

MSX2 TECHNICAL HANDBOOK

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

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

Changes from the original:

- In Figure 5.2, unused bits are marked as "x", and inverted signals are marked with "*", for easiest readability.
- Figure 5.17B was added.
- In List 5.4, the last line before the work area, "JR START", has been corrected to "JR SCAN".
- In Figure 5.18, the addresses for GETPNT y PUTPNT were swapped. They have been corrected.
- In description of BIOS routines PINLIN and INLIN, "BUF" address has been corrected from F55DH to F55EH.
- In Figure 5.22 (B), "Arabaic mode display" has been changed to "Arabic or kana mode display".
- In description of BIOS routine GTTRIG, the input needed for reading B buttons has been added in the "Input" field.
- In Table 5.5, in the Note 4, "the trigger button of the mouse or the trigger button" has been changed to "the trigger button of the mouse or the trigger button of the track ball".
- In Figure 5.29, "1200 or 2400 hours" indication has been corrected to "12 or 24 hours".
- In Figure 5.32, "Register 3 #11" indication has been corrected to "Register #11".
- In Figure 5.33, "Adjust Y (8 to +7)" has been corrected to "Adjust Y (-8 to +7)".
- In description of BIOS routine WRTCLK, the input needed in the A register has been added in the "Input" field.

CHAPTER 5 - ACCESS TO PERIPHERALS THROUGH BIOS (Parts 1 to 6)

The basic philosophy of MSX is to have a standard interface, independent of machines or versions, to access peripherals through BIOS. Thus, the user should get to know about using BIOS first. In chapter 5, accessing peripherals using BIOS and the structure used for each peripheral are described.

1. PSG AND SOUND OUTPUT

MSX has the following three kinds of sound output functions, but function (3) is not installed in the standard MSX, so it is not described in this manual. This section describes functions (1) and (2).

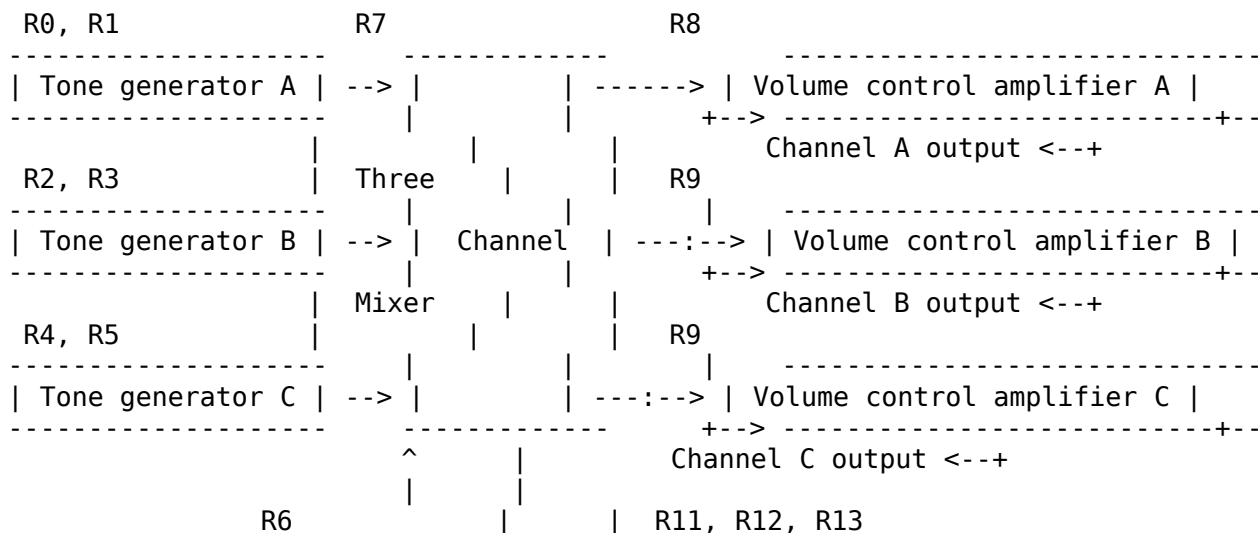
- (1) PSG sound output (3 channels, 8 octaves)
- (2) Sound output by 1 bit I/O port
- (3) Sound output by MSX-AUDIO (FM sound generator) not described in this manual

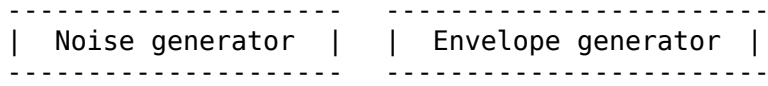
1.1. PSG functions

An AY-3-8910 compatible LSI is used for the MSX music play function and for BEEP tone generation. This LSI is referred to as the PSG (Programmable Sound Generator), and can generate complex music and various tones. It has the following features:

- * There are three tone generators, each of which can independently specify 4096 scales (equivalent to 8 octaves) and 16 volume levels.
- * It can generate piano and organ tones by using envelope patterns. Note that, since there is only one envelope generator, the tone of only one channel can be modified fundamentally.
- * With the noise generator inside, tones such as the wind or waves can easily be generated. Note that since there is only one noise generator, only one channel can generate the noise.
- * Any necessary frequency, such as the tone or the envelope, is obtained by dividing the input clock (in MSX, it is defined that $f_c = 1.7897725$ MHz). So there is no unsteady pitch or rhythm.

Figure 5.1 PSG block diagram





The PSG has two additional I/O (input/output) ports used for other than tone generating functions, which are omitted in the block diagram above. MSX uses them as general-purpose I/O ports to connect to I/O devices such as joystick, a touch pad, a paddle, or a mouse. These general-purpose I/O ports are described in section 5.

* PSG registers

Since the PSG generates tones, the CPU simply notifies PSG when the tone is to be changed. This is done by writing values in 16 8-bit registers inside the PSG as shown in Figure 5.2.

Roles and uses of these registers are described below.

* Setting the tone frequency (R0 to R5)

Each tone frequency of channel A, B, and C is set by R0 to R5. The input clock frequency ($f_c = 1.7897725$ MHz) is divided by 16 and the result is the standard frequency. Each channel divides the standard frequency by the 12-bit data assigned for each, and the objective pitch is obtained. The following relation exists between 12-bit data (TP) and the tone frequency to be generated (f_t).

$$\begin{aligned}
 f_t &= f_c / (16 * TP) \\
 &= 0.11186078125 / TP \text{ [MHz]} \\
 &= 111860.78125 / TP \text{ [Hz]}
 \end{aligned}$$

A 12-bit data TP is specified for each channel by 4 high order bit coarse tune CT and 8 low order bit fine tune value FT, as shown in Figure 5.3. Table 5.1 shows the register settings to make the scales.

Figure 5.2 PSG register structure

Register	Bit	B7	B6	B5	B4	B3	B2	B1	B0
R0	Channel A note	8 low order bits							
R1	Dividing rate	x	x	x	x	4 high order bits			
R2	Channel B note	8 low order bits							
R3	Dividing rate	x	x	x	x	4 high order bits			
R4	Channel C note	8 low order bits							
R5	Dividing rate	x	x	x	x	4 high order bits			
R6	Noise div. rate	x	x	x					

R7	Enable*	IN*/OUT	NOISE*			TONE*		
		I0B I0A	C	B	A	C	B	A
R8	Chan. A volume	x	x	x	M			
R9	Chan. B volume	x	x	x	M			
R10	Chan. C volume	x	x	x	M			
R11	Envelope Cycle	8 low order bits						
R12		8 high order bits						
R13	Env. wave shape	x	x	x	x			
R14	I/O port A							
R15	I/O port B							

NOTE: x = unused bit
 * = inverted signal

Figure 5.3 Setting the pitch

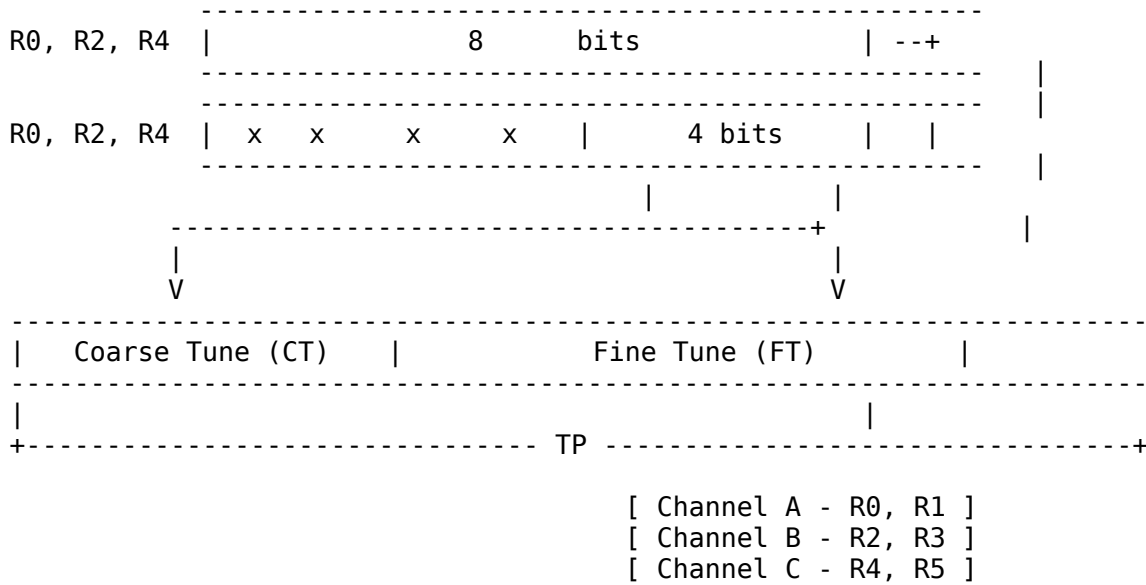


Table 5.1 Setting the tone frequency (scale data)

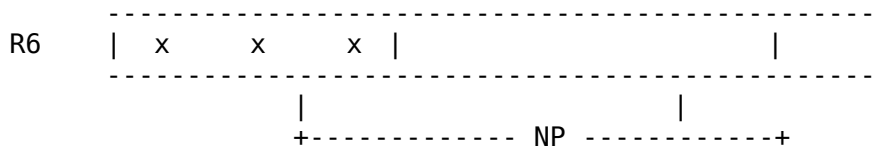
Octave	1	2	3	4	5	6	7	8
Note								
C	D5D	6AF	357	1AC	D6	6B	35	1B

C#	C9C	64E	327	194	CA	65	32	19
D	BE7	5F4	2FA	17D	BE	5F	30	18
D#	B3C	59E	2CF	168	84	5A	2D	16
E	A9B	54E	2A7	153	AA	55	2A	15
F	A02	501	281	140	A0	50	28	14
F#	973	4BA	25D	12E	97	4C	26	13
G	8EB	476	23B	11D	8F	47	24	12
G#	88B	436	21B	10D	87	43	22	11
A	7F2	3F9	1FD	FE	7F	40	20	10
A#	780	3C0	1E0	F0	78	3C	1E	F
B	714	38A	1C5	E3	71	39	1C	E

* Setting the noise frequency (R6)

The noise generator is used for synthesizing explosion sounds or wave sounds. The PSG can send the noise output by the noise generator to channels A to C. Since there is only one noise generator, the same noise is sent to all channels. By changing the average frequency, various noise effects can be obtained and this is done by R6 register settings. The 5 low order bit data (NP) of this register is divided into the standard frequency (fc/16) and this determines the average frequency of the noise (fn).

Figure 5.4 Setting the noise frequency



The following relation exists between NP and fn.

$$\begin{aligned}
 fn &= fc / (16 * NP) \\
 &= 0.11186078125 / NP \text{ [MHz]} \\
 &= 111860.78125 / NP \text{ [Hz]}
 \end{aligned}$$

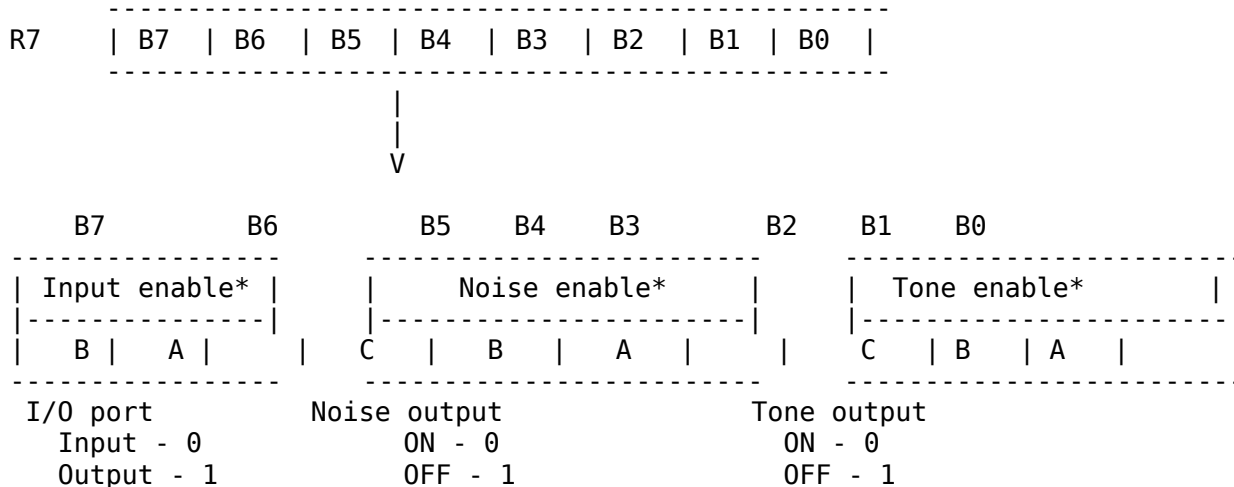
Since the value of NP is from 1 to 31, the average frequency of the noise can be set from 3.6kHz to 111.9kHz.

* Mixing the sound (R7)

R7 is used to select the output of the tone and noise generator, or a mixture of both. As shown in Figure 5.5, the 3 low order bits (B0 to B2) of R7 control the tone output and the next 3 bits (B3 to B5) control the noise

output. In both cases, when the corresponding bit is 0, the output is ON and, when 1, it is OFF.

Figure 5.5 Output selection for each channel

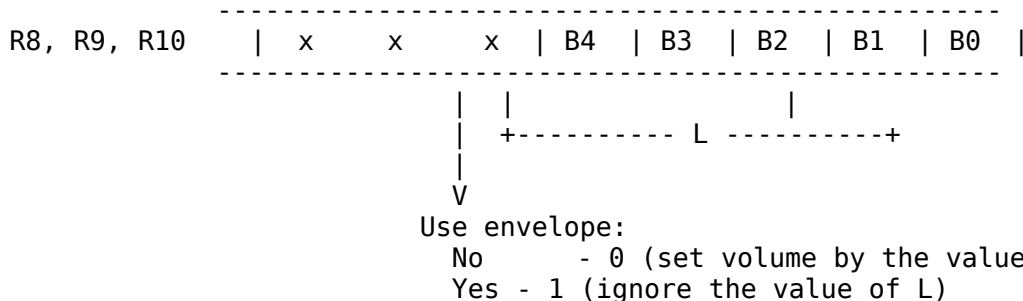


The 2 high order bits of R7 do not affect sound output. These are used to determine the direction of the data of two I/O ports which PSG has. When the corresponding bit is 0, the input mode is selected and, when 1, the output mode is selected. In MSX, port A is used for the input and port B for the output, so it should always be set so that bit 6 = "0" and bit 7 = "1".

* Setting the volume (R8 to R10)

R8 to R10 are used to specify the volume of each channel. Two ways can be selected by these registers: specifying the fixed volume by 4-bit data (0 to 15) and generating sound effects such as vibrato or fade-out by using the envelope.

Figure 5.6 Setting the volume

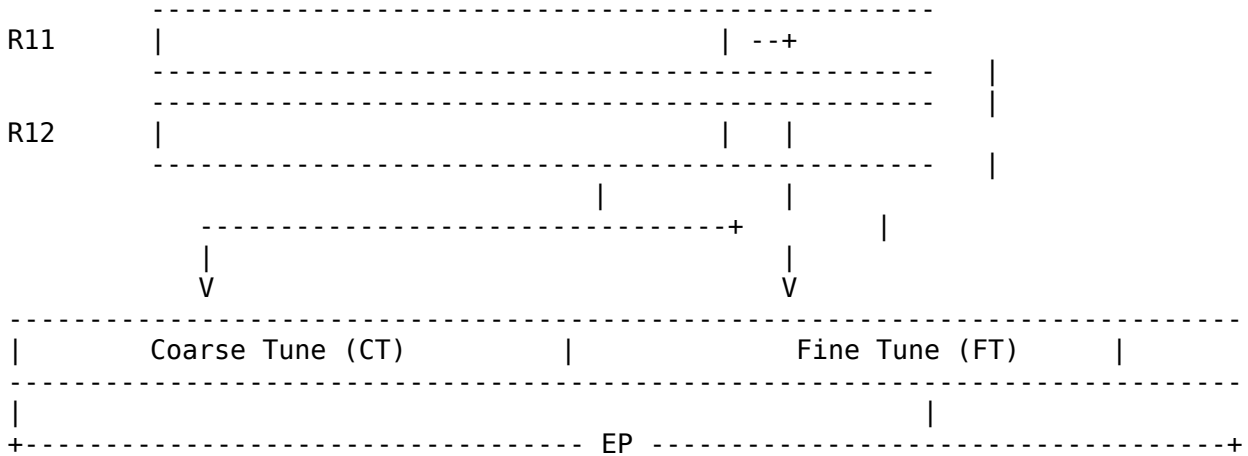


When bit 4 of these registers is "0", the envelope is not used and the 4 low order bit value L (0 to 15) of the registers specify the volume. When bit 4 is "1", the volume depends on the envelope signals and the value L is ignored.

* Setting the envelope cycle (R11, R12)

R11 and R12 specify the envelope cycle in 16-bit data. The 8 high order bits are set in R12 and the 8 low order bits are set in R11.

Figure 5.7 Setting the envelope cycle



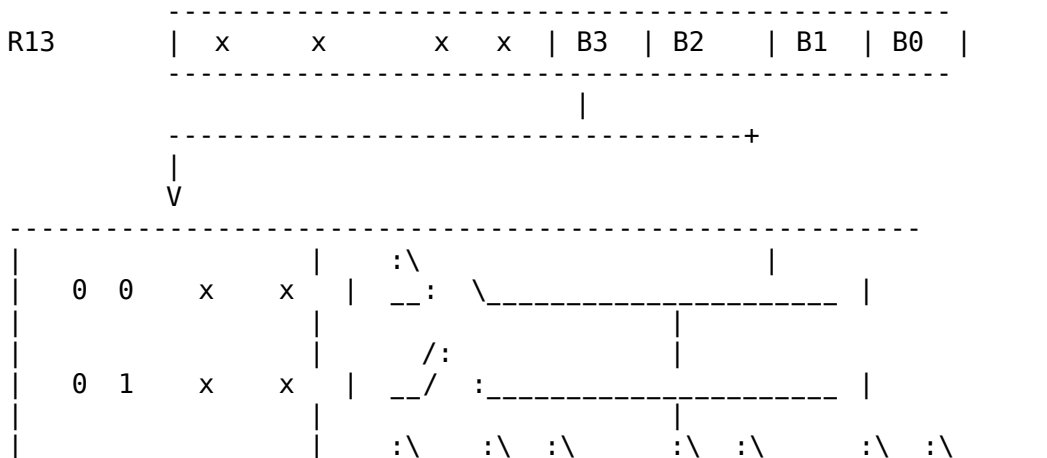
The following relation exists between the envelope cycle T and 16-bit data EP.

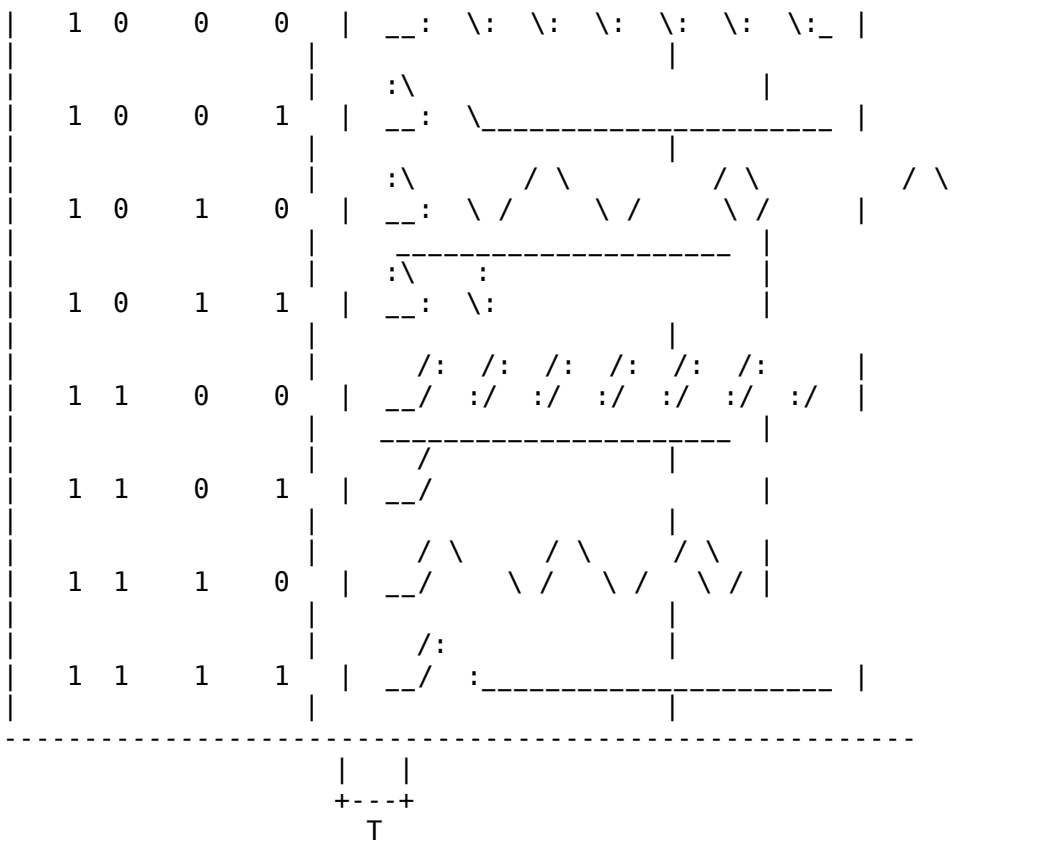
$$\begin{aligned}
 T &= (256 * EP) / fc \\
 &= (256 * EP) / 1.787725 \text{ [MHz]} \\
 &= 143.03493 * EP \text{ [micro second]}
 \end{aligned}$$

* Setting the envelope pattern (R13)

R13 sets the envelope pattern by the 4 low order bit data as shown in Figure 5.8. The intervals of T specified in the figure correspond to the envelope cycle specified by R11 and R12.

Figure 5.8 Setting the wave forms of the envelopes





* I/O port (R14, R15)

R14 and R15 are the ports to send and receive 8-bit data in parallel. MSX uses these as the general-purpose I/O interface. For more information, see section 5.

1.2 Access to the PSG

For access the PSG from assembly language programs, several BIOS routines described below are available.

* GICINI (0090H/MAIN) PSG initialization

Input: ---
 Output: ---
 Function: initializes PSG registers and does the initial settings of the work area in which PLAY statement of BASIC is executed. Each register of PSG is set to the value as shown in Figure 5.9.

Figure 5.9 Initial values of PSG registers

	Bit	7	6	5	4	3	2	1	0
Register									

R0	Channel A	0	1	0	1	0	1	0	1
R1	frequency	0	0	0	0	0	0	0	0
R2	Channel B	0	0	0	0	0	0	0	0
R3	frequency	0	0	0	0	0	0	0	0
R4	Channel C	0	0	0	0	0	0	0	0
R5	frequency	0	0	0	0	0	0	0	0
R6	Noise frequency	0	0	0	0	0	0	0	0
R7	Channel setting	1	0	1	1	1	0	0	0
R8	Chan. A volume	0	0	0	0	0	0	0	0
R9	Chan. B volume	0	0	0	0	0	0	0	0
R10	Chan. C volume	0	0	0	0	0	0	0	0
R11	Envelope Cycle	0	0	0	0	1	0	1	1
R12		0	0	0	0	0	0	0	0
R13	Env. pattern	0	0	0	0	0	0	0	0
R14	I/O port A								
R15	I/O port B								

* WRTPSG (0093H/MAIN) writing data in PSG registers

Input: A <-- PSG register number
 E <-- data to be written

Output: ---

Function: writes the contents of the E register in the PSG register
 whose number is specified by the A register.

* RDPSG (0096H/MAIN) reading PSG register data

Input: A <-- PSG register number

Output: A <-- contents of the specified register

Function: reads the contents of PSG register whose number is specified
 by the A register and stores the value in the A register.

* STRTMS (0099H/MAIN) starting the music

Input: (Queue) <-- MML which is translated into the intermediate
 language

Output: ---

Function: examines whether the music is played as the background task,

and plays the music which is set in the queue, if the music has not yet been played.

List 5.1 Single tone generation

```

=====
;*****
;
; List 5.1 440 Hz tone
;
;*****
;
WRTPSG EQU 0093H

ORG 0B000H

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

LD A,7 ;Select Channel
LD E,00111110B ;Channel A Tone := 0n
CALL WRTPSG

LD A,8 ;Set Volume
LD E,10
CALL WRTPSG

LD A,0 ;Set Fine Tune Channel A
LD E,0FEH ;Data 0FEH
CALL WRTPSG

LD A,1 ;Set Coarse Tune Channel A
LD E,0 ;Data 0H
CALL WRTPSG

RET

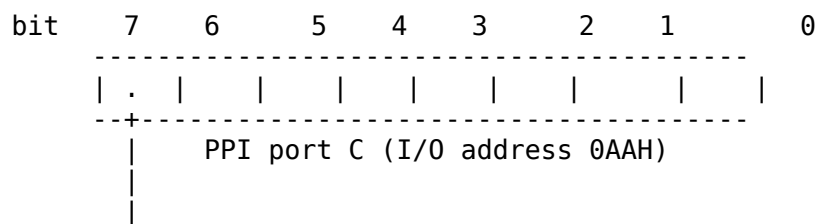
END
=====

```

1.3 Tone Generation by 1-bit Sound Port

MSX has another sound generator in addition to the PSG. This is a simple one that generates sound by turning ON/OFF the 1-bit I/O port output repeatedly using software.

Figure 5.10 1-bit sound port




```

JR    NC,LBL01

XOR   A           ;stop cassette motor
CALL  STMOTR
RET

END

```

2. CASSETTE INTERFACE

Cassette tape recorders are the least expensive external storage devices available for the MSX. Knowledge of the cassette interface is required to treat information in cassette tapes within assembly language programs. This section offers the necessary information.

2.1 Baud Rate

The following two baud rates can be used by the MSX cassette interface (see Table 5.2). When BASIC is invoked, 1200bps is set by default.

Table 5.2 MSX baud rate

Baud rate	Characteristics
1200 bps	Low speed / high reliability
2400 bps	High speed / low reliability

The baud rate is specified by the fourth parameter of the SCREEN instruction or the second parameter of the CSAVE instruction. Once the baud rate is set, it stays at that value.

```

SCREEN      ,,,<baud rate>
CSAVE      "filename",<baud rate>
           (<baud rate> is 1 for 1200bps, 2 for 2400 bps)

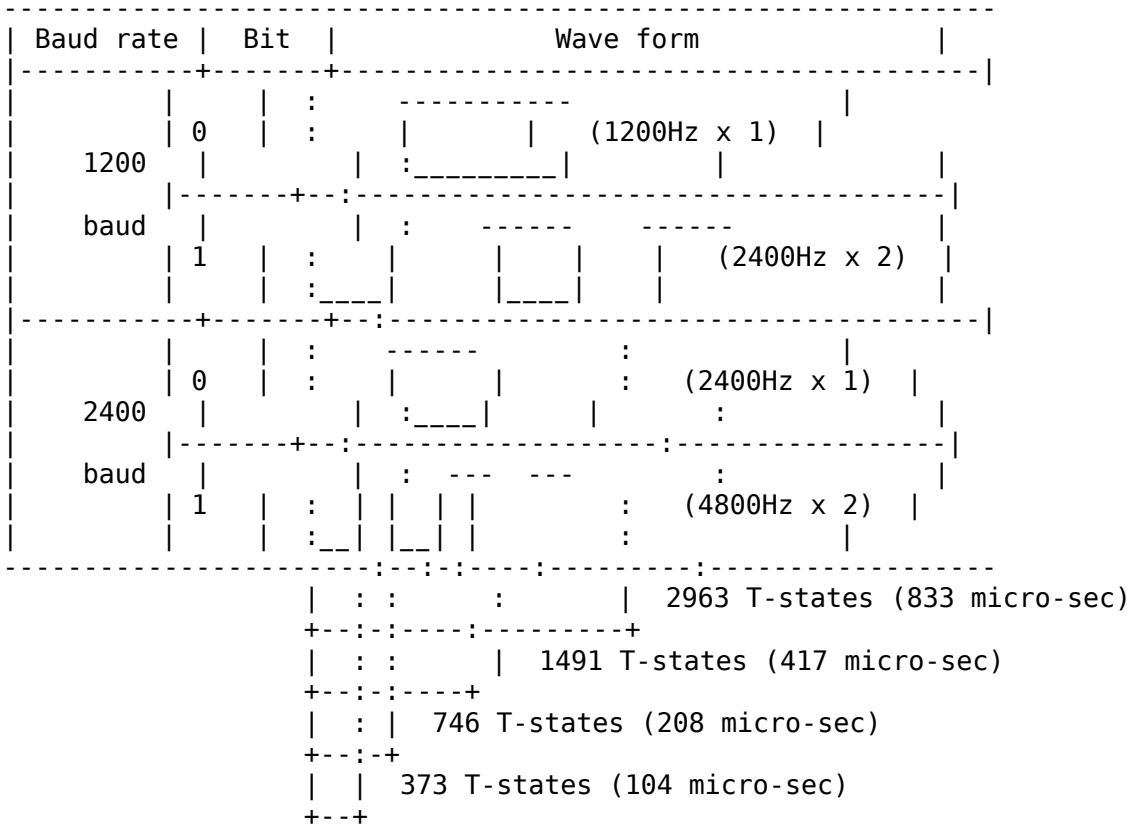
```

2.2 One bit composition

One bit data, the basis of I/O, is recorded as shown in Figure 5.11. The pulse width is determined by counting the T-STATE of the CPU, so, while the cassette interface is active, any interrupt is inhibited.

The bit data from the cassette can be read through the seventh bit of port B of the general-purpose I/O interface (register 15 of the PSG). This function was used in the program example of List 5.3, section 1 of chapter 5.

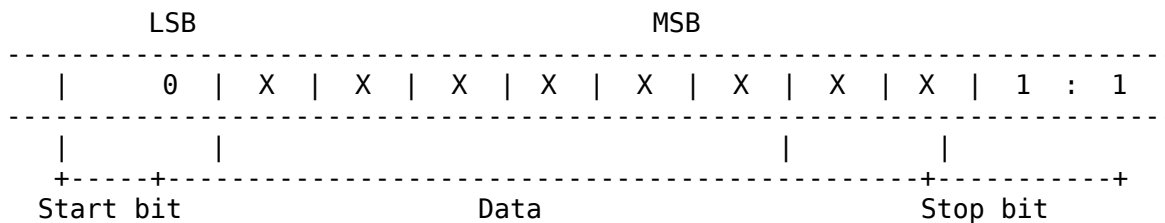
Figure 5.11 One bit composition



2.3 One byte composition

One byte data is recorded in the array of bits as shown in Figure 5.12. There is one "0" bit as the start bit, followed by the 8-bit data body from LSB to MSX and by two "1" bit as the stop bits, so 11 bits are used.

Figure 5.12 One byte composition



2.4 Header Composition

The header is the portion where the signal of the specific frequency is recorded on the tape for a certain period. This allows the cassette tape speed to stabilize after it is started, or divides two files. There is a long header and a short header. The long header is used to wait until the motor is stabilized. The baud rate at reading the tape is determined by reading the long header. The short header is used to divide file bodies. Table 5.3 shows the compositions of both.

Table 5.3 Header composition

Baud rate	Header	Header composition
1200 baud	Long header	2400 Hz x 16000 (about 6.7 sec)
	Short header	2400 Hz x 4000 (about 1.7 sec)
2400 baud	Long header	4800 Hz x 32000 (about 6.7 sec)
	Short header	4800 Hz x 8000 (about 1.7 sec)

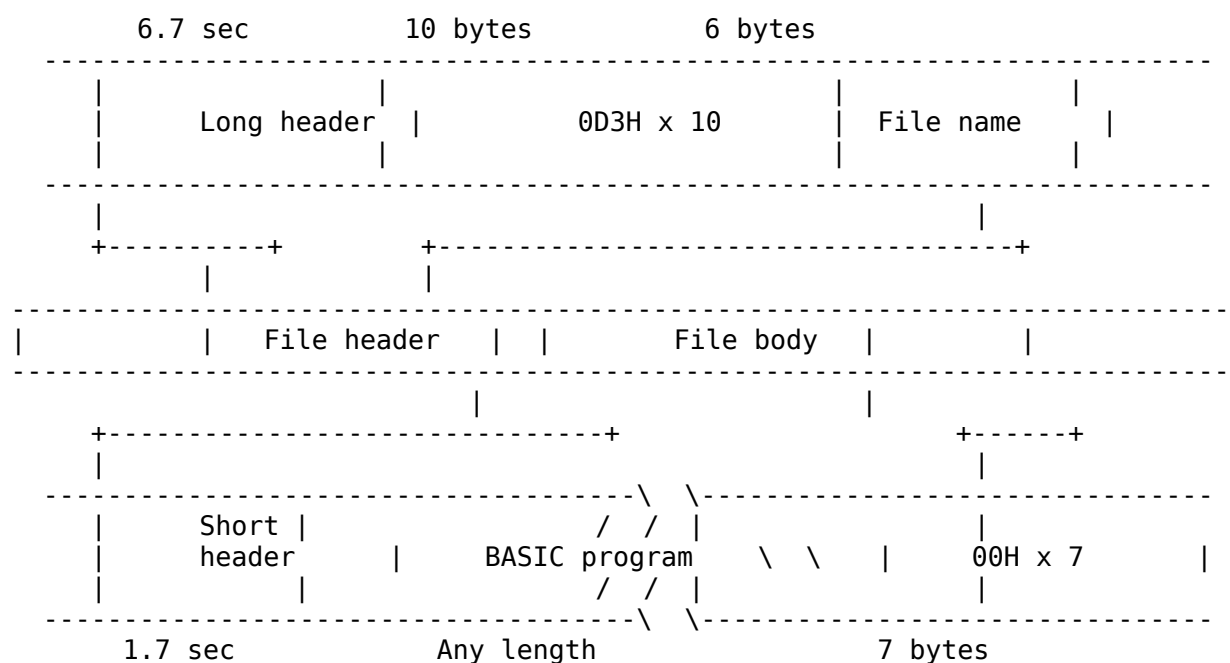
2.5 File Formats

MSX BASIC supports the following three kinds of cassette format files.

(1) BASIC text file

BASIC programs saved with the CSAVE command are recorded in this format. The file is divided into the preceding file header and the succeeding the body.

Figure 5.13 Binary file format

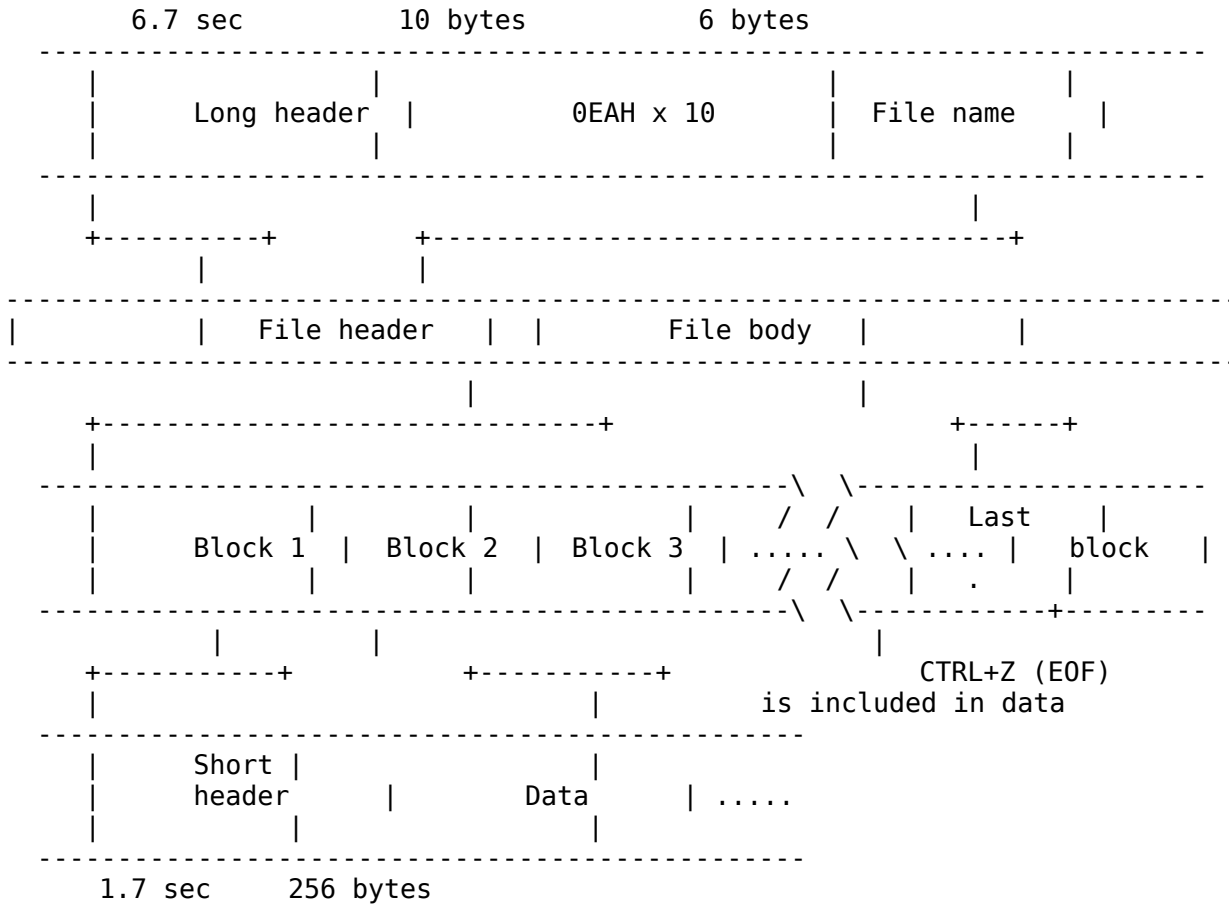


In the file header, ten bytes each of the value 0D3H follow after the long header and six bytes containing the file name are placed after them. In the file body, program body follows the short header and the end of the file is indicated by seven bytes of 00H.

(2) ASCII text file

BASIC programs saved in ASCII format by the SAVE command and data files created by the OPEN command are recorded in this format.

Figure 5.14 ASCII file format

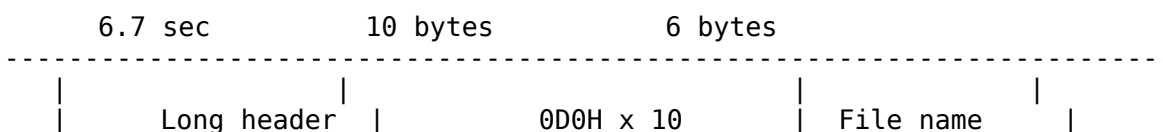


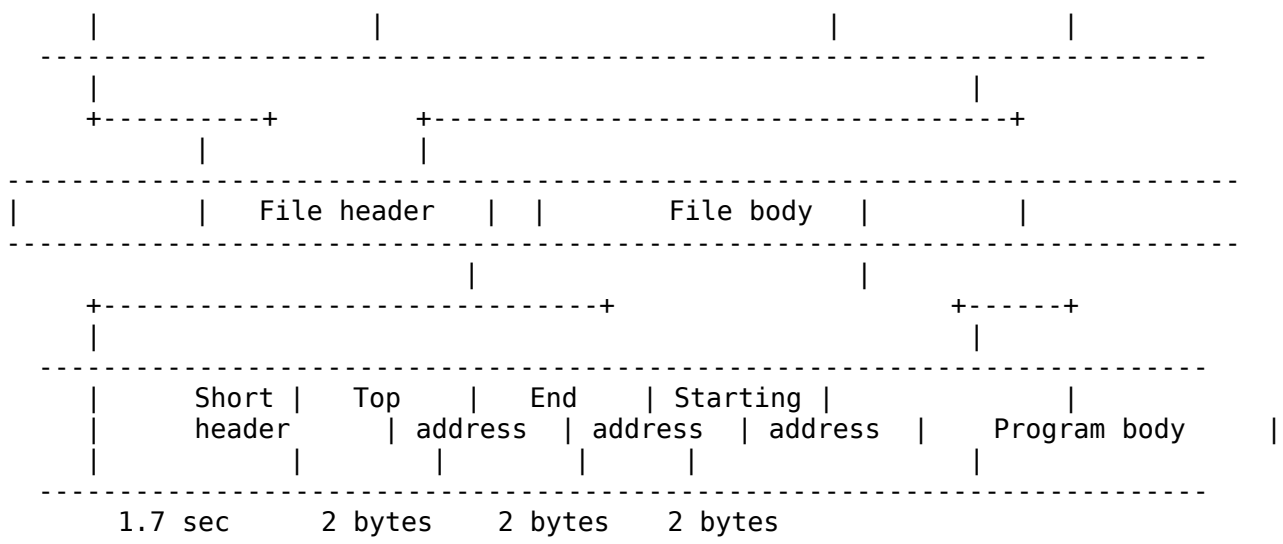
(3) Machine code file

Machine code files saved by the BSAVE command are recorded in the following format. In the file header, 10 bytes each of the value 0D0H follow after the long header and 6 bytes containing the file name are placed after them.

In the file body, the starting address, the end address, and the entry address are recorded in order after the short header, and the machine codes follow after them. Since the amount of data can be calculated from the starting and ending addresses, there is no special mark for the end of the file. The entry address is the address where the program is executed when the R option of the BLOAD command is used.

Figure 5.15 Machine code file format





2.6 Access to cassette files

The following BIOS routines are offered to access cassette files.

* TAPION (00E1H/MAIN) OPEN for read

Input: ---
 Output: CY flag = ON at abnormal terminations
 Function: starts the motor of the tape recorder and reads the long header or the short headet. At the same time, the baud rate in which the file is recorded is detected and the work area is set according to it. Interrupts are inhibited.

* TAPIN (00E4H/MAIN) read one byte

Input: ---
 Output: A <-- data which has been read
 CY flag = ON at abnormal terminations
 Function: reads one byte of data from the tape and stores it in the A register.

* TAPIOF (00E7H/MAIN) CLOSE for read

Input: ---
 Output: ---
 Function: ends reading from the tape. At this point, interrupts are allowed.

* TAPOON (00EAH/MAIN) OPEN for write

Input: A <-- type of header (0 = short header, others = long header)
 Output: CY flag = ON at abnormal terminations
 Function: starts the motor of the tape recorder and writes the header of the type specified in the A register to the tape. Interrupts are inhibited.

* TAPOUT (00EDH/MAIN) write one byte

Input: A <-- data to be written
Output: CY flag = 0N at abnormal terminations
Function: writes the contents of the A register to the tape.

* TAP00F (00F0H/MAIN) CLOSE writing

Input: ---
Output: ---
Function: ends writing the tape. At this point, interrupts are allowed.

* STMOTR (00F3/MAIN) specify the actions of the motor

Input: A <-- action (0 = stop, 1 = start, 255 = reverse the current status)
Output: ---
Function: sets the status of the motor according to the value specified in the A register.

When READ/WRITE routines for the cassette files are created using these BIOS calls, only READ or WRITE, without any other action, should be done. For example, reading data from the tape and displaying it on the CRT might cause a READ error.

List 5.3 is a sample program which uses BIOS routines.

List 5.3 Listing names of files saved in the cassette

```
=====
;*****
;
; List 5.3 Cassette files
;
; Set cassette tape into recorder and run this program.
; Then all the names and attributes of the programs
; in that tape will be listed.
;
;*****
;
CHPUT EQU 00A2H
TAPION EQU 00E1H
TAPIN EQU 00E4H
TAPIOF EQU 00E7H

ORG 0C000H

;----- program start ----- Note: View program names on cassette tape.
START: CALL TAPION ;motor on and read header

LD B,16
```

```

LD HL,WORK ;work area address
LBL01: PUSH HL
      PUSH BC
      CALL TAPIN ;read a byte of data from tape
      POP BC
      POP HL
      JR C,ERROR ;set carry flag if read error
      LD (HL),A
      INC HL
      DJNZ LBL01

      LD HL,FILNAM ;write file name
      CALL PUTSTR
      LD HL,WORK+10
      CALL PUTSTR
      CALL CRLF

      LD A,(WORK) ;check file attributes

      LD HL,BINFIL
      CP 0D3H ;check binary file
      JR Z,LBL03

      LD HL,ASCFIL
      CP 0EAH ;check ascii file
      JR Z,LBL03

      LD HL,MACFIL
      CP 0D0H ;check machine code file
      JR Z,LBL03

ERROR: LD HL,ERRSTR

LBL03: CALL PUTSTR
      CALL TAPIOF
      RET

;----- put CRLF -----
CRLF: LD HL,STCRLF
      CALL PUTSTR
      RET

;----- put string -----
PUTSTR: LD A,(HL) ;get a character from strings
        CP '$' ;check end of strings
        RET Z
        CALL CHPUT ;write a character to CRT
        INC HL
        JR PUTSTR

;----- strings data -----
FILNAM: DB 'FILE NAME :$'
ASCFIL: DB 'ASCII FILE',0DH,0AH,'$'
BINFIL: DB 'BINARY FILE',0DH,0Ah,'$'
MACFIL: DB 'BSAVE FILE',0DH,0AH,'$'

```

```

ERRSTR: DB 'TAPE READ ERROR',0DH,0AH,'$'
STCRLF: DB 0DH,0AH,'$'

;----- WORK AREA -----

WORK: DS 16,0
      DB '$' ;end of strings

      END

```

3. KEYBOARD INTERFACE

Although the MSX2 keyboard has the same design as that of the MSX1, it is more convenient to use because of the Romand-to-kana translation available for kana input. This chapter describes the keyboard interface of the MSX2.

Descriptions of the key arrangement are based on the Japanese keyboard standard; note that data is slightly different for the international MSX versions.

3.1 Key Scanning

MSX uses the key matrices as shown in Figure 5.16, Figure 5.17 and Figure 5.17B. The key status can be obtained in real time by examining this key matrix and is available for reading input.

Scanning the key matrix is done by the following BIOS routine.

* SNSMAT (0141H/MAIN) reads the specified line of the key matrix

Input: A <-- key matrix line to be read (0 to 10)
Output: A <-- status of the specified line of the key matrix (when pressed, the bit of the key is 0)
Function: specifies a line of the key matrix shown in Figure 5.16, Figure 5.17 or Figure 5.17B and stores its status in the A register. The bit corresponding with the key being pressed is "0", and "1" for the key not being pressed.

Figure 5.16 MSX USA version key matrix

MSB	7	6	5	4	3	2	1	0	LSB
0	B	L	/	/	1	S	X	,	
1	V	J	=	`	Q	A	C	N	
2	G	8	0]	W	F	Z	M	
3	T	I	~	;	2	D	U	\	

4	6	K	P	'	3	R	7	H	
5	5	0	9	[4	E	Y	.	
6	F3	F2	F1	CODE	CAPS	GRAPH	CTRL	SHIFT	
7	RETURN	SELECT	BS	STOP	TAB	ESC	F5	F4	
8	RIGHT	DOWN	UP	LEFT	DEL	INS	HOME	SPACE	

[TEN KEY]									

9	4	3	2	1	0	option	option	option	
10	.	,	-	9	8	7	6	5	

Figure 5.17 MSX International version key matrix

MSB	7	6	5	4	3	2	1	0	LSB
0	B	L	deadkey	/	1	S	X	,	
1	V	J	^]	Q	A	C	N	
2	G	8	0	[W	F	Z	M	
3	T	I	~	;	2	D	U	\	
4	6	K	P	:	3	R	7	H	
5	5	0	9	@	4	E	Y	.	
6	F3	F2	F1	CODE	CAPS	GRAPH	CTRL	SHIFT	
7	RETURN	SELECT	BS	STOP	TAB	ESC	F5	F4	
8	RIGHT	DOWN	UP	LEFT	DEL	INS	HOME	SPACE	

[TEN KEY]									

9	4	3	2	1	0	option	option	option	
10	.	,	-	9	8	7	6	5	

Figure 5.17B MSX European version key matrix

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

1	;]	[\	=	-	9	8	
2	B	A	accent	/	.	,	`	'	
3	J	I	H	G	F	E	D	C	
4	R	Q	P	O	N	M	L	K	
5	Z	Y	X	W	V	U	T	S	
6	F3	F2	F1	CODE	CAPS	GRAPH	CTRL	SHIFT	
7	RETURN	SELECT	BS	STOP	TAB	ESC	F5	F4	
8	RIGHT	DOWN	UP	LEFT	DEL	INS	HOME	SPACE	

[TEN KEY]									

9	4	3	2	1	0	option	option	option	
10	.	,	-	9	8	7	6	5	

List 5.4 Use of the key scanning routine

```

;*****
;
; List 5.4 scan key-matrix and display it
;
;*****
;
CHPUT EQU 00A2H
BREAKX EQU 00B7H
POSIT EQU 00C6H
SNSMAT EQU 0141H

ORG 0B000H

;----- program start ----- Note: read key matrix and display key
                                pattern.

SCAN: LD C,0 ;C := line of key matrix

SC1: LD A,C
CALL SNSMAT ;Read key matrix

LD B,8
LD HL,BUF ;HL := buffer address
SC2: LD D, '.'
RLA ;Check bit
JR C,SC3
LD D,'#'

SC3: LD (HL),D ;store '.' or '#' to buffer
INC HL

```

```

DJNZ SC2

LD H,05H ;x := 5
LD L,C ;y := C+1
INC L
CALL POSIT ;set cursor position

LD B,8 ;put out bit patterns to CRT
LD HL,BUF
SC4: LD A,(HL)
CALL CHPUT
INC HL
DJNZ SC4

CALL BREAKX ;check Ctrl-STOP
RET C

INC C ;line No. increment
LD A,C
CP 09
JR NZ,SC1
JR SCAN

;----- work area -----
BUF: DS 8

END

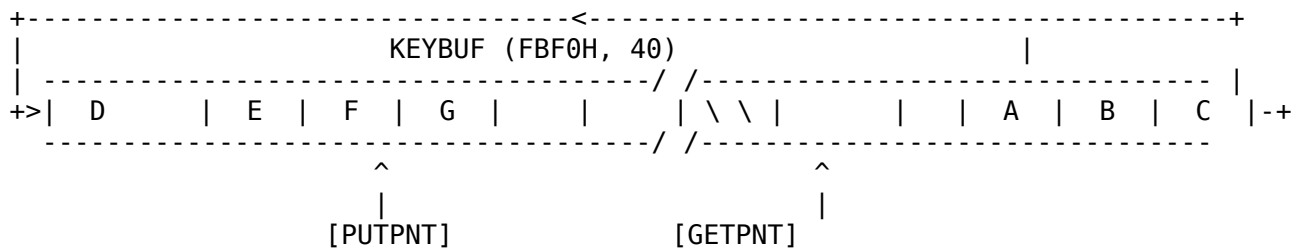
```

=====

3.2 Character Input

MSX scans the key matrix every 1/60 second using the timer interrupt and, when a key is pressed, stores the character code in the keyboard buffer as shown in Figure 5.18. Key input to MSX is generally done by reading this keyboard buffer.

Figure 5.18 Keyboard ring buffer



GETPNT (F3FAH, 2) points to the next character to be obtained in CHGET routine.

PUTPNT (F3F8H, 2) points to the next location for the character to be put when the keyboard is pressed next time.

BIOS routines having functions for key input using this keyboard buffer and functions related to it are described below. Inhibiting the timer interrupt renders them useless, of course.

* CHSNS (009CH/MAIN) checks the keyboard buffer

Input: ---
Output: Z flag = ON when the buffer is empty
Function: examines whether any characters remain in the keyboard buffer and sets the Z flag when the buffer is empty.

* CHGET (009FH/MAIN) one character input from the keyboard buffer

Input: ---
Output: A <-- character code
Function: reads one character from the keyboard buffer and stores it in the A register. When the buffer is empty, it displays the cursor and waits for a key input. While a key input is waited for, the CAP lock, KANA lock, and Roman-to-kana translation lock are valid. The related work area is listed below. In the list, since SCNCNT and REPCNT are initialised after the execution of CHGET routine, this area should be set at each CHGET call to change the interval of the auto-repeat.

Work area

- CLIKSW (F3DBH, 1) key click sound (0 = OFF, others = ON)
- SCNCNT (F3F6H, 1) key scanning interval (1, normally)
- REPCNT (F3F7H, 1) delay until beginning auto-repeat (50, normally)
- CSTYLE (FCAAH, 1) figure of the cursor (0 = block, others = underline)
- CAPST (FCABH, 1) CAPS lock (0 = OFF, others = ON)
- DEADST (FCACH, 1) dead key lock
 - 0 = on preceding dead key
 - 1 = dead key
 - 2 = shifted dead key
 - 3 = code dead key
 - 4 = code shift dead key

* KILBUF (0156H/MAIN) empty the keyboard buffer

Input: ---
Output: ---
Function: empties the keyboard buffer.

List 5.5 Use of one character input routine

```
=====
;*****
;
; List 5.5 get key code
;
; this routine doesn't wait for key hit
```

```

;
;*****
;
CHSNS EQU 009CH ;check keyboard buffer
CHGET EQU 009FH ;get a character from buffer
CHPUT EQU 00A2H ;put a character to screen
BREAKX EQU 00B7H ;check Ctrl-STOP
KILBUF EQU 0156H ;clear keyboard buffer
REPCNT EQU 0F3F7H ;time interval until key-repeat
KEYBUF EQU 0FBF0H ;keyboard buffer address

```

```

ORG 0B000H

```

----- program start ----- Note: Real-time input using CHGET

```

KEY: CALL CHSNS ;check keyboard buffer
      JR C,KEY1

      LD A,1
      LD (REPCNT),A ;not to wait until repeat
      CALL CHGET ;get a character (if exists)
      JR KEY2

KEY1: LD A,'-' ;A := '-'

KEY2: CALL CHPUT ;put the character
      CALL KILBUF ;clear keyboard buffer
      CALL BREAKX ;check Ctrl-STOP
      JR NC,KEY

      END

```

=====

* CNVRCHR (00AB/MAIN) graphic character operation

Input: A <-- character code
Output: A <-- translated graphic character
(normal characters are not translated)
CY flag = OFF (input was the graphic header byte 01H)
CY flag = 0N, Z flag = 0N (input was the graphic character
and was translated)
CY flag = 0N, Z flag = OFF (input was the normal character
and was not translated)

Function: executing CNVCHR after CHGET causes the graphic character
to be translated to one byte code as shown in Figure 5.19
and causes other character not to be translated and to be
returned. Since the graphic character is represented by
irregular 2-byte code with the graphic header byte (01H),
annoying procedures are required for the character
operations; this routine makes it somewhat easy.

Figure 5.19 Graphic character translation chart

Before	After	Before	After

conversion	conversion	conversion	conversion
		0150H -->	50H
0141H -->	41H	0151H -->	51H
0142H -->	42H	0152H -->	52H
0143H -->	43H	0153H -->	53H
0144H -->	44H	0154H -->	54H
0145H -->	45H	0155H -->	55H
0146H -->	46H	0156H -->	56H
0147H -->	47H	0157H -->	57H
0148H -->	48H	0158H -->	58H
0149H -->	49H	0159H -->	59H
014AH -->	4AH	015AH -->	5AH
014BH -->	4BH	015BH -->	5BH
014CH -->	4CH	015CH -->	5CH
014DH -->	4DH	015DH -->	5DH
014EH -->	4EH	015EH -->	5EH
014FH -->	4FH	015FH -->	5FH

* PINLIN (00AEH/MAIN) one line input

Input: ---
Output: HL <-- F55DH
[F55EH] <-- input string (the end of the line is represented by 00H)
CY flag <-- terminated by STOP=ON, terminated by RETURN=OFF
function: stores input string in the line buffer BUF (F55EH). All functions of the screen editing are available at the string input. Pressing RETURN or STOP causes the input to be finished. The work area is listed below.

Work area
BUF (F55EH, 258) the line buffer where the string is stored
LINTTB (FBB2H, 24) 00H when the one physical line is the succession of the line above

* INLIN (00B1H/MAIN) one line input (prompt available)

Input: ---
Output: same as PINLIN
Function: stores input string in the line buffer BUF (F55EH), as PINLIN routine. Note that the portion before the cursor location at the time when the routine begins to execute is not received. List 5.6 shows the difference between PINLIN and INLIN.

List 5.6 Difference between INLIN and PINLIN

```
=====
;*****
;
; List 5.6 INLIN and PINLIN
;
;*****
```

```

;
CHPUT EQU 00A2H
INLIN EQU 00B1H
PINLIN EQU 00AEH
KILBUF EQU 0156H

BUF EQU F55EH

ORG 0B000H

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

LD HL,PRMPT1
CALL PUTMSG ;put prompt message
CALL INLIN ;use INLIN routine
LD HL,BUF
CALL PUTMSG

LD HL,PRMPT2
CALL PUTMSG ;put prompt message
CALL PINLIN ;use PINLIN routine
LD HL,BUF
CALL PUTMSG

RET

;----- put a string -----

PUTMSG: LD A,(HL)
CP '$'
RET Z
CALL CHPUT
INC HL
JR PUTMSG

;----- string data -----

PRMPT1: DB 0DH,0AH,'INLIN:$'
PRMPT2: DB 0DH,0AH,'PINLIN:$'

END

```

3.3 Function Keys

MSX has ten function keys, which can be defined by the user at will. A 16 byte work area is allocated for the definition of each key. The following list shows their addresses.

```

=====

FNKSTR (F87FH, 16) ..... F1 key definition address
+ 10H (F88FH, 16) ..... F2 key definition address
+ 20H (F89FH, 16) ..... F3 key definition address
+ 30H (F8AFH, 16) ..... F4 key definition address
+ 40H (F8BFH, 16) ..... F5 key definition address
+ 50H (F8CFH, 16) ..... F6 key definition address
+ 60H (F8DFH, 16) ..... F7 key definition address

```

```

+ 70H (F8EFH, 16) ..... F8 key definition address
+ 80H (F8FFH, 16) ..... F9 key definition address
+ 90H (F90FH, 16) ..... F10 key definition address

```

Pressing a function key causes the string defined in that key to be stored in [KEYBUF]. The end of the string is indicated by 00H and a maximum of 15 keystrokes can be defined for one function key (definitions longer than 16 keystrokes are defined over more than one function key definition area). To restore the initial settings of the function keys, use the following BIOS routine.

```
* INIFNK (003EH/MAIN) ..... initialize function keys
```

```

Input:      ---
Output:     ---
Function:   restores the function key definition to the setting when
            BASIC starts.

```

3.4 STOP Key During Interrupts

CHGET, the one-character input routine described in 3.3, determines the pressed key in the timer interrupt routine. Thus, when the timer interrupt is inhibited, such as during cassette data I/O, pressed keys cannot be detected. By using the BIOS routine described below, the CTRL key + STOP key combination can be detected even when interrupts are inhibited.

```
* BREAKX (00B7H/MAIN) ..... CTRL + STOP detection
```

```

Input:      ---
Output:     CY flag = ON, when CTRL + STOP is pressed
Function:   scans keys and decides whether CTRL key and STOP key are
            pressed at the same time. When both are pressed, this routine
            sets "1" to the CY flag and returns. Otherwise, it resets "0"
            to the CY flag and returns. This routine is available while
            interrupts are inhibited.

```

4. PRINTER INTERFACE

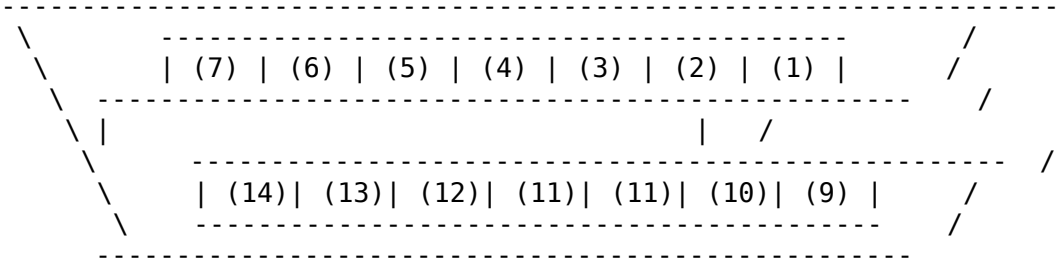
This section describes how to access the MSX printer interface from assembly language. The information described here is helpful if the printer is going to be used to print bit image graphics.

4.1 Print Interface Overview

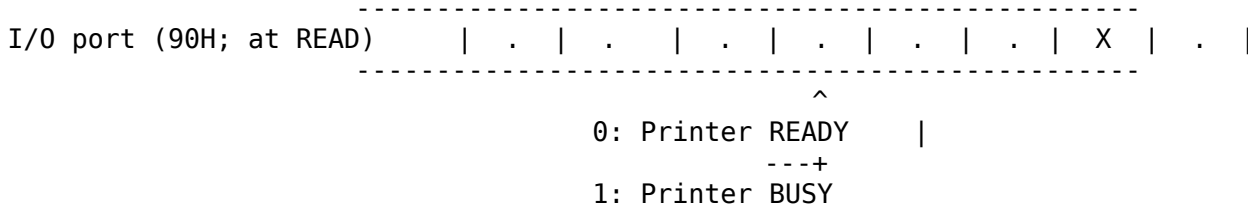
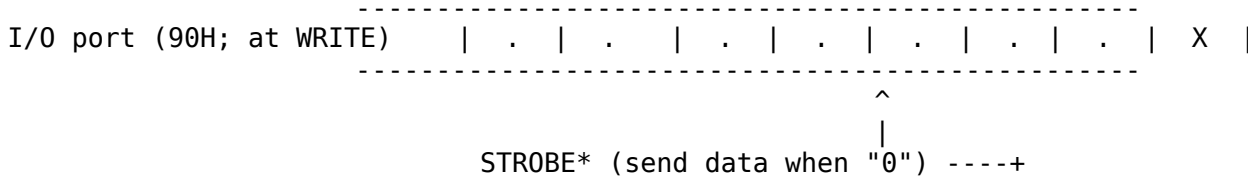
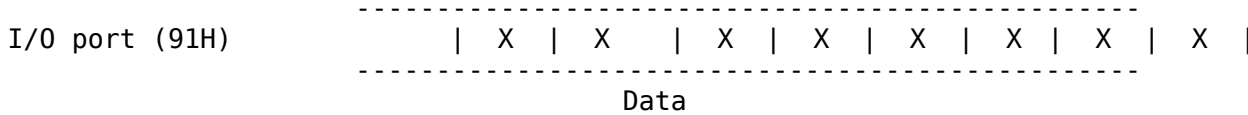
The printer interface is supported by BIOS and BASIC. MSX drives the printer through an 8-bit parallel output port and uses a handshaking method with BUSY and STROBE signals. The standard connector is also defined (Amphenol 14-pin, female side to the machine). Figure 5.20 shows the signal lines.

Figure 5.20 Printer interface

Printer interface pin connections



- (1) STROBE*
- (2) to (9) Data (b0 to b7)
- (11) BUSY
- (14) BGND



4.2 Output to the MSX Standard Printer

If data is sent from MSX to the printer, the action depends on whether the printer receiving the data is of the MSX standard. The use of MSX standard printers is described in this section. Descriptions about other printers are in the next section.

An MSX standard printer can print any character that can be displayed on the screen. Special graphic characters corresponding to character codes $n = 01H$ to $1FH$ can be also printed by sending the code $40H + n$ after the graphic character header ($01H$). In addition to these, the control codes shown in Table 5.4 can be used with MSX standard printers (see the manual of the printer for controlling a printer which has other functions such as printing Chinese characters).

To feed lines in MSX standard printers, send $0DH$ and $0AH$ successively. To print the bit image, send $nnnn$ bytes data, where $nnnn$ means four decimal figures, after the escape sequence $ESC + "Snnnn"$. Note that, MSX has a function to transform the tab code ($09H$) to the adequate number of space codes ($20H$) for printers not having a tab function. This transformation is normally done. To print a bit image which includes the value $09H$ correctly,

change the following work area.

* RAWPRT (F418H, 1) replaces a tab by spaces when the contents are 00H, otherwise not.

Table 5.4 Control codes of the printer

code	function
0AH	line feed
0CH	form feed
0DH	carriage return
ESC + "A"	normal line spacing (spaces between lines; characters are read easily)
ESC + "B"	line spacing for graphics (no space between lines)
ESC + "Snnnn"	bit image printing

4.3 Access to the printer

To send output to the printer, the following BIOS routines are offered.

* LPTOUT (00A5H/MAIN)

Input: A register <-- character code
Output: CY flag = 0N at abnormal termination
Function: sends a character specified by the A register to the printer.

* LPTSTT (00A8/MAIN)

Input: ---
Output: A register <-- printer status
Function: examines the current printer status. After calling this routine, the printer can be used when the A register is 255 and the Z flag is 0; when the A register is 0 and the Z flag is 1, the printer cannot be used.

* OUTDLP (014DH,MAIN)

Input: A register <-- character code
Output: CY flag = 0N at abnormal termination
Function: sends a character specified by the A register to the printer. Differences between this routine and LPTOUT routine is as following:
* prints corresponding number of spaces for TAB code
* transforms hiragana to katakana for printers other than

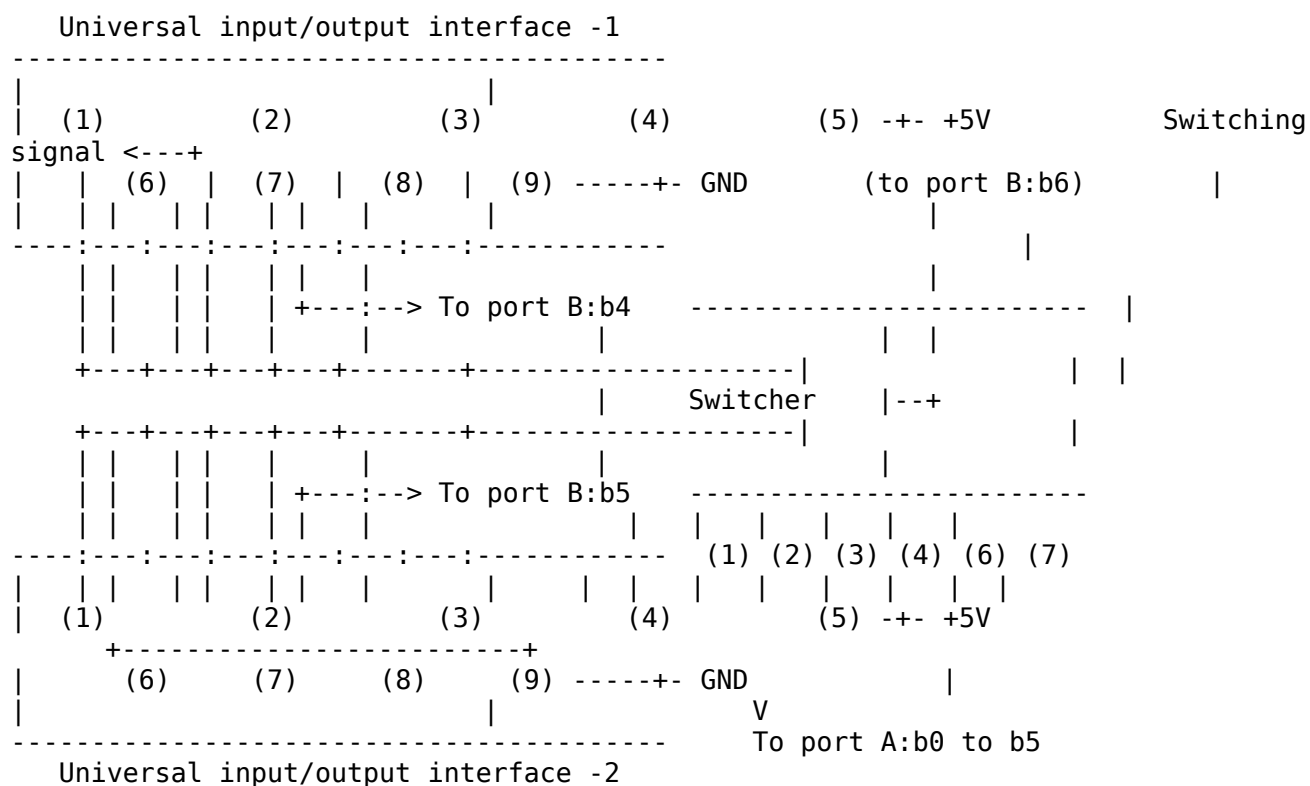
MSX standard
 * returns Device I/O error at abnormal termination

5. UNIVERSAL I/O INTERFACE

As described in section 1, the PSG used by MSX has two 8-bit I/O ports, port A and port B, in addition to the sound output function. In MSX, these two ports are connected to the universal I/O interface (joystick port) and are used to exchange data with the joystick or the paddle (see Figure 5.21). Various devices to be connected to this universal I/O interface have the necessary BIOS routine in ROM, so they are easily accessible.

In this section, the function of each I/O device and the method for accessing with BIOS routines are described.

Figure 5.21 Universal I/O interface

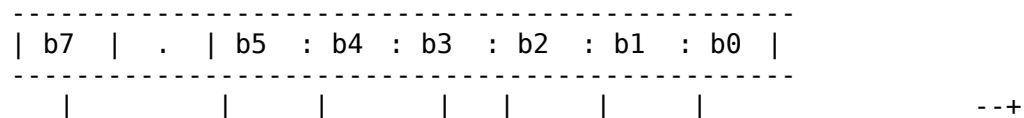


5.1 Functions of the Ports

Two I/O ports of PSG are used as shown in Figure 5.22.

Figure 5.22 (A) Functions of PSG port A

Port A (PSG#14)



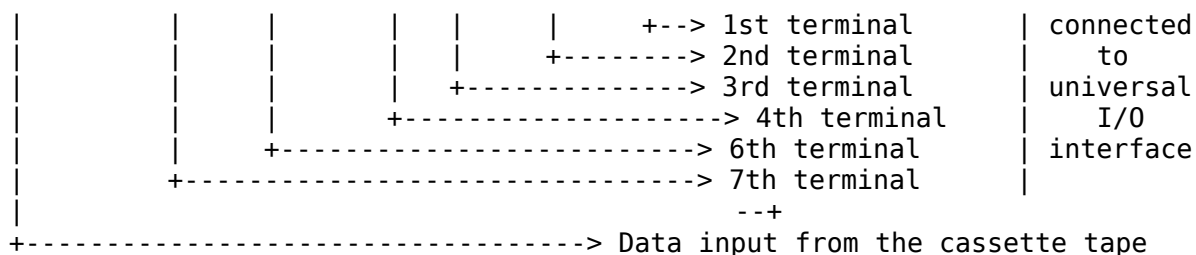
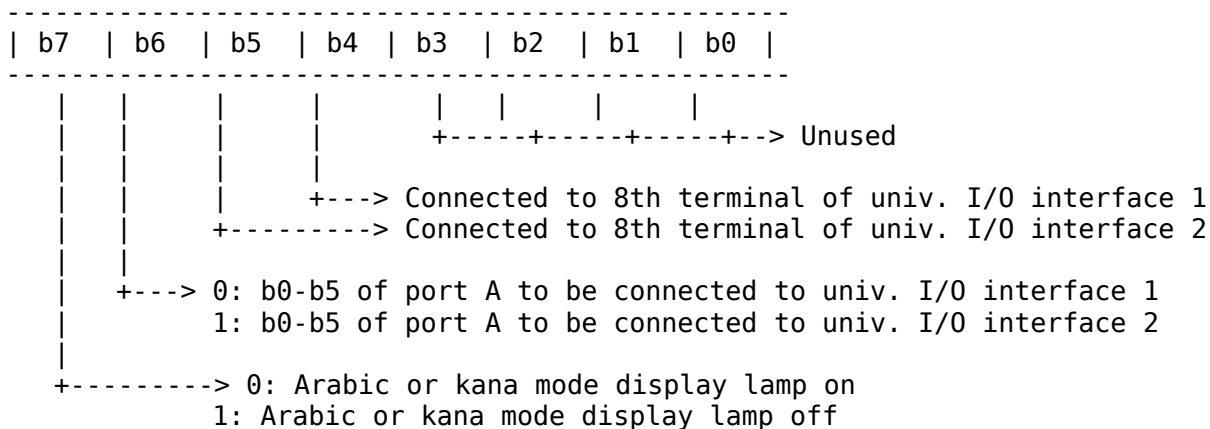


Figure 5.22 (B) Functions of PSG port B

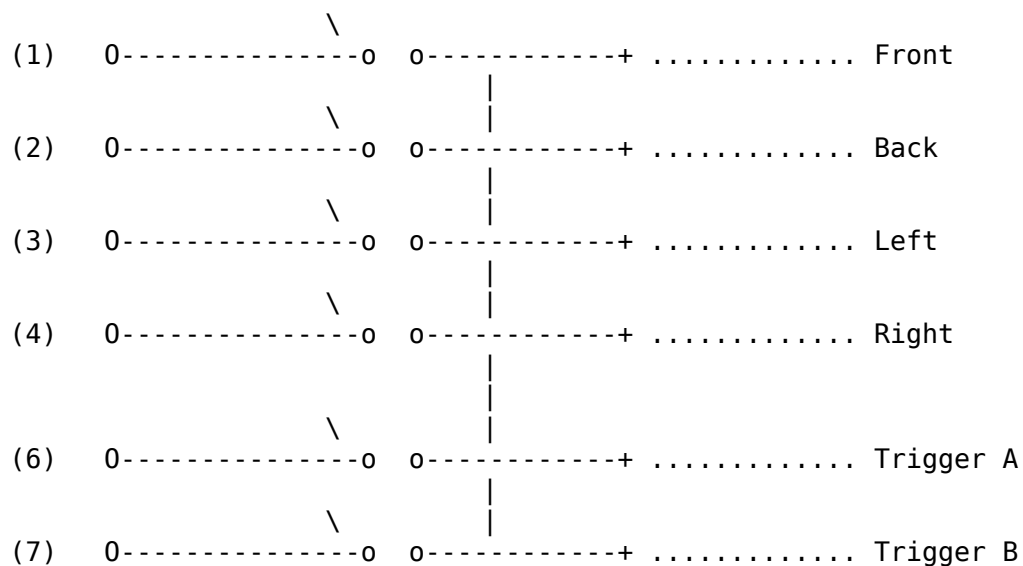
Port B (PSG#15)



5.2 Joystick Use

Figure 5.23 shows the joystick circuit. As the circuit shows, sending "0" to the 8th terminal and reading the 1st to 4th and 6th to 7th terminals enable information about the stick and the trigger buttons to be obtained. However, it is advisable to use BIOS for accessing the joystick, in order to give portability to the program.

figure 5.23 Joystick circuit



(8) 0-----+
|

The following BIOS routines are offered for accessing the joystick. These routines have similar functions to the STICK function and STRIG function of BASIC. The status of the cursor keys or the space bar, in addition to the joystick, can be read in real time.

* GTSTCK (00D5H/MAIN) read joystick

Input: A <-- joystick number (0 = cursor key, 1 and 2 = joystick)
Output: A <-- direction of joystick or cursor key
Function: returns the current status of the joystick or the cursor keys
 in the A register. The value is the same as the STICK
 function in BASIC.

* GTTRIG (00D8H/MAIN) read trigger button

Input: A <-- trigger button number (0 = space bar,
 1 and 2 = trigger button A, 3 and 4 = trigger button B)
Output: A <-- status of trigger button or space bar
 (0FFH = pressed, 00H = released)
Function: returns the current status of the trigger buttons or the
 space bar in the A register. The value is 0FFH when the
 trigger is pressed, otherwise it is 0.

List 5.7 Joystick use

=====

```
*****  
;  
; List 5.7 Joystick and trigger access  
;  
*****
```

```
CHPUT EQU 00A2H  
BREAKX EQU 00B7H  
GTSTCK EQU 00D5H  
GTTRIG EQU 00D8H
```

```
ORG 0D00H
```

```
;-;-;- program start -;-;-    Note: display joystick status
```

```
STICK:     LD    A,1            ;choose joystick 1  
          CALL GTSTCK         ;read joystick status  
          LD    (WK1),A  
          LD    A,1            ;choose joystick 1  
          CALL GTTRIG         ;read trigger status  
  
          OR    A  
          JR    Z,STCK1  
          LD    HL,WDON        ;trigger ON
```



```

        JR      STCK2
STCK1:  LD      HL,WDOFF      ;trigger OFF
STCK2:  CALL   PUTSTR
        LD      A,(WK1)
        OR      A
        JR      Z,BRKCH0      ;do not use joystick
        LD      C,0
STCK3:  DEC     A
        JR      NZ,STCK4
        INC     C
        JR      STCK3

STCK4:  SLA     C              ;C := C*16
        SLA     C
        SLA     C
        LD      B,0           ;Accounting Strings data address
        LD      HL,WSTK
        ADD     HL,BC
        CALL   PUTSTR

BRKCH0: LD      A,0DH         ;put carriage return
        CALL   CHPUT         ;code := 0DH

BRKCHK: CALL    BREAKX       ;break check
        RET     C
        JR      STICK

;----- put strings to screen -----

PUTSTR: LD      A,(HL)
        CP      '$'
        RET     Z
        INC     HL
        CALL   CHPUT
        JR      PUTSTR

;----- string area -----

WDON:  DB      'Trigger ON: $'
WDOFF: DB      'Trigger OFF: $'
WSTK:  DB      'UP only      ',0DH,0AH,'$'
        DB      'Up and Right ',0DH,0AH,'$'
        DB      'Right only   ',0DH,0AH,'$'
        DB      'Right & Down ',0DH,0AH,'$'
        DB      'Down only    ',0DH,0AH,'$'
        DB      'Down and Left',0DH,0AH,'$'
        DB      'Left only    ',0DH,0AH,'$'
        DB      'Left and Up  ',0DH,0AH,'$'

WK1:  DW      0

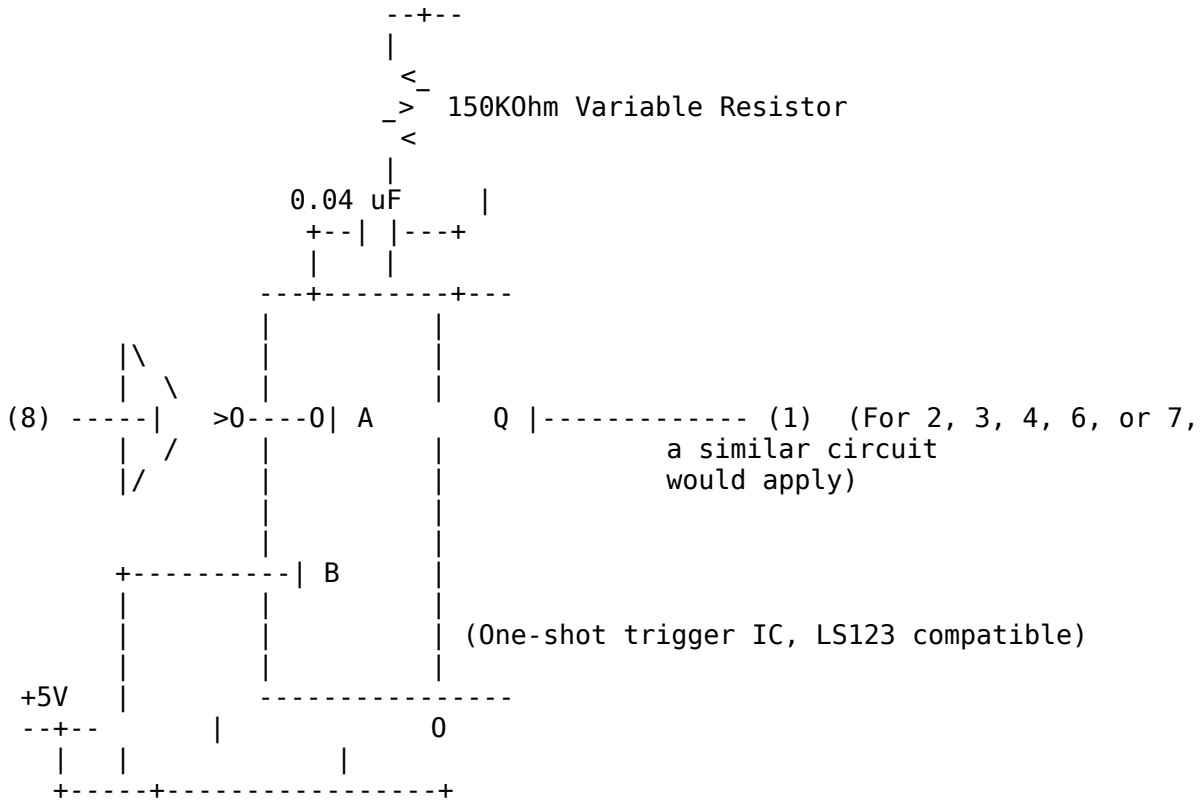
        END

```

5.3 Paddle Use

Figure 5.24 shows the paddle circuit. Sending a pulse to the 8th terminal causes the single stable multi-vibrator to generate a pulse with a specified interval. This interval depends on the value of the variable register which can range from 10 to 3000 microseconds (0.01 to 3.00 ms). Measuring the pulse length enables the value in the variable register and the turning angle to be obtained.

Figure 5.24 Paddle circuit



Input to 8 _____ : : _____

Output to 1 _____ : : _____
 |<----- 10 us to 3 ms ----->|

BIOS routines for accessing the paddle are described below.

* GTPDL (00DEH/MAIN) read paddle information

Input: A <-- paddle number (1 to 12)
 Output: A <-- turning angle (0 to 255)
 Function: examines the status of the paddle specified in the A register and returns the result in the A register.

5.4 Use of Touch Panel, Light Pen, Mouse, and Track Ball

The touch panel, light pen, mouse, and track ball (cat) are accessible using the same BIOS routine. This routine is described below.

* GTPAD (00DBH/MAIN) access to various I/O devices

Input: A <-- device ID (0 to 19)

Output: A <-- objective information

Function: obtains various information as shown in Table 5.5 according to the value specified in the A register. This is the same as the PAD function of BASIC. "XXX1" in the table means the "XXX" device connected to the universal I/O interface 1; "XXX2" means the one connected to the universal I/O interface #2.

Table 5.5 GTPAD BIOS Function

Device ID	Device specified	Information returned
0		0FFH when touching panel surface, 00H when not
1	Touch panel 1	X-coordinate (0 to 255)
2		Y-coordinate (0 to 255)
3		0FFH when button is pressed, 00H when not
4		
5	Touch panel 2	Same as above
6		
7		
8		0FFH: valid data, 00H: invalid data
9	Light pen	X-coordinate (0 to 255)
10		Y-coordinate (0 to 255)
11		0FFH when switch is pressed, 00H when not
12		Always 0FFH (used to request for input)
13	Mouse 1 or track ball 1	X-coordinate (0 to 255)
14		Y-coordinate (0 to 255)

15		Always 00H (no meaning)
16		
17	Mouse 2 or track ball 2	Same as above
18		
19		

Note 1: Though information of the coordinate of the light pen (A = 9, 10) and the switch (A = 11) are read at the same time when BIOS is called with A = 8, other values are valid only when the result is 0FFH. In the case that the result of BIOS which is called with A = 8 is 00H, the coordinate values and the status of the switch contained after that are meaningless.

Note 2: Mouse and track ball are automatically distinguished.

Note 3: To obtain the coordinate value of the mouse or the track ball, do the input request call (A = 12 or A = 16), then execute the call to obtain the coordinate value actually. In this case, the interval of these two calls must be minimized as possible. Too much interval between the input request and the coordinate input causes the obtained data to be unreliable.

Note 4: To obtain the status of the trigger button of the mouse or the trigger button of the track ball, use GTTRIG (00D8H/MAIN), not GTPAD routine.

List 5.8 Touch panel use

=====

```

;*****
;
; List 5.8 touch pad access
;
;*****

```

```

BREAKX EQU 00B7H
GTPAD EQU 00D8H
WRTVRM EQU 004DH

```

```

ORG 0B000H

```

```

;----- program start ----- Note: Displays "*" at position specified
;                               by touch pad.

```

```

PAD: XOR A ;check sense
CALL GTPAD
OR A
JR NZ,PAD1
LD A,3
CALL GTPAD ;break check
OR A
RET NZ
JR PAD

```

```

PAD1: LD    A,1          ;get X axis
      CALL  GTPAD
      SRL  A            ;A := A/8
      SRL  A
      SRL  A
      LD   (WORK),A     ;reserve X axis
      LD   A,2          ;get Y axis
      CALL  GTPAD
      LD   L,A          ;HL := Y data (0-255)
      LD   H,0
      LD   C,A
      LD   B,0
      ADD  HL,BC        ;HL := HL*3 (HL := 0-767)
      ADD  HL,BC
      LD   A,L
      AND  11100000B
      LD   L,A
      LD   A,(WORK)
      ADD  A,L
      LD   L,A
      LD   BC,1800H     ;VRAM start address
      ADD  HL,BC
      LD   A,2AH
      CALL  WRTVRM      ;write VRAM
      LD   A,3
      CALL  GTPAD      ;break check
      OR   A
      RET  NZ
      JR   PAD

```

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

```

WORK: DW    0          ;work

```

END

List 5.9 Mouse and track ball use

```

;*****
;
; List 5.9 mouse and track ball access
;
;*****
;
GTPAD EQU    00DBH
WRTVRM EQU    004DH
RDVRM EQU    004AH
BREAKX EQU    00B7H

```

ORG 0D000H

;----- program start ----- Note: Displays "*" at position specified by mouse or track ball.

```

TEST: CALL  VADR          ;Put old data
      LD   A,(WKOLD)
      CALL WRTVRM
      LD   A,12
      CALL GTPAD          ;Request mouse/track ball data
      LD   A,13
      CALL GTPAD          ;Read X val.
      LD   (WKXVAL),A
      LD   A,14
      CALL GTPAD          ;Read Y val.
      LD   (WKYVAL),A

      LD   A,(WKX)
      LD   B,A
      LD   A,(WKXVAL)
      ADD  A,B
      CP   245            ;X<0?
      JR   C,TEST01
      XOR  A              ;X=0
      JR   TEST02

TEST01: CP   32            ;X>31?
        JR   C,TEST02
        LD   A,31

TEST02: LD   (WKX),A

        LD   A,(WKY)
        LD   B,A
        LD   A,(WKYVAL)
        ADD  A,B
        CP   245            ;Y<0?
        JR   C,TEST03
        XOR  A              ;Y=0
        JR   TEST04

TEST03: CP   24            ;Y>23?
        JR   C,TEST04
        LD   A,23

TEST04: LD   (WKY),A

        CALL VADR
        CALL RDVRM          ;Read old data
        LD   (WKOLD),A

        CALL VADR
        LD   A,2AH
        CALL WRTVRM          ;Put cursor ("*").

        CALL BREAKX          ;Break check
        RET  C

        CALL WAIT

        JR   TEST

```

```

VADR: LD    A,(WKY)    ;Make SCREEN Address:
      LD    H,A        ; From X,Y axis on WORK AREA
      LD    L,0        ; To Hl reg.
      SRL   H
      RR    L
      SRL   H
      RR    L
      SRL   H
      RR    L
      LD    A,(WKX)
      ADD   A,L        ; Y=32+X
      LD    L,A
      LD    BC,1800H   ; VRAM start address
      ADD   HL,BC
      RET

```

```

WAIT: LD    A,0        ;WAIT routine
WLP1: INC   A
      LD    B,(IX+0)
      LD    B,(IX+0)
      LD    B,(IX+0)
      JR    NZ,WLP1
      RET

```

;----- data -----

```

WKX:  DB    10        ;X axis
WKY:  DB    10        ;Y axis
WKOLD: DB    0        ;Character code on (X,Y)
WKXVAL: DB  0        ;X variable
WKYVAL: DB  0        ;Y variable

```

END

=====

6. CLOCK AND BATTERY-POWERED MEMORY

MSX2 uses a CLOCK-IC to for its timer function. Since this IC is battery-powered, it remains active even after MSX2 is turned off. MSX2 uses a small amount of RAM inside to set the PASSWORD or to set the screen mode at startup automatically, in addition to the CLOCK functions.

6.1 CLOCK-IC Functions

This IC has the following three functions:

* CLOCK function

- set/read the settings of "year, month, day, day of week, hour, minute, second"
- for the expression of time, 24-hour clock/12-hour clock available
- for months, months of 31 days and of 30 days are distinguished (leap years are also recognised)

* Alarm function

- when the time for alarm is set, CLOCK generates signals at that time.
- the time for alarm is set as "XXday XXhour XXminute".

* Battery-powered memory function

- has 26 sets of 4-bit memory, and can be battery-powered.
- MSX2 stores the following data in this memory:

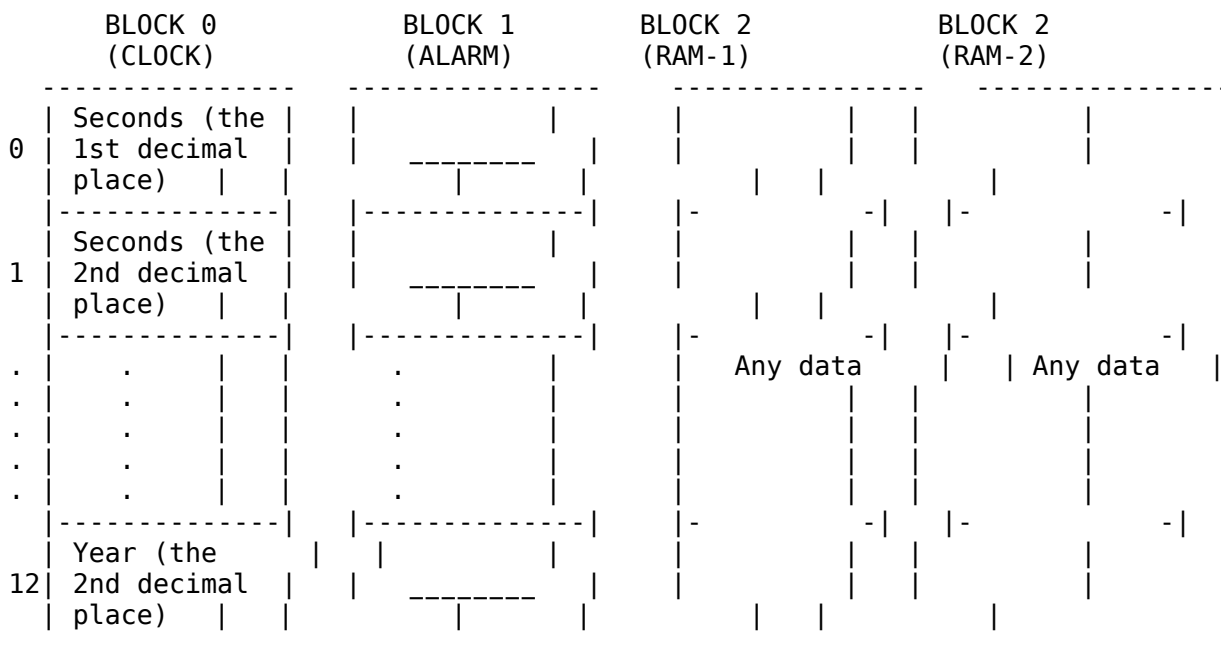
1. adjustment value of CRT display width and height
2. initial values of SCREEN, WIDTH, colour
3. BEEP tone and volume
4. title screen colour
5. country code
6. password --+
7. BASIC prompt | (one of 6 to 8)
8. title caption --+

6.2 Structure of the CLOCK-IC

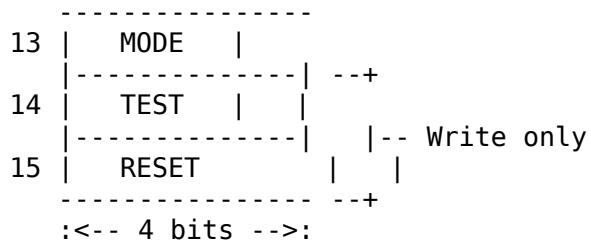
The CLOCK-IC has four blocks inside as shown in Figure 5.25. Each block consists of 13 sets of 4-bit registers, which are specified by addresses from 0 to 12. In addition, it has three 4-bit registers for selecting the block or controlling functions; they are specified by the addresses from 13 to 15.

The registers inside the block (#0 to #12) and the MODE register (#13) can be read from and written to. The TEST register (#14) and RESET register (#15) can only be written to.

Figure 5.25 Clock IC structure



:<-- 4 bits -->: :<-- 4 bits -->: :<-- 4 bits -->: :<-- 4 bits -->:



6.3 MODE Register Functions

The MODE register has the following 3 functions:

* Selecting block

To read from or write to registers from #0 to #12, select the block to be used and then access the objective address. The 2 low order bits of the MODE register are used to select the block.

Registers from #13 to #15 are accessible whichever block is selected.

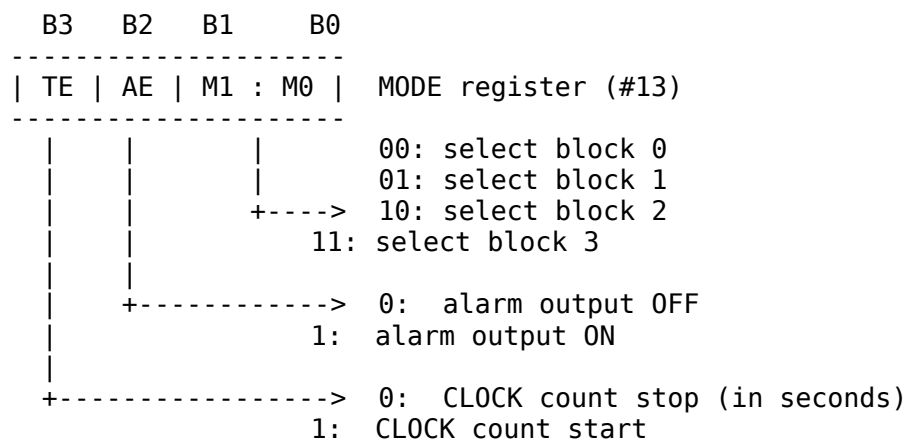
* Alarm output ON/OFF

To switch the alarm input ON/OFF, use bit 2 of the MODE register. Since the standard MSX2 does not support the alarm, modifying this bit causes nothing to happen in general.

* Terminating CLOCK count

By writing "0" in bit 3 of the MODE register, the count in seconds is stopped (the stages before the seconds are not stopped) and the clock function is terminated. By writing "1" in bit 3, the count is resumed.

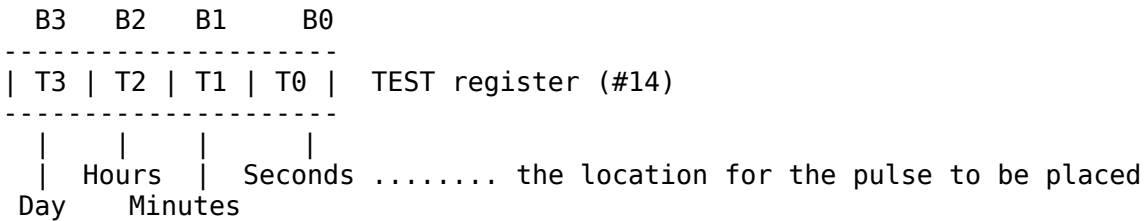
Figure 5.26 MODE register functions



6.4 TEST Register functions

The TEST register (#14) is used to increment the upper counter quickly and to confirm that date and time carries are done correctly. Setting "1" in each bit of the register, the pulse of 2^{14} (=16384)[Hz] is directly set in day, hour, minute, and second counters.

Figure 5.27 TEST register functions



6.5 RESET Register Functions

The RESET register (#15) has the following functions:

* Resetting the alarm

Setting "1" in bit 0 causes all alarm registers to be reset to 0.

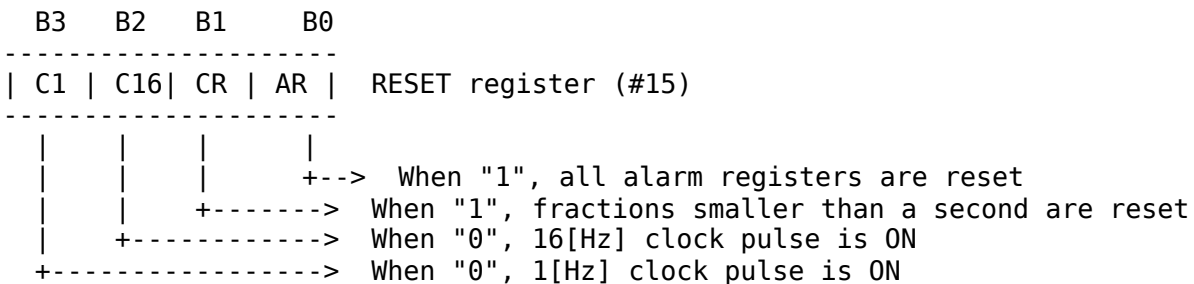
* Setting the seconds

Setting "1" in bit 1 causes the stage before the seconds to be reset. Use this function to set the seconds correctly.

* Clock pulse ON/OFF

Setting "1" in bit 2 turns the 16Hz clock pulse output ON, and setting "0" in bit 3 turns the 1Hz clock pulse output ON. Note that both are not supported by the MSX2 standard.

Figure 5.28 RESET register function



6.6 Setting the Clock and Alarm

* Setting date and time

Block 0 is used to set the clock. Selecting block 0 in the MODE register and writing data in the objective register causes the date and the time to be set. The current time is acquired by reading the contents of the register. See Figure 5.29 for the meaning of the register and its address.

Block 1 is used to set the alarm. Note that the time of the alarm can be set only in days, hours, and minutes. Nothing happens, in general, when the time of the clock meets the time of the alarm.

In the clock, the year is represented by 2 digits (registers #11 and #12). In MSX-BASIC, the 2 low order digits of the year is represented by adding the offset 80 to this value. For example, after setting register #11 to 0 and register #12 to 0, the year would be 80, as "80/XX/XX", when the date is read by using the GET DATE instruction of BASIC.

The day of the week is represented by 0 to 6. This is only a mod 7 counter which is renewed along with the date, and the correspondence between the actual day of the week and the number value 0 to 6 is not defined.

Figure 5.29 Setting the CLOCK and ALARM

block 0 : CLOCK

		B3	B2	B1	B0
0	Seconds (the 1st decimal place)	X X	X	X	
1	Seconds (the 2nd decimal place)	. X	X	X	
2	Minutes (the 1st decimal place)	X X	X	X	
3	Minutes (the 2nd decimal place)	. X	X	X	
4	Hours (the 1st decimal place)	X X	X	X	
5	Hours (the 2nd decimal place)	. .	X	X	
6	Day of the week	. X	X	X	
7	Day (the 1st decimal place)	X X	X	X	
8	Day (the 2nd decimal place)	. .	X	X	
9	Month (the 1st decimal place)	X X	X	X	

10	Month (the 2nd decimal place)	.	.	.	X
11	Year (the 1st decimal place)	X	X	X	X
12	Year (the 2nd decimal place)	X	X	X	X

block 1 : ALARM

		B3	B2	B1	B0
0	_____
1	_____
2	Minutes (the 1st decimal place)	X	X	X	X
3	Minutes (the 2nd decimal place)	.	X	X	X
4	Hours (the 1st decimal place)	X	X	X	X
5	Hours (the 2nd decimal place)	.	.	X	X
6	Day of the week	.	X	X	X
7	Day (the 1st decimal place)	X	X	X	X
8	Day (the 2nd decimal place)	.	.	X	X
9	_____
10	12 or 24 hours	.	.	.	X
11	Leap year counter	.	.	X	X
12	_____

Bits indicated by an "." are always 0 and cannot be modified.

* Selecting 12-hour clock/24-hour clock

Two clocks can be selected; one is a 24-hour clock which represents one o'clock in the afternoon as 13 o'clock, and the other is a 12-hour clock which represents it as 1 p.m. Register #10 is used to select between them. As shown in Figure 5.30, the 12-hour clock is selected when B0 is "0" and the 24-hour clock when B0 is "1".

Figure 5.30 Selecting 12-hour clock/24-hour clock

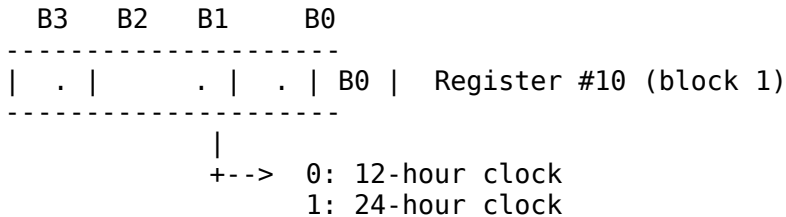
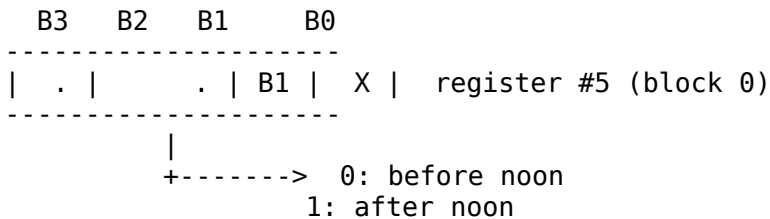


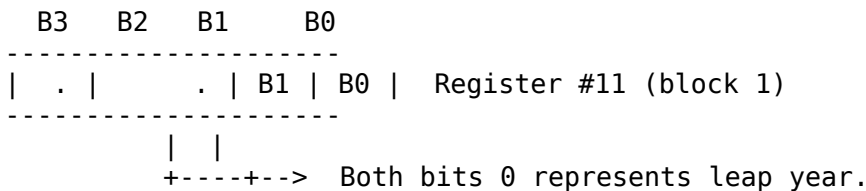
Figure 5.31 Morning/afternoon flag for 12-hour clock



* Leap year counter

Register #11 of block 1 is a mod 4 counter which is renewed along with the count of the year. When the 2 low order bits of this register are 00H, that is considered as a leap year and 29 days are counted in February.

Figure 5.32 Leap year determination



6.7 Contents of the Battery-powered Memory

Blocks 2 and 3 of the CLOCK-IC are used as the battery-powered 4-bit x 13 memory blocks. MSX2 uses this area as shown below.

* Contents of block 2

Figure 5.33 Contents of block 2

	B3	B2	B1	B0
0	ID			
1	Adjust X (-8 to +7)			
2	Adjust Y (-8 to +7)			
3	_____	_____	Interlace mode	Screen mode
4	WIDTH value (Lo)			
5	WIDTH value (Hi)			
6	Foreground color			
7	Background color			
8	Border color			
9	Cassette speed	Printer mode	Key click	Key ON/OFF
10	BEEP tone		BEEP volume	
11	_____	_____	Title colour	
12	Native code			

* Contents of block 3

Block 3 has three functions, depending on the contents of the ID value (register #0). Figure 5.34 shows the functions.

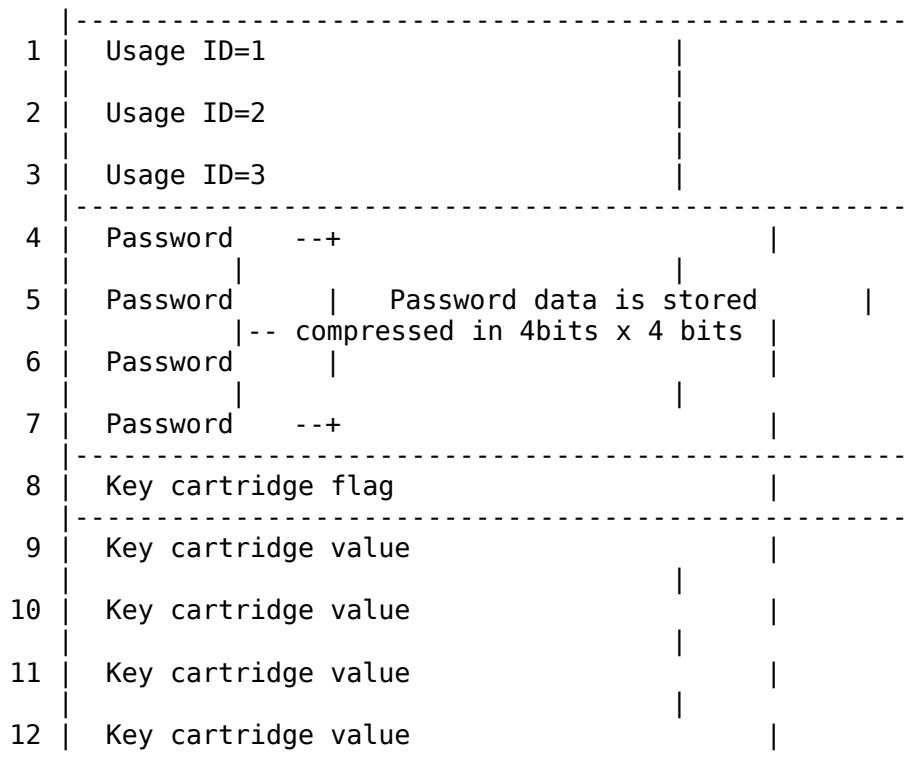
Figure 5.34 Contents of block 3

ID=0: displays the title (within 6 characters) on the initial screen

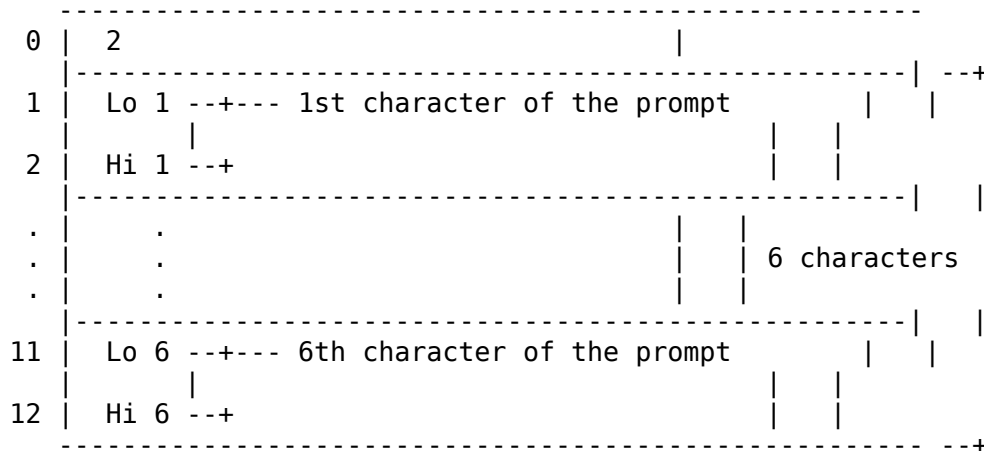
0	0	
1	Lo 1 ---+---	1st character of the title
2	Hi 1 ---+	
.	.	
.	.	6 characters
.	.	
11	Lo 6 ---+---	6th character of the title
12	Hi 6 ---+	

ID=1: sets the password

0	1
---	---



ID=2: sets the prompt on BASIC



6.8 Access to the CLOCK-IC

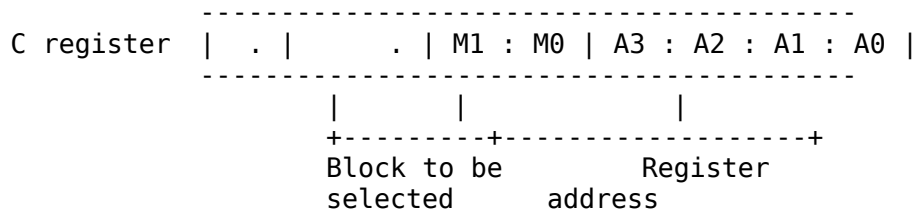
The following BIOS routines are offered to access the clock and the battery-powered memory. Since these routines reside in SUB-ROM, they are called by using the inter-slot call.

* REDCLK (015FH/SUB) read CLOCK-IC data

Input: C <-- CLOCK-IC address (see Figure 5.35)
Output: A <-- data obtained (only 4 low order bits valid)
Function: reads CLOCK-IC register in the address specified by the C register and stores in the A register. Since the address specification includes the block selection information as

shown in Figure 5.35, it is not necessary to set the MODE register and then read the objective register.

Figure 5.35 CLOCK-IC register specification method



* WRTCLK (01F9H/SUB) write CLOCK-IC data

Input: C <-- CLOCK-IC address (see Figure 5.35)

A <-- data to be written (4 low order bits)

Output: ---

Function: write the contents of the A register in the CLOCK-IC at the address specified by the C register. The address is specified in the format shown in Figure 5.35 as REDCLK.

List 5.10 shows an example of this BIOS routine.

List 5.10 Setting the prompt

```

=====
;*****
;
; List 5.10 set prompt message
;
;*****
;
WRTCLK: EQU 01F9H
EXTROM: EQU 015FH

        ORG 0B000H

;----- program start ----- ;Note: Set prompt message for BASIC.

START:  LD C,00110000B ;address data
        LD A,2 ;ID := prompt mode
        CALL WRTRAM ;write to back-up RAM

        LD B,6 ;loop counter
        LD HL,STRING ;prompt data
L01:   LD A,(HL) ;read string data
        AND 0FH ;A := hi 4 bit
        INC C ;increment address
        CALL WRTRAM ;write data to back-up RAM
        LD A,(HL)
        RRCA
        RRCA
        RRCA
  
```



```
RRCA
AND 0FH
INC C ;increment address
CALL WRTRAM ;write low 4 bits
INC HL
DJNZ L01
RET
```

;----- write data to back-up RAM -----

```
WRTRAM: PUSH HL
        PUSH BC
        LD IX,WRTCLK
        CALL EXTRM ;use interslot call
        POP BC
        POP HL
        RET
```

;----- string data -----

```
STRING: DB 'Ready?'
```

```
END
```

=====