MSX HOME PERSONAL COMPUTER SYSTEM

HARDWARE SPECIFICATION

(RELEASE 3.1)

04/05/84

Comments about this documentation may be sent to:

Microsoft Corporation
Microsoft Building
10700 Northup Way
Bellevue, WA 98004

Hardware specification for MSX HOME PERSONAL COMPUTER SYSTEM

*This document specifies the U.S./European version of MSX system.

MSC TECHNICAL REFERENCE DOCUMENT

INDEX

Hardware specification for MSX HOME PERSONAL COMPUTER SYSTEM

CONTENTS

CHAPTER 1

SUMMARY OF HARDWARE

SUMMARY OF HARDWARE

1.1 SPECIFICATION

1.1.1 Required Components

o	CPU	4 MHz Z-80A compatible
o	Memory	ROM 32K (MSX system software) RAM minimum 8K (16K* recommended)
o	Screen Display	Text display capability 40 x 24 (refer to section 2.4) Graphic 256 x 192 Color 16
o	Cassette tape	FSK format 1200/2400 Baud
o	Sound	8 Octave, 3 Voices
o	Character Set	Alphanumeric, Japanese, Graphic (Japanese version) Alphanumeric, European, Graphic (International version)
o	Keyboard	U.S./Europe, French*, German*, Japanese
o	Expansion Slot	Software cartridge, expansion BUS slots
o	Joystick	1 or 2*

1.1.2 Recommended Extensions for U.S./Europe

o	Memory	RAM 64K* total
o	Expansion Slot	Second
o	Video	RF output

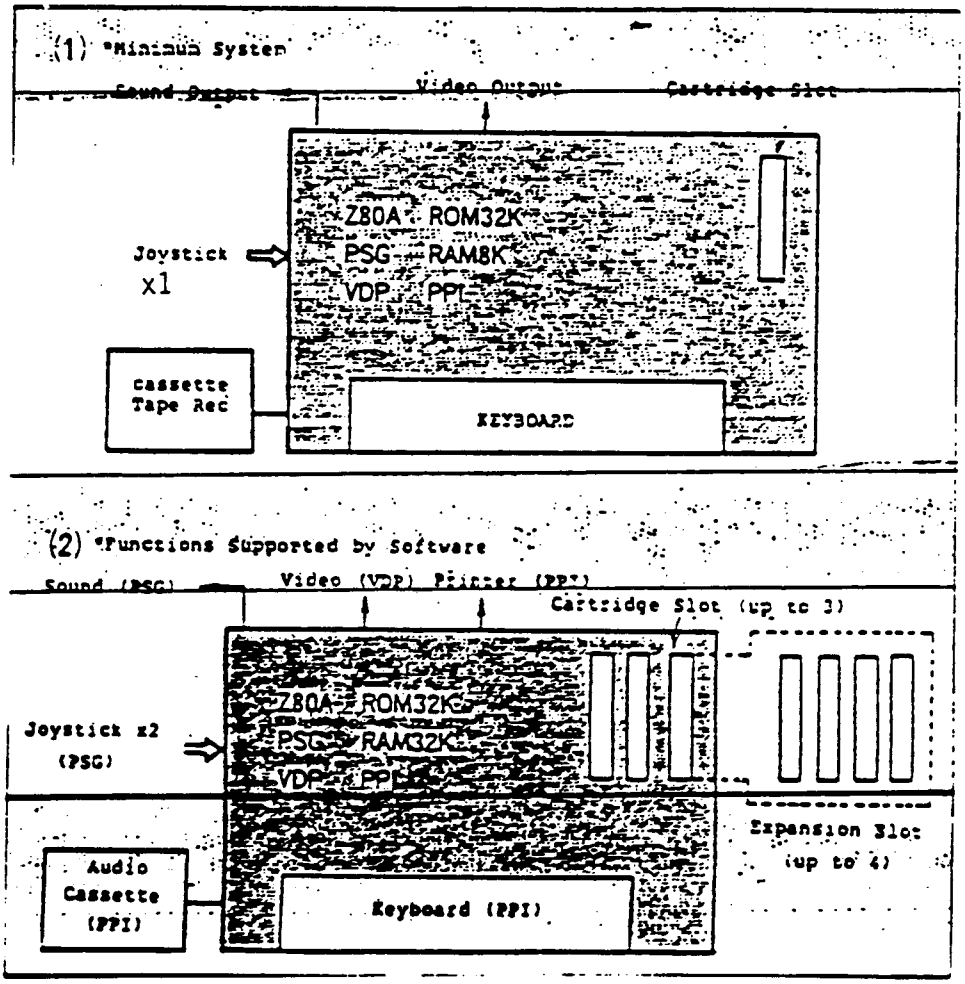
1.1.3 Standardized Optional Extensions

o	Screen Display*	80-column text
o	Clock*	Battery backed-up CMOS
o	Communications*	RS-232
o	Floppy Disk*	According to each company. Format is MS-DOS compatible
o	Printer*	8 bit parallel

* Items with asterisk may not be build-in in the minimum system.

SUMMARY OF HARDWARE

1.2 SYSTEM CONFIGURATION



HARDWARE SPECIFICATION

2.1 LSI

- o CPU — Z-80A 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

2.2 MEMORY

- o ROM — MSX BASIC 32KB
- o RAM — 8KB or more

NOTE

Basic unit has four logical slots, so the total memory space can be expanded up to 256KB. Each logical slot can be expanded to have up to 4 physical slots, total of 16 slots. So in this case, maximum memory space is 1 megabyte.

BASIC ROM occupies address 0 to 7FFF, RAM address starts from FFFF and grows downward on the memory map to 8000.

For details refer to 4.1 memory map.

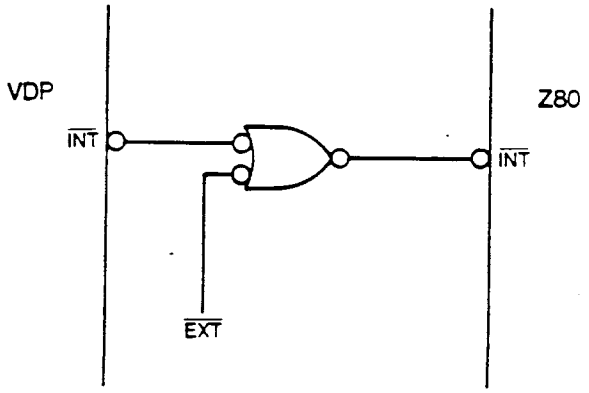
For U.S. Market we recommend 64K so the machine can easily be upgradable to MSX-DOS, although BASIC ROM will only use 32K RAM.

2.3 INTERRUPT

- MNI — Not used. MSX ROM only provides RAM hook.
- INT — Accept interrupts from VDP and cartridge. The interrupt is Z-80 mode 1. (Branch to 38H) MSX system software uses interrupt from VDP for timer count. The interval of the interrupt is 60Hz in NTSC and 50Hz in PAL/SECAM version.

NOTE

It is not possible to support NMI under MSX DOS environment because address 66H, which is the entry vector for NMI, is occupied by FCB data for DOS.



HARDWARE SPECIFICATION

2.4 SCREEN DISPLAY

- o LSI — TI TMS9918A Compatible
- o Character set — Alphanumerical + European + Graphic
  256 patterns 6x8 dots
- o Color — 16 colors
- o Sprites — 32 sprites. Maximum 4 sprites on the same horizontal line.
- o List of display modes

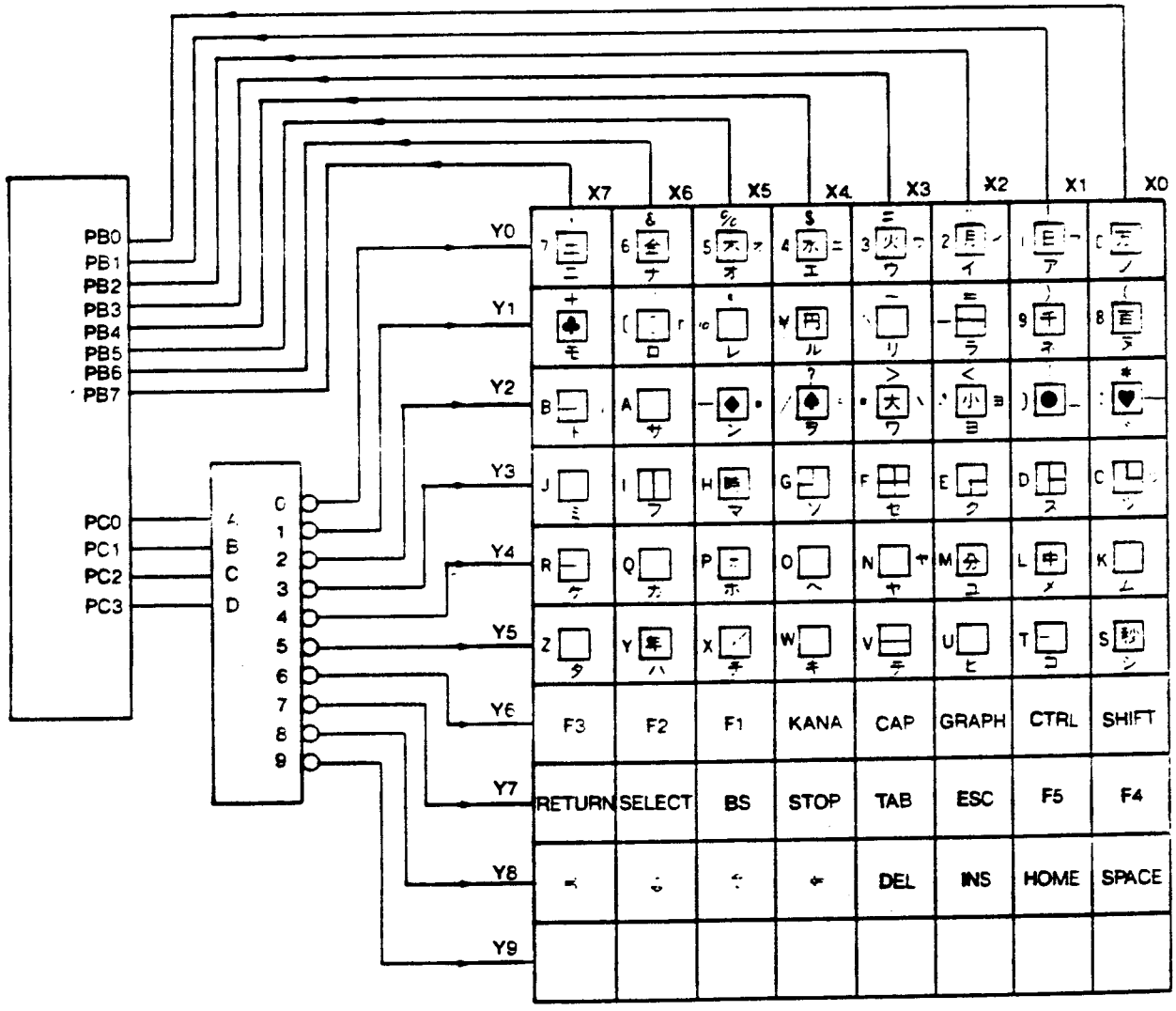
MODE		RES.	SIZE	Δ NO.	COLOR	SPRITE	NUMBER OF CHARACTERS
Graphic I	LSI Spec.	256 x 192	8 x 8	256	16 colors	YES	32 x 24
	Suggested value	240 x 192					29 x 24
Graphic II	LSI Spec.	256 x 192	8 x 8	768	16 colors	YES	32 x 24
	Suggested value	240 x 192					29 x 24
Multi-color	LSI Spec.	64x48blk	4 x 4 /block	-	16 colors	NO	32 x 24
	Suggested value	64x40blk					29 x 24
Text	LSI Spec.	256 x 192	6 x 8	256	2 colors out of 16 colors	YES	40 x 24
	Suggested value	240 x 192					39 x 24

Suggested value to use: the 8 pixels from left and right of horizontal are not used by software.

Δ The number of patterns

2.5 KEYBOARD

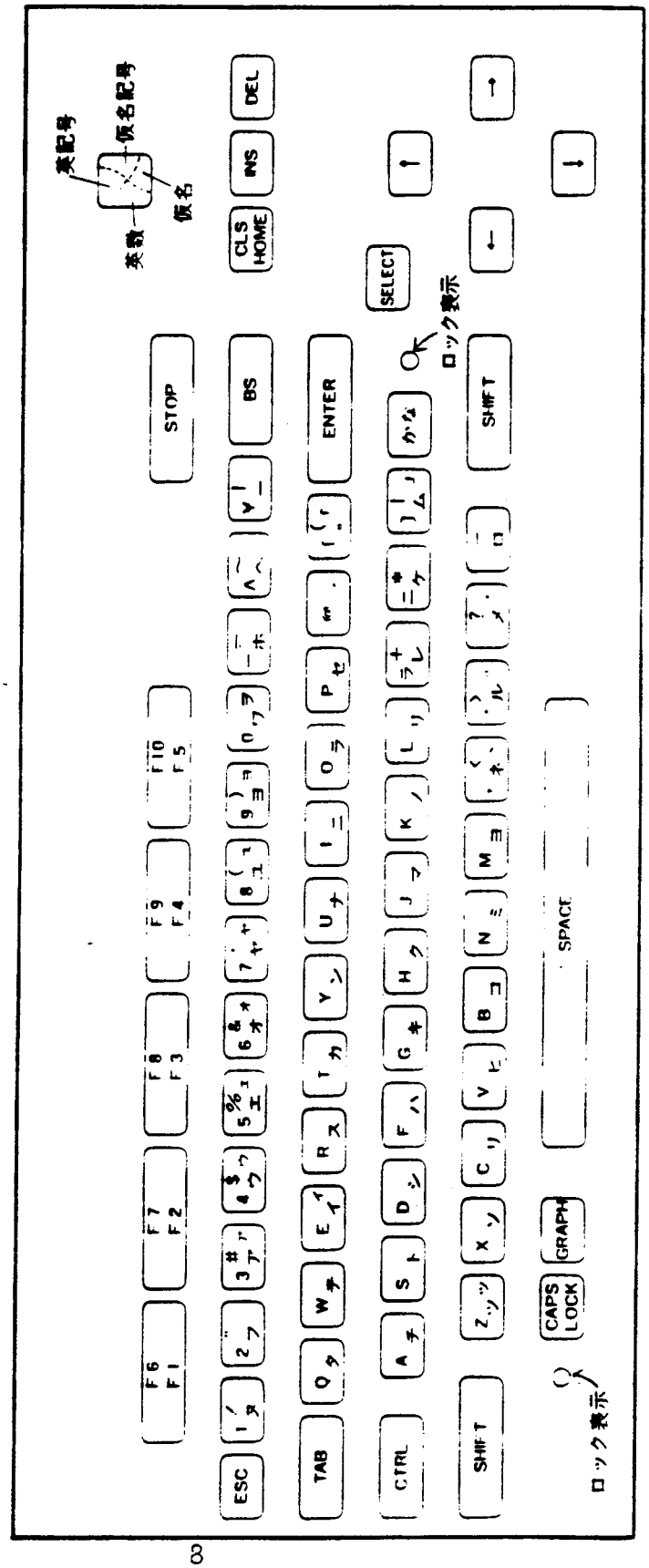
- o Layout — Refer to figure
  - U.S./European
  - French
  - German
- o Scanning — Software scanning driven by VDP interrupt
- o Number of keys — 70 plus optional dead key
- o Matrix diagram



Y	X	0		1		2		3		4		5		6		7	
		char	code	char	code	char	code	char	code	char	code	char	code	char	code	char	code
0		0	30	/	31	2	32	3	33	4	34	5	35	6	36	7	37
	S	)	29	!	21	@	40	#	23	$	24	%	25	^	5E	&	26
	G	○	D9	¼	AC	½	AB	¾	BA	∩	EF	‰	BD	⌈	F4	⌈	FB
	G S	◙	0A			²	FD	ⁿ	FC					J	FS		
	C	δ	EB	ƒ	9F	₹	D9	ϕ	BF	¢	9B	ÿ	98	∝	EO	β	E1
	CS	Δ	D8	¡	AD	₧	9E	¶	BE	£	9C	¥	9D	--			
1		8	38	9	39	—	2D	=	3D	\	5C	[	5B	]	5D	;	3B
		*	2A	(	28	_	5F	+	2B	|	7C	{	7B	}	7D	:	3A
	G	∞	EC	•	07	═	17	±	F1	/	1E	☺	01	♪	0D	♠	06
	G S			■	08	┼	1F	═	FO	|	16	☻	02	♫	0E	♦	04
	C	ϒ	E7	ς	87	∈	EE	e	E9			∅	ED	ω	DA	ă	B7
	CS	⌐	EZ	Ç	80							Φ	EB	Ω	EA	Ŭ	B6
2		'	27	¨	60	;	2C	.	2E	/	2F	¨		a	61	b	62
		"	22	~	7E	<	3C	>	3E	?	3F	~		A	41	B	42
	G	♣	05	∽	BB	≤	F3	≥	F2	/	1D	¨	key	▪	C4		11
	G S	♥	03	≈	F7	«	AE	»	AF	÷	F6	~	dead	▮	FE	-	
	C	ij	B9	σ	E5	å	86	a	A6	O	A7	´		ä	84	ù	97
	CS	IJ	B8	Σ	E4	Å	8F			¿	A8	^		Ä	8E		
3		c	63	d	64	e	65	f	66	g	67	h	68	i	69	j	6A
		C	43	D	44	E	45	F	46	G	47	H	48	I	49	J	4A
	G	◇	BC	▀	C7	▼	CD	╬	14	┼	15	┤	13	▒	DC	▯	C6
	G S	•	FA	■	C1	▲	CE	▄	D4	+	1C	▌	D6	▒	DF	▓	CA
	C	ì	8D	ï	8B	î	8C	ö	94	ü	81	ã	B1	í	A1	æ	91
	CS							Ö	99	Ü	9A	Ã	B0			Æ	92
4		k	6B	l	6C	m	6D	n	6E	o	6F	p	70	q	71	r	72
		K	4B	L	4C	M	4D	N	4E	O	4F	P	50	Q	51	R	52
	G	▌	DD	▐	C8	♂	0B	┘	1B	▀	C2	▒	DB	▨	CC	┌	18
	G S	▌	DE	·	C9	♀	0C	▀	D3		C3	▩	D7	▨	CB	┌	A9
	C	ĩ	B3	õ	B5	µ	E6	ñ	A4	ó	A2	ú	A3	â	83	ô	93
	CS	Ĩ	B2	Õ	B4			Ñ	A5			Ú	E3	—			
5		s	73	t	74	u	75	v	76	w	77	x	78	y	79	z	7A
		S	53	T	54	U	55	V	56	W	57	X	58	Y	59	Z	5A
	G	▚	D2	┼	12	▐	C0	┘	1A	▶	CF	▩	1C	┌	19	▯	0F
	G S	▚	D1			▌	C5	▗	D5	◢	D0	·	F9	—	AA	°	F8
	C	ë	89	û	96	é	82	ò	95	ê	88	è	8A	á	A0	ã	85
	CS					É	90										

HARDWARE SPECIFICATION

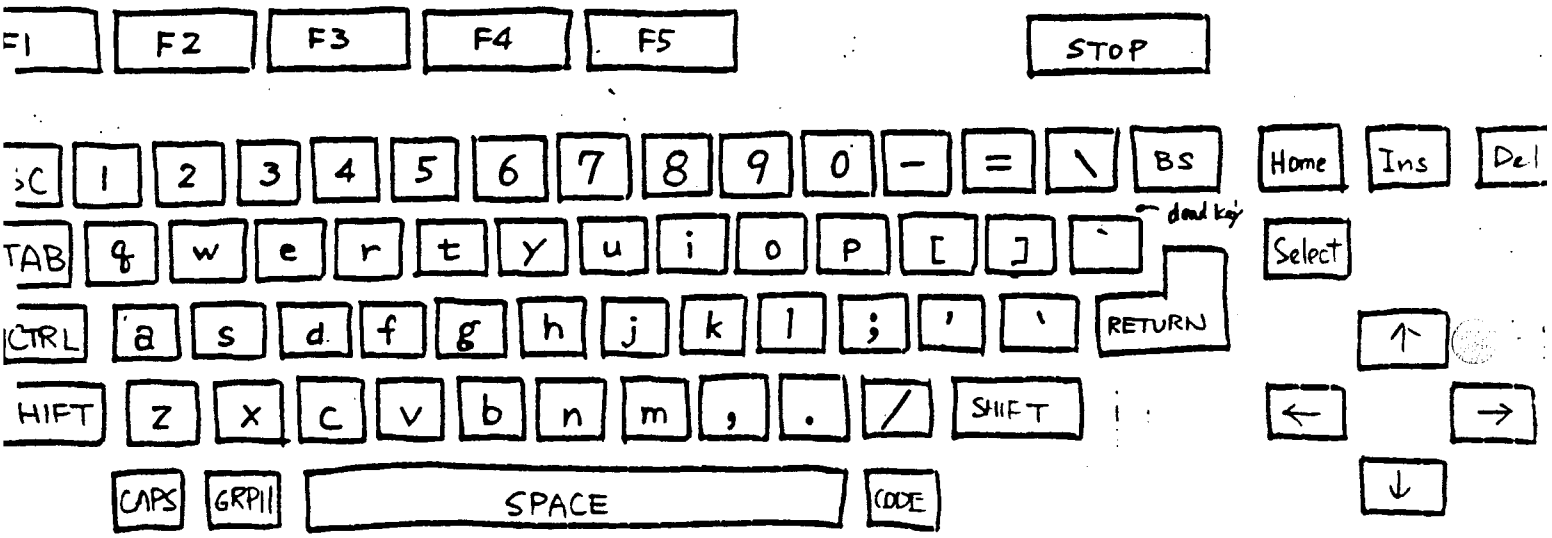
o Keyboard layout



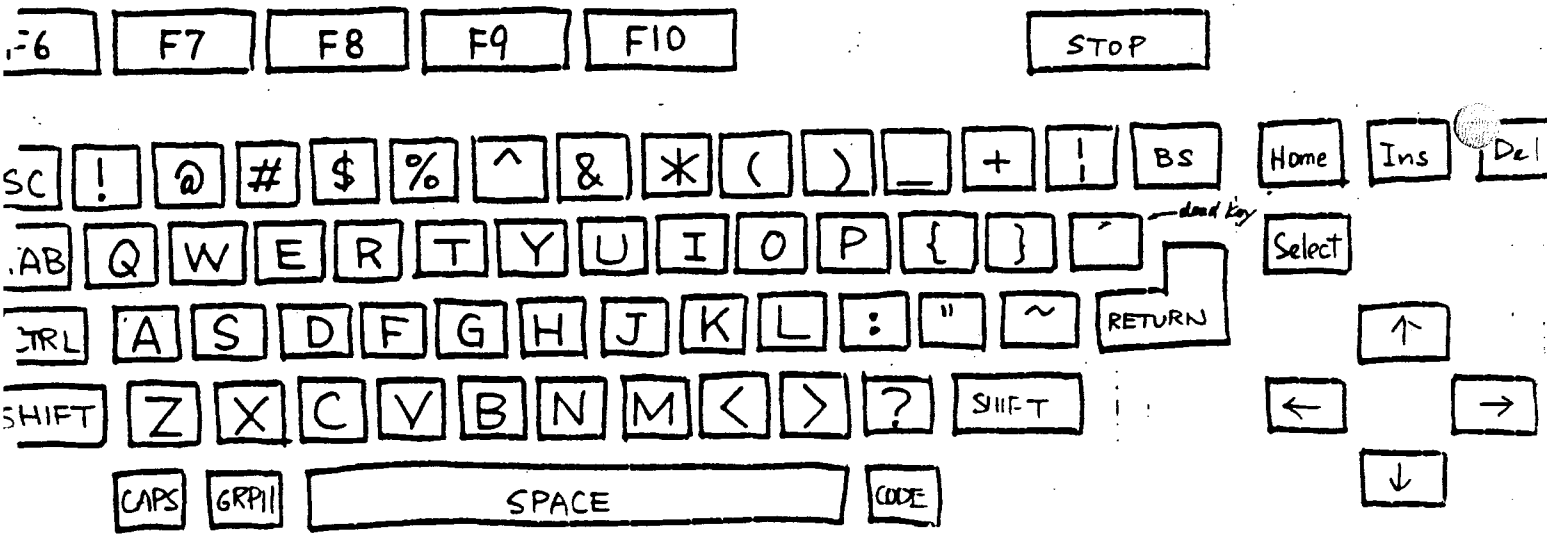
The following keyboard diagrams contain an optional dead-key. This dead-key is useful for European accented character input. For example, when one wishes to enter "â", one must first press the dead-key "^" and then press "a".

The dead-key will not be useful in the U.S. or U.K. Nor will it be useful in France and Germany, where specific French and German keyboards have been designed which include different dead-keys. Therefore this general dead-key should only be included on machines which will be marketed into minor European countries.

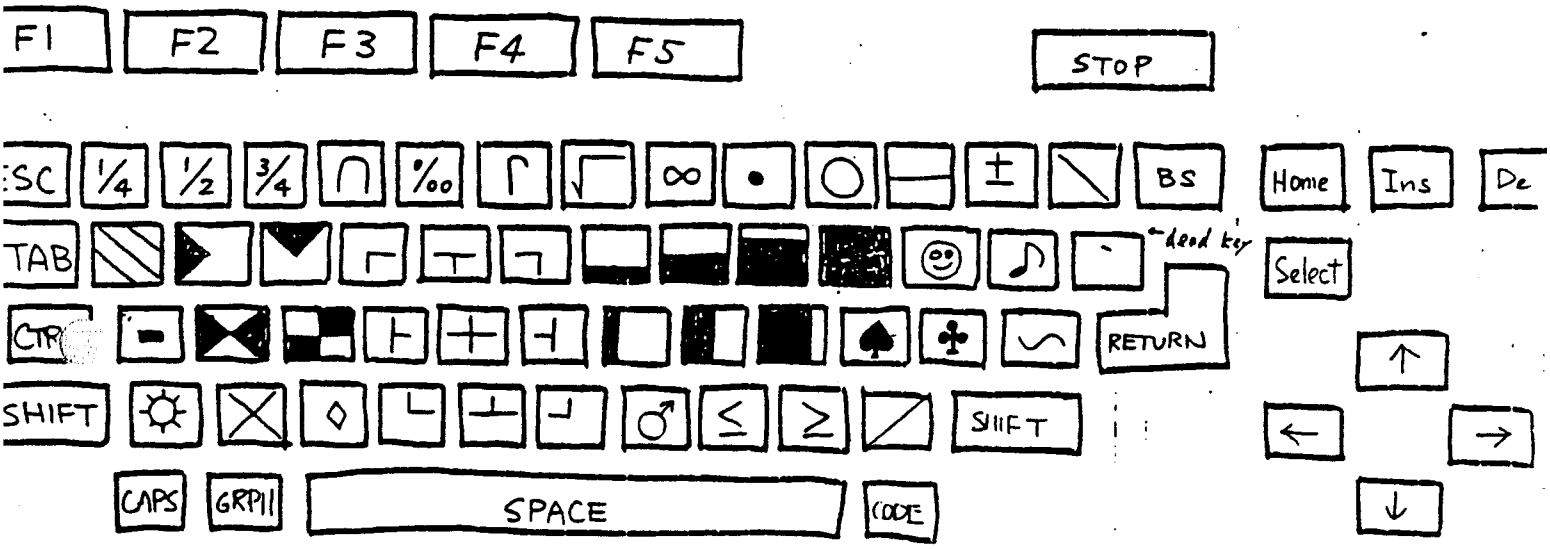
The keyboard diagrams show the dead-key to the left of the carraige return key, but this is probably not a good place for it, because it pushes the carraige return key too far to the right. Manufacturers may place this key where they wish.



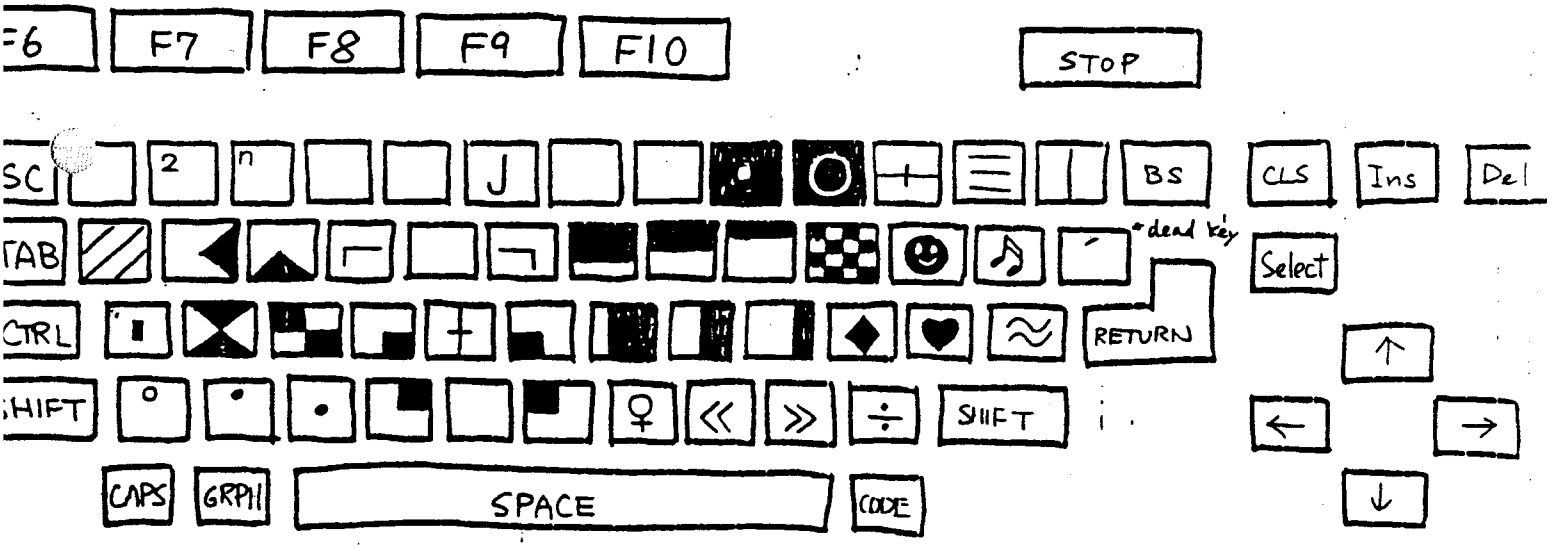
without Shift
without Graph
without code



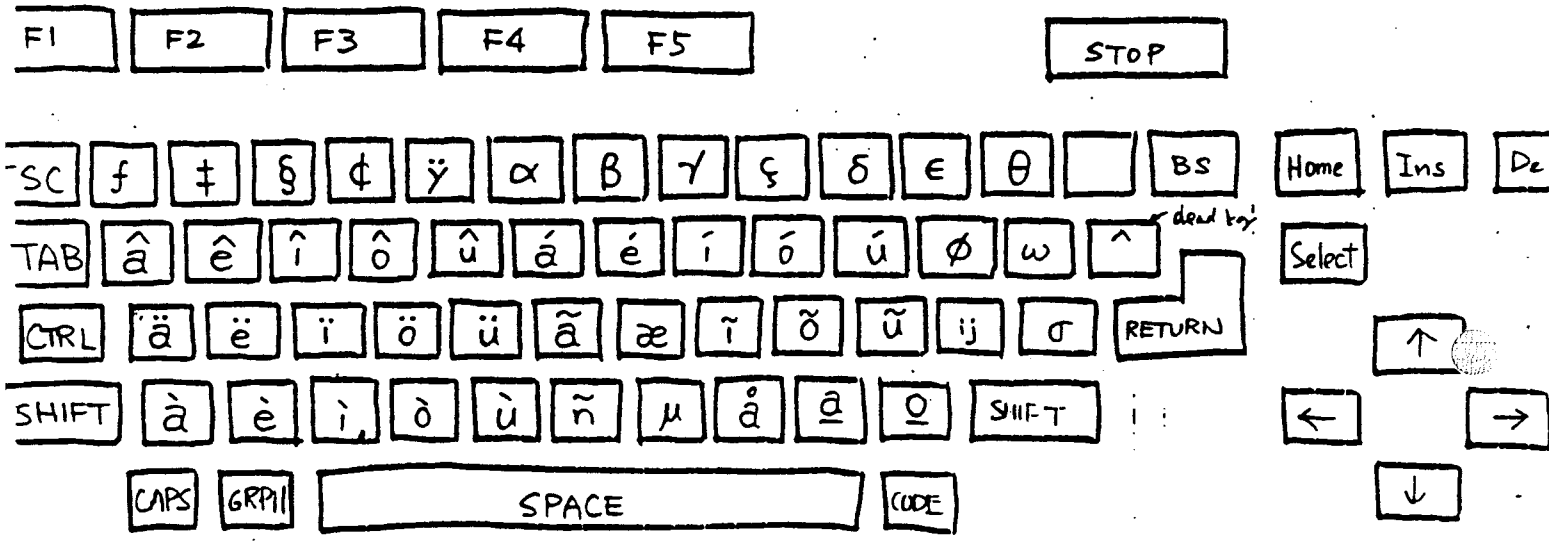
with Shift
without Graph
without Code



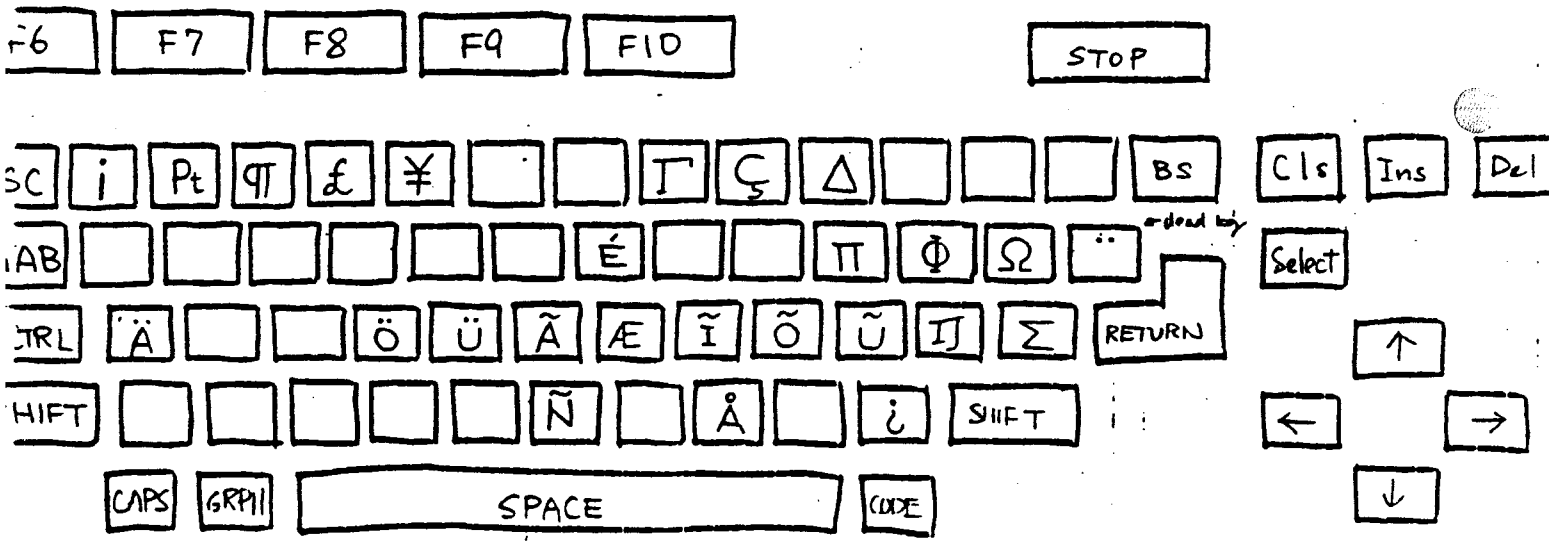
without Shift
with Graph
without code



with Shift
with Graph
without Code



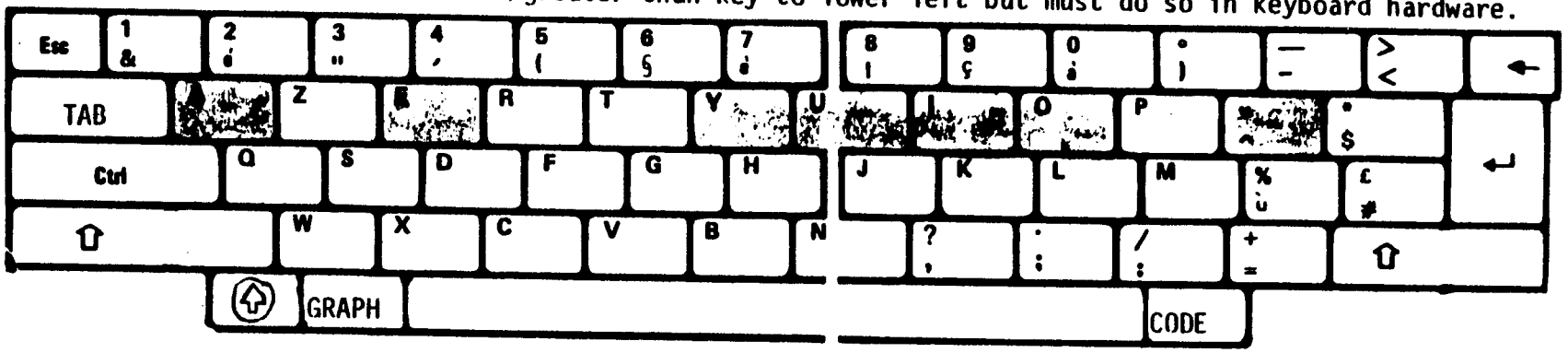
without shift
without Graph
with code



with shift
without Graph
with Code

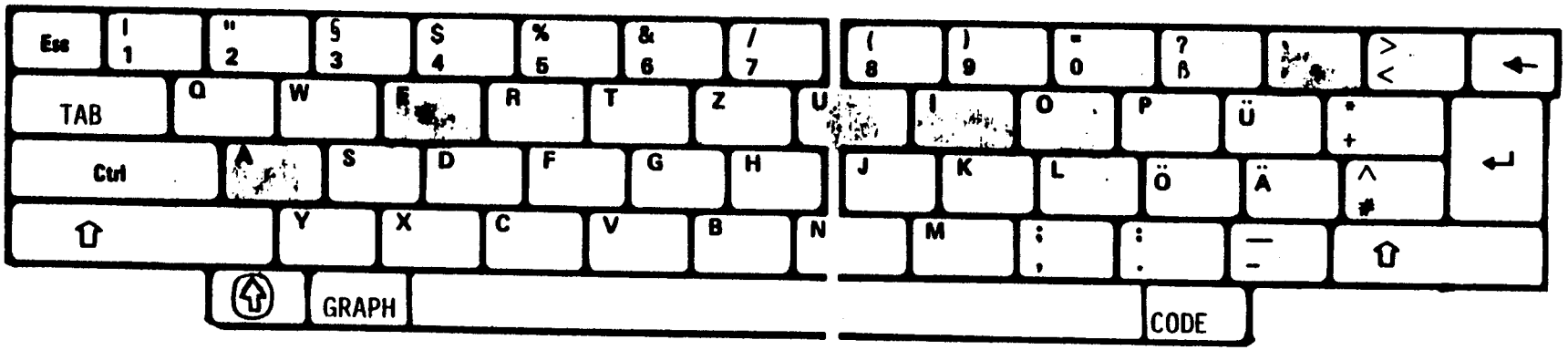
MSX French Keyboard

Caps Lock shifts numeric keys and)/° key. Dead keys are shaded.
Manufacturer may move less than/greater than key to lower left but must do so in keyboard hardware.



MSX German Keyboard

Caps Lock shifts A, O, and U umlaut keys. Dead keys are shaded.
Manufacturer may move less than/greater than key to lower left but must do so in keyboard hardware.



BASIC SPECIFICATION

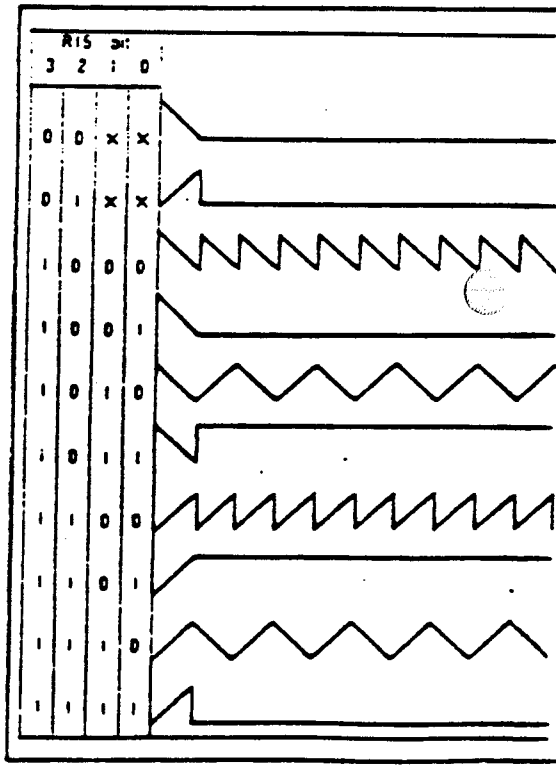
2.6 SOUND

- o LSI GI AY-3-8910 Compatible
- o OCTAVE 8 Octaves (3 Voices output)
- o SOUND EFFECT Yes
- o SOFTWARE SOUND OUTPUT 1 bit from output port
- o OUTPUT LEVEL -5dbm (If the system has output connector)
- o CONNECTOR RCA 2 pins (If the system has audio output connector)

REGISTER \ BIT	B7	B6	B5	B4	B3	B2	B1	B0
R0	Channel A Tone Period			8-BIT Fine Tune A				
R1					4-BIT Coarse Tune A			
R2	Channel B Tone Period			8-BIT Fine Tune B				
R3					4-BIT Coarse Tune B			
R4	Channel C Tone Period			8-BIT Fine Tune C				
R5					4-BIT Coarse Tune C			
R6	Noise Period				5-BIT Period Control			
R7	Enable	IN/OUT		Noise			Tone	
		IOB	IOA	C	B	A	C	B	A
R10	Channel A Amplitude				M	L3	L2	L1	L0
R11	Channel A Amplitude				M	L3	L2	L1	L0
R12	Channel A Amplitude				M	L3	L2	L1	L0
R13	Envelope Period	8-BIT Fine Tune E							
R14		8-BIT Coarse Tune E							
R15	Envelope Shape Cycle					CONT	ATT	ALT	HOLD
R16	I/O Port A Data Store	8-BIT PARALLEL I/O on Port A							
R17	I/O Port B Data Store	8-BIT PARALLEL I/O on Port B							

図2 AY-3-8910のレジスタ構成

図3 AY-3-8910のエンベロープ波形

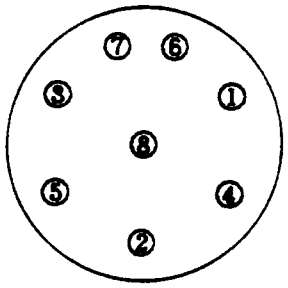


HARDWARE SPECIFICATION

2.7 CASSETTE INTERFACE

- o INPUT — From the earphone terminal of tape recorder
- o OUTPUT — To the microphone terminal of tape recorder
- o SYNCHRONIZATION — Asynchronous by the software
- o BAUD RATE — 1200 Baud (1200Hz - 1 wave "0", 2400Hz - 2 waves "1") (Default)

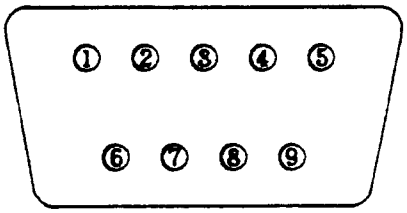
  2400 Baud (2400Hz - 1 wave "0", 4800Hz - 2 waves "1")
  Change by software
  (Tape recorder may have to be specified by the manufacturer when used under 2400 Baud mode)
- o MODULATION — FSK (Frequency Shift Keying) by the software
- o DEMODULATION — By the software. The system software automatically detects the baud rate when receiving the data.
- o MOTOR CONTROL — Yes
- o CONNECTOR — DIN 45326 (8 pin)
- 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	---	

HARDWARE SPECIFICATION

2.8 INPUT/OUTPUT (JOYSTICK) PORT (1 OR 2* PORTS)

- o LSI — AY-3-8910 compatible
- o I/O — Input 4 bit, output 1 bit, bidirectional 2 bit per each port
- o LOGIC — Active high
- o LEVEL — TTL
- o CONNECTOR — AMP 9 pin 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	---	

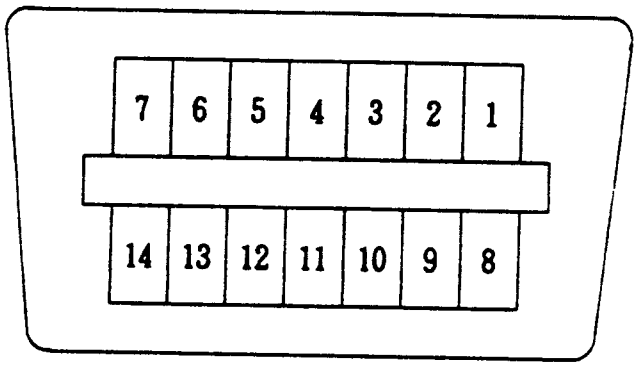
△ Current capacity is 50mA each

HARDWARE SPECIFICATION

2.9 *PRINTER INTERFACE

- o SPECIFICATION 8 bit parallel Handshakes by BUSY and STROBE signal
- o LEVEL TTL
- o CHARACTER CODE SAME AS MSX DISPLAY CODE
- o CONNECTOR AMP 14 pin compatible
- o LIST OF PINS

PIN NO.	SIGNAL NAME	PIN CONNECTION
1	PSTB	
2	PDB0	
3	PDB1	
4	PDB2	
5	PDB3	
6	PDB4	
7	PDB5	
8	PDB6	
9	PDB7	
10	N.C.	
11	BUSY	
12	N.C.	
13	N.C.	
14	GND	



2.10 FLOPPY DISK INTERFACE

- Contains 16K bytes of ROM at 4000H that includes:
  * MSX-DOS KERNEL
  * MSX DISK BASIC
  * PHYSICAL DISK I/O DRIVER (Supplied by each manufacturer)

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

- It is desirable to have a mechanism in the disk drive to detect whether the drive door has been opened. It reduces disk accesses which check for disk changes.

- Floppy formats are MS-DOS compatible

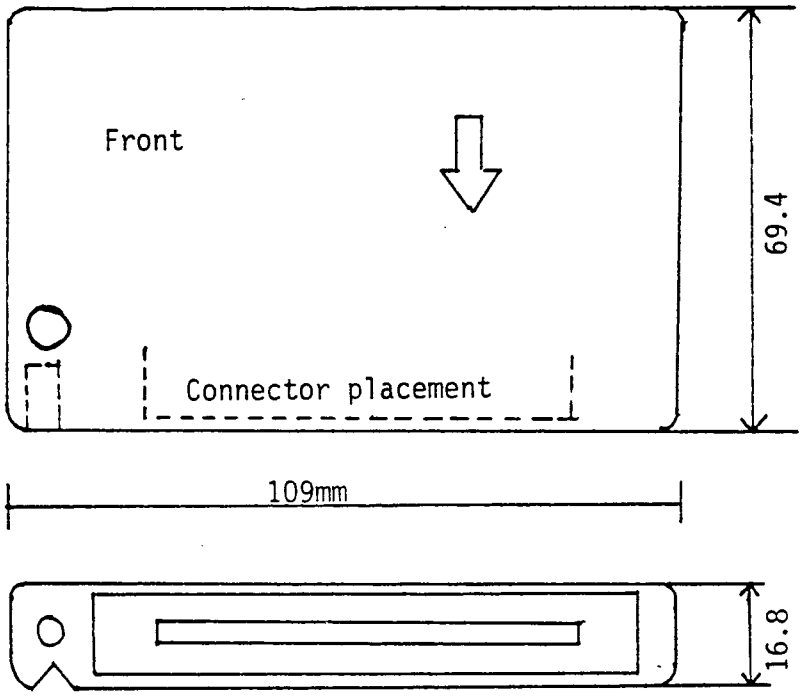
8 inch	SD	128 byte/sector
8 inch	DD	1024 byte/sector
5-1/4 inch	DD	512 byte/sector
3.5 inch	CFD	512 byte/sector (exactly same as 5-1/4" 96TPI
3 inch	CFD	512 byte/sector (exactly same as 5-1/4" 48TPI

CHAPTER 3

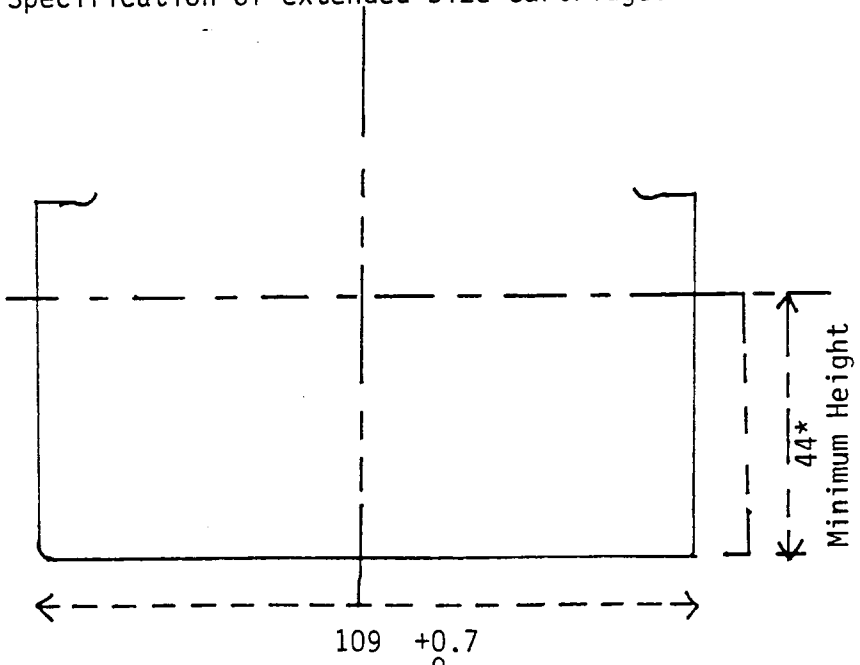
CARTRIDGE

3.1 PHYSICAL CARTRIDGE SPECIFICATION

The internal specification of standard size cartridge
(all measurements in mm)



Specification of extended size cartridge.

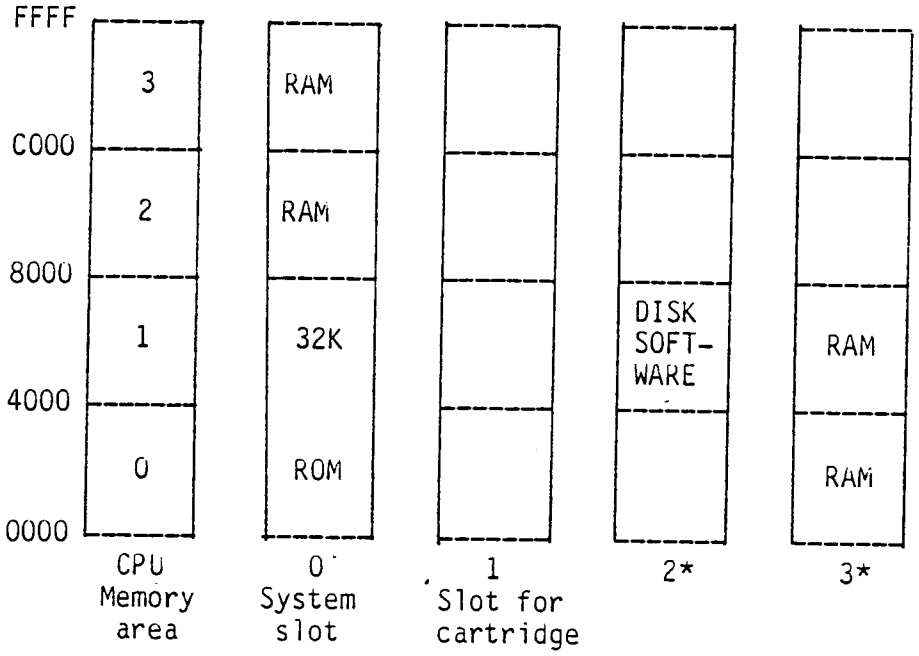


* No standard will be enforced beyond the above minimum height

ADDRESS MAP

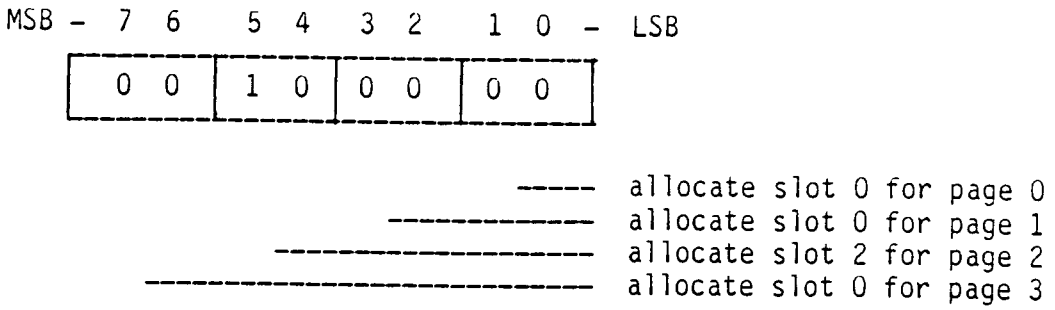
4.1 MEMORY MAP

- Following is an example of memory map.



- MSX BASIC uses the largest available contiguous RAM area that is installed from FFFF to 8000 for its system working RAM area. This can be placed in any slots including expandsion slots.

- Slot select register, which is port A of 8255, maps thephysical 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.



Physical memory is always allocated to the same memory page in the CPU address space. It is not possible to allocate to a different page, like page 3 of slot 3, to page 0 of CPU memory space.

- Minimum system must have two slots, one for system, the other for cartridge.

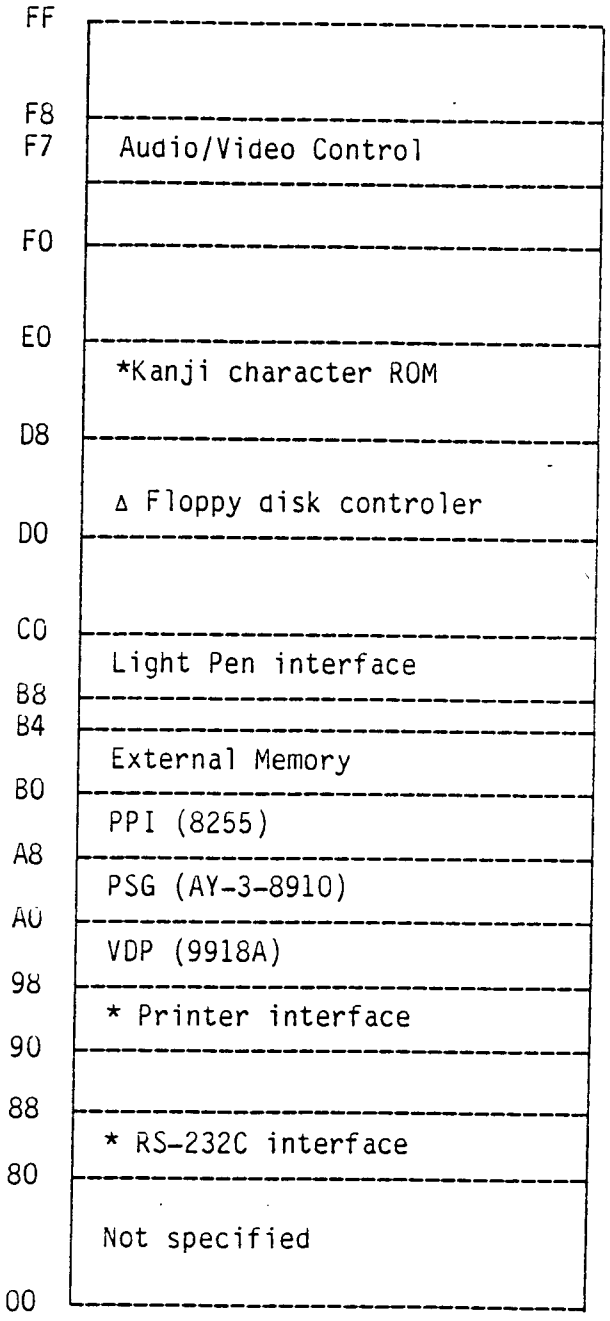
NOTE

The word "slot" does not imply that it must have a connector for cartridges, however, a slot for cartridges must have a connector, of course. Refer to APPENDIX C.

- MSX-DOS requires 64K RAM

ADDRESS MAP

4.2 I/O ADDRESS MAP



ADDRESS MAP

4.3 I/O DEVICE DESCRIPTION

4.3.1 RS-232C

4.3.1.1 LSI Components -

i-8251 Communication interface chip
i-8253 Programmable interval timer chip

4.3.1.2 Port Address -

80H	R/W	8251 data port
81H	R/W	8251 command/status port
82H	R	Baud rate setting switches
83H	R	Configuration setting switches
83H	W	Interrupt mask register
84H	R/W	8253 counter 0
85H	R/W	8253 counter 1
86H	R/W	8253 counter 2
87H	W	8253 mode register

4.3.1.3 The Usage of Switch Port at Address 82H And 83H -

82H read - Baud rate select

bit 0 - 3 : baud rate for receiver
bit 4 - 7 : baud rate for transmitter

value	baud rate
0	50
1	75
2	110
3	150
4	300
5	600
6	1200
7	2400
8	4800
9	9600
A	19200
B	N.A.
C	N.A.
D	N.A.
E	N.A.
F	disable*

*When value F is set as a baud rate, that function is disabled by software.

ADDRESS MAP

83H read - Set various functions

bit 0	- CD (carrier detect)*	
1	- auto line feed on receive**	1 - auto line feed
2	- Full/Half duplex	1 - Full duplex
3	- XON/OFF control	1 - Enable control
4	- Word length	1 - 8bits, 0 - 7 bits
5	- Parity Even/Odd	1 - Even
6	- Parity enable	1 - Enable
7	- Stop bit length	1 - 2 bits, 0 - 1 bit

* CD is a signal directly connected to carrier detect (pin 8) on the DB-25 connector.

** Add line feed on receiving carriage return.

NOTE

Bit 0 of the switch pulls up the CTS line of the 8251 (or [actually] it pulls down since CTS on 8251 is negative logic) to make it possible to send data even when CTS is not supplied from outside.

83H write - Set interrupt mask for receive

bit 0 - mask interrupt for receive 1 - mask interrupt
The initial value of this mask is 1 (disable interrupt).

4.3.1.4 Usage of 8253 Timer-counter To Generate Baud Rate - clock for 8251

- Frequency of crystal
  The frequency of the crystal:
  1.2288 MHz

- Usage of counter channel
  CH0 - Rx baud rate clock
  CH1 - Tx baud rate clock
  CH2 - General interrupt timer .. connect to IRQ

4.3.2 PRINTER PORT

4.3.2.1 Port Address -

90H	R	Busy status : bit 1
90H	W	Strobe output : bit 0
91H	W	Print data

ADDRESS MAP

4.3.3 VDP PORT

98H	R/W	Vido Ram data
99H	R/W	Commanc and status register

4.3.4 PSG PORT

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

4.3.5 PPI PORT

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

4.3.6 External Memory (Sony)

B0H through B3H

4.3.7 Light Pen (Sanyo)

B8H through BBH

4.3.8 Audio/Visual 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

4.4 NOTES ON I/O ADDRESS ASSIGNMENT

- I/O address 80~FF are assigned for system usage. The empty areas are reserved for system use.
  Although these addresses are defined here, software should not access those devices directly through the addresses listed above. Every access to the I/O must be done through the BIOS calls. This is to keep software independent from hardware differences. Manufacturers may change some hardware from the standard MSX system and is still able to maintain software compatibility by

supporting the hardware differences within the BIOS, so that the difference can be transparent to software.
The only exception is access to the VDP. Locations 6 and 7 of the MSX system ROM contain read and write addresses of VDP register. The software needing to access VDP very quickly may access VDP directly through those addreses stored in ROM.

- o 00~7F are free addresses, however when different devices use the same address, they may not be accessed at the same time. Basically, special I/O devices not defined here should be placed in the memory space as memory mapped I/O. Refer to Appendix B.3.

- Δ FDC may be placed in I/O space, but it must have a mechanism to disable it and only at the moment when the system accesses the FDC, is it enabled. This makes it possible to have more than one FDC interface in the system to handle different kinds of media.

ADDRESS MAP

4.5 8255 (PPI) BIT ASSIGNMENT

PORT	BIT	I/O	SIGNAL NAME	DESCRIPTION
A	0 1	OUTPUT	CS0L CS0H	0000~3FFF address slot select signal
	2 3		CS1L CS1H	4000~7FFF address slot select signal
	4 5		CS2L CS2H	8000~BFFF address slot select signal
	6 7		CS3L CS3H	C000~FFFF address slot select signal
B	0 thru 7	INPUT		Keyboard return signal
C	0 1 2 3	OUTPUT	KB0 KB1 KB2 KB3	Keyboard scan signal
	4		CASON	Cassette control signal (L-ON)
	5		CASW	Cassette write signal
	6		CAPS	CAPS lamp signal ("L" --> ON)
	7		SOUND	Sound input by software

ADDRESS MAP

4.6 PSG BIT ASSIGNMENT

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

1 Available when bit 6 of port B is low used by JOYSTICK1
2 Available when bit 6 of port B is high used by JOYSTICK2
3 Turn to "H" level when use those pins as an input port.
Tied an open collector buffer to the output. (Refer to Appendix C-1)
4 JIS layout - "H" level, syllable layout - "L" level

<Remark> PIN5 +5V
PIN9 GND

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

APPENDIX A

LIST OF CONNECTORS

PIN NAME	SPECIFICATION
1. Video output and composite video	DIN 5 PIN CONNECTOR Δ 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	AMPHENOL 14 PIN CONNECTOR
CARTRIDGE BUS	0.10 INCH (2.54mm) SPACE, 50 PIN CONNECTOR
AUDIO	RCA 2 PIN CONNECTOR

MSX-DOS CP/M-80
Technical Information

Introduction

MSX-DOS Function Requests

MSX-DOS FCB Format

Introduction

The following information is provided so programmers can run CP/M-80 programs under MSX-DOS. MSX-DOS function requests and the FCB format are described.

MSX-DOS Function Requests

The user requests a function by

1. Placing a function number in the C register.
2. Supplying additional information in other registers for the specific function.
3. Executing a CALL 5 instruction.

Unless otherwise specified, single-byte values are passed in E and double-byte values in DE. When MSX-DOS takes control, it switches to an internal stack. No registers are guaranteed to be preserved. MSX-DOS returns single-byte values in register A and double-byte values in register pair HL. In addition, the contents of register A will always equal those of register L, and the contents of B will equal H for those calls which have a CP/M-80 counterpart. A value of zero will be returned in A (as well as HL) for non-supported function numbers.

Function numbers are as follows. All values are in hex:

0 Program Terminate: All file buffers are flushed, but files which have been changed in length (but not closed) will not be recorded properly in the disk directory. Control transfers back to MSX-DOS. This is the same as JMP 0.

1 Keyboard: Waits for a character to be typed at the keyboard, then echoes the character to the screen and returns it in A. A console status check is done to check to Control-C, printer echo, and Control-S. These characters will *not* be passed through this call. Execution will be suspended until a character is typed, if none was waiting when the call was made. Carriage Return, Line Feed, Backspace, and Bell are echoed as such. Tabs are expanded.

2 Screen Output: The character in E is output to the screen in a fashion similar to Function 1. A console status check is performed after the character is output. A VT52 driver is built into the BIOS.

Escape sequences are:

j	Clear screen
E	Clear screen
K	Erase to end of line
J	Erase to end of screen
l	(Lowercase L) Erase entire line
L	Insert a line
M	Delete a line
Y	Locate cursor
A	Up
B	Down
C	Right
D	Left
H	Home
x4	Block cursor
y4	Underscore cursor
x5	Cursor off
y5	Cursor on

3 Auxiliary Input: Waits for a character from the serial port and returns it in A. A console status check is done beforehand, suspending reading from this device if Control-S is detected.

4 Auxiliary Output: The character in E is sent to the serial port. A console status check is performed beforehand.

5 Printer Output: The character in E is output to the printer. A console status check is performed beforehand. Tabs are not expanded by the DOS.

6 Direct Console IO: If E is FFH, then A returns with keyboard input character if one is ready and zero flag is not set; otherwise, A is zero and zero flag is set. If E is not FFH, the E is assumed to have a valid character which is output to the screen. A console status check is *not* performed, passing through the values Control-C and Control-S.

7 Direct Console Input: Waits for a character to be typed at the keyboard, then returns the character in A. As with Function 6, no console status check is performed.

Note that CP/M-80 Function 7 is Set I/O Byte. I/O byte is not supported under MSX-DOS.

8 Keyboard Input without Echo: Identical to Function 1, without displaying the character.

Note that CP/M-80 Function 8 is Set I/O Byte. I/O byte is not supported under MSX-DOS.

9 Display String: On entry DE must point to a character string in memory terminated with a "$" (24H). Each character in the string will be displayed to the screen in the same form as Function 2,

including console status check.

A **Buffered Keyboard Input:** On entry, DE points to an input buffer. The first byte must not be zero and specifies the number of characters the buffer can hold. Characters are read from the keyboard and placed in the buffer at the third byte. Reading the keyboard and filling the buffer continues until the carriage return is typed. If the buffer fills to the maximum, additional keyboard input is ignored and a bell is rung until carriage return is entered. The second byte of the buffer is set to the number of characters received, excluding the carriage return (which is always the last character) unless the buffer was full prior to the carriage return. Editing of this buffer is described below and is somewhat different from CP/M-80.

—>	Copy one character
DEL	Skip one character
SEL	Copy until first occurrence of next character typed
CLS	Skip until first occurence of next character typed
↓	Copy to end of line
↑, ESC	Kill new line
HOME	Re-edit line
<—, BS	Backspace one character
INS	Toggle insert mode.

Note that CP/M-80 does not place the terminating carriage return in the buffer and that this byte will be uninitialized. However, because the count of characters does not include the carriage return, the count returned is the same as that under CP/M-80.

B **Keyboard Status Check:** If a character is available from the keyboard, A will return FFH. Otherwise A will be zero. If the available character is Control-C, Control-S, or printer echo, the appropriate action will be taken.

C **Return Version Number:** MSX-DOS returns 22H in L (and zero in H) to indicate compatibility with CP/M-80 version 2.2.

D **Disk Reset:** Flushes all file buffers. Files that have been changed in size and not closed will not be properly recorded in the disk directory until they are closed.

Unlike CP/M-80 this function need not be called before a disk change if all files which have been written have been closed.

E **Select Disk:** The drive specified in E (0=A, 1=B,etc.) is selected as the default drive.

Unlike CP/M-80, if the disk is changed the drive does *not* revert to Read Only status. Also, the number

of drives is returned in A.

F **Open File:** On entry, DE points to an unopened file control block (FCB). The disk directory is searched for the named file and A returns FFH if it is not found. If it is found, A returns zero. The extent field is used and the record count field is appropriately filled in. The size of the file (in bytes), the modification date and time are set in the FCB from information obtained from the directory. It is the application's responsibility to set the record size to the desired size if it uses the block read and write calls. It is also the application's responsibility to set the random record field and/or the extent and current record fields.

Unlike CP/M-80, there is no concept of partial or missing extents in MSX-DOS. It is not possible to create a file with "holes" in it. Therefore, this function may not fail in all the cases CP/M-80 would. Note that different directory information is copied into bytes 10-1F4 of the FCB than in CP/M-80. Also, MSX-DOS always returns a zero if successful, whereas CP/M-80 returns a number from 0 to 3, indicating the position within the logical directory sector of this file.

10 **Close File:** This function must be called after file writes to ensure all directory information and the File Allocation Table is updated. On entry, DE points to an opened FCB.

Unlike CP/M-80, MSX-DOS checks to ensure that the directory entry for this file is in the same position it was on file open. If it is different, MSX-DOS assumes that the disk has been changed and A returns FFH. Otherwise, the directory is updated to reflect the status in the FCB and A returns zero.

11 **Search First:** On entry, DE points to an unopened FCB. The disk directory is searched for the first matching name (the name could have "?"s indicating any letter matched) and if none is found. A returns FFH. Otherwise 33 (as opposed to CP/M-80's 128) locations at the disk transfer address contain a valid unopened FCB with the first byte indicating the drive number used (1=A, etc.) and A returns a zero. The extent field returned will match the one that was searched for, with the record count initialized appropriately.

Unlike CP/M-80, the directory position returned is always zero (with CP/M-80 it can be 0 to 3). Also, if the drive field contains a "?", the filename and extent field are ignored, causing anything to match. Deleted directory entries are not accessible and looking beyond the first directory entry at the disk transfer address will not work. If the extent field is a "?", the resultant extent field is set to the highest extent and record count.

12 **Search Next:** After a Function 11 has been called and has found a match, Function 12 may be called to find the next match to an ambiguous request ("?"s in the search filename). Both inputs and outputs are the same as Function 11. No intervening calls should be made between Search First and Search Next or successive Search Next requests. Search Next assumes the address of the FCB used in Search First.

13 **Delete File:** On entry, DE points to an unopened FCB. All matching directory entries are deleted. If no directory entries match, A returns FFH. Otherwise, A returns zero.

14 **Sequential Read:** On entry, DE points to an opened FCB. The record addressed by the current extent and current record is loaded at the disk transfer address, then the record number is incremented. If end-of-file is encountered, A returns 01H. A returns zero if the transfer was completed successfully.

Unlike CP/M-80, which does not recognize record sizes other than 80H, partial records are zero-filled.

15 **Sequential Write:** On entry, DE points to an opened FCB. The record addressed by the current extent and current record is written from the disk transfer address, then the record number is incremented. If the disk is full, A returns with 01H. A returns zero if the transfer was completed successfully.

Note that in the case of records smaller than sector sizes, the sector is buffered up for an eventual write when a sector's worth of data is accumulated.

16 **Create File:** On entry, DE points to an unopened FCB. The disk directory is searched for an empty directory entry, and A returns FFH if none is found. Otherwise the entry is initialized to a zero length file, the file is opened (see Function F), and A returns zero.

Unlike CP/M-80, MSX-DOS always returns zero as the directory code, instead of 0 to 3. If the file already exists and the extent field contains a zero, MSX-DOS deletes the existing file. If the extent field is non-zero, the DOS opens the file and points to that extent.

17 **Rename File:** On entry, DE points to a modified FCB which has a drive code and filename in the usual position, and a second filename starting 6 bytes after the first (DE+11H) in what is normally a reserved area. Every matching occurrence of the first is changed to the second (with the restriction that two files cannot have the same name and extension). If no match was found, A returns FFH.

Wild cards may be used in both source and destination files. If "?" appears in the second name, then the corresponding positions in the original name will be unchanged.

18 **Login Vector:** Returns 1 bit for all drives on the system in HL, with drive A the low-order bit.

19 **Current Disk:** A returns with the code of the default drive (0=A, etc.).

1A **Set Disk Transfer Address:** The disk transfer address is set to DE. On a Disk Reset (Function D) the disk transfer address is reset to 80H.

1B **FAT Address:** A drive code is passed in E. On return, A contains the number of sectors per cluster for the default drive or FFH if the drive number is invalid. BC is the sector size. DE is the number of clusters on the disk. HL is the number of free clusters. IY points to the FAT. IX points to the BPB.

CP/M-80 returns with HL pointing to a bit vector of free or allocated clusters. The cluster size can be

obtained from CP/M-80 Function 1F. No such structure exists within MSX-DOS.

1C **Write Protect Drive:** NOT IMPLEMENTED. This call will return with no processing.

1D **Get R/O Vector:** NOT IMPLEMENTED. This call will return with zeros, indicating no drives write-protected.

1E **Set File Attributes:** NOT IMPLEMENTED.

1F **Get Disk Parameter Address:** NOT IMPLEMENTED.

20 **Set/Get User Code:** NOT IMPLEMENTED.

21 **Random Read:** On entry, DE points to an opened FCB. The current extent and current record are set to agree with the random record field. This record is loaded at the disk transfer address. If end-of-file is encountered, A returns 01H. Otherwise, A returns zero if successful.

Unlike CP/M-80, which doesn't support them, partial records are zero-filled. The high byte (r2) need not be zero and is always used, which addresses files up to 2 gigabytes.

22 **Random Write:** On entry, DE points to an opened FCB. The current extent and current record are set to agree with the random record field. This record is written from the disk transfer address. A returns 01H if the disk is full. Otherwise, A returns zero.

As with Sequential Write, the sector is buffered if the record size is smaller than the sector size.

23 **File Size:** On entry, DE points to an unopened FCB. The disk directory is searched for the first matching entry and if none is found, A returns FFH. Otherwise, the random record field is set with the size of the file in records, and A returns zero.

Unlike CP/M-80, MSX-DOS supports ambiguous filenames, using the first match.

24 **Set Random Record:** On entry, DE points to an opened FCB. This function sets the random record field to the same file address as the current extent and current record fields.

25 **Disk Reset:** NOT IMPLEMENTED. This call will return after no processing.

28 **Random Block Write:** Essentially the same as Function 27 above, except for writing and a write protect indication. If there is insufficient space on the disk, A returns 01H and no records are written. If HL is zero upon entry, no records are written, but the file is set to the length specified by the random record field, whether longer or shorter than the current file size. (Allocation units are released or allocated as appropriate.)

27 **Random Block Read:** On entry, DE points to an opened FCB, and HL contains a record count which must not be zero. The specified number of records (in terms of the record size field) are

read from the file address specified by the random record field (3 bytes if sector size is greater than or equal to 40H; otherwise 4 bytes) into the disk transfer address. If end-of-file is reached, A returns 01H, zero filling any partial record. A returns zero on a successful read. In any case, HL returns the actual number of record read, and the random record field is set to address the next record.

28 **Zero Fill Random Write:** Same as call 22 except whenever a write request would extend a file contiguously, the space in between is zeroed as well as allocated.

29 NOTIMPLEMENTED

2A **Get Date:** Returns the date in DE. HL has the year, D has the month (1=Jan, etc.) and E has the day. A has the day of the week (0=Sun). If the time clock changes to the next day, the date will be adjusted accordingly, taking into account the number of days in each month and leap years.

2B **Set Date:** On entry, DE and HL must contain a valid date as described in Function 2A above. If the date is valid, A returns zero; otherwise, A returns FFH.

2C **Get Time:** Returns time of day. H has the hours, L has the minutes, D has the seconds, and E has the hundredths of seconds. This form is easily converted to a printable form, yet it can also be used to calculate (e.g., subtracting two times).

2D **Set Time:** On entry, DE and HL must contain a valid time as described in Function 2C. If the time is valid, A returns zero; otherwise A returns FFH.

2E **Verify Read After Write:** Supported.

2F **Direct Sector Read:** Supported.

30 **Direct Sector Write:** Supported.

MSX-DOS FCB Format

The following figure illustrates the MSX-DOS FCB format:

Offset	Function
0	Drive number. 0 = default, 1 = A, 2 = B.
1-8	Filename left justified with trailing blanks. All 8 bits of characters are significant. If the name of a device (PRN, CON) is placed here, do not include the optional colon.
9-B	Filename extension, left justified with trailing blanks (can be all blanks). All 8 bits of characters are significant.

C — Extent field. Used on Open, Create, Search, Sequential Read, and Sequential Write. Set by Random Read and Random Write.

D — S1. Reserved.

E — S2. An 8-bit extension of the extent field. Zeroed on Open, Create, and Search First. Used as low byte of record size for Block Read and Block Write.

F — Record Count. Normally 8DH, but set to number of 128-byte records left in extent after Open and Create. Incremented as appropriate by Sequential Write and Random Write. Used as high byte of record size for Block Read and Block Write.

10-13 — File size in bytes. In this 2-word field, the first word is the low order part of the size.

14-15 — Date the filenames were created or last modified.

Bits F-9 (year) = 0-119 (1980-2099)
Bits 8-5 (month) = 1-12
Bits 4-0 (day) = 1-31

16-17 — Time the file was created or last modified.

Bits F-B (hours) = 0-23
Bits A-5 (minutes) = 0-59
Bits 6-0 (seconds) = 0-29 (two second increments)

18-1F — Reserved

20 — Current relative record number (0-127) within the current block. You must set this field before doing sequential read/write operations to the disk. This field is not initialized by the Open function call.

21-24 — Relative record number relative to the beginning of the file, starting with zero. You must set this field before doing random read/write operations to the disk. This field is not initialized by the Open function call.

If the record size is less than 64 bytes, both words are used. Otherwise, only the first 3 bytes are used. Note that if you use the File Control Block at 5CH in the program segment, the last byte of the FCB overlaps the first byte of the unformattted parameter block.

III. MSX-BASIC LANGUAGE SPECIFICATION

Language specification for MSX-BASIC

Ver 1.4 (31st August '83)

CHAPTER 1

GENERAL INFORMATION ABOUT MSX BASIC

MSX BASIC is an extended version to the Microsoft standard Basic version 4.5, which includes supports to graphics, music and various peripherals attached to MSX Home and Personal computer. Generally, MSX BASIC is designed to follow the GW-BASIC which is a standard Basic in 16-bit machine world. But the major effort was made to make the whole system as flexible and expandable as possible.

Also MSX BASIC is featured with up to 14 digits accuracy double precision BCD arithmetic function. This means arithmetic operations no more generate strange round errors that confuse novice users. Every trancendental functions are also calculated with this accuracy. 16 bit signed integer operation is also available for faster execution.

1.1 MODES OF OPERATION

When MSX BASIC is initialized, it displays the prompt "Ok". "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 direct mode, MSX BASIC statements and commands are not preceded by line numbers. They are executed as they are entered. Results of arithmetic and logical operations may be displayed immediately and stored for later use, but the instructions themselves are lost after execution. Direct mode is useful for debugging and for using MSX BASIC as a "calculator" for quick computations that do not require a complete program.

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.

1.2 LINE FORMAT

MSX BASIC program lines have the following format (square brackets indicate optional input):

nnnnn BASIC statement [:BASIC statement...] (carraige return)

More than one BASIC statement may be placed on a line, but each must be separated from the last by a colon.

An MSX BASIC program line always begins with a line number and ends with a carraige return. A line may contain a maximum of 255 characters.

1.2.1 Line Numbers

Every MSX BASIC program line begins with a line number. Line numbers indicate the order in which the program lines are stored in memory. Line numbers are also used as references in branching and editing. Line numbers must be in the range 0 to 65529 and only integer type numbers can be used.

A period (.) may be used in LIST, AUTO, and DELETE commands to refer to the current line.

1.3 CHARACTER SET

The MSX BASIC character set consists of alphabetic characters, numeric characters, special characters, graphic characters and European characters.

The alphabetic characters in MSX BASIC are the upper case and lower case 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
\	Back slash or integer division symbol
@	At sign
_	Underscore
(backspace)	Deletes last character typed.
(escape)	Escapes
(tab)	Moves print position to next tab ston Tab stops are set every eight columns.
(line feed)	Moves to next physical line.
(carraige return)	Terminates input of a line

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 10-64 to 10+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

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

Examples:

&B01110110
&B11100111

1.4.1 Single And Double Precision Form For 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 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 constant 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
```

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.

1.5.1 Variable Names And Declaration Characters

MSX BASIC variable names may be 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 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. 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.

Numeric variable names may declare integer, single precision, or double precision values. The type declaration characters for these variable names are as follows:

%	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.

There is a second method by which variable types may be declared. The MSX BASIC statements DEFINT, DEFSTR, DEFSNG, and DEFDBL may be included in a program to declare the types for certain variable names. Refer to the description for these statements.

1.5.2 Array Variables

An array is a group or table of values referenced by the same variable name. Each element in an array is referenced by an array variable that is subscripted with an integer or an integer expression. An array variable name has as many subscripts as there are dimensions in the array. For example V(10) would reference a value in a one-dimension array, T(1,4) would reference a value in a two-dimension array, and so on. The maximum number of dimensions for an array is 255. The maximum number of elements is determined by memory size.

1.5.3 Space Requirements

The following table lists only the number of bytes occupied by the values represented by the variable names.

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 overhead plus the present contents of the string.

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 equal to a numeric variable of a different type, the number will be stored as the type declared in the variable name. (If a string variable is set equal to a numeric value or vice versa, a "Type mismatch" error occurs.)

   Example:

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

2. During expression evaluation, all of the operands in an arithmetic or relational operation are converted to the same degree of precision, i.e., that of the most precise operand. Also, the result of an arithmetic operation is returned to this degree of precision.

   Examples:

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

   The arithmetic was performed in double precision and the result was returned in D as a double precision value.

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

   The arithmetic was performed in double precision and the result was returned to D! (single precision variable), rounded, and printed as a single precision value.

3. Logical operators convert their operands to integers and return an integer result. Operands must be in the range -32768 to 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 assigned a single precision value, only the first six digits of the converted number will be valid. This is because only six digits of accuracy were supplied with the single precision value.

   Example:

```
10 A!=SQR(2)
20 B=A!
```

```
30 PRINT A!,B
RUN
 1.41421        1.41421
```

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. The MSX BASIC operators may be divided into four categories:

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

Each category is described in the following sections.

1.7.1 Arithmetic Operators

The arithmetic operators, in order of precedence, are:

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

To change the order in which the operations are performed, use parentheses. Operations within parentheses are performed first. Inside parentheses, the usual order of operations is maintained.

1.7.1.1 Integer Division And Modulus Arithmetic

Two additional operators are available in MSX BASIC:

Integer division is denoted by the yen symbol. The operands are truncated to integers (must be in the range -32768 to 32767) before the division is performed, and the quotient is truncated to an integer.

Example:

```
10$4=2
25.68$6.99=4
```

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

Modulus arithmetic is denoted by the operator MOD. Modulus arithmetic yields the integer value that is the remainder of an integer division.

Example:

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

Modulus arithmetic follows integer division in order of precedence.

1.7.1.2 Overflow And Division By Zero -

If, during the evaluation of an expression, division by zero is encountered, the "Division by zero" error message is displayed and execution of program terminates.

If overflow occurs, the "Overflow" error message is displayed and execution terminates.

1.7.2 Relational Operators

Relational operators are used to compare two values. The result of the comparison is either "true" (-1) or "false" (0). This result may then be used to make a decision regarding program flow. (See description for "IF" statements.)

The relational operators are:

Operator	Relation Tested	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 equal sign is also used to assign a value to a variable.)

When arithmetic and relational operators are combined in one expression, the arithmetic is always performed first. For example, the expression

```
X+Y<(T-1)/Z
```

is true if the value of X plus 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
```

1.7.3 Logical Operators

Logical operators perform tests on multiple relations, bit manipulation, or Boolean operations. The logical operator returns a bitwise result which is either "true" (not zero) or "false" (zero). In an expression, logical operations are performed after arithmetic and relational operations. The outcome of a logical operation is determined as shown in Table 1. The operators are listed in order of precedence.

Table 1. MSX BASIC Relational Operators Truth Table

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

Just as the relational operators can be used to make decisions regarding program flow, logical operators can connect two or more relations and return a true or false value to be used in a decision .

Example:

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

Logical operators work by converting their operands to 16-bit, signed, two's complement integers in the range -32768 to 32767. (If the operands are not in this range, an error results.) If both operands are supplied as 0 or -1, logical operators return 0 or -1. The given operation is performed on these integers in bitwise fashion, i.e., each bit of the result is determined by the corresponding bits in the two operands.

Thus, it is possible to use logical operators to test bytes for a particular bit pattern. For instance, the AND operator may be used to "mask" all but one of the bits of a status byte at a machine I/O port. The OR operator may be used to "merge" two bytes to create a particular binary value. The following examples will help demonstrate 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, which is the two's complement representation of -1.
NOT X=-(X+1)	The two's complement of any integer is the bit complement plus one.

1.7.4 Functional Operators

A function is used in an expression to call a predetermined operation that is to be performed on an operand. MSX BASIC has "intrinsic" functions that reside in the system, such as SQR (square root) or SIN (sine).

MSX BASIC also allows "user-defined" functions that are written by the programmer. See descriptions for "DEF FN".

1.7.5 String Operations

Strings may be concatenated by using +.

Example:

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

Strings may be compared using the same relational operators that are used with numbers:

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

String comparisons are made by taking one character at a time from each string and comparing the ASCII codes. If all the ASCII codes are the same, the strings are equal. If the ASCII codes differ, the lower code number precedes the higher. If during string comparison the end of one string is reached, the shorter string is said to be smaller. Leading and trailing blanks are significant.

Examples:

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

Thus, string comparisons can be used to test string values or to alphabetize strings. All string constants used in comparison expressions must be enclosed in quotation marks.

1.8 PROGRAM EDITING

The Full Screen Editor equiped with MSX BASIC allows the user to enter program lines as usual, then edit an entire screen before recording the changes. This time-saving capability is made possible by special

keys for cursor movement, character insertion and deletion, and line or screen erasure. Specific functions and key assignments are discussed in the following sections.

With the Full Screen Editor, a user can move quickly around the screen, making corrections where necessary. The changes are entered by placing the cursor on the first line changed and pressing <RETURN> at the beginning of each line. A program line is not actually changed until <RETURN> is entered from somewhere within the line.

Writing Programs

Within MSX BASIC, the editor is in control any time after an OK prompt and before a RUN command is issued. Any line of text that is entered is processed by the editor. Any line of text that begins with a number is considered a program statement.

Program statements are processed by the editor in one of the following ways:

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

2. An existing line is modified. This occurs if the line number matches that of an existing line in the program. The existing line is replaced with the text of the new line.

3. An existing line is deleted. This occurs if the line number matches that of an existing line, and the new line contains only the line number.

4. An error is produced.

   If an attempt is made to delete a non-existent line, an "Undefined line number" error message is displayed.

   If program memory is exhausted, and a line is added to the program, an "Out of memory" error is displayed and the line is not added.

More than one statement may be placed on a line. If this is done, the statements must be separated by a colon (:). The colon does not have to be surrounded by spaces.

The maximum number of characters allowed in a program line, including the line number, is 255.

Editing Programs

Use the LIST statement to display an entire program or range of lines on the screen so that they can be edited. Text can then be modified by moving the cursor to the place where the change is needed and performing one of the following actions:

1. Typing over existing characters
2. Deleting characters to the right of the cursor
3. Deleting characters to the left of the cursor
4. Inserting characters
5. Appending characters to the end of the logical line

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

Changes to a line are recorded when a carriage return is entered while the cursor is somewhere on the line. The carriage return enters all changes for that logical line, no matter how many physical lines are included and no matter where the cursor is located on the 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 standard conversions.

Individual control functions are described following the table.

Table 1. MSX BASIC Control Functions. The ASCII control key is entered by pressing the key while holding down the Control key.

Hex. Code		Control Key	Special Key	Function
01		A		Ignored
02	*	B		Move cursor to start of previous word
03	*	C		Break when MSX BASIC is waiting for input
04	*	D		Ignored
05	*	E		Truncate line (clear 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 (moves 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 to end of line
0F	*	O		Ignored
10	*	P		Ignored
11	*	Q		Ignored
12	*	R	INS	Toggle insert/typeover mode

13 *	S		Ignored
14 *	T		Ignored
15 *	U		Clear logical line
16 *	V		Ignored
17 *	W		Ignored
18 *	X	Select	Ignored
19 *	Y		Ignored
1A *	Z		Ignored
1B	[	ESC	Ignored
1C *	\	Right arrow	Cursor right
1D *	]	Left arrow	Cursor left
1E *	^	Up arrow	Cursor up
1F *	_	Down arrow	Cursor down
7F	DEL	DEL	Delete character at cursor

Note: Those keys marked with asterisk(*) cansels insert mode when editor is in insert mode.

PREVIOUS WORD

The cursor is moved left to the previous word. The previous word is defined as the next character to the left of the cursor in the sets A-Z, a-z, or 0-9.

BREAK

Returns to MSX BASIC direct mode, without saving changes that were made to the line currently being edited.

TRUNCATE

The cursor is moved to the end of the logical line. The characters it passes over are deleted. Characters typed from the new cursor position are appended to the line.

NEXT WORD

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

BEEP

The beep sound will be produced.

BACKSPACE

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

TAB

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

CURSOR HOME

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

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 sends it to MSX BASIC.

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

INSERT
Toggle switch for insert mode. When insert mode is on, the size of the cursor is reduced and characters are inserted at the current cursor position. Characters to the right of the cursor move right as new ones are inserted. Line wrap is observed.
When insert mode is off, the size of cursor returned to normal size and typed characters will replace existing characters on the line.

CLEAR LOGICAL LINE
When this key is entered anywhere in the line, the entire logical line is erased.

CURSOR RIGHT
Moves the cursor one position to the right. Line wrap is observed.

CURSOR LEFT
Move the cursor one position to the left. Line wrap is observed.

CURSOR UP
Moves the cursor up one physical line (at the current position).

CURSOR DOWN
Move the cursor down one physical line (at the current position).

Logical line Definition with INPUT

Normally, a logical line consists of all the characters on each of the physical lines that make up the logical line. During execution of an INPUT or LINE INPUT statement, however, this definition is modified slightly to allow for forms input. When either of these statements is executed, the logical line is restricted to characters actually typed or passed over by the cursor. Insert mode and the delete function only move characters which are within that logical line, and Delete will decrement the size of the line.

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. In this case, the non-blank characters not part of the logical line are preserved and the characters at the end of the logical line are thrown out. This preserves labels that existed prior to the INPUT statement. If an incorrect character is entered as a line is being typed, it can be deleted with the <Back Space> key or with Control-H. and they backspacing over a character and erasing it. Once a character(s) has been deleted, simply continue typing the line as desired.

To delete a line that is in the process of being typed, type Control-U.

To correct program lines for a program that is currently in memory, simply retype the line using the same line number. MSX BASIC will automatically replace the old line with the new line.

To delete the entire program currently residing in memory, enter the NEW command. NEW is usually used to clear memory prior to entering a new program.

1.9 Special keys

MSX BASIC supports several special keys as follows.

1.9.1 Function Keys

MSX BASIC has 10 pre-defined function keys. The current contents of these keys are displayed on the last line on the screen and can be re-defined by program with KEY statement. The initial values for each keys are:

F1	color[b]
F2	auto[b]
F4	goto[b]
F5	list[b]
F5	run[cr]
F6	color 15,4,7[cr]
F7	cload"
F8	cont[cr]
F9	list.[cr][u][u]
F10	[cls]run[cr]

[b] = blank character
[cr]= carriage return
[u] = cursor up character
[cls]=clear screen character

Function keys are also used as event trap keys. See ON KEY GOSUB and KEY ON/OFF/STOP statement for details.

1.9.2 Stop key

When MSX BASIC is in command mode, the STOP key has no effects to the operation, MSX BASIC just ignores it.

When MSX BASIC is executing the program, pressing the STOP key causes

suspension of the program execution, and MSX BASIC turn on the cursor display to indicate that the execution is suspended. Another STOP key input resumes the execution. If the STOP key and control key are pressed simultaneously, MSX BASIC terminates the execution and return to command mode with following message.

```
Break in nnnn
```

where nnnn is the program line number where the execution stopped.

1.10 ERROR MESSAGES

If an error causes program execution to terminate, an error message is printed. For a complete list of MSX BASIC error codes and error messages, see Appendix A.

CHAPTER 2

MSX BASIC COMMANDS, STATEMENTS AND FUNCTIONS

2.1 Commands, Statements, and Functions except I/O

2.1.1 Commands except I/O

AUTO [<line number>[,<increment>]]
To generate a line number automatically after every carriage return.

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

If AUTO generates a line number that is already being used, an asterisk is printed after the line number to warn the user that any input will replace the existing line. However, typing a carriage return immediately after the asterisk will save the line and generate the next line number.

AUTO is terminated by typing Control-C or Control-STOP. The line in which Control-C is typed is not saved. After Control-C is typed, BASIC returns to command level.

CONT
To continue program execution after BREAK or STOP in execution.

DELETE [<line number>][-<line number>]
To delete program lines.

BASIC always returns to command level after a DELETE is executed. If <line number> does not exist, an 'Illegal function call' error occurs.

LIST [<line number>[-[<line number>]]]
To list all or 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> is specified, that line is listed.

If the first <line number> and "-" are specified, that line and all higher-numbered lines are listed.

If "-" and the second <line number> are specified, all lines from the beginning of the program through that line are listed.

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

Listing is terminated by typing "CTRL" and "STOP" keys at same time. Listing is suspended by typing "STOP" key. and it resumed by typing "STOP" key again.

LLIST [<line number>[-[<line number>]]]
To list all or part of the program on the printer. (See the LIST command for details of the parameters)

NEW
To delete entire program from working memory and reset all variables.

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

<new number> is the first line number to be used in the new sequence. The default is 10. <old number> is the line in the current program where renumbering is to begin. The default is the first line of the program. <increment> is the increment to be used in the new sequence. The default is 10.

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

NOTE: RENUM cannot be used to change the order of program lines (for example, RENUM 15,30 when the program has three lines numbered 10, 20 and 30) or to create line numbers greater than 65529. An 'Illegal function call' error will result.

RUN [<line number>]
To execute a program.

If <line number> is specified, execution begins on that line. Otherwise, execution begins at the lowest line number.

TRON/TROFF
To trace the execution of program statements.

As an aid in debugging, the TRON statement (executed in either the direct or indirect mode) enables a trace flag that prints each line number of the program as is executed. The numbers appear enclosed in square brackets. The trace flag is disabled with the TROFF statement (or when a NEW command is executed).

CLEAR [<string space>[,<highest location>]]
To set all numeric variables to zero, all string variables to null, and close all open files, and optionally, to set the end of memory.

<string space>
Space for string variables. Default size is 200 bytes.
<Highest location>
The highest memory location available for use by BASIC.

DATA <list of constants>
To store the numeric and string constants that are accessed by the program's READ statement(s).

DATA statements are nonexecutable and may be placed anywhere in the program. A DATA statement may contain as many constants as will fit on a line (separated by commas), and any number of DATA statements may be used in a program. The READ statements access the DATA statements in order (by line number) and the data contained there in may be thought of as one continuous list of items, regardless of how many items are on a line or where the lines are placed in the program.

<list of constants> may contain numeric constants in any format; i.e.,fixed point, floating point, or integer. (No numeric expressions are allowed in the list.) String constants in DATA statements must be surrounded by double quotation marks only if they contain comma, colons, or significant leading or trailing spaces. Otherwise, quotation marks are not needed.

The variable type (numeric or string) given in the READ statement must agree with the corresponding constant in the DATA statement. DATA statements may be read from the beginning or specified line by use of the RESTORE statement.

DIM <list of subscripted variables>
To specify the maximum values for array variable subscripts and allocate storage accordingly.

If an array variable name is used without a DIM statement, the maximum value of its subscript(s) is assumed to be 10. If a subscript is used that is greater than the maximum specified, a 'Subscript out of range' error occurs. The minimum value for a subscript is always 0.

DEFINT <range(s) of letters>
DEFSNG <range(s) of letters>
DEFDBL <range(s) of letters>
DEFSTR <range(s) of letters>

To declare variable type as integer, single precision, double precision, or string.

DEFINT/SNG/DBL/STR statements declare that the variable names beginning with the letter(s) specified will be that type variable. However, a type declaration character always takes precedence over a DEFxxx statement in the typing of a variables. (See the end of section 1.5.1, for details of declaration characters.)

DEF FN<name>[(<parameter list>)]=<function definition>

To define and name a function that is written by the user.

<name> must be a legal variable name. This name, preceded by FN. becomes the name of the function. <parameter list> comprised of those variable name in the function definition that are to be replaced when the function is called. The items in the list are separated by commas. <function definition> is an expression that performs the operation of the function. It is limited to one line. Variable names that appear in this expression serve only to define the function; they do not affect program variables that have the same name. A variable name used in a function definition may or may not appear in the parameter list. If it does, the value of the parameter 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 a type is specified in the function name, the value of the expression is forced to that type before it is returned to the calling statement. If a type is specified in the function name and the argument type does not match, a 'Type mismatch' error occurs.

A DEFFN statement must be executed before the function it defines may be called. If a function is called before it has been defined, an 'Undefined user function' error occurs. DEFFN is illegal in the direct mode.

DEFUSR[<digit>]=<integer expression>

To specify the starting address of an assembly language subroutine.

<digit> may be any digit from 0 to 9. The digit 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 starting address of the USR routine

Any number of DEFUSR statements may appear in a program to redefine subroutine starting addresses, thus allowing access to as many subroutines as necessary.

ERASE <list of array variables>
To eliminate 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 first ERASEing it, a '"Redimensioned array' error occurs.

END

To terminate program execution, close all files and return to command level.

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

ERROR <integer expression>
To simulate the occurrence of an error or to allow 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 your own error code, use a value that is greater than any used by BASIC for error codes. See Appendix A for a list of error codes and messages. (It is preferable to use the highest available values, so compatibility may be maintained when more error codes are added to BASIC.) This user defined error code may then be conveniently handled in an error trap routine.

Example:

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

If an ERROR statement specified a code for which no error message has been defined, BASIC responds with the message 'Unprintable error'. Execution of an ERROR statement for which there is no error trap routine causes an 'Unprintable error' error message to be printed and execution to halt.

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

NEXT [<variable>][,<variable>...]

note: <Variable> can be integer,single-precision or double-precision. where x,y,z are numeric expressions.

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

<variable> is used as a counter. 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 amount specified by STEP. A check is performed to see if the value of the counter is now greater than the final value (y). If it is not greater, BASIC branches back to the statement after the FOR statement and the process is repeated. If it is greater, execution continues with the statement following the NEXT statement. This is a FOR...NEXT loop. If STEP is not specified, the increment is assumed to be one.

If step is negative, the final value of the counter is set to 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 body of the loop is executed one time at least 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 the context of another FOR...NEXT loop. When loops are nested, each loop must have a unique variable name as its counter. The NEXT statement for the inside loop must appear before that for the outside loop. If nested loops have the same end point, a single NEXT statement may be used for all of them. Such nesting of FOR...NEXT loops is limited only by 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>]

To branch to subroutine beginning at <line number> and return from a subroutine.

<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. Such nesting of subroutines is limited only by available memory.

The RETURN statement(s) in a subroutine cause BASIC to branch back to the statement following the most recent GOSUB statement. A subroutine may contain more than one RETURN statement, should logic dictate a return at different points in the subroutine. Subroutines may appear anywhere in the program, but it is recommended that the subroutine be readily distinguishable from the main program. To prevent inadvertent entry into the 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 message is issued and execution is terminated.

GOTO <line number>

To branch 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>]

To make a decision regarding 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,

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

will not print "A<>C" when A<>B. It 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>
To allow input from the keyboard during program execution.

When an INPUT statement is encountered, program execution pauses and a question mark is printed to indicate the program is waiting for data. If "<prompt string>" is included, the string is printed before the question mark. The required data is then entered at 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. Data items are separated by commas.

The names in the <list of variables> may be numeric or string variable names (including subscripted variables). The type of each data item that is input must agree with the type specified by the variable name. (Strings input to an INPUT statement need not be surrounded by quotation marks.)

Responding to input with the wrong type of value (string instead of the numeric, etc.) causes the message "?Redo from start" to be printed. No assignment of input value is made until an acceptable response is given.

Example:
```
list
10 INPUT "A and B";A,B
20 PRINT A+B
Ok
run
A and B? 10,@0
?Redo from start
A and B? 10,20
 30
Ok
```

Responding to INPUT with too many items causes the message "?Extra ignored" to be printed and the next statement to be executed.

Example:
```
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 INPUT with too few item 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
```

Escape INPUT by typing Control-C or the "CTRL" and "STOP" keys simultaneously. BASIC returns to command level and types "Ok". Typing CONT resumes execution at the INPUT statement.

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

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

The prompt string is a string literal that is printed at the console before input is accepted. A question mark is not printed unless it is part of the prompt string. All input from the end of the prompt to the carriage return is assigned to <string variable>.

Escape LINE INPUT by typing Control-C or the "CTRL" and "STOP" keys simultaneously. BASIC returns to command level and types "Ok". Typing CONT resumes execution at the LINE INPUT statement.

[LET] <variable>=<expression>

; To assign value of an expression to a variable.

Notice the word LET is optional; i.e., the equal sign is sufficient when assigning an expression to a variable name.

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

; To print data at the line printer. (see PRINT and PRINT USING statements below for details.)

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

; To replace a portion of one string with another string.

The character 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 replacement of characters never goes beyond the original length of <string exp.1>.

ON ERROR GOTO <line number>

; To enable error trapping and specify the first line of the error

handling subroutine.

Once error trapping has been enabled all errors detected, including direct mode errors (e.g., SN (Syntax) errors), will cause a jump to the specified error handling subroutine. If <line number> does not exist, an 'Undefined line number' error results. To disable error trapping, execute an ON ERROR GOTO 0. Subsequent errors will print an error message and halt execution. An ON ERROR GOTO 0 statement that appears in an error trapping subroutine causes BASIC to stop and print 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 number>
ON <expression> GOSUB <list of line number>
; To branch 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 a noninteger, 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 zero or greater than the number of items in the list (but less than or equal to 255), BASIC continues with the next executable statement. If the value of <expression> is negative or greater than 255, a 'Illegal function call' error occurs.

POKE <address of the memory>,<integer expression>
; To write a byte into a memory location.

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

PRINT [<list of expressions>]
; To output data to the console.

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

expressions are printed at 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 printed item is determined by the punctuation used to separate the items in the list. BASIC divides the line into print zones of 14 spaces each. In the <list of expressions>, a comma causes the next value to be printed at the beginning of the next zone. A semicolon causes the next value to be printed immediately after the last value. Typing one or more spaces between expressions has the same effect as typing a semicolon.

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, BASIC goes to the next physical line and continues printing.

Printed numbers are 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 in place of the word PRINT in a PRINT statement.

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

<list of expressions> comprises the string expressions or numeric expressions that are to be printed, separated by semicolons. <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 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&"

Specifies that 2+n characters from the string are tc 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
```

"@"

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. 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 "###.##-";12.5,-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
$$
```

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

Example:

```
PRINT USING "$$###.##";12.35,-12.35
   $12.35  $12.35-
OK
"**$"
```

The **$ at the beginning of a format string combines the effects of the above two symbols. Leading spaces will be asterisk-filled and a $ sign will be printed before the number. **$ specifies three more digit positions, one of which is the $ 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>
; To read values from a DATA statement and assign 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>
; To allow 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 into (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 this in a DATA statement as it would be considered legal data.

RESTORE [<line number>]
; To allow 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>

; To continue program execution after an error recovery procedure has been performed.

Any one of the four formats shown above 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

; To terminate program execution and return to command level.

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>

; To exchange the value of two variables.

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

2.1.3 Functions, except 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.

n is a numeric expression in the range -32768 to 65535. If n is negative, the two's complement from is used. That is, BIN$(-n) is the same as BIN$(65536-n).

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

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

CINT(X)
; Converts X to a integer number by truncating the fractional portion. If X isn't the range -32768 to 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
; 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. The ERR and ERL variables are usually used in IF...THEN statements to direct program flow in the error trap routine.

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

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

Otherwise, use

```
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 equal 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 program, text file, machine language program file, etc. 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.

n is a numeric expression in the range -32768 to 65535. If n is negative, the two's complement from 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-C, which terminates the program.

INPUT$(X)
; Returns a string of X characters, read from the keyboard. No character will be echoed and all characters are passed through except Control-C, terminates the execution of the INPUT$ function.

INSTR([I,]X$,Y$)
; Searches for the first occurrence of string Y$ in X$ and returns the position at which the match is found. 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 must be greater than zero.

LPOS(X)
; Returns the current position of the line printer print head within the line printer buffer. Does not necessarily give 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.

n is a numeric expression in the range -32768 to 65535. If n is negative, the two's complement from is used. That is, 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. For 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 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 an 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.

2.2 Device specific statements and functions.

--- Expanded statements and functions for MSX ---

2.2.1 Statements

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

; To assign 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 the key click
non zero:enable the 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. Baud rate can also be determined using CSAVE command with baud rate option.

Note that when reading cassette, baud rate is automatically determined, so the user don't have to know in what baud rate the cassette is written. <printer option> determines if the printer in operation is 'MSX printer' (which has 'graphics symbol' and 'European' 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 European MSX characters are converted to equivalent ISO characters.

Width <width of screen in text mode>
; To Set the width of display during text mode. Legal value is 1..40 in 40x24 text mode, 1..32 in 32x24 text mode.

CLS ; To clear the screen. Valid in all screen modes.

LOCATE [<x>] [,<y>] [,<cursor display switch>]
; To locate character 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>]
;To define the color. Defaults to 15,4,7. The argument can be in the range of 0..15. Actual color corresponding to each value is as follows.

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>]
; To set up sprite attributes.

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

<coordinates specifier> always can 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 form is more common and directly refers to a point without regard to the last point referenced. Examples are:

```
(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. It may seem strange to start numbering y at the top so the bottom left corner is (0,191) in both high-resolution and medium resolution, but this is the standard.

Above description can be applied wherever graphic coordinate is used.

X coordinate <x> may range from -32 to 255. Y coordinates <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>, then that sprite disappears from the screen. (Refer to VDP manual for further details.)

When a field is omitted, the current value is used. At start up, 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 size of sprites is 2 or 3. <pattern number> defaults to the <sprite plane number>. (See also SCREEN statement and SPRITE$ variable)

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

; To draw 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 the detail of <coordinate specifier>, see the description at 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>
; To draw figure according to the graphic macro language.

The graphic macro language commands are contained in the string expression string. The string defines an object, which is 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 detailed 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 und right
F n     ;Moves diagonally down and right
G n     ;Moves diagonally down and left
H n     ;Moves diagonally up and left
```

n in each of the preceding commands indicating 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.
```

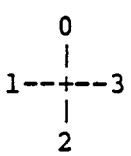
The aspect ratio of the screen is 1. So 8 horizontal points are equal in length to 8 vertical points.

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 degree, 1 is 90, 2 is 180, 3 is 270.
```




```
C n     ;Sets color n. n may range 0 to 15.
```

S n ;Sets scale factor. n may range from 0 to 255. n divided by 4 is the scale factor. For example, if n=1, then the scale factor is 1/4. The scale factor multiplied by the distance given with the U,D,L,R,E,F,G,H, and relative M commands gives the actual distance moved. The default value is 0, which means 'no-scaling' (i.e., 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 when you use a variable this way, or in the X command. Otherwise, a semicolon is optional between commands. Spaces are ignored in string. Foe example, you could use variables in a move command 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>]
; To draw line connecting the two specified coordinate. For the detail of the <coordinate specifier>, see description at PUT SPRITE statement.

If 'B'is specified, draws rectangle. If 'BF' is specified, fills rectangle.

PAINT <coordinate specifier>[,<paint color>][,<color regarded as border>]
; To fill in an arbitrary graphics figure of the specified fill color starting at <coordinate specifier>. For the detail of the <coordinate specifier>, see the description at PUT SPRITE statement. PAINT does not allow <coordinate specifier> to be out of the screen.

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

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

; To set/reset the specified coordinate. For the detail of the <coordinate specifier>, see the description at PUT SPRITE statement

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>

; To set 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

; To list the contents of all function keys.

Example:

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

"color" aligns with 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 a function key is converted to spaces.

KEY ON|OFF

; To turn on/off function key display on 24th line of text screen.

ON KEY GOSUB <list of line numbers>

; To set 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
; To activate/deactivate 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 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 not remembered even if it does take place.

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 an immediate trap will take place when KEY(n)ON is executed.

KEY(n)ON has no effect on whether the function key value are displayed at the bottom of the console.

ON STRIG GOSUB <list of line numbers>
; To set 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 this automatically disables all trapping (including ERROR, STRIG, STOP, SPRITE, INTERVAL and KEY).

STRIG (<n>) ON/OFF/STOP
; To activate/deactivate trapping of trigger buttons of joy sticks in a BASIC program.

<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 joy-stick 1 is used. When <n> is either 2 or 4, joy-stick 2.

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

of trigger button. After STRIG(n)ON statement, 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>

; To set up a line numbers for BASIC to trap to when the Control-STOP key is pressed.

When the trap occurs an automatic STOP STOP 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 executiog 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).

The user must be VERY careful when using this statement. For example, following program cannot be aborted. The only way left is to reset the system!

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

STOP ON/OFF/STOP

; To activate/deactivate trapping of a 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>

; To set 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 executiog 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

; To activate/deactivate 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>

; To set up a line number for BASIC to trap to time interval.

Generates a timer interrupt at 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

; To activate/deactivate trapping of time interval in a BASIC program.

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 occur, this is remembered so an immediate trap will take place when INTERVAL ON is executed.

VPOKE <address of VRAM>,<value to be written>
; To poke a value to specified location of VRAM. <address of VRAM> can be in the range of 0..16383. <value to be written> should be a byte value.

BEEP
; To generate a beep sound. Exactly the same with outputting CHR$(7).

MOTOR [<ON|OFF>]

; To change 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>

; To write value directly to the <register of PSG>.

PLAY <string exp for voice 1>[,<string exp for voice 2> [,<string exp for voice 3>]]

; To play music according to 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.

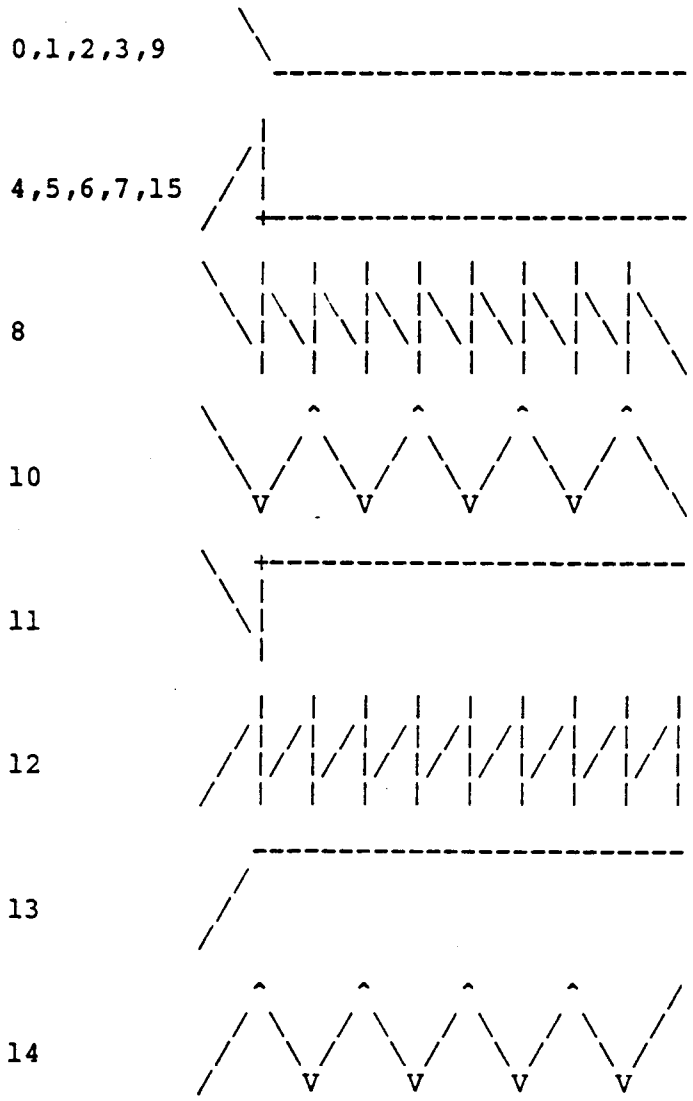
;(Dot or period) After a note, causes the note to be played as a dotted note. That is, its length is multiplied by 3/2. More than one dot may appear after the note, and the length is adjusted accordingly. For example, "A..." will play 27/8 as long, etc. Dots may also appear after the pause(P) to scale 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 values specified with above commands will be reset to the system default when beep sound is generated.

MAXFILES=<expression>
; To specify the maximum number of files opened at a time. <expression> can be in the range of 0..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>

; To allocate 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, 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 SLOT.MEM for further details.

<mode> is one of the following:

OUTPUT	: Specifies sequential output mode
INPUT	: Specifies sequential input mode
APPEND	: Specifies 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>

; To write data to the specified channel. (See PRINT/PRINT USING statements for details.)

INPUT #<file number>,<variable list>

; To read data items from the specified channel and assign 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, 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 the 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>
; To read 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>)
; To Return a string of n characters, read from the file. <file number> is the number which the file was OPENed.

CLOSE [[#]<file number>[,<file number>]]
; To close the channel and releases the buffer associated with it. If no <file number>'s are specified, all open channels are closed.

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

LOAD "<device_descriptor>[<file name>]"
; To load 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>]"
; To merge 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 the <file name> is omitted, the next program files, it should be ASCII file, file encountered on the tape is MERGEd. Control-Z is treated as end-of-file.

BSAVE "<device descriptor>[<file name>]",<top adrs>,<end adrs>
[,<execution adrs>]
; To save 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>.

Example:

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

BLOAD "<device_descriptor>[<file name>]"[,R][,<offset>]
; To load 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>]
; To save a BASIC program file to the 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 intended to be MERGEd must be saved in ASCII format. Programs saved in ASCII may be read as BASIC data files and text files. In that case, use the SAVE command.

<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>"]
; To load a BASIC program file from the CMT.

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, baud rate is determined automatically.

CLOAD? ["<file name>"]
; To verify a BASIC program on CMT with one in memory.

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

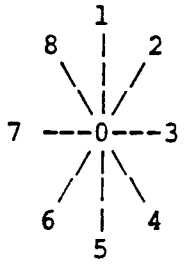
```
CALL TALK("Yamashita","Hayashi","Suzuki GSX400FW")
_TALK("Yamashita","Hayashi","Suzuki GSX400FW")
```

2.2.2 Functions

POINT(<X coordinate>,<Y coordinate>)
; Returns 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..16383.

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



STRIG(<n>)
; Returns the status of a trigger button of a joy-stick. <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 joy-stick 1 is used. When <n> is either 2 or 4, joy-stick 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..12. When <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..7.

When 0..3 is specified, touch pad connected to joy stick port 1 is selected, when 4..7, port 2.

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 immediate after the PLAY statement is issued, the PLAY function returns -1 regardless to the actual status of the music queue.

EOF(<file number>)
; Return -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.

2.2.3 Special variables

Following are the 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 pattern of sprite.

<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 assign the string that is shorter than 32 character, 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: Following two are system variables which can be evaluated
*        or assigned like other ordinary variables. Prepared for
*        advanced programmers only. If you don't know the meaning,
*        never use.
*
*************************************************************************
```

VDP(<n>) (type: unsigned byte)
; If <n> is in the range of 0..7, specifies the current value of VDP's write only register. If <n> is 8, specifies the status register of 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 - meaningless                                                 \
 2 - base of pattern generator table for text mode.  > 40 * 24
 3 - meaningless                                                 /
 4 - meaningless                                                /

 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 - meaningless
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.
```

2.2.4 Machine dependent statements and function

```
**************************************************************************
*                                                                        *
*  Note: Following statements and function access machine's I/O port     *
*        directly. So, the programs that use those statements and        *
*        functions will not be compatible with MSX systems released      *
*        future. Programs distributed to the public should not use       *
*        those  statements and functions.                                *
*                                                                        *
**************************************************************************
```

OUT <port number>,<integer expression>
; To send 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]
; To suspend 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 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 to be 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 above statements and functions, <port number> is handled with 16bit number to support Z80's capability that accesses I/O port with [BC] register pair. However, statndard MSX system does not support those extended I/O address space, port number larger than 255 is meaningless.

CHAPTER 3

APPENDIX

A. 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 as the result of:

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 Unprintable errors
.
49 These codes have no definitions. Should be reserved for future expansion in 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.

IV. BIOS ENTRY POINT LIST

```
        COMMENT %

                Following RST's (RST 0  thru  RST  5)  are  reserved  for  BASIC
                interpreter, RST  6  for  inter-slot  calls,  RST 7 for hardware
                interrupt.

                Following notations are used.

                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  we  want.   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  e
        ;                       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  read  the
        ;                       content 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  - content of memory
;       Modifies:               AF, BC, DE
;       Note:                   Interrupts are disabled automatically but 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 write to
;                               the 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 never
;                               enabled by this routine.
;
0014            ENT     WRSLT
                HOLE    1
;
;       Name:                   OUTDO
;       Function:               Outputs to 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:                  IYH - FxxxSSPP
;                                     |   ||||
;                                     |   ||++-- primary slot # (0-3)
;                                     |   ++---- secondary slot # (0-3)
;                                     +--------- 1 if secondary slot # specified
```

```
;
;                       IX  - address to call
;       Returns:        Who knows?
;       Modifies:       Who knows?
;       Note:           Interrupts are disabled automatically but never
;                       enabled by this routine.  You can never pass
;                       arguments via alternate registers of Z80 or
;                       IX, 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 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
;
;       Following 5 bytes are reserved to store version number of MSX.
;       First versions hold 5 zeros.
;
                HOLE    5
;
;       Name:           CALLF
;       Function:       Performs far call (i.e., inter-slot call)
;       Entry:          None
```

```
        ;       Returns:        Who knows?
        ;       Modifies:       ditto
        ;       Note:           Calling sequence is as follows.
        ;
        ;                       RST     6
        ;                       DB      destination slot
        ;                       DW      destination address
        ;
        ;                       For precise description  about  parameters,  see
        ;                       CALSLT.
        ;
0030            ENTR    CALLF
                HOLE    5
        ;
        ;       Name:           KEYINT
        ;       Function:       Performs hardware interrupt procedures necessary
        ;       Entry:          None
        ;       Returns:        None
        ;       Modifies:       None
        ;
0038            ENTR    KEYINT
```

```
        COMMENT %

                Following 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 string contents
        ;       Entry:          None
        ;       Returns:        None
        ;       Modifies:       All
        ;
003E            ENT     INIFNK
```

```
        COMMENT %

                Following are used to access 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 VDP register
        ;       Entry:          Register # in [C], data in [B]
        ;       Returns:        None
        ;       Modifies:       AF, BC
        ;
0047            ENT     WRTVDP
        ;
        ;       Name:           RDVRM
        ;       Function:       Reads VRAM addressed by [HL]
        ;       Entry:          HL
        ;       Returns:        A
        ;       Modifies:       AF
        ;
004A            ENT     RDVRM
        ;
        ;       Name:           WRTVRM
        ;       Function:       Writes to VRAM addressed by [HL]
        ;       Entry:          HL, A
        ;       Returns:        None
        ;       Modifies:       AF
        ;
004D            ENT     WRTVRM
        ;
        ;       Name:           SETRD
        ;       Function:       Sets up VDP for read
        ;       Entry:          HL
        ;       Returns:        None
        ;       Modifies:       AF
        ;
0050            ENT     SETRD
        ;
        ;       Name:           SETWRT
```

```
                ;       Function:       Sets up VDP for write
                ;       Entry:          HL
                ;       Returns:        None
                ;       Modifies:       AF
                ;
0053                    ENT     SETWRT
                ;
                ;       Name:           FILVRM
                ;       Function:       Fills VRAM with specified data
                ;       Entry:          Address in [HL], length in [BC], data in [Acc]
                ;       Returns:        None
                ;       Modifies:       AF, BC
                ;
0056                    ENT     FILVRM
                ;
                ;       Name:           LDIRMV
                ;       Function:       Moves block of memory from VRAM 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 VRAM.
                ;       Entry:          Address of  source in [HL], destination in [DE],
                ;                       length in [BC].
                ;       Returns:        None
                ;       Modifies:       All
                ;
005C                    ENT     LDIRVM
                ;
                ;       Name:           CHGMOD
                ;       Function:       Sets VDP mode according to SCRMOD
                ;       Entry:          SCRMOD (0..3)
                ;       Returns:        None
                ;       Modifies:       All
                ;
005F                    ENT     CHGMOD
                ;
                ;       Name:           CHGCLR
                ;       Function:       Changes color of 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 ar
        ;                       set to foreground color, vertical position
        ;                       are set to 209.
        ;       Entry:          SCRMOD
        ;       Returns:        None
        ;       Modifies:       All
        ;
0069            ENT     CLRSPR
        ;
        ;       Name:           INITXT
        ;       Function:       Initializes screen for text mode (40*24), sets
        ;                       VDP.
        ;       Entry:          TXTNAM, TXTCGP
        ;       Returns:        None
        ;       Modifies:       All
        ;
006C            ENT     INITXT
        ;
        ;       Name:           INIT32
        ;       Function:       Initializes screen for text mode (32*24), sets
        ;                       VDP.
        ;       Entry:          T32NAM, T32CGP, T32COL, T32ATR, T32PAT
        ;       Returns:        None
        ;       Modifies:       All
        ;
006F            ENT     INIT32
        ;
        ;       Name:           INIGRP
        ;       Function:       Initializes screen for hiresolution mode, sets
        ;                       VDP.
        ;       Entry:          GRPNAM, GRPCGP, GRPCOL, GRPATR, GRPPAT
        ;       Returns:        None
        ;       Modifies:       All
        ;
0072            ENT     INIGRP
        ;
        ;       Name:           INIMLT
        ;       Function:       Initializes screen for multicolor mode, sets
        ;                       VDP.
        ;       Entry:          MLTNAM, MLTCGP, MLTCOL, MLTATR, MLTPAT
        ;       Returns:        None
        ;       Modifies:       All
        ;
0075            ENT     INIMLT
        ;
        ;       Name:           SETTXT
        ;       Function:       Sets VDP for text (40*24) mode
        ;       Entry:          TXTNAM, TXTCGP
        ;       Returns:
```

```
        ;       Modifies:
        ;
0078            ENT     SETTXT
        ;
        ;       Name:           SETT32
        ;       Function:       Sets VDP for text (32*24) mode
        ;       Entry:          T32NAM, T32CGP, T32COL, T32ATR, T32PAT
        ;       Returns:        None
        ;       Modifies:       All
        ;
007B            ENT     SETT32
        ;
        ;       Name:           SETGRP
        ;       Function:       Sets VDP for hiresolution mode
        ;       Entry:          GRPNAM, GRPCGP, GRPCOL, GRPATR, GRPPAT
        ;       Returns:        None
        ;       Modifies:       All
        ;
007E            ENT     SETGRP
        ;
        ;       Name:           SETMLT
        ;       Function:       Sets 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 current sprite size
        ;       Entry:          None
        ;       Returns:        Sprite size (# of bytes) in [Acc].
        ;                       Carry   set   if   16*16   sprite   in   use,   reset
        ;                       otherwise.
        ;       Modifies:       AF
        ;
008A            ENT     GSPSIZ
        ;
        ;       Name:           GRPPRT
        ;       Function:       Prints a character on graphic screen
```

```
        ;       Entry:          Code to output in [Acc]
        ;       Returns:        None
        ;       Modifies:       None
        ;
008D            ENT     GRPPRT
```

```
        COMMENT %

                Following are used to access 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 a data to PSG register
        ;       Entry:          Register number in [Acc], data in [E]
        ;       Returns:        None
        ;       Modifies:       None
        ;
0093            ENT     WRTPSG
        ;
        ;       Name:           RDPSG
        ;       Function:       Reads a data from PSG register
        ;       Entry:          Register number in [Acc]
        ;       Returns:        Data in [Acc]
        ;       Modifies:       None
        ;
0096            ENT     RDPSG
        ;
        ;       Name:           STRTMS
        ;       Function:       Checks and starts the background task for PLAY
        ;       Entry:          None
        ;       Returns:        None
        ;       Modifies:       All
        ;
0099            ENT     STRTMS-
```

```
        COMMENT %

                Following are used to access console. I.e., keyboard, and CRT.

        %
        ;
        ;       Name:           CHSNS
        ;       Function:       Checks the status of keyboard buffer.
        ;       Entry:          None
        ;       Returns:        Z flag reset if there's any character in buffer
        ;       Modifies:       AF
        ;
009C            ENT     CHSNS
        ;
        ;       Name:           CHGET
        ;       Function:       Waits until any characters are typed, and retur
        ;                       with the character code.
        ;       Entry:          None
        ;       Returns:        Character code in [Acc]
        ;       Modifies:       AF
        ;
009F            ENT     CHGET
        ;
        ;       Name:           CHPUT
        ;       Function:       Outputs a character to console.
        ;       Entry:          Character code to be output in [Acc]
        ;       Returns:        None
        ;       Modifies:       None
        ;
00A2            ENT     CHPUT
        ;
        ;       Name:           LPTOUT
        ;       Function:       Outputs a character to LPT
        ;       Entry:          Character code to be output in [Acc]
        ;       Returns:        Carry flag set if aborted
        ;       Modifies:       F
        ;
00A5            ENT     LPTOUT
        ;
        ;       Name:           LPTSTT
        ;       Function:       Checks 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 convert code
        ;       Entry:          Character code in [Acc]
        ;       Returns:        Carry flag reset - graphic header byte
        ;                       Carry flag set, Z flag set - converted graphic
        ;                       Carry flag set, Z flag reset - non converted co
        ;       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 buffer
                ;       Entry:          None
                ;       Returns:        Address of buffer top-1 in [HL], carry flag
                ;                       set if STOP is typed.
                ;       Modifies:       All
                ;
00AE            ENT     PINLIN
                ;
                ;       Name:           INLIN
                ;       Function:       Same as PINLIN, except in case AUTFLG is set.
                ;       Entry:          None
                ;       Returns:        Address of buffer top-1 in [HL], carry flag
                ;                       set if STOP is pressed.
                ;       Modifies:       All
                ;
00B1            ENT     INLIN
                ;
                ;       Name:           QINLIN
                ;       Function:       Outputs a '?' mark and a space then fall into
                ;                       INLIN.
                ;       Entry:          None
                ;       Returns:        Address of buffer top-1 in [HL], carry flag
                ;                       set if STOP is pressed.
                ;       Modifies:       All
                ;
00B4            ENT     QINLIN
                ;
                ;       Name:           BREAKX
                ;       Function:       Checks the status of 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 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:       Beeps buzzer
        ;       Entry:          None
        ;       Returns:        None
        ;       Modifies:       All
        ;
00C0            ENT     BEEP
        ;
        ;       Name:           CLS
        ;       Function:       Clears screen
        ;       Entry:          None
        ;       Returns:        None
        ;       Modifies:       AF, BC, DE
        ;
00C3            ENT     CLS
        ;
        ;       Name:           POSIT
        ;       Function:       Locates cursor at 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.
        ;                       so, displays it, otherwise do nothing.
        ;       Entry:          FNKFLG
        ;       Returns:        None
        ;       Modifies:       All
        ;
00C9            ENT     FNKSB
        ;
        ;       Name:           ERAFNK
        ;       Function:       Erases function key display
        ;       Entry:          None
        ;       Returns:        None
        ;       Modifies:       All
        ;
00CC            ENT     ERAFNK
        ;
        ;       Name:           DSPFNK
        ;       Function:       Displays function key display
        ;       Entry:          None
        ;       Returns:        None
        ;       Modifies:       All
        ;
00CF            ENT     DSPFNK
        ;
        ;       Name:           TOTEXT
        ;       Function:       Forces screen to text mode
        ;       Entry:          None
        ;       Returns:        None
        ;       Modifies:       All
```

```
Sep 14 19:34 1983  bioent.val

;
00D2            ENT     TOTEXT
```

```
        COMMENT %

                Following are used to access game I/O

        %
        ;
        ;       Name:           GTSTCK
        ;       Function:       Returns the current status of joy stick
        ;       Entry:          Joy stick ID in [Acc]
        ;       Returns:        Direction in [Acc]
        ;       Modifies:       All
        ;
00D5            ENT     GTSTCK
        ;
        ;       Name:           GTTRIG
        ;       Function:       Returns the current status of trigger button
        ;       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 current status of touch PAD
        ;       Entry:          ID in [Acc]
        ;       Returns:        Value in [Acc]
        ;       Modifies:       All
        ;
00DB            ENT     GTPAD
        ;
        ;       Name:           GTPDL
        ;       Function:       Returns the value of paddle
        ;       Entry:          Paddle ID in [Acc]
        ;       Returns:        Value in [Acc]
        ;       Modifies:       All
        ;
00DE            ENT     GTPDL
```

```
        COMMENT %

                Following are used to access 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:       Sets cassette motor
        ;       Entry:          0 in [Acc] to stop, 1 to start, 255 to flip.
        ;       Returns:        None
        ;       Modifies:       AF
        ;
00F3            ENT     STMOTR
```

```
        COMMENT %

                Following are used to handle queues

        %
        ;
        ;       Name:           LFTQ
        ;       Function:       Returns how many bytes are left in queue
        ;       Entry:
        ;       Returns:
        ;       Modifies:
        ;
00F6            ENT     LFTQ
        ;
        ;       Name:           PUTQ
        ;       Function:       Puts a byte in queue
        ;       Entry:
        ;       Returns:
        ;       Modifies:
        ;
00F9            ENT     PUTQ
```

```
        COMMENT %

                Following are used by 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 X Y coordinates
        ;       Entry:
        ;       Returns:
        ;       Modifies:
        ;
010E            ENT     SCALXY
        ;
        ;       Name:           MAPXYC
        ;       Function:       Maps coordinate to 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 to 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 specified attribute
        ;       Entry:
        ;       Returns:
        ;       Modifies:
        ;
0120            ENT     SETC
        ;
```

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

```
        COMMENT %

                Following are the 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 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 what is currently output to primary slot
        ;                       register.
        ;       Entry:          None
        ;       Returns:        Result in [Acc]
        ;       Modifies:       A
        ;
0138            ENT     RSLREG
        ;
        ;       Name:           WSLREG
        ;       Function:       Writes to primary slot register.
        ;       Entry:          Value in [Acc]
        ;       Returns:        None
        ;       Modifies:       None
        ;
013B            ENT     WSLREG
        ;
        ;       Name:           RDVDP
        ;       Function:       Reads VDP's status register.
        ;       Entry:          None
        ;       Returns:        Data in [Acc]
        ;       Modifies:       A
        ;
013E            ENT     RDVDP
        ;
        ;       Name:           SNSMAT
        ;       Function:       Returns the status of specified row of a
        ;                       keyboard matrix.
        ;       Entry:          Row # in [Acc]
        ;       Returns:        Status in [Acc], corresponding bit is reset
        ;                       to 0 if being pressed.
        ;       Modifies:       AF
```

```
        ;
0141            ENT     SNSMAT
        ;
        ;       Name:           PHYDIO
        ;       Function:       Performs operation for mass storage devices
        ;                       (such as disks).
        ;       Entry:          ???
        ;       Returns:        ???
        ;       Modifies:       ???
        ;       Note:           In minimum configuration, only a hook is
        ;                       provided.
        ;
0144            ENT     PHYDIO
        ;
        ;       Name:           FORMAT
        ;       Function:       Performs mass storage devices initialization.
        ;       Entry:          ???
        ;       Returns:        ???
        ;       Modifies:       ???
        ;       Note:           In minimum configuration, only a hook is
        ;                       provided.
        ;
0147            ENT     FORMAT
        ;
        ;       Name:           ISFLIO
        ;       Function:       Checks if we're doing device I/O
        ;       Entry:          None
        ;       Returns:        Non zero if so, zero otherwise
        ;       Modifies:       AF
        ;
014A            ENT     ISFLIO
        ;
        ;       Name:           OUTDLP
        ;       Function:       Outputs to LPT
        ;       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 symbol are converted
        ;                          when 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:           Only used to play music as the background task.
        ;
0150            ENT     GETVCP
        ;
        ;       Name:           GETVC2
```

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

V. BIOS WORK AREA LIST

```
        ;
        ;       Following short routines are to perform inter-slot read/write
        ;       and call facility.
        ;
                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 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  ;Addr of queue tables used by QUEUTL
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(    CS120,  5*2)
                ;
                ;       Some parameters for cassette
                ;
                HEDLEN= 2000            ;Length of  header bits (mark) for short
                                        ;header
                ;
```

```
                ;       followings 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
                ;
                ;       followings are for 2400 baud
                ;
                        INTERN  LOW02,HIGH02,LOW12,HIGH12
                LOW02=  37              ;Width of low state for 0  1200Hz
                                        ;416.7usec
                HIGH02= 45              ;Width of high state for 0
                LOW12=  14              ;Width of low state for 1  2400Hz
                                        ;208.3usec
                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      ":"     ;FAKE END OF  PROGRAM  FOR  RESUME  NEX
        ;
        ;       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 LPT PRINT  HEAD  -initial
                                        ;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 KRUNCH BUFFER
F55D  RMB(    BUFMIN, 1)          ;A COMMA (PRELOAD OR ROM) USED BY INPUT
                                  ;STATEMENT SINCE THE DATA POINTER ALWAYS
                                  ;STARTS ON A COMMA OR TERMINATOR
F55E  RMB(    BUF,    BUFLEN+3)   ;TYPE IN  STORED  HERE DIRECT STATEMENTS
                                  ;EXECUTE OUT OF HERE.  REMEMBER "INPUT"
                                  ;SMASHES  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  "DIM"  OR  NOT
                                  ;DIMFLG AND  VALTYP  MUST BE CONSECUTIVE
                                  ;LOCATIONS
F663  RMB(    VALTYP, 1)          ;THE 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  RES'D
                                  ;WORDS TURNED ON IN THE 8K  WHEN  "DATA"
                                  ;BEING SCANNED  BY  CRUNCH  SO  UNQUOTED
                                  ;STRINGS WON'T BE CRUNCHED.
F665  RMB(    DONUM,  1)          ;FLAG  FOR  CRUNCH  =0  MEANS  NUMBERS
                                  ;ALLOWED, (FLOATING,INT,  DBL)  1  MEANS
                                  ;NUMBERS ALLOWED,  KRUNCH  BY  CALLING
                                  ;LINGET  -1  (377)  MEANS  NUMBERS
                                  ;DISALLOWED (SCANNING  VARIABLE  NAME)
F666  RMB(    CONTXT, 2)          ;SAVED TEXT  POINTER  USED  BY  CHRGET
                                  ;TO SAVE THE TEXT POINTER AFTER 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  USE  FOR  THE  STACK
                                  ;INITIALLY SET  UP  BY  INIT  ACCORDING
                                  ;TO MEMORY SIZE TO ALLOW  FOR  50  BYTE
                                  ;OF STRING  SPACE.  CHANGED  BY A CLEAR
                                  ;COMMAND WITH AN ARGUMENT.
F676  RMB(    TXTTAB, 2)          ;POINTER  TO  BEGINNING  OF  TEXT DOESN'T
                                  ;CHANGE AFTER  BEING  SETUP  BY  INIT.
F678  RMB(    TEMPPT, 2)          ;POINTER AT  FIRST  FREE TEMP DESCRIPTOR
                                  ;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 STRING ADDRESS IS STORE IN DSCTMI
F69B    RMB(    FRETOP, 2)              ;TOP OF STRING FREE SPACE
F69D    RMB(    TEMP3,  2)              ;USED TO STORE THE ADDRESS OF THE ENI
                                        ;OF STRING ARRAYS IN GARBAGE COLLECTION
                                        ;AND USED MOMENTARILY BY FRMEVL USEI
                                        ;IN EXTENDED BY FOUT AND USER DEFINEI
                                        ;FUNCTIONS ARRAY VARIABLE HANDLINC
                                        ;TEMPORARY
F69F    RMB(    TEMP8,  2)              ;7/3/79 Now used by garbage collectior
                                        ;not TEMP3 due to conflict
F6A1    RMB(    ENDFOR, 2)              ;SAVED TEXT POINTER AT END OF "FOR"
                                        ;STATEMENT
F6A3    RMB(    DATLIN, 2)              ;DATA LINE # -- REMEMBER FOR ERRORS
F6A5    RMB(    SUBFLG, 1)              ;FLAG WHETHER SUBSCRIPTED VARIABLI
                                        ;ALLOWED "FOR" AND USER-DEFINED FUNCTIO
                                        ;POINTER FETCHING TURN THIS ON BEFORI
                                        ;CALLING PTRGET SO ARRAYS WON'T BI
                                        ;DETECTED. STKINI AND PTRGET CLEAI
                                        ;IT.
F6A6    RMB(    USFLG,  0)
F6A6    RMB(    FLGINP, 1)              ;FLAGS WHETHER WE ARE DOING "INPUT'
                                        ;OR A 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 POINTEI
                                        ;HERE, CLEARC SAVES [H,L] HERE.
F6A9    RMB(    PTRFLG, 1)              ;=0 IF NO LINE NUMBERS CONVERTED
                                        ;POINTERS, NON ZERO IF POINTERS EXIST
F6AA    RMB(    AUTFLG, 1)              ;FLAG TO INICATE 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)              ;THE AUTO INCREMENT
F6AF    RMB(    SAVTXT, 2)              ;PLACE WHERE NEWSTT SAVES TEXT POINTER
                                        ;FOR "RESUME" STATEMENT
F6B1    RMB(    SAVSTK, 2)              ;NEWSTT SAVES STACK HERE BEFORE SC
                                        ;THAT ERROR RECOVERY CAN RESTORE THI
                                        ;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)              ;THE LINE TO GOTO WHEN AN ERROR OCCURS
F6BB    RMB(    ONEFLG, 1)              ;ONEFLG=1 IF WERE ARE EXECUTING AN
                                        ;ERROR TRAP ROUTINE, OTHERWISE 0
F6BC    RMB(    TEMP2,  2)              ;FORMULA EVALUATOR TEMP. MUST BI
                                        ;PRESERVED BY OPERATORS USED IN EXTENDEI
                                        ;BY FOUT AND USER-DEFINED FUNCTION
                                        ;ARRAY VARIABLE HANDLER TEMPORARY
F6BE    RMB(    OLDLIN, 2)              ;OLD LINE NUMBER (SETUP 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 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 CHANGED BY
                                        ;"DEFSTR" "DEFINT" "DEFSNG" "DEFDBL"
                                        ;AND USED BY PTRGET WHEN ! # % OR
                                        ;$ DON'T FOLLOW A VARAIBLE 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)                ;THE NUMBER OF BYTES IN THE ACTIVE TABLE
F6E8  RMB(    PARM1,  PRMSIZ)           ;THE ACTIVE PARAMETER DEFINITION TABLE
F74C  RMB(    PRMPRV, 2)                ;INIIALLY 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                      ;TEMP COMPLEMENT OF SIGN
```

```
F7F6   RMB(     DAC,     16)
               INTERN   FACLO
       FACLO=  DAC+2
       ;
       ;       HOLDING REGS FOR DECIMAL MULTIPLY
       ;
F806   RMB(     HOLD8,   48)              ;80*X
F836   RMB(     HOLD2,   8)               ;2*X
F83E   RMB(     HOLD,    8)               ;1*X
       ;
       ;       ARGUMENT ACCUMULATOR
       ;
       ZX==     ZX+1                      ;TEMP 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 #
F860    RMB(    FILTAB, 2)              ;Points to adr 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 if should run after load
F866    RMB(    FILNAM,11)              ;Holds filename for LOADF, from NAME
F871    RMB(    FILNM2,11)              ;Holds other filename for NAME
F87C    RMB(    NLONLY, 1)              ;Non-zero if loading program
        ;
        ;       Set up by NULOPN and BSAVE, used by BSAVE and CREATE.
        ;
F87D    RMB(    SAVEND, 2)              ;End of binary or mem 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 no. of pts in circle
F93B    RMB(    CPCNT8,2)               ;No. of pts in circle
F93D    RMB(    CRCSUM,2)               ;Circle sum
F93F    RMB(    CSTCNT,2)               ;start count
F941    RMB(    CSCLXY,1)               ;scaling x 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 loc
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 = #strings parsed
                                        ;D7=0 if 1st 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)               ;# 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        ;# 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(    CSAVE,1)                ;Code save area for cursor
FBCD    RMB(    FNKSWI,1)               ;Indicates  which  function  key  is
                                        ;displayed
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-text,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)              ;Whether we're 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..3)
```

```
        COMMENT %

                Following are definition of hooks and their functions

                name            - name of hook
                where           - where in what module it is used
                purpose         - what purpose it is used for

        %

        GSX==   ZX
FD9A    RMB(    HOKJMP,0)
        ;
        ;       name:           H.KEYI
        ;       where:          MSXIO, at the beginning of interrupt handler
        ;       purpose:        to do additional interrupt handling such as
        ;                       RS232C
        ;
FD9A    RMB(    H.KEYI,5)
        ;
        ;       name:           H.TIMI
        ;       where:          MSXIO, in timer interrupt handler
        ;       purpose:        to allow other interrupt handling invoked by
        ;                       timer
        ;
FD9F    RMB(    H.TIMI,5)
        ;
        ;       name:           H.CHPU
        ;       where:          MSXIO, at the beginning of CHPUT (CHaracter
        ;                       outPUT) routine
        ;       purpose:        to allow other console output devices to be used
        ;
FDA4    RMB(    H.CHPU,5)
        ;
        ;       name:           H.DSPC
        ;       where:          MSXIO, at the beginning of DSPCSR (DiSPlay
        ;                       CurSoR) routine
        ;       purpose:        to allow other console output devices to be used
        ;
FDA9    RMB(    H.DSPC,5)
        ;
        ;       name:           H.ERAC
        ;       where:          MSXIO, at the beginning of ERACSR (ERAse CurSoR)
        ;                       routine
        ;       purpose:        to allow other console output devices to be used
        ;
FDAE    RMB(    H.ERAC,5)
        ;
        ;       name:           H.DSPF
        ;       where:          MSXIO, at the beginning of DSPFNK (DiSPlay
        ;                       FuNction Key) routine
        ;       purpose:        to allow other console output devices to be used
        ;
FDB3    RMB(    H.DSPF,5)
        ;
```

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

```
        ;
        ;       name:           H.QINL
        ;       where:          MSXINL, at the beginning of QINLIN (Question
        ;                       mark and INput LINe) routine
        ;       purpose:        to allow other console input devices or other
        ;                       input design to be used
        ;
FDE0    RMB(    H.QINL,5)
        ;
        ;       name:           H.INLI
        ;       where:          MSXINL, at the beginning of INLIN (INput LINe)
        ;                       routine
        ;       purpose:        to allow other console input devices or other
        ;                       input design to be used
        ;
FDE5    RMB(    H.INLI,5)
        ;
        ;       name:           H.ONGO
        ;       where:          MSXSTS, at the beginning of ONGOTP (ON GOTo
        ;                       Procedure) routine
        ;       purpose:        to allow other interrupting devices to be used
        ;
FDEA    RMB(    H.ONGO,5)
        ;
        ;       name:           H.DSKO
        ;       where:          MSXSTS, at the beginning of DSKO$ (DiSK Output)
        ;                       routine
        ;       purpose:        to install disk driver
        ;
FDEF    RMB(    H.DSKO,5)
        ;
        ;       name:           H.SETS
        ;       where:          MSXSTS, at the beginning of SETS (SET
        ;                       attributeS) routine
        ;       purpose:        to install disk driver
        ;
FDF4    RMB(    H.SETS,5)
        ;
        ;       name:           H.NAME
        ;       where:          MSXSTS, at the beginning of NAME (reNAME) routine
        ;       purpose:        to install disk driver
        ;
FDF9    RMB(    H.NAME,5)
        ;
        ;       name:           H.KILL
        ;       where:          MSXSTS, at the beginning of KILL (KILL file)
        ;                       routine
        ;       purpose:        to install disk driver
        ;
FDFE    RMB(    H.KILL,5)
        ;
        ;       name:           H.IPL
        ;       where:          MSXSTS, at the beginning of IPL (Initial Program
        ;                       Load) routine
        ;       purpose:        to install disk driver
        ;
```

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

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

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

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

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

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

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

```
        ;
FF2F    RMB(    H.NTFN,5)
        ;
        ;       name:
        ;       where:          BINTRP
        ;       purpose:
        ;
FF34    RMB(    H.NOTR,5)
        ;
        ;       name:
        ;       where:          BINTRP
        ;       purpose:
        ;
FF39    RMB(    H.SNGF,5)
        ;
        ;       name:
        ;       where:          BINTRP
        ;       purpose:
        ;
FF3E    RMB(    H.NEWS,5)
        ;
        ;       name:
        ;       where:          BINTRP
        ;       purpose:
        ;
FF43    RMB(    H.GONE,5)
        ;
        ;       name:
        ;       where:          BINTRP
        ;       purpose:
        ;
FF48    RMB(    H.CHRG,5)
        ;
        ;       name:
        ;       where:          BINTRP
        ;       purpose:
        ;
FF4D    RMB(    H.RETU,5)
        ;
        ;       name:
        ;       where:          BINTRP
        ;       purpose:
        ;
FF52    RMB(    H.PRTF,5)
        ;
        ;       name:
        ;       where:          BINTRP
        ;       purpose:
        ;
FF57    RMB(    H.COMP,5)
        ;
        ;       name:
        ;       where:          BINTRP
        ;       purpose:
        ;
FF5C    RMB(    H.FINP,5)
```

```
        ;
        ;       name:
        ;       where:          BINTRP
        ;       purpose:
        ;
FF61    RMB(    H.TRMN,5)
        ;
        ;       name:
        ;       where:          BINTRP
        ;       purpose:
        ;
FF66    RMB(    H.FRME,5)
        ;
        ;       name:
        ;       where:          BINTRP
        ;       purpose:
        ;
FF6B    RMB(    H.NTPL,5)
        ;
        ;       name:
        ;       where:          BINTRP
        ;       purpose:
        ;
FF70    RMB(    H.EVAL,5)
        ;
        ;       name:
        ;       where:          BINTRP
        ;       purpose:
        ;
FF75    RMB(    H.OKNO,5)
        ;
        ;       name:
        ;       where:          BINTRP
        ;       purpose:
        ;
FF7A    RMB(    H.FING,5)
        ;
        ;       name:           H.ISMI
        ;       where:          BINTRP, at the ISMID$ (IS MID$) routine
        ;       purpose:
        ;
FF7F    RMB(    H.ISMI,5)
        ;
        ;       name:           H.WIDT
        ;       where:          BINTRP, at the WIDTHS (WIDTH) routine
        ;       purpose:
        ;
FF84    RMB(    H.WIDT,5)
        ;
        ;       name:           H.LIST
        ;       where:          BINTRP, at the LIST routine
        ;       purpose:
        ;
FF89    RMB(    H.LIST,5)
        ;
        ;       name:           H.BUFL
```

```
        ;       where:          BINTRP, at the BUFLIN (BUFfer LINe) routine
        ;       purpose:
        ;
FF8E    RMB(    H.BUFL,5)
        ;
        ;       name:           H.FRQI
        ;       where:          BINTRP, at the FRQINT routine
        ;       purpose:
        ;
FF93    RMB(    H.FRQI,5)
        ;
        ;       name:
        ;       where:          BINTRP
        ;       purpose:
        ;
FF98    RMB(    H.SCNE,5)
        ;
        ;       name:           H.FRET
        ;       where:          BISTRS, at the FRETMP (FREe up TeMPorari
        ;                       routine
        ;       purpose:
        ;
FF9D    RMB(    H.FRET,5)
        ;
        ;       name:           H.PTRG
        ;       where:          BIPTRG, at the PTRGET (PoinTeR GET) routine
        ;       purpose:        to use other variable names than default
        ;
FFA2    RMB(    H.PTRG,5)
        ;
        ;       name:           H.PHYD
        ;       where:          MSXIO, at the PHYDIO (PHYsical Disk I/O) routine
        ;       purpose:        to install disk driver
        ;
FFA7    RMB(    H.PHYD,5)
        ;
        ;       name:           H.FORM
        ;       where:          MSXIO, at the FORMAT (disk FORMATter) routine
        ;       purpose:        to install disk driver
        ;
FFAC    RMB(    H.FORM,5)
        ;
        ;       name:           H.ERRO
        ;       where:          BINTRP, at the ERROR routine
        ;       purpose:        to trap errors from application programs
        ;
FFB1    RMB(    H.ERRO,5)
        ;
        ;       name:           H.LPTO
        ;       where:          MSXIO, at the LPTOUT (Line PrinTer OUTput)
        ;                       routine
        ;       purpose:        to use other printer than default
        ;
FFB6    RMB(    H.LPTO,5)
        ;
        ;       name:           H.LPTS
```

```
        ;       where:          MSXIO, at the LPTSTT (Line PrinTer STaTu
        ;                       routine
        ;       purpose:        to use other printer than default
        ;
FFBB    RMB(    H.LPTS,5)
        ;
        ;       name:           H.SCRE
        ;       where:          MSXSTS, at the entry to SCREEN statement.
        ;       purpose:        To expand SCREEN statement.
        ;
FFC0    RMB(    H.SCRE,5)
        ;
        ;       name:           H.PLAY
        ;       where:          MSXSTS, at the entry to PLAY statement.
        ;       purpose:        To expand PLAY statement.
        ;
FFC5    RMB(    H.PLAY,5)
        ;
FFCA    RMB(    ENDWRK,0)       ;end of work area
```

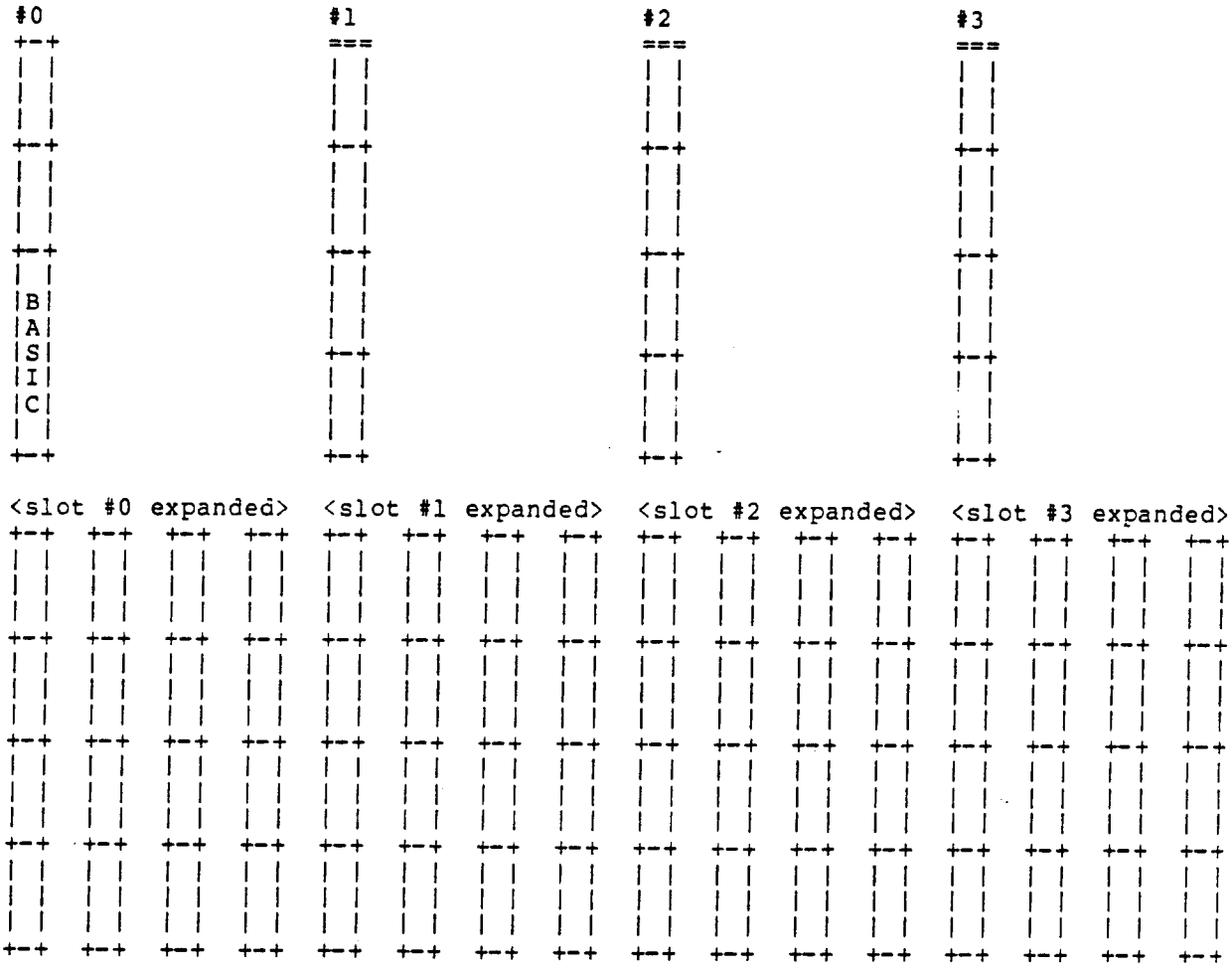
VI. SLOT MANAGEMENT MECHANISM

MSX BASIC interpreter's slot management mechanism

August 20th. 1983.

MSX BASIC interpreter's slot management mechanism

[Memory structure of MSX]



Total: 1024 Kbytes (16*64 Kbytes)

Terminology:

primary slot	- Slot which is enabled by slot select register within 8255 PPI.
secondary slot	- Slot which is enabled by expansion slot register placed at 0FFFFH.
page	- Block of memory (maximum 16K) in each slot. A slot is divided into 4 pages. (0000H to 3FFFH, 4000H to 7FFFH, 8000H to 0BFFFH, 0C000H to 0FFFFH)

1. Minimum configuration

MSX BASIC interpreter's slot management mechanism

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)

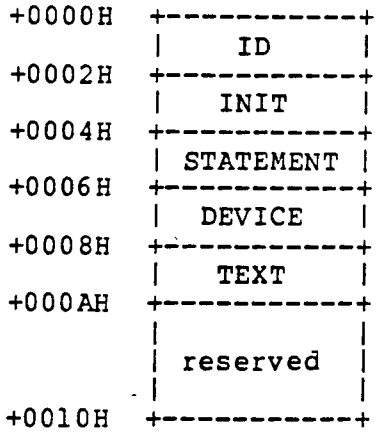
2. RAM search procedure

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

3. 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 ID and others are as follows.

Offset from top



- ID is a 2 byte code used to distinguish ROM cartridge from empty pages. 'AB' (41H,42H) is used for this purpose.

- INIT holds an address of the initialization procedure specific to this cartridge. 0 when no such procedure is necessary. Programs that need to work co-operatively with BASIC interpreter should return control to it by a Z80's 'RET' instruction (all registers except [SP] can be destroyed). Other programs (such as game programs) need not to do so, however.

- STATEMENT holds an address of the expanded statement handler if such is contained in this cartridge. 0 when no such handler is inside. When BASIC encounters a 'CALL' statement, it calls

this address with the statement name in the system area. Following are the notes to be remembered. (In the notes below, [HL] register pair is called a 'text pointer')

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

2) Syntax for expanded statement is,

CALL <statement_name> [(<arg> [,<arg>]..)]

Key word "CALL" can be substituted with an under score character, "_".

3) Statement name is stored in the system area terminated by 0. The buffer for statement name is of fixed length (16 bytes) so statement name cannot be longer than 15 characters.

4) If the handler for that statement is not inside the cartridge, return with carry flag set. Text pointer must be returned unchanged.

5) If the handler for that statement is inside the cartridge, the cartridge should do the function, update text pointer to the end of the statement (usually, pointing to 0 which indicates the end of line, or ':' which indicates the end of statement), and return with carry flag reset (registers except [SP] can be destroyed). At the entry to the expanded statement handler, text pointer is set up to point to the first non-blank character after the statement name.

- DEVICE holds an address of the expanded device handler if such is contained in this cartridge. 0 when no such handler is inside. BASIC calls this address with the device name in the system area. Following are notes to be remembered.

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

2) Device name is stored in the system area terminated by 0. The buffer for statement name is of fixed length (16 bytes) so device name cannot be longer than 15 characters.

3) A cartridge (16K) can have up to 4 logical devices.

4) When BASIC encounters a device name which is not known to itself, it calls DEVICE entry with 0FFH in [Acc]. If the handler for that device is not inside the cartridge, carry should be returned set. If it's inside, device ID (from 0 to 3) should be returned in [Acc], and carry reset. All registers can 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'.

(Further descriptions about I/O operations will be prepared later.)

- TEXT holds the beginning address of BASIC text (tokenized) contained in the cartridge. 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. Following are the notes to be remembered.

    1) When there are more than one such slots, only the leftmost one (in the figure above) is enabled and executed.

    2) The cartridge must be placed at 8000H..0BFFFH, thus the maximum length of BASIC text cannot exceed 16K bytes.

    3) Even if there is a RAM block equipped at 8000H..0BFFFH, it can never 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, GOSUB, etc) had better be translated to pointers in advance because they are never converted to pointers when executed. They CAN be line numbers however, but the execution would become slower then.

Note: INIT, STATEMENT, DEVICE and TEXT are placed low order byte first.

4. How slot informations are 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 what value is currently output to 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
        |||+------Statement expander inside
        ||+-------Device expander inside
        |+--------BASIC text inside
        +---------BASIC text inside
```

SLTWRK - Holds working storage for each page.

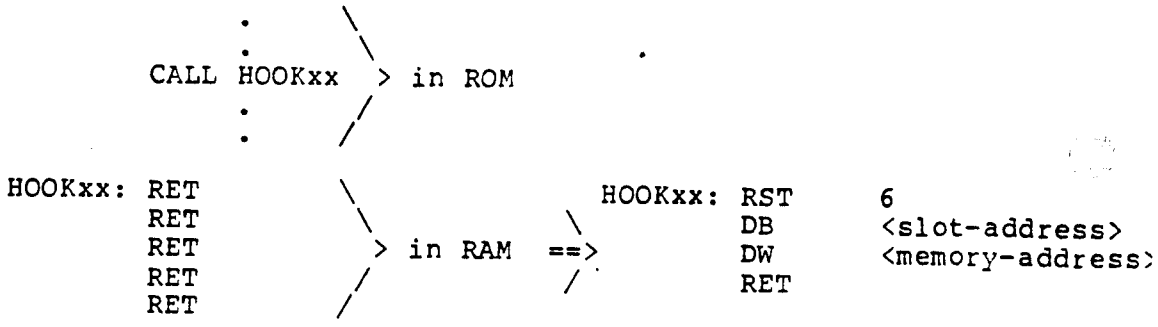
```
SLTWRK: DS      128
```

Each word in the SLTWRK table can be exclusively used by each page. The usage of this work area is completely up to the page.

5. Usage of hooks

Hooks are one of those means by which MSX-BASIC can be expanded. Some procedures (such as 'console input', 'console output') has a Z80's 'CALL' instruction which is directed to common RAM area. Those areas consist of 5 byte storage per hook, and initialized with 5 Z80's 'RET' instructions at cold start. Expansions can be done by re-directing this entry to somewhere else.

Example:



RST 6 performs an inter-slot call to a different slot. See BIOENT.MAC for further details of inter-slot call facility.

To connect the hook to the desired routine, the routine has to know where (i.e., in what slot) he is. This is very important because there's no telling in what slot the routine is placed. This is done by a 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               ;Finaly form slot address
                RET
```

< CAUTION >

A machine language program in cartridges must be able to run in any slots (including secondary slots) There's no telling in what slot the cartridge is to run.

177

6. Usage of USR function

There are 10 USR functions, USR0 through USR9. USR0 can be abbreviated as USR. Defining address to which a USR function jumps is done as follows.

```
DEFUSR0=&HE000  (This can be DEFUSR=&HE000)
DEFUSR3=&HE023
```

USR functions can be invoked as follows.

```
A=USR0(12)      (This can also be A=USR(12))
PRINT USR("ABCD")+" This is a test"
```

Argument to a USR function is passed to machine language programs by the following manner.

Integer

0F663H has 2. Real value is in 0F7F8H and 0F7F9H, lower byte first.

String

0F663H has 3. 0F7F8H and 0F7F9H have the address of the string descriptor. A string descriptor consists of 3 bytes, first byte has the length of string, second and third the address of the string.

Single precision

0F663H has 4. Real value is in 0F7F6H through 0F7F9H.

Double precision

0F663H has 8. Real value is in 0F7F6H through 0F7FDH.

Value from a USR function can be returned to BASIC by the following manner.

Integer

0F663H should be set to 2. Real value should be in 0F7F8H and 0F7F9H, lower byte first.

String

0F663H should be set to 3. 0F7F8H and 0F7F9H should have the address of the string descriptor. A string descriptor consists of 3 bytes, first byte should be set to the length of string, second and third the address of the string.

Single precision

0F663H should be set to 4. Real value should be in 0F7F6H through 0F7F9H.

Double precision

0F663H should be set to 8. Real value should be in 0F7F6H through 0F7FDH.

7. Appendix

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 it needs to run with BASIC interpreter and other programs in other cartridges, RAM usage is restricted.

There are three ways to allocate RAM to be used exclusively by each cartridge.

1) Put a RAM on the cartridge. (the easiest and the best)

2) If the work area is less than 3 bytes, use the SLTWRK.

3) If the work area is greater than 2 bytes, make the SLTWRK point to the system variable BOTTOM (0FC48H), then update it by the amount of memory required. BOTTOM is set up by the initialization code to point to the bottom of equipped RAM.

   Ex. if the program is at 4000H..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
                JRNC    NOROOM          ;Yes, cannot allocate this much
                SHLD    BOTTOM
                RET
;
;               BOTTOM became  greater  than  0EFFFH,  there  is
;               no RAM left to be allocated.
;
NOROOM:                                 ;Print messages      or
                                        ;something like     that
```