaded is also executed. A Control-Z is treated as the end-of-file character.

MERGE "COM[<n>:]"

Merges lines from a program in ASCII format received through the communication channel into the program currently in memory.

If some of the line numbers of the program in memory match line numbers of the incoming (channel) program, the lines from the program from the channel replaces the matching lines. A Control-Z is treated as the end-of-file character.

After the MERGE command, the merged program will reside in memory, and control will return to BASIC at the command level.

RUN "COM[<n>:]"[,R]

Loads a program from the channel into memory and runs it.

RUN closes all open files and deletes the current contents of memory before loading the designated program. When the "R" option is specified, however, all data files remain OPEN.

5) Event Trap Control Statements

CALL COMON("[<n>:]")
Enables event trapping caused by incoming character from the communication channel.

CALL COMOFF("[<n>:]")
Disables event trapping caused by incoming character from the communication channel. The communication buffer is flushed.

CALL COMSTOP("[<n>:]")
Suspends event trapping caused by incoming character from the communication channel.

CALL COM ([<n>:],GOSUB <line number>)
Sets the line numbers for BASIC to trap when characters are received at the communication channel.

When trap occurs, since CALL COMSTOP is automatically executed, received traps can never take place. The RETURN from the trap routine will automatically do CALL COMON unless CALL COMOFF has been explicitly performed inside the trap routine.

Event trapping does not take place when BASIC is not executing a program. When an error trap (resulting from an ON ERROR statement) takes place, it automatically disables all trapping (including ERROR, STRIG, STOP, SPRITE, INTERVAL and KEY).

6) Miscellaneous Control Statements

An OPEN statement must be executed before any one of following statements may be executed. The default channel number is 0 for all the following statements.

CALL COMBREAK(["<n>:"],<expression>)
Sends break characters specified by <expression> to the channel specified by <n>. The range of the <expression> should be between 3 and 32767.

CALL COMDTR(["<n>:"],<expression>)
Turns off the DTR signal when the <expression> is zero, otherwise turns on the DTR signal.

CALL COMSTAT(["<n>:"],<name of variable>)
Reads the status of the communication channel. The status returned by the hardware is assigned to the variable. The bit assignments are as follows:

BIT NO.	Description
15	Buffer Overflow Error 0: No buffer overflow 1: Buffer overflow
14	Time Out Error (TMENBT) 0: No time out error occurred 1: Time out error occurred
13	Framing Error 0: No framing error occurred 1: Framing error occurred
12	Over Run Error 0: No over run error occurred 1: Over run error occurred
11	Parity error 0: Character has no parity error 1: Character has parity error
10	Control break key was pressed (BRONBT) 0: Control break key not pressed 1: Control break key was pressed
9	Not used: Reserved
8	Not used: Reserved
7	Clear To Send 0: False 1: True
6	Timer/Counter Output-2 0: Timer/Counter Output-2 negated 1: Timer/Counter Output-2 asserted
5	Not used: Reserved
4	Not used: Reserved
3	Data Set Ready 0: False 1: True
2	Break Detect 0: Not detected 1: Detected
1	Ring Indicator 0: False 1: True
0	Carrier Detect 0: False 1: True

7) Functions

EOF(<file number>)
Returns -1 (true) if the EOF character is received. Otherwise, returns 0. Use EOF to test for end-of-transmission during INPUT to avoid 'Input past end' errors.

LOC(<file number>)
Returns the number of characters received in the communication buffer. The size of the communication buffer is 255 characters.

LOF(<file number>)
Returns the size of the free space remaining in the communication buffer.

8) Terminal Mode

CALL COMTERM[("<n>:")
Enters a terminal emulator mode. The channel should be closed when this statement is invoked. The function keys have a special use in the terminal mode as described below.

F-6: Toggles the literal mode on/off. In the literal mode, control characters are displayed, offset by 40H. As an example,a character whose code is 01H is displayed as "A".
Initial mode: Literal mode off

F-7: Toggles the Half/Full duplex modes. In Half duplex mode, the characters typed in are echoed to the screen as well as sent to the communication channel.
Initial mode: Full duplex

F-8: Turn on/off printer echo. When the printer echo is on, all characters sent to the screen are also echoed to the printer.
Initial mode: Printer echo off

9) Help Function (Optional)

CALL COMHELP[(<n>:)]

Prints out a brief description of parameters set by a COMINI statement on the screen as follows.

```
Initialize statement options

CALL COMINI ("
<Device# {0,1,2...9}>:
<Character length {5,6,7,8}>
<Parity {E,O,I,N}>
<Stop bits {1,2,3}>
<XON/XOFF {X,N}>
<CTS handshaking {H,N}>
<Auto LF on receive {A,N}>
<Auto LF on transmit {A,N}>
<SI/SO {S,N}>"
,<Receiver baud rate>
,<Transmitter baud rate>
                          )
Default:
 CALL COMINI("0:8N1XHNNN"
      ,1200,1200,0)
```

10) Behavior of Control Signals

	RESET	COMINI	OPEN	CLOSE
RTS:	Inactive	No effect	Active	Inactive
DTR:	Active	Active	No effect	No effect

The RTS signal is affected in the following cases:

1. OPEN statement executed: activated
2. CLOSE statement executed: inactivated
3. The remaining contents of the communication is less than 16 bytes and the CTS-RTS handshake is enabled: inactivated.
4. When inactive and the remaining contents of the communication buffer has more than one byte and CTS-RTS handshaking is enabled: activated.

DTR is affected by the CALL COMDTR and CALL COMINI statements.

11) Handling of EOF

An EOF is transmitted when a CLOSE statement is executed during the open mode was output.

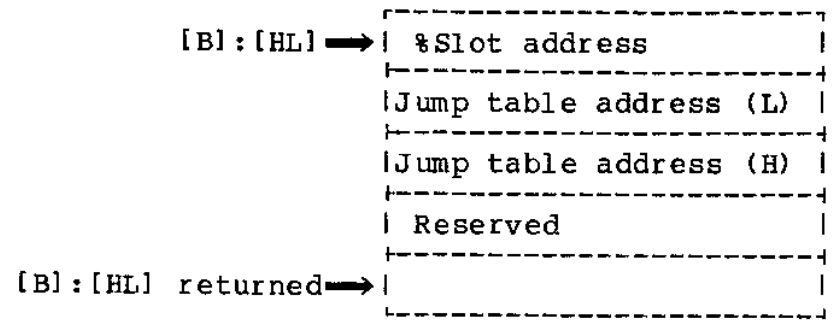
4.1.2 Extended BIOS Calls Handling RS-232C

The RS-232C driver can be used by application programs using the "EXTENDED BIOS CALL" routine. Such programs can access the functions in the RS-232C driver through the entry jump table with inter-slot call function provided in the BIOS. Programmers can determine the location of this table by using an EXTENDED BIOS CALL 0 and 1.

The RS-232C driver is addressed by device number 8. The functions for the RS-232C driver supported by extended BIOS calls are described below.

Build a Slot Address Table Entry to the Jump Table

Number:	0
Function:	Builds entry address to the jump table in the device driver pointed by [HL].
Entry:	[B] = Slot address of table entry for the device driver [HL] = Points to table entry for this device driver
Exit:	[B] = Slot address of next table entry [HL] = Points to next table entry
Description:	The RS-232C device driver call routine can issue this function call to determine the slot number and the location of the jump table to access the device driver for the desired channel.



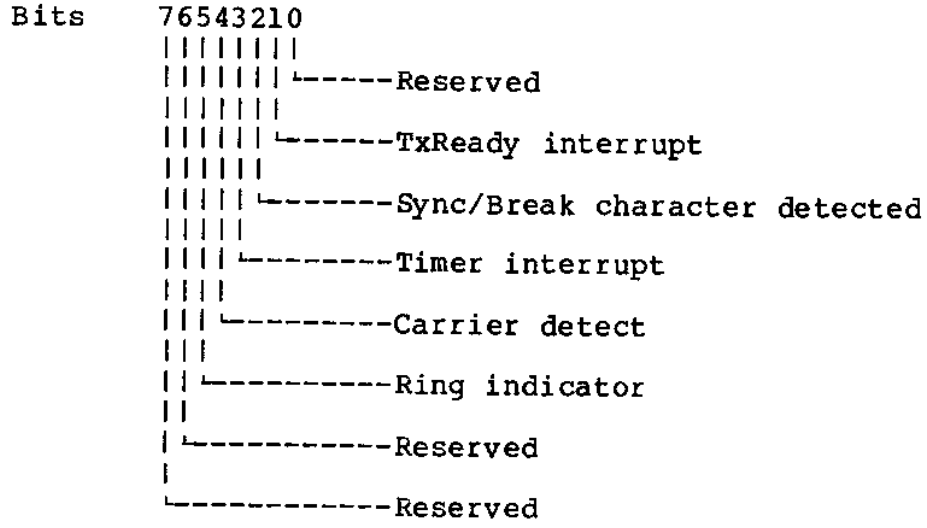
% The format of the slot address is same to the MSX standard notation, which is:

```
Bit # 76543210
      F000SSPP
      |   ||||
      |   ||└┴── Primary slot (0-3)
      |   └┴──── Secondary slot (0-3)
      └──────── 1 if secondary slot specified
```

Return Number Of Channels

Number: 1
Function: Returns the number of channels available to the device driver.
Entry: [A] = Contains number of RS-232 channels so far.
Exit: [A] = Number of RS-232 channels updated.
Description: This function is provided for each RS232C driver so as to find the channel number for the driver. Each driver can call this function to get the number of RS-232C channels installed so far.

The device information byte indicates whether the following options are installed or not:



The RS-232C driver has entries as follows. Application programs can use the RS-232C driver by an 'inter-slot call' to those entries.

```
EXBTBL: DEFB    DVINFB,0,0      ; Device information
        JP      INIT            ; Initialize RS-232C port
        JP      OPEN            ; Open RS-232C port
        JP      STAT            ; Read status
        JP      GETCHR          ; Receive data
        JP      SNDCHR          ; Send data
        JP      CLOSE           ; Close RS-232C port
        JP      EOF             ; EOF code received
        JP      LOC             ; Reports the number of characters in
                                ;  the receiver buffer
        JP      LOF             ; Reports the number of free spaces
                                ;  left in the receiver buffer
        JP      BACKUP          ; Back up a character
        JP      SNDBRK          ; Send break character
        JP      DTR             ; Turn on/off DTR line
        NOENT                   ; Reserved for future expansion
        NOENT
        NOENT
```

NOTE

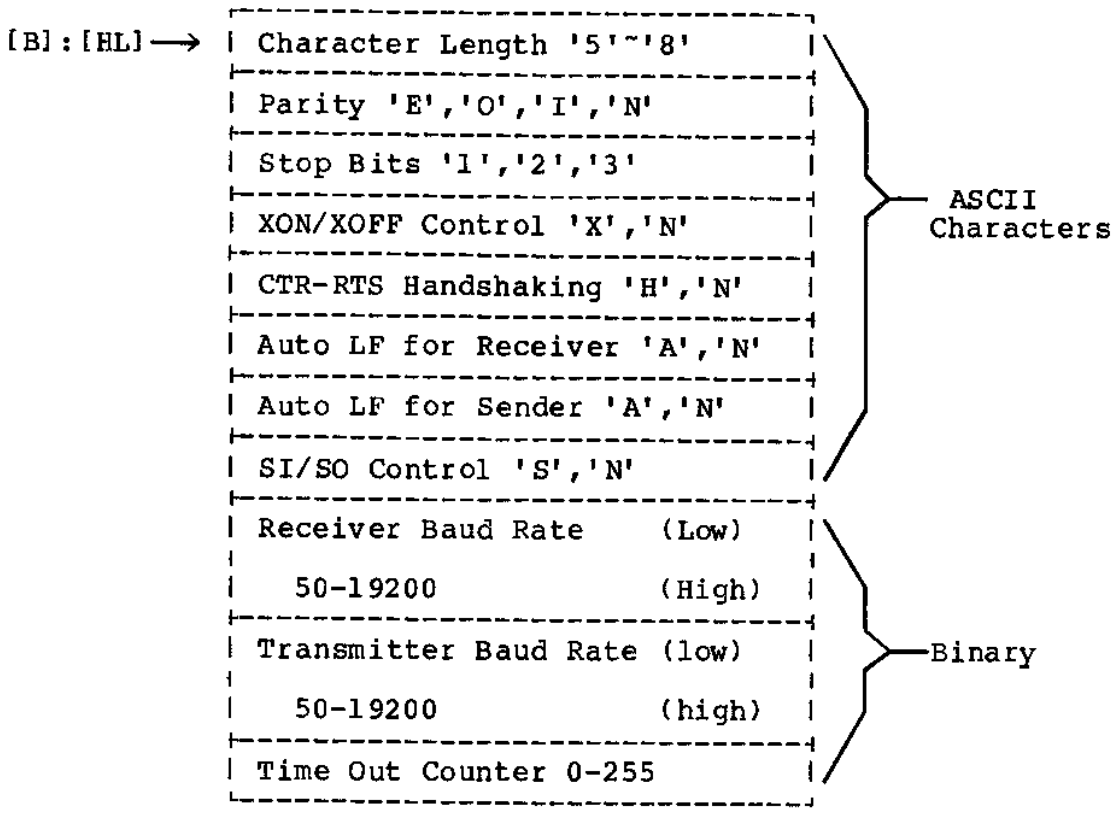
The RS-232C receiver is driven by the interrupt generated by the receiver ready. However, the inter-slot call handler disables the interrupt automatically. Thus, when control returns to the application program, it must enable an interrupt as soon as possible, or the RS-232C receiver routine will lose some of the characters.

Description of each Extended BIOS Call

1) Initialize RS-232C Port (INIT)

Entry: [HL]= Address of the parameter table
[B] = Slot address of the parameter table
Return: The carry flag is set if illegal parameters are set.
Modify: [AF]

Description:
Initializes the RS-232C port with the specified parameters. This entry must be called before any other function calls are made. The parameters are similar to the _COMINI expanded statement of BASIC. However, note that all the ASCII parameters must be specified with uppercase characters only. (See section 4.1.1 CALL COMINI for details of BAUD RATE and TIME OUT.)



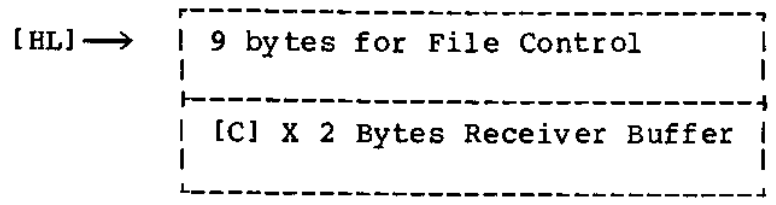
2) Open RS-232C port (OPEN)

Entry: [HL]= Address of FCB (must be located higher address than 8000H)
[C] = Buffer length (32~254)
[E] = Open mode (one of following):

Open Mode	Meaning
1	<Input> mode
2	<Output> mode
4	<Raw> and <Input/Output> mode

Returns: The carry flag is set if an error occurs.
Modifies: [AF]

Description:
Opens the RS-232C port with the specified File Control Block (FCB). An Open must be executed before any I/O operations can be done. Each character received occupies two bytes in the buffer. One is the received character code itself and the other is the error status of the received character. An extra 9 bytes are necessary for the working storage for file control. Note that the buffer length passed by [C] specifies the number of characters, so the actual length of buffer is [C] x 2 + 9 bytes. This buffer area can also be accessed without slot handling whenever the RS232C driver is called (including the timing when the interrupt from the receiver is generated).



3) Read Status (STAT)

Entry: None
Returns: [HL]= Status Data
Modifies: None

Description:
Returns the status information and error code of the character just read from the buffer (not the character just received).

BIT NO.	Description
15	Buffer over flow error 0: No buffer over flow 1: Buffer over flow
14	Time out error (TMENBT) 0: No time out error occurred 1: Time out error occurred
13	Framing error 0: No framing error occurred 1: Framing error occurred
12	Over run error 0: No over run error occurred 1: Over run error occurred
11	Parity error 0: Character has no parity error 1: Character has parity error
10	Control break key was pressed (BRONBT) 0: Control-break was not pressed 1: Control-break was pressed
9	Reserved
8	Reserved
7	Clear To Send 0: False 1: True
6	Timer/Counter Output-2 0: Timer/Counter Output-2 negated 1: Timer/Counter Output-2 asserted
5	Reserved
4	Reserved
3	Data Set Ready 0: False 1: True
2	Break Detect 0: Not detected 1: Detected
1	Ring Indicator 0: False 1: True
0	Carrier Detect 0: False 1: True

4) Get A Character From The Receive Buffer (GETCHR)

Entry: None
Returns: [A] = character received
The sign flag is set if any error occurred.
The carry flag is set if the character is an EOF code when the port is opened for input mode.
Modifies: [F]

Description:
Gets a character from the receiver buffer. Returns backed up character if any.

5) Send A Character To The RS-232C Port (SNDCHR)

Entry: [A] = Character to send
Returns: The carry flag is set if a control-break was entered.
The zero flag is set if a time out error has occurred while waiting for XON or/and CTS signal.
Modifies: [F]

Description:
Sends the specified character to the RS-232C port. The character flow control by XON/OFF characters and/or the CTS (Clear To Send) line signal is handled if they had been initialized. A time out error will be generated if the specified time has elapsed while waiting for transmission permission, and the character will not be sent.

6) Close The RS-232C Port (CLOSE)

Entry: None
Returns: The carry flag is set if an error occurs,
Modifies: [AF]

Description:
Closes the RS232C port. The buffer is released, and a EOF code is sent if the port was opened for <output> mode. The RTS signal is placed in an inactive state.

7) Check For The EOF Code (EOF)

Entry: None
Returns: [HL]= -1, carry flag set, if next character is EOF.
= 0, carry flag reset, if next character not EOF.
Modifies: [AF]

Description:
Tests whether the next character is an EOF or not. Returns 0 if the next character is not EOF.

8) Returns The Number Of Characters In The Receive Buffer (LOC)

Entry: None
Returns: [HL] = Number of character in the receiver buffer.
Modifies: [AF]

Description:
Returns the number of valid characters in the receive buffer. This value includes number of backed-up characters. The characters after the EOF code are ignored if the transmission was opened in the <input> mode; however, they will occupy space in the buffer.

9) Returns Number Of Free Space In The Receive Buffer (LOF)

Entry: None
Returns: [HL]= number of free space
Modifies: [AF]

Description:
Returns the number of free spaces for characters in the receiver buffer.

10) Back Up A Character (BACKUP)

Entry: [C] = Character to back up
Returns: None
Modifies: [F]

Description:
Backs up a character in the special buffer. Last backed up character will be lost if any.

11) Send Break Character (SNDBRK)

Entry: [DE] = Number of break characters to send
Returns: The carry flag is set if control break key was pressed.
Modifies: [AF], [DE]

Description:
Transmits the specified number of break characters. Aborts if a Control-Break is entered during the transmission and returns with the carry flag set.

12) Turn On/Off DTR Line (DTR)

Entry: [A] = 0 to turn off
[A] = Non-zero to turn on
Returns: None
Modifies: [F]

Description:
The DTR (Data terminal Ready) line is turned on when a power-on /reset initialize or an INIT routine is called.

```
**************************************************
*                      NOTE                      *
*                                                *
* Stack pointer must be located in PAGE-3 (higher *
* address than 0C000H). No registers except those *
* described here should be changed.              *
**************************************************
```

4.2 Other MSX Extended BIOS Calls

The extended BIOS call provides a way to access the extended device drivers via an additional HOOK entry. The device type is specified by register D, and the function of the call is specified by register E. To build a link of an extended BIOS call, each device driver should nest the Hook properly.

The address of the Hook for the extended BIOS call is: 0FFCAH. The flag bit which indicates whether the HOOK is valid or not is: LSB of 0FB20H.

NOTE

> The stack pointer must be located in PAGE-3 (addresses higher than 0C000H). No other registers except those described here should be changed.

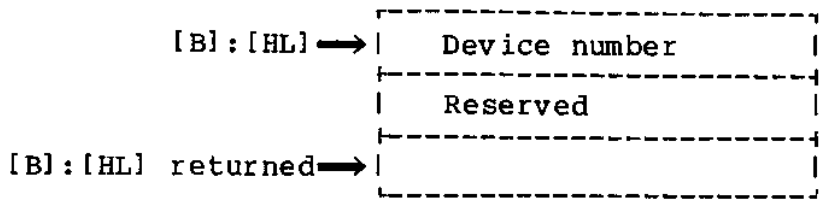
4.2.1 Extended BIOS Calls

1) Broad Cast Command

If the device number specified by register [D] is 0, this call should handle all extended device drivers added to the system.

Build Device Name Table

Number:	0
Function:	Build table which contains device number.
Entry:	[B] = Slot address of table entry for the device driver. [HL] = Points to table entry for the device driver
Exit:	[B] = Slot address of next table entry [HL] = Points to next table entry
Description:	Using this call, the user is provided information on the type of device driver that must be installed in the system. To obtain detailed information, such as the slot number and the address to access driver, issue a call with the device number in [D] and the function number (which is zero) in [E].



Return Number Of Trap Entries Used

Number: 1
Function: Adds number of traps used in device driver to [A].
Entry: [A] = Contains number of traps used by extended device driver so far.
Exit: [A] = Number of traps updated
Description: There is a limited number (six) of flags for the event trap function. This call is provided to determine the flag to use with this device.

Disable Interrupt

Number: 2
Function: Disables device driver interrupts.
Entry: None
Exit: None
Description: This function call is provided to inhibit interrupts. This feature is useful for improving the interrupt service response time or to inform the interrupt-drive routine that the DI instruction is going to issue.

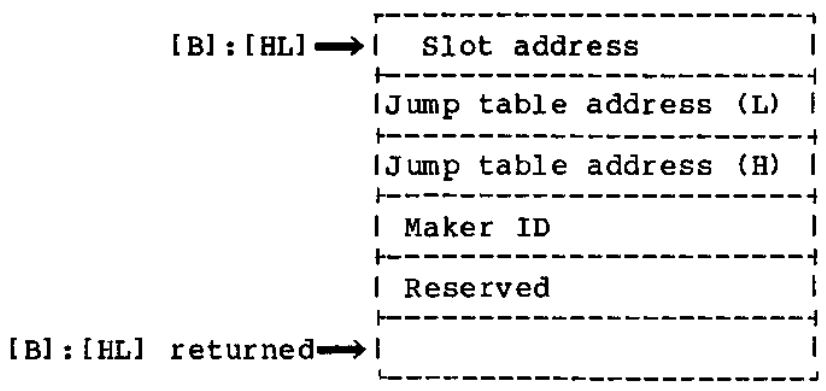
Enable Interrupt

Number: 3
Function: Enables device driver interrupts.
Entry: None
Exit: None
Description: This function call is provided to allow device drivers to generate interrupts.

2) System Exclusive Extended BIOS Call

This call is provided to allow the installation of special system software for proprietary use. The sole function specified follows. All other functions are not specified.

Device number:255
Number: 0
Function: Builds a table containing the pointer to the BIOS functions and device information.
Entry: [B] = Slot address of table entry for the device driver.
[HL] = Points to table entry for this device driver.
Exit: [B] = Slot address of next table entry
[HL] = Points to next table entry
Description: The caller of the device driver can issue this function call to determine the slot number and the location of the jump table access the device driver.



NOTE

The Maker ID is assigned in response to requests. Manufacturers who provide unique Maker IDs must also provide the BIOS specifications to the public.

3) Summary of Extended BIOS Calls

[D]	Description	
	[E]	Broad cast
0	0	Build device name table
	1	Return number of Trap Entries used
	2	Disable interrupt
	3	Enable interrupt
	[E]	RS-232C
8	0	Build a slot address table
	1	Return number of channels
	[E]	System exclusive
255	0	Build a slot address table

4.2.2 Extended BIOS Maker ID Number

The Maker ID is assigned in response to requests. A computer manufacturer not providing a Maker ID listed below must also provide the BIOS specification to the public.

ID code	Name of manufacturer
0	ASCII
1	MICROSOFT
2	CANON
3	CASIO KEISANKI
4	FUJITSU
5	GENERAL
6	HITACHI SEISAKUSYO
7	KYOCERA
8	MATSUSHITA DENKI
9	MITSUBISHI DENKI
10	NIHON DENKI
11	NIHON GAKKISEIZOU
12	NIHON VICTOR
13	PHILLIPS
14	PIONEER
15	SANYO
16	SHARP
17	SONY
18	SPECTRAVIDEO
19	TOSHIBA
20 *	MITSUMI DENKI

* Added on August 21, 1984

4.3 Tenkey Support on MSX

The standard MSX uses nine rows of a key matrix, but two more rows (Y9 and Y10) can be added to support an additional sixteen keys. The following is a list of the assignments of the additional keys.

	X7	X6	X5	X4	X3	X2	X1	X0
Y 9	4	3	2	1	0	Option	Option	Option
Y10	. (Period)	, (Comma)	- (Minus)	9	8	7	6	5

(The Option keys may be used for any purpose.)

PART D

SOFTWARE DEVELOPMENT GUIDE

5. International MSX Versions and their Differences

5.1 Introduction

At present, the MSX computer has the following versions. At a later time, it is possible that other versions will be released for other countries.

Japanese
USA
International (abbreviated INT in this document)
UK
DIN
French
Korea

5.2 Keyboard

5.2.1 Keyboard Hardware

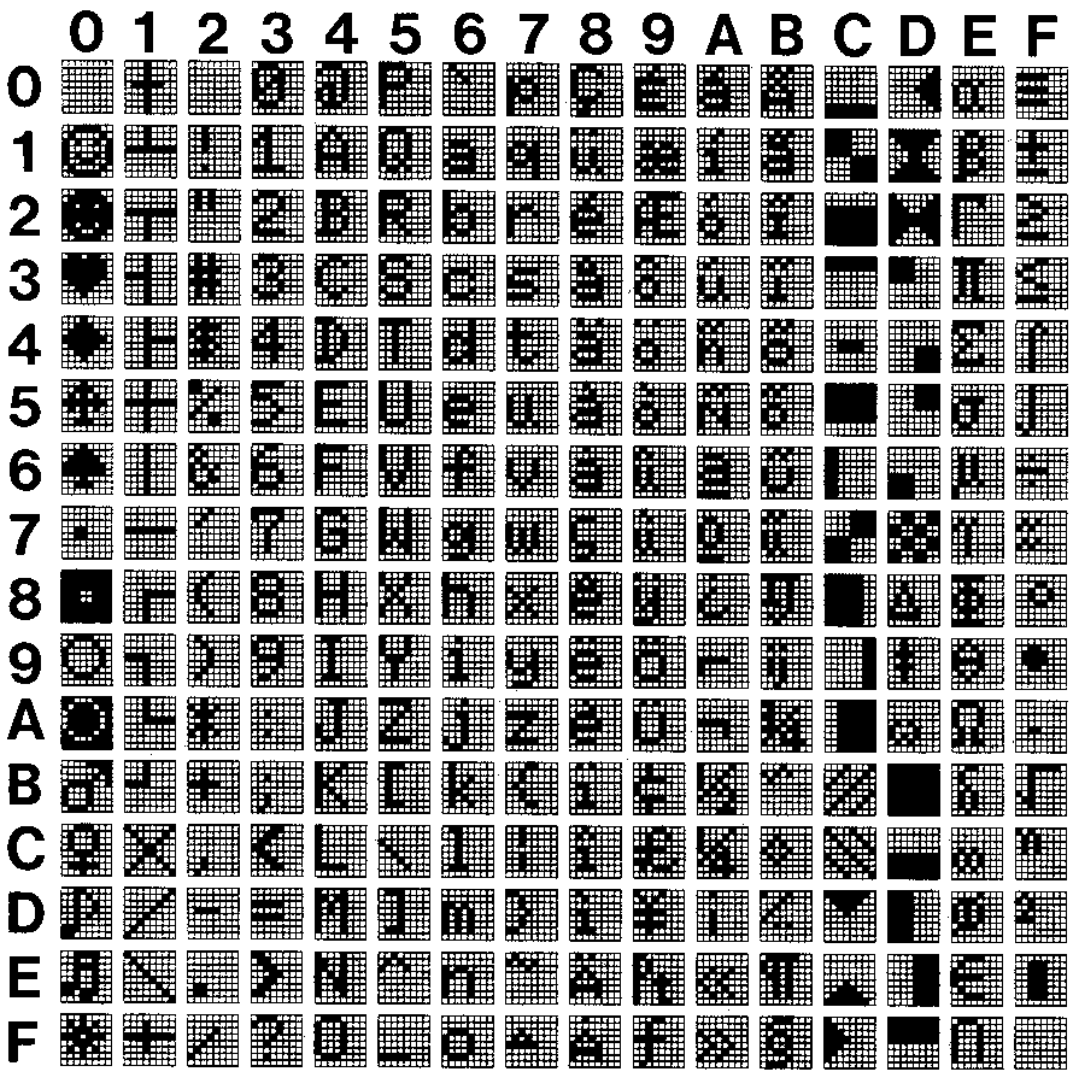
The KANA key of the Japanese version toggles Kana mode and alpha mode, but the CODE key of international versions, while occupy ing the same position on the keyboard matrix, the keyboard input mode for the entry of the next key. Thus, the LED to indicate the CODE shift status is unnecessary.

Three keys are pressed simultaneously in the Shift-Graph and Shift-Code modes. Using ordinary keyboard sense techniques, the SHIFT, GRAPH, and CODE shift keys must have a diode to prevent the loopback current that causes scanning conflicts.

5.2.2 Character Set

The USA, INT, UK, DIN, and French versions have a common international character set.

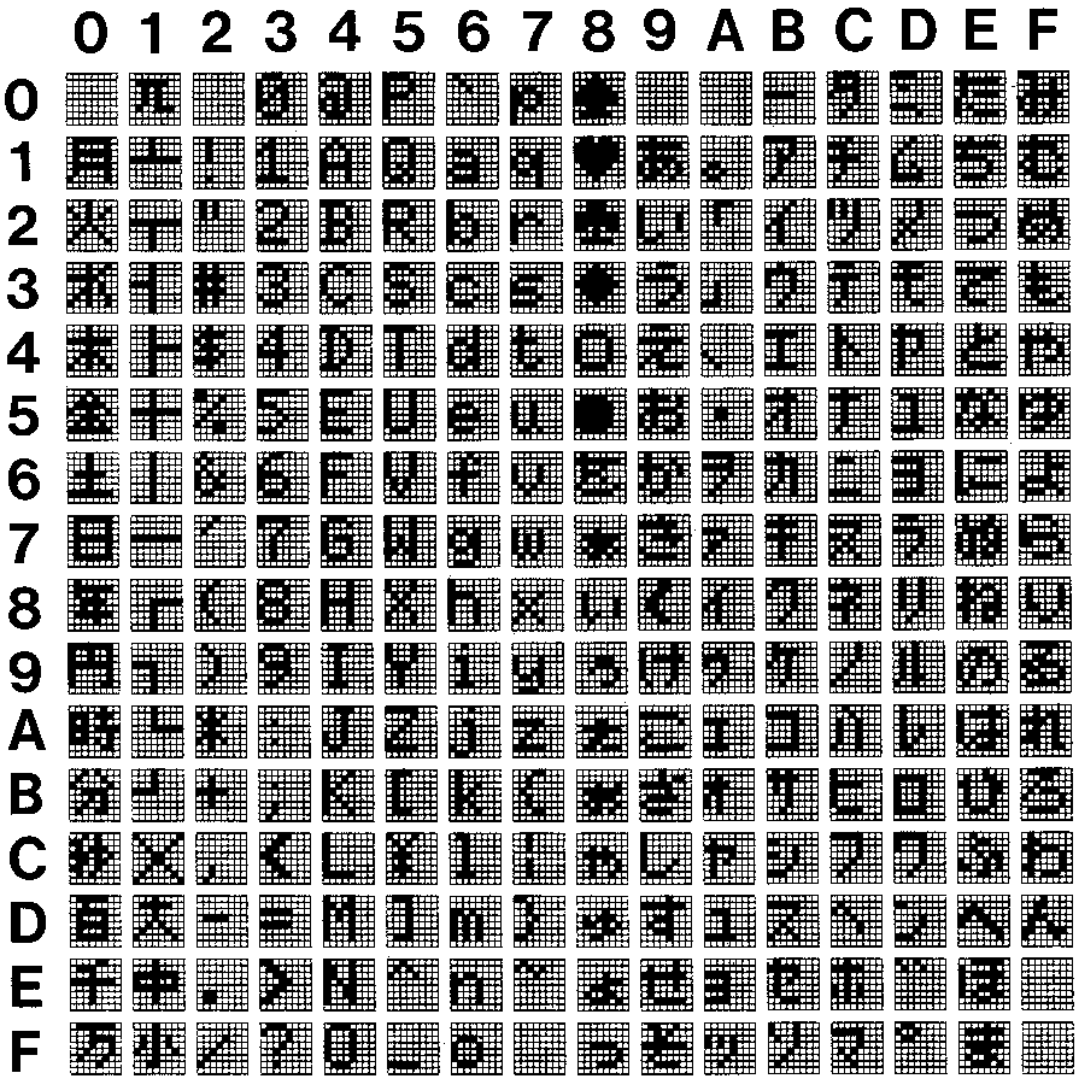
- Character Code Table (International)



NOTE: The font of the character '0' (zero) is different in DIN version. See figure.

```
 ***
*   *
*   *
* * *
*   *
*   *
 ***
```

o Character Code Table (Japanese)



5.2.3 Keyboard Layout

See the figures in the next section. The USA and INT versions have the same keyboard.

About USA, UK, INT versions:
The keyboard diagrams show a dead-key to the left of the carriage return key, but this is probably not a good place for it, because it pushes the carriage return key too far to the right. Manufacturers may place this key another place, for example, the right of the right-hand shift key.

About DIN, French versions :
Manufacturers may move the less than and greater than keys (<, >) to the left of the left-hand shift key, but must also revise the keyboard hardware.

5.2.4 CAPS Lock

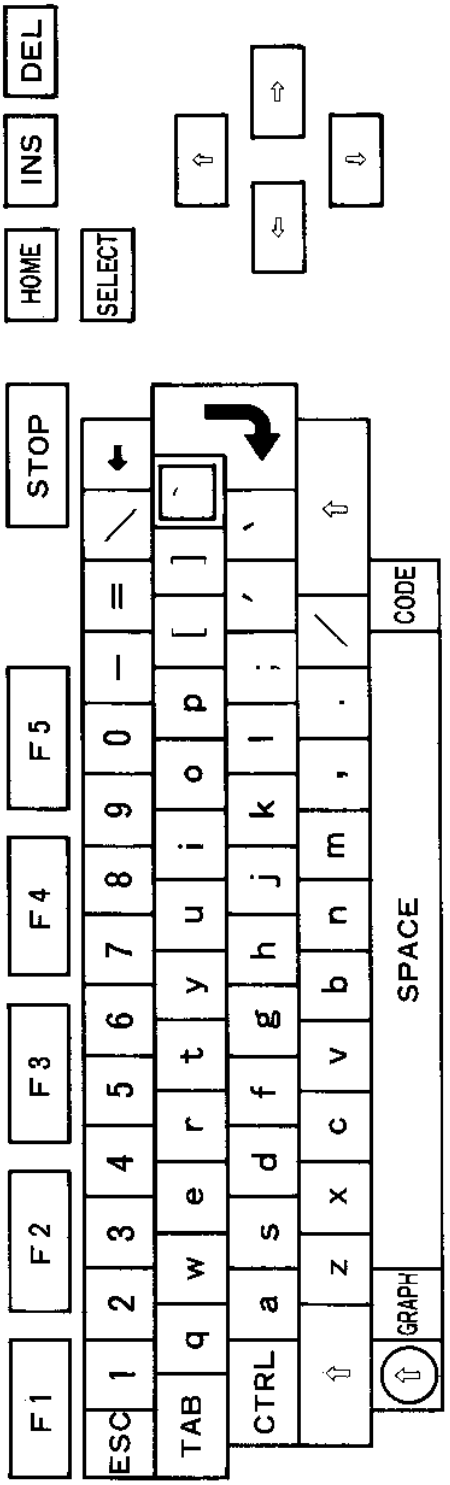
In the CAPS-lock mode, the uppercase of characters (having both a lowercase and an uppercase) is entered. In the CAPS-lock mode of the Japanese and French versions, when the shift key and an alphabet key are pressed, the lowercase letter is entered. When this is done in other (USA, UK, DIN, and INT) versions, an uppercase letter is entered.

For the French version, see figure. The marked keys in the figure are shifted by the CAPS-lock. The CAPS-lock is not valid for the graphics and code characters.

KANA characters in the Japanese version are valid when the KANA-Lock key is valid. Normally HIRAGANA charcters are entered, and with the CAPS-Lock key together, KATAKANA characters are entered. Most of them are not affected by the SHIFT key. However, some of the KANA characters have both upper- and lowercase letters and are shifted by the SHIFT key. Notice the differences in the decoding charts.

INTERNATIONAL MSX VERSIONS

o Layout International (USA)



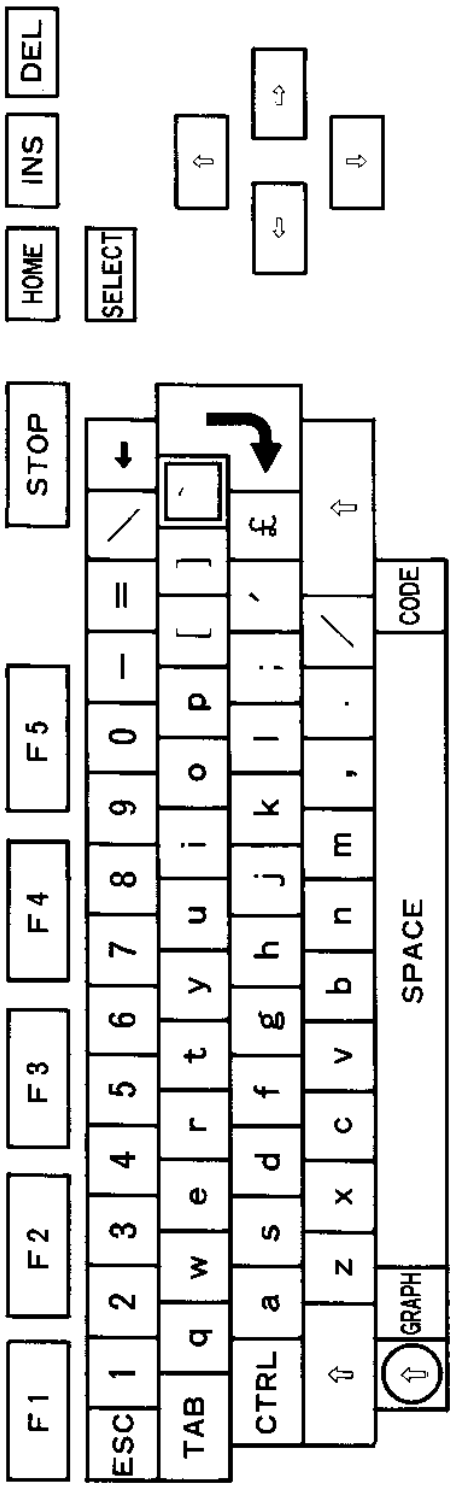
INTERNATIONAL MSX VERSIONS

o Decode International (USA)

INT			0	1	2	3	4	5	6	7
0	Normal		0 30	1 31	2 32	3 33	4 34	5 35	6 36	7 37
		Shift	) 29	! 21	@ 40	# 23	$ 24	% 25	^ 5E	& 26
	Graph		○ 09	¼ AC	½ AB	¾ BA	⁄ EF	‰ BD	ƒ F4	√ FB
		Shift	◙ 0A		² FD	³ FC			∫ F5	
	Code		δ EB	ƒ 9F	‡ D9	§ BF	¢ 9B	ÿ 98	α E0	β E1
		Shift	Δ D8	¡ AD	Pt 9E	¶ BE	£ 9C	Y 9D		
1	Normal		8 38	9 39	− 2D	= 3D	\ 5C	[5B	| 5D	; 3B
		Shift	* 2A	(28	_ 5F	+ 2B	¦ 7C	{ 7B	} 7D	: 3A
	Graph		▽ EC	• 07	— 17	± F1	╲ 1E	☺ 01	♪ 0D	♠ 06
		Shift		■ 08	┼ 1F	≡ F0	| 16	☻ 02	♬ 0E	♦ 04
	Code		⁄ E7	ç 87	ε EE	θ E9		φ ED	ω DA	ū B7
		Shift	Γ E2	Ç 80				Φ E8	Ω EA	Ũ B6
2	Normal		' 27	` 60	, 2C	. 2E	/ 2F	~ dead key	a 61	b 62
		Shift	" 22	~ 7E	< 3C	> 3E	? 3F	´	A 41	B 42
	Graph		♣ 05	BB	≤ F3	≥ F2	⁄ 1D	`	▬ C4	┴ 11
		Shift	♥ 03	≈ F7	《 AE	》 AF	÷ F6	¨	▮ FE	
	Code		ij B9	σ E5	å 86	a̲ A6	º A7	^	ä 84	û 97
		Shift	IJ B8	Σ E4	Å 8F		¿ A8	¨	Ä 8E	
3	Normal		c 63	d 64	e 65	f 66	g 67	h 68	i 69	j 6A
		Shift	C 43	D 44	E 45	F 46	G 47	H 48	I 49	J 4A
	Graph		◇ BC	▟ C7	▼ CD	├ 14	┼ 15	┤ 13	▄ DC	▌ C6
		Shift	FA	▙ C1	▲ CE	▪ D4	┼ 10	▪ D6	▀ DF	▉ CA
	Code		ì 8D	ï 8B	î 8C	ö 94	ü 81	ã B1	í A1	æ 91
		Shift				Ö 99	Ü 9A	Ã B0		Æ 92
4	Normal		k 6B	l 6C	m 6D	n 6E	o 6F	p 70	q 71	r 72
		Shift	K 4B	L 4C	M 4D	N 4E	O 4F	P 50	Q 51	R 52
	Graph		▌ DD	▉ C8	♂ 0B	┘ 1B	▬ C2	▉ DB	╲╲ CC	┌ 18
		Shift	▌ DE	▌ C9	♀ 0C	▪ D3	▬ C3	▚ D7	╱╱ CB	┌ A9
	Code		ï B3	o B5	µ E6	ñ A4	ó A2	ú A3	â 83	ô 93
		Shift	I B2	Õ B4		Ñ A5		II E3		
5	Normal		s 73	t 74	u 75	v 76	w 77	x 78	y 79	z 7A
		Shift	S 53	T 54	U 55	V 56	W 57	X 58	Y 59	Z 5A
	Graph		▚ D2	┬ 12	▬ C0	└ 1A	▶ CF	× 1C	┐ 19	☼ 0F
		Shift	▙ D1		▉ C5	▪ D5	◀ D0	● F9	¬ AA	○ F8
	Code		ë 89	û 96	é 82	ò 95	ê 88	è 8A	á A0	à 85
		Shift			É 90					

INTERNATIONAL MSX VERSIONS

o Layout UK

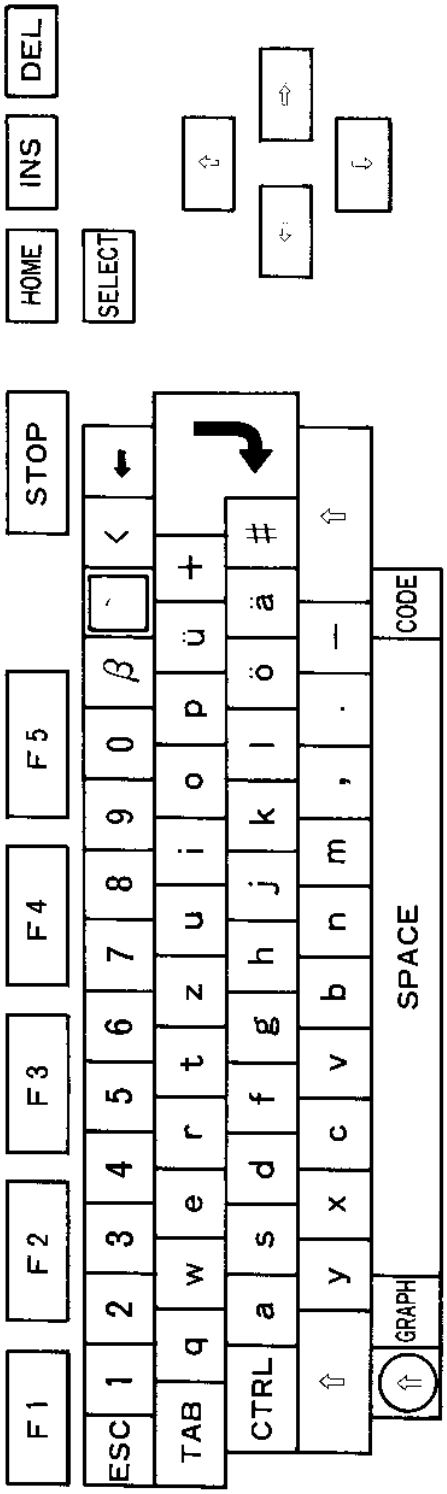


o Decode UK

UK			0	1	2	3	4	5	6	7
0	Normal		0 30	1 31	2 32	3 33	4 34	5 35	6 36	7 37
		Shift	) 29	! 21	@ 40	# 23	$ 24	% 25	^ 5E	& 26
	Graph		○ 09	¼ AC	½ AB	¾ BA	η EF	‰ BD	ƒ F4	√ FB
		Shift	◙ 0A		² FD	ª FC			∫ F5	
	Code		∂ EB	ƒ 9F	‡ D9	§ BF	¢ 9B	ÿ 98	α E0	β E1
		Shift	Δ D8	¡ AD	Pt 9E	¶ BE	£ 9C	¥ 9D		
1	Normal		8 38	9 39	− 2D	– 3D	\ 5C	[5B	] 5D	; 3B
		Shift	* 2A	(28	_ 5F	+ 2B	¦ 7C	¦ 7B	¦ 7D	: 3A
	Graph		∞ EC	• 07	— 17	± F1	\ 1E	☺ 01	♪ 0D	♠ 06
		Shift		◘ 08	┼ 1F	≡ F0	| 16	☻ 02	♫ 0E	◆ 04
	Code		γ E7	ç 87	ε EE	θ E9	` 60	φ ED	ω DA	ü B7
		Shift	Γ E2	Ç 80				Φ E8	Ω EA	Ü B6
2	Normal		' 27	£ 9C	, 2C	. 2E	/ 2F	dead key	a 61	b 62
		Shift	" 22	~ 7E	< 3C	> 3E	? 3F		A 41	B 42
	Graph		♣ 05	⌒ BB	≤ F3	≥ F2	⁄ 1D		▬ C4	⊥ 11
		Shift	♥ 03	≈ F7	◊ AE	∫ AF	÷ F6		▌ FE	
	Code		ij B9	σ E5	å 86	a A6	o A7	¨	ä 84	ù 97
		Shift	IJ B8	Σ E4	Å 8F		¿ A8	¨	A 8E	
3	Normal		c 63	d 64	e 65	f 66	g 67	h 68	i 69	j 6A
		Shift	C 43	D 44	E 45	F 46	G 47	H 48	I 49	J 4A
	Graph		◇ BC	▟ C7	▼ CD	├ 14	┼ 15	┤ 13	▄ DC	▌ C6
		Shift	- FA	▙ C1	▲ CE	▪ D4	┼ 10	■ D6	▀ DF	▉ CA
	Code		ì 8D	ï 8B	î 8C	ö 94	ù 81	à B1	í A1	æ 91
		Shift				Ö 99	Ù 9A	Ã B0		Æ 92
4	Normal		k 6B	l 6C	m 6D	n 6E	o 6F	p 70	q 71	r 72
		Shift	K 4B	L 4C	M 4D	N 4E	O 4F	P 50	Q 51	R 52
	Graph		▌ DD	▉ C8	♂ 0B	┘ 1B	▄ C2	▉ DB	╲╲ CC	┌ 18
		Shift	▐ DE	▏ C9	♀ 0C	▪ D3	▬ C3	▚ D7	╱╱ CB	⌐ A9
	Code		ì B3	õ B5	µ E6	ñ A4	ó A2	ú A3	â 83	ô 93
		Shift	Ì B2	Õ B4		Ñ A5		Û E3		
5	Normal		s 73	t 74	u 75	v 76	w 77	x 78	y 79	z 7A
		Shift	S 53	T 54	U 55	V 56	W 57	X 58	Y 59	Z 5A
	Graph		⋈ D2	┬ 12	▬ C0	└ 1A	▶ CF	× 1C	┐ 19	☼ 0F
		Shift	⧗ D1		▉ C5	▪ D5	◀ D0	● F9	¬ AA	○ F8
	Code		ë 89	û 96	é 82	ò 95	ê 88	è 8A	á A0	à 85
		Shift			É 90					

INTERNATIONAL MSX VERSIONS

o Layout DIN

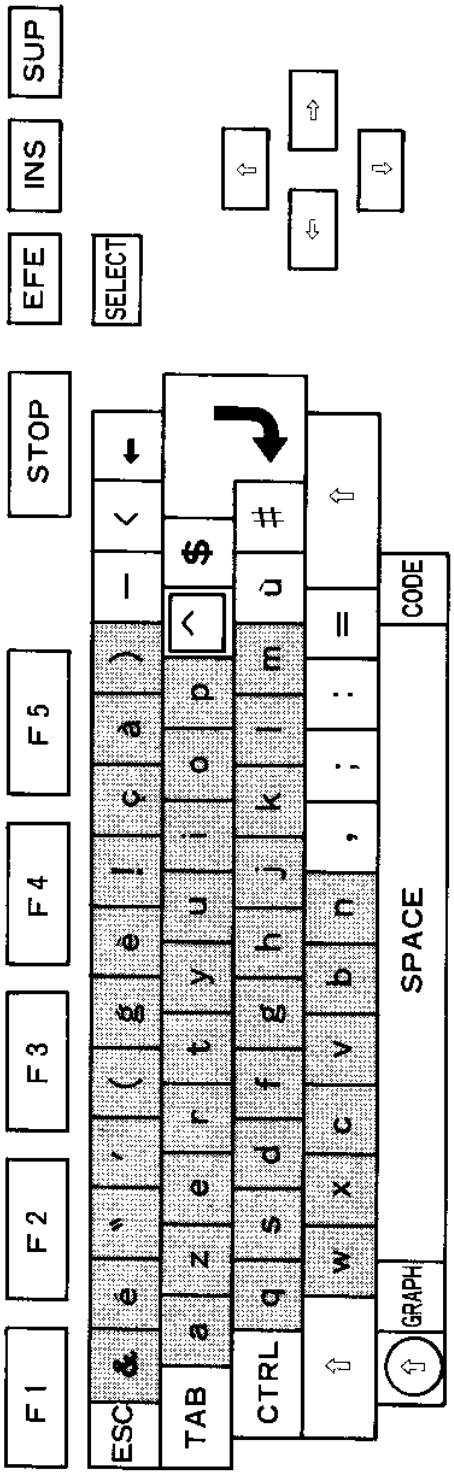


o Decode DIN

DIN			0	1	2	3	4	5	6	7
0	Normal		0 30	1 31	2 32	3 33	4 34	5 35	6 36	7 37
		Shift	= 3D	! 21	" 22	§ BF	$ 24	% 25	& 26	/ 2F
	Graph		○ 09	¼ AC	½ AB	¾ BA	ŋ EF	‰ BD	∫ F4	⁄ 1D
		Shift	◙ 0A		² FD	³ FC		÷ F6	∫ F5	\ 1E
	Code		δ EB	~ 7C	@ 40	€ EE	ç 87	¢ 9B	γ E7	\ 5C
		Shift	Δ D8	¡ AD	Pt 9E	¶ BE	Ç 80	£ 9C	Γ E2	
1	Normal		8 38	9 39	β E1	` dead key	< 3C	ü 81	+ 2B	ö 94
		Shift	(28	) 29	? 3F	´ dead key	> 3E	Ü 9A	* 2A	Ö 99
	Graph		∞ EC	• 07	♪ 0D	` 60	《 AE	☺ 01	± F1	♠ 06
		Shift		■ 08	♬ 0E	´ 27	》 AF	☻ 02	┼ 1F	◆ 04
	Code		[5B	] 5D	θ E9	^ dead key	≦ F3	ϕ ED	ω DA	ü B7
		Shift	| 7B	| 7D	¿ A8	¨ dead key	≧ F2	Φ E8	Ω EA	Ü B6
2	Normal		ä 84	# 23	, 2C	. 2E	− 2D		a 61	b 62
		Shift	Ä 8E	^ 5E	; 3B	: 3A	_ 5F		A 41	B 42
	Graph		♣ 05	~ 7E	√ FB	| 16	─ 17		▬ C4	┴ 11
		Shift	♥ 03	‾ BB	≈ F7		≡ F0		▌ FE	
	Code		ij B9	σ E5	å 86	a A6	o A7		α E0	ù 97
		Shift	IJ B8	Σ E4	Å 8F					
3	Normal		c 63	d 64	e 65	f 66	g 67	h 68	i 69	j 6A
		Shift	C 43	D 44	E 45	F 46	G 47	H 48	I 49	J 4A
	Graph		◇ BC	▟ C7	▼ CD	├ 14	┼ 15	┤ 13	▄ DC	▌ C6
		Shift	· FA	▙ C1	▲ CE	■ D4	┼ 10	■ D6	▄ DF	▉ CA
	Code		ì 8D	ï 8B	î 8C	ƒ 9F	y 98	ã B1	í A1	æ 91
		Shift						Ã B0		Æ 92
4	Normal		k 6B	l 6C	m 6D	n 6E	o 6F	p 70	q 71	r 72
		Shift	K 4B	L 4C	M 4D	N 4E	O 4F	P 50	Q 51	R 52
	Graph		▌ DD	▉ C8	♂ 0B	┘ 1B	▄ C2	▄ DB	╲ CC	┌ 18
		Shift	▌ DE	▉ C9	♀ 0C	■ D3	▬ C3	▚ D7	╱ CB	┌ A9
	Code		ĩ B3	õ B5	μ E6	ñ A4	ó A2	ú A3	á 83	ô 93
		Shift	Ĩ B2	Õ B4		Ñ A5		Π E3		
5	Normal		s 73	t 74	u 75	v 76	w 77	x 78	z 7A	y 79
		Shift	S 53	T 54	U 55	V 56	W 57	X 58	Z 5A	Y 59
	Graph		⋈ D2	┬ 12	▬ C0	└ 1A	▶ CF	× 1C	┐ 19	☼ 0F
		Shift	⧗ D1	‡ D9	▄ C5	■ D5	◀ D0	● F9	┐ AA	○ F8
	Code		ë 89	û 96	é 82	ò 95	ê 88	è 8A	à A0	a 85
		Shift			É 90					¥ 9D

INTERNATIONAL MSX VERSIONS

o Layout French

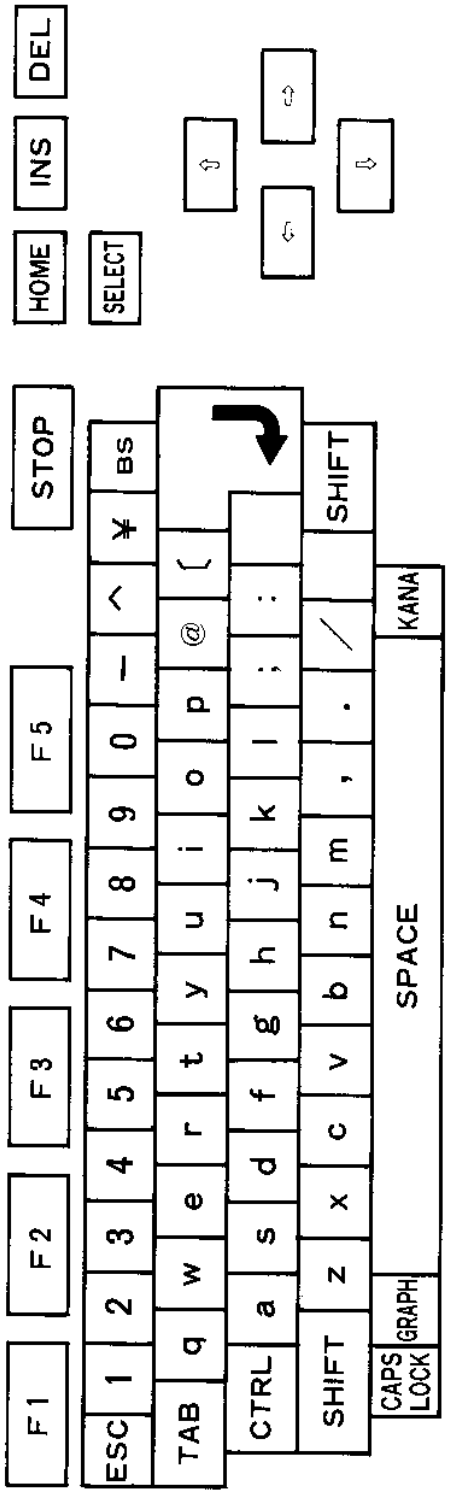


o Decode French

FR			0	1	2	3	4	5	6	7
0	Normal		à 85	& 26	é 82	' 22	' 27	(28	§ BF	è 8A
		Shift	0 30	1 31	2 32	3 33	4 34	5 35	6 36	7 37
	Graph		○ 09	£ AC	½ AB	¼ BA	‾ BB	η EF	∫ F4	√ FB
		Shift	◙ 0A	¦ 16	² FD	ⁿ FC	≈ F7		∫ F5	
	Code		δ EB	7C	@ 40	α E0	` 60	¦ 7B	^ 5E	ϵ EE
		Shift	Δ D8	¡ AD	É 90	Pt 9E		[5B	¶ BE	~ 7E
1	Normal		' 21	ç 87	) 29	— 2D	< 3C	^ dead key	$ 24	m 6D
		Shift	8 38	9 39	○ F8	_ 5F	> 3E	¨ dead key	* 2A	M 4D
	Graph		∞ EC	• 07	☺ 01	— 17	《 AE	^ dead key	♪ 0D	♠ 06
		Shift		■ 08	☻ 02	┼ 1F	》 AF	¨ dead key	♬ 0E	◆ 04
	Code		γ E7	θ E9	| 7D	ϕ ED	≤ F3	^ dead key	¢ 9B	ũ B7
		Shift	Γ E2	Ç 80	] 5D	Φ E8	≧ F2	¨ dead key		Ũ B6
2	Normal		ù 97	# 23	; 3B	: 3A	= 3D		q 71	b 62
		Shift	% 25	£ 9C	. 2E	/ 2F	+ 2B		Q 51	B 42
	Graph		♣ 05	‰ BD	÷ F6	\ 1E	± F1		▬ C4	┴ 11
		Shift	♥ 03			／ 1D	≡ F0		▌ FE	
	Code		ij B9	σ E5	à 86	a A6	º A7		ä 84	β E1
		Shift	IJ B8	Σ E4	À 8F	\ 5C			Ä 8E	
3	Normal		c 63	d 64	e 65	f 66	g 67	h 68	i 69	j 6A
		Shift	C 43	D 44	E 45	F 46	G 47	H 48	I 49	J 4A
	Graph		◇ BC	▗ C7	▼ CD	├ 14	┼ 15	┤ 13	▄ DC	▌ C6
		Shift	- FA	▖ C1	▲ CE	▮ D4	╂ 10	▮ D6	▀ DF	▉ CA
	Code		ì 8D	ï 8B	î 8C	ö 94	ü 81	ã B1	í A1	æ 91
		Shift				Ö 99	Ü 9A	Ã B0		Æ 92
4	Normal		k 6B	l 6C	, 2C	n 6E	o 6F	p 70	a 61	r 72
		Shift	K 4B	L 4C	? 3F	N 4E	O 4F	P 50	A 41	R 52
	Graph		▌ DD	▉ C8	♂ 0B	┘ 1B	▬ C2	▉ DB	╲╲ CC	┌ 18
		Shift	▌ DE	▏ C9	♀ 0C	■ D3	▔ C3	▚ D7	╱╱ CB	┌ A9
	Code		ĩ B3	õ B5	μ E6	ñ A4	ó A2	ú A3	â 83	ô 93
		Shift	Ĩ B2	Õ B4	¿ A8	Ñ A5		Π E3		
5	Normal		s 73	t 74	u 75	v 76	z 7A	x 78	y 79	w 77
		Shift	S 53	T 54	U 55	V 56	Z 5A	X 58	Y 59	W 57
	Graph		⋈ D2	┬ 12	▂ C0	└ 1A	▶ CF	× 1C	┐ 19	☼ 0F
		Shift	⧗ D1	‡ D9	▇ C5	▮ D5	◀ D0	● F9	¬ AA	
	Code		ë 89	û 96	ÿ 98	ò 95	ê 88	ƒ 9F	á A0	ω DA
		Shift							¥ 9D	Ω EA

INTERNATIONAL MSX VERSIONS

o Layout Japanese



o Decode Japanese 1

JIS			0	1	2	3	4	5	6	7
0	Normal		0 30	1 31	2 32	3 33	4 34	5 35	6 36	7 37
		Shift		! 21	" 22	# 23	$ 24	% 25	& 26	' 27
	Graph		力 0F	口 07	月 01	火 02	水 03	木 04	金 05	土 06
	Kana		わ FC	ぬ E7	ふ EC	あ 91	う 93	え 94	お 95	や F4
		Caps	ワ DC	ヌ C7	フ CC	ア B1	ウ B3	エ B4	オ B5	ヤ D4
1	Normal		8 38	9 39	2D	^ 5E	¥ 5C	@ 40	[5B	; 3B
		Shift	(28	) 29	3D	~ 7E	| 7C	` 60	{ 7B	+ 2B
	Graph		百 0D	千 E0	— 17		円 09		○ 84	♣ 82
	Kana		ゆ F5	よ F6	は EE	へ ED	ー B0	゛ DE	° DF	れ FA
		Caps	ユ D5	ヨ D6	ホ CE	ヘ CD	・ B0	゛ DE	° DF	レ DA
2	Normal		: 3A	] 5D	, 2C	. 2E	/ 2F		a 61	b 62
		Shift	* 2A	} 7D	< 3C	> 3E	? 3F	_ 5F	A 41	B 42
	Graph		♥ 81	● 85	小 1F	大 1D	♠ 80	◆ 83		┘ 1B
	Kana		け 99	む F1	ね E8	る F9	め F2	ろ FB	ち E1	こ 9A
		Caps	ケ B9	ム D1	ネ C8	ル D9	メ D2	ロ DB	チ C1	コ BA
3	Normal		c 63	d 64	e 65	f 66	g 67	h 68	i 69	j 6A
		Shift	C 43	D 44	E 45	F 46	G 47	H 48	I 49	J 4A
	Graph		┘ 1A	├ 14	─ 18	┼ 15	┤ 13	時 0A	・ 16	
	Kana		そ 9F	し 9C	い 92	は EA	き 97	く 98	に E6	ま EF
		Caps	ソ BF	シ BC	イ B2	ハ CA	キ B7	ク B8	ニ C6	マ CF
4	Normal		k 6B	l 6C	m 6D	n 6E	o 6F	p 70	q 71	r 72
		Shift	K 4B	L 4C	M 4D	N 4E	O 4F	P 50	Q 51	R 52
	Graph			中 1E	分 0B			π 10		┌ 12
	Kana		の E9	り F8	も F3	み F0	ら F7	せ 9E	た E0	す 9D
		Caps	ノ C9	リ D8	モ D3	ミ D0	ラ D7	セ BE	タ C0	ス BD
5	Normal		s 73	t 74	u 75	v 76	w 77	x 78	y 79	z 7A
		Shift	S 53	T 54	U 55	V 56	W 57	X 58	Y 59	Z 5A
	Graph		秒 0C	- 19		┘ 11		× 1C	年 08	
	Kana		と E4	か 96	な E5	ひ EB	て E3	さ 9B	ん FD	つ E2
		Caps	ト C4	カ B6	ナ C5	ヒ CB	テ C3	サ BB	ン DD	ツ C2

o Decode Japanese 2

KANA+SHIFT		0	1	2	3	4	5	6	7
0		を 86			ぁ 87	ぃ 89	ぇ 8A	ぉ 8B	ゃ 8C
	Caps	ヲ A6			ァ A7	ウ A9	ェ AA	ォ AB	ャ AC
1		ゅ 8D	ょ 8E					「 A2	
	Caps	ュ AD	ョ AE					」 A2	
2			｜ A3	、 A4	。 A1	・ A5			
	Caps		」 A3	、 A4	。 A1	・ A5			
3				゛ 88					
	Caps			ィ A8					
5									っ 8F
	Caps								ッ AF

5.2.5 DEAD-KEY Functions

When an "a", "e", "i", "o", "u", or "y" key is entered after the SHIFT, GRAPHICS, CODE, or DEAD keys are entered, the accented character is entered instead. The dead-key is valid only for the "a", "e", "i", "o", "u", "y", and SPACE keys.

If a designated character does not exist in the character set, a normal (non-accented) character is entered. For example, when the dead key and a "Y" key of the international keyboard are pressed, an accent grave "y" is not entered, and a normal "y" is entered.

The dead-key is an optional provision. The dead-key is less useful in the French and German versions, where special keyboards must be used, and in English-speaking countries. Application programs that must use this dead-key are less compatible with other versions.

USA, UK, INT Versions

Mode	Function	
Normal	Accent grave	(`)
Normal shift	Accent egu	(')
Graph	Accent grave	(`)
Graph shift	Accent egu	(')
Code	Accent circonflex	(^)
Code shift	Umlaut	( )

INTERNATIONAL MSX VERSIONS

DIN version

Mode	Function	
Normal	Accent grave (`)	
Normal shift	Accent egu (')	
Graph	Accent grave (`)	*See Note 1.
Graph shift	Accent egu (')	*See Note 1.
Code	Accent circonflex (^)	
Code shift	Umlaut ( )	

*Note 1: In the DIN version, when the SHIFT, GRAPH, or DEAD keys are pressed, an accent sign without a letter is entered.

French version

Mode	function	
Normal	Accent circonflex (^)	
Normal shift	Umlaut ( )	
Graph	Accent circonflex (^)	
Graph shift	Umlaut ( )	
Code	Accent circonflex (^)	
Code shift	Umlaut ( )	

In the DIN and French versions, when the SPACE key is pressed either the SHIFT, GRAPH, CODE, or DEAD keys are pressed, an accent sign without alphabet is entered as follows.

Mode	DIN	French
Normal	Accent grave (`)	Accent circonflex (^)
Normal shift	Accent egu (')	Space
Graph	(*See Note 1.)	Accent circonflex (^)
Graph shift	(*See Note 1.)	Space
Code	Accent circonflex (^)	Accent circonflex (^)
Codeshift	Space	Space

5.3 Screen Mode

The vertical synchronize frequencies and the default screen modes of the different versions are as follows.

Version	V.Sync.	Default screen mode	Default border color	Default screen width	
				SCREEN 0	SCREEN 1
Japan	60Hz	1	7	39	29
USA					
UK					
DIN	50Hz	0	4	37	
French					
INT					

5.4 Other Differences among Versions

The default function for the F6 key differs as follows.

Japanese	color 15,4,7
Others	color 15,4,4

The Japanese version has a Hiragana-to-Katakana converter for non-MSX printers; however, other versions do not have this feature.

The format symbols for the PRINT USING statement that differ among international versions are as follows.

Purpose	Japanese	UK	Others
Currency sign	Yen ¥	Pound £	$
Fixed-length string field	&	\	
Variable-length string field	@	&	

The VDP interrupt interval is equal to the vertical synchronize frequency, or 1/60 second in the Japanese and USA versions, and 1/50 second in other versions. This has an effect on the interval to increment the TIME variable.

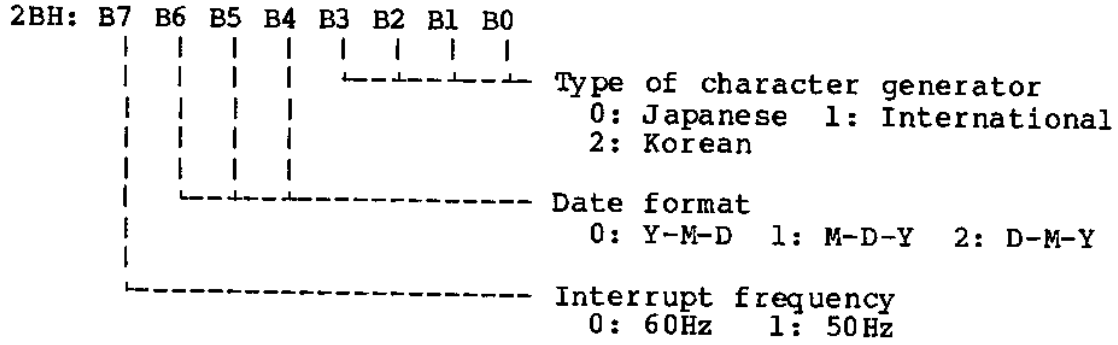
The symbol for integer division is the Yen sign in the Japanese version and "\" in all other versions.

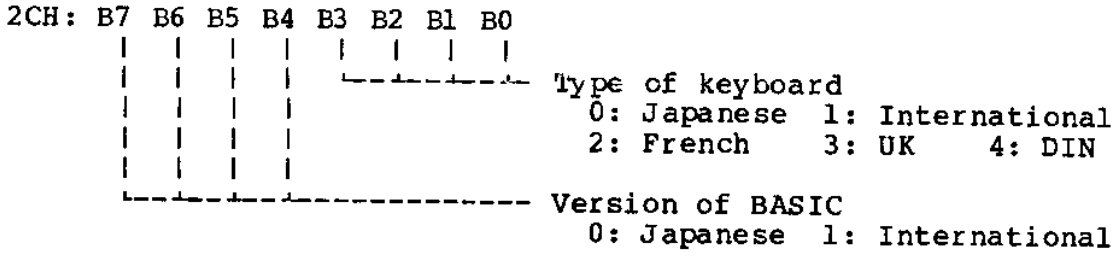
The format of DATE used for MSX-DOS is 'yy-mm-dd' (Year, Month, Date) in Japanese version, 'mm-dd-yy' in USA version and 'dd-mm-yy' in European versions.

5.5 ID Bytes

You can build software having compatibility with all MSX versions by using the following information supplied in the system ROM.

The format of the ID bytes are as follows:





6. Notes for MSX Software Developers

1) Do not write programs to directly handle the hardware. Use routines prepared in BIOS so as to isolate the software from the hardware and make future changes to the hardware without affecting the existing software possible. The BIOS is built to access its functions via a jump table beginning at address 0000. The jump table contains jump vectors functions that handle the hardware of the MSX computer. By using the functions provided by BIOS, application programs can access the MSX hardware without modification, even though the hardware is different.

   Fot example, the current MSX scans the keyboard by using 8255 PPI. In the near future, however, there may be computers having separate keyboards using an infrared communication link. This new computer may not use the 8255 PPI; it might use some other chip to do serial communications to handle the keyboard. If the software scanning the keyboard uses the 8255 directly, the new computer would not support the software.

   The only exception to the above rule is the VDP. To allow fast data transfer with the VDP, the ROM contains the locatios of the VDP in addresses 0006 and 0007. Address 0006 contains the read address of the VDP and address 0007 contains the write address to the VDP. If the software needs to transfer data at a high speed, the program can access the VDP directly using these addresses contained in ROM with the indirect addressing mode.

   In addition, address 0004 contains the address of the character pattern generator table stored in ROM. This may be of use to some programs that must keep track of the location of the table.

2) Do not use RAM locations above F380H if you do not have detailed documentation on the meanings of these locaations. This area is used by the system for working storage and access to these locations may cause your program to malfunction or to be incompatible with versions released in the future. All locations that are unused in the current MSX version within the above area are reseved for future expansion.

3) Software that has to interact with other programs must be designed in a way that it does not alter the programming environment. Major considerations are as follows.

   o Allocate work space
   o Share HOOKs

4) There are differences among versions of MSX sold in different countries. These differences have been restricted to the keyboard layout and the character generators. The locations 2BH and 2CH contain the special ID bytes that indicate the characteristics of the ROM. Software should be written to refer to these locations so as to work on any international version. See section 5.5 'ID bytes' for details.

5) Programs distributed in ROM cartridges must run in any slot, primary or secondary. Some of the game software that have been developped can run only in slot 1, or only in non-expanded slot. This DOES cause a big problem.

   Programs which use the MSX-BASIC interpreter with system CALL statements or device expansion mechanisms, must also determine the number of the slot in which the cartridge is inserted.

   Programs which run independently from the MSX-BASIC interpreter (such as game programs) do not have to determine the location of the cartridge unless they use the CALSLT routine, the CALBAS routine, or the 'RST 30H'; or if the program occupies more than one page (for example the first 16K in 4000H..7FFFH, or the second 16K in 8000H....0BFFFH). This is because slot exchanges are not done during program execution. For example, if you want to call a routine in your program from an interrupt hook, simply do a 'JP' instruction, not 'RST 30H', because your routine will always be there.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;     Use the following routine to know where you are:                          ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; This routine returns the slot address in the following format
; in [Acc].
;
;               FxxxSSPP
;               |   ||||
;               |   ||++-- Primary slot # (0-3)
;               |   ++---- Secondary slot # (0-3)
;               +--------- 1 if secondary slot # specified
;
; This value can later be used as an input parameter for the
; RDSLT, WRSLT, CALSLT, ENASLT and 'RST 30H'.
;
RSLREG  EQU     138H
EXPTBL  EQU     0FCC1H
B8000   EQU     0               ;Set this to non-zero if the program
                                ;  resides at 8000..0BFFFH
WHERE_AM_I:
        CALL    RSLREG          ;Read primary slot #
        RRC                     ;Move it to bit 0,1 of [Acc]
        RRC
IF      B8000
        RRC
        RRC
ENDIF
        ANI     11B
        MOV     C,A
        MVI     B,0
        LXI     H,EXPTBL        ;See if this slot is expanded or not
        DAD     B
        ORA     M               ;Set MSB if so
        RP                      ;Not expanded, all done
        MOV     C,A             ;Save primary slot number
        INX     H               ;Point to SLTTBL entry
        INX     H
        INX     H
        INX     H
        MOV     A,M             ;Get what is currently output to
                                ;  expansion slot register
IF      B8000
        RRC                     ;Move it to bit 2,3 of [Acc]
        RRC
ENDIF
        ANI     1100B
        ORA     C               ;Finally form slot address
        RET
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;    If your program occupies 32K from 4000H..0BFFFH, and                       ;
;    the execution begins from the INIT entry of the ROM                        ;
;    at 4000H..7FFFH, use the following routine to enable the                   ;
;    last half:                                                                 ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

ENASLT  EQU     24H
;
        CALL    WHERE_AM_I
        MVI     H,80H
        CALL    ENASLT
```

6) When changing the contents of register 0 or 1 of the VDP to change the display modes, do not write immediate values to them. Their current contents are always stored in RG0SAV and RG1SAV, respectively, so get their contents first, then mask off the unnecessary bits and add your own bits by using AND and OR instructions. Leave the other bits unchanged. This is necessary because some manufacturers use the EV (External Video) bits to allow superimposing.

7) When writing values to the various base address registers of the VDP, always write 0's to the unused bits near the MSB. This is important because it makes it possible to expand the capabilities of the VDP while maintaining compatibility with the current 9918-compatible VDP.

8) BIOS provides two ways to read the keyboard. Use the routines depending on to your needs.

   1. Physical key position and real-time keyboard status as returned value needed.

      Use SNSMAT entry

   2. Logical character code needed as a returned value.

      Use CHSNS and CHGET