

Dreamcast (VMU) Visual Memory Unit

Table of Contents

Dreamcast VMU Specifications	i
VMU Specifications	VMU-1
Overview	VMU-1
VMU Overview	VMU-1
VMU Configuration	VMU-3
VMU Functions	VMU-6
Mode Settings	VMU-8
File Management	VMU-9
Management Area	VMU-10
Data Area	VMU-11
Reserved Area	VMU-11
LCD Display	VMU-11
XRAM	VMU-11
Screen Mode	VMU-11
Icons	VMU-12
Screen Configuration	VMU-12
LCD Characteristics	VMU-12
Miscellaneous	VMU-13
Executable File Initiation	VMU-13
Downloading an Executable File	VMU-13
File Size	VMU-13
Subroutine	VMU-14
Interrupts	VMU-14
RAM	VMU-14
Save Processing During Executable File Operations	VMU-14
Auto Power Off	VMU-14

Communications Function	VMU-15
Maple Bus Protocol	VMU-15
Synchronous Serial Communications	VMU-15
Clock Function	VMU-15
Settings	VMU-15
Alarm Function	VMU-16
SLEEP Function	VMU-16
SLEEP Operation	VMU-16
Buttons	VMU-16
Batteries	VMU-17
Battery Life	VMU-17
Processing When Battery Power Is Exhausted	VMU-17
Battery Replacement	VMU-17
Postscript	VMU-18
 Visual Memory Unit (VMU) Hardware Manual	 VMU-i
 1. Overview	 VMD-1
General	VMD-1
Features	VMD-1
System Block Diagram	VMD-5
 2. Internal System Configuration	 VMD-7
Memory Space	VMD-7
Program Counter (PC)	VMD-8
Internal Program ROM	VMD-10
Internal Data Memory	VMD-10
Flash Memory	VMD-15
Accumulator	VMD-16
B Register, C Register	VMD-16
Program Status Word	VMD-17
Stack Pointer	VMD-19
The Table Reference Register (TRR)	VMD-20
CHANGE Instruction	VMD-21

3. Peripheral System Configuration	VMD—23
Input/Output Ports	VMD—23
Port 1	VMD—25
Port 3	VMD—31
Port 7	VMD—34
Timer/Counter 0 (T0)	VMD—35
Timer 1 (T1)	VMD—51
Base Timer	VMD—67
Serial Interface	VMD—73
Dot Matrix LCD Controller/Driver	VMD—94
External Interrupt Function	VMD—102
Port Interrupt Functions	VMD—109
VMU Work RAM	VMD—115
Flash EEPROM	VMD—118
 4. Control Functions	 VMD—121
Interrupt Function	VMD—121
Types of Interrupts	VMD—122
Interrupt Function Operation	VMD—123
Circuit Configuration	VMD—124
Related Registers	VMD—124
Interrupt Priority Ranking	VMD—127
System Clock Generation Function	VMD—129
Overview	VMD—129
Functions	VMD—129
Circuit Configuration	VMD—130
Related Registers	VMD—132
System Clock Operation Mode	VMD—135
Standby function	VMD—137
Overview	VMD—137
4.3.2. Related Registers	VMD—137
Operating Statuses When in Standby	VMD—138
HALT Mode	VMD—139
HOLD Mode	VMD—140
Reset Function	VMD—141
Overview	VMD—141
Function	VMD—141
Hardware Status During a Reset	VMD—141

5. Instructions	VMD-145
Instruction Overview	VMD-145
Arithmetic Operation Instructions	VMD-146
Logical Operation Instructions	VMD-147
Data Transfer Instructions	VMD-147
Jump Instructions	VMD-147
Conditional Branching Instructions	VMD-147
Subroutine Instructions	VMD-147
Bit Manipulation Instructions	VMD-147
Miscellaneous Instruction	VMD-147
Macro Instruction	VMD-148
Addressing	VMD-148
Program Memory (ROM) Addressing	VMD-148
Data Memory (RAM) and Special Function Register (SFR) Addressing	VMD-150
Arithmetic Operation Instructions	VMD-153
Logical Operation Instructions	VMD-171
Data Transfer Instructions	VMD-184
Jump Instructions	VMD-195
Conditional Branching Instructions	VMD-199
Subroutine Instructions	VMD-212
Bit Manipulation Instructions	VMD-217
Miscellaneous Instruction	VMD-220
Macro Instruction	VMD-220
 Visual Memory Unit (VMU) Programing Manual	 VMC-i
 1. Environment Variables	 VMC-1
Environment Variables for the L86K Series	VMC-1
Setting the Environment Variables (MS-DOS Version)	VMC-2
Setting the Environment Variables (UNIX Version)	VMC-2
 2. File Specification for the Assembler	 VMC-3
File Name Specification	VMC-3
MS-DOS Version File Specification	VMC-3
UNIX Version File Specification	VMC-4
Specifying Parameters through the Command Line	VMC-4
Specifying Parameters in Response to Prompts	VMC-5
 3. Assembler Option Specification	 VMC-7
Specification for Upper- & Lower-case Letters in Identifiers	VMC-7
Specification for Outputting Debugging Information	VMC-7
Specification for Not Optimizing Branching Instructions	VMC-8
Specification for Suppressing the Copyright Notice	VMC-8

Reserved Word File Specification	VMC-8
Work Buffer Size Specification	VMC-9
Option List Display	VMC-9
4. Environment Variables and the Reserved Word File	VMC-11
Environment Variables VMC-11	
Setting the Environment Variables (MS-DOS Version)	VMC-12
Setting the Environment Variables (UNIX Version)	VMC-12
Reserved Word File	VMC-12
5. Source File Input Format	VMC-13
Statements	VMC-13
Label Names and Symbol Names	VMC-14
Comments	VMC-14
Operators	VMC-15
Numeric constants	VMC-16
Character Constants	VMC-17
Character String Constant	VMC-18
Special Symbols	VMC-18
6. Errors	VMC-19
Warnings	VMC-20
Errors	VMC-22
Fatal Errors	VMC-27
7. Pseudo Instructions	VMC-31
ORG (Specify origin)	VMC-33
ORG expression	VMC-33
WORLD (Select ROM for code storage)	VMC-34
WORLD selection	VMC-34
CSEG (Declare start of code segment)	VMC-34
CSEG mode	VMC-34
DSEG (Declare start of data segment)	VMC-35
DESG	VMC-35
END (End program)	VMC-36
END	VMC-36
PUBLIC (Specify external definition name)	VMC-37
PUBLIC symbol [, symbol]	VMC-37
EXTERN (Specify external reference name)	VMC-38
EXTERN [segmanet:]symbol {[segment:]symbol}	VMC-38
OTHER_SIDE_SYMBOL (Declare CHANGE instruction jump label)	VMC-38
OTHER SIDE SYMBOL label [,label]	VMC-38

EQU (Assign value)	VMC-39
symbolname EQU expression	VMC-39
SET (Assign temporary value)	VMC-40
symbolname SET expression	VMC-40
DB (Define byte data)	VMC-41
labelname DB expression {,expression}	VMC-41
DW (Define word data)	VMC-42
labelname DW expression {,expression}	VMC-42
DC (Define character string data)	VMC-43
labelname DC "string"	VMC-43
DS (Define byte area)	VMC-44
labelname DS absolute_expression	VMC-44
MACRO (Define macro)	VMC-45
name MACRO parameter {, parameter}	VMC-45
REPT (Repeat macro)	VMC-47
REPT count	VMC-47
IRP (Continuous macro)	VMC-48
IRP parameter, argument {,argument}	VMC-48
IRPC (Character string macro)	VMC-49
IRPC parameter, string	VMC-49
ENDM (End macro definition)	VMC-50
ENDM	VMC-50
EXITM (Interrupt macro expansion)	VMC-51
EXITM	VMC-51
LOCAL (Define local label)	VMC-52
LOCAL name {, name}	VMC-52
IFDEF (Assemble if defined)	VMC-54
IFDEF symbol	VMC-54
IFNDEF (Assemble if undefined)	VMC-55
IFNDEF symbol	VMC-55
IFB (Assemble if operand is empty)	VMC-56
IFB <argument>	VMC-56
IFNB (Assemble if operand is not empty)	VMC-57
IFNB <argument>	VMC-57
IFE (Assemble if value of expression is "0")	VMC-58
IFE expression	VMC-58
IFNE (Assemble if value of expression is not "0")	VMC-59
IFNE expression]	VMC-59
IFIDN (Assemble if two character strings are identical)	VMC-60
IFIDN <string1>, <string2>	VMC-60
IFDIF (Assemble if two character strings are not identical)	VMC-61
IFDIF <string1>, <string2>	VMC-61
ELSE (Assemble in case of condition opposite of the above IF condition)	VMC-61
ELSE	VMC-61

ENDIF (End conditional assembly)	VMC-61
ENDIF	VMC-61
PRINTX (Display on VDT during assembly)	VMC-62
.PRINTX "string"	VMC-62
LIST (Output list)	VMC-63
.LIST	VMC-63
.XLIST (Interrupt list output)	VMC-64
.XLIST	VMC-64
.MACRO (Output macro expansion)	VMC-64
.MACRO	VMC-64
.XMACRO (Interrupt macro expansion output)	VMC-64
.XMACRO	VMC-64
.IF (Output conditional skip)	VMC-65
.IF	VMC-65
.XIF (Interrupt conditional skip output)	VMC-66
.XIF	VMC-66
INCLUDE (Load file)	VMC-66
INCLUDE filename	VMC-66
TITLE (Specify list title)	VMC-67
TITLE string	VMC-67
PAGE (End of page)	VMC-68
PAGE	VMC-68
CHIP (Define chip that is target of assembly)	VMC-69
CHIP chipname	VMC-69
COMMENT (Output comments to object file)	VMC-69
COMMENT comment_string	VMC-69
WIDTH (Specify number of columns in list file)	VMC-70
WIDTH number	VMC-70
BANK (Specify RAM area bank)	VMC-71
BANK expression	VMC-71
CHANGE (Jump between external and internal ROM)	VMC-72
CHANGE symbol	VMC-72
RADIX (Specify default base)	VMC-72
RADIX expression	VMC-72
JMPO (Generate optimal JMP instruction)	VMC-73
JMPO expression	VMC-73
BRO (Generate optimal BR instruction)	VMC-74
BRO expression	VMC-74
CALLO (Generate optimal CAL instruction)	VMC-75
CALLO expression	VMC-75
BZO (Generate BZ instruction that will not generate an address error)	VMC-75
BZO expression	VMC-75
BNZO (Generate BNZ instruction that will not generate an address error)	VMC-76
BNZO expression	VMC-76

BPO (Generate BP instruction that will not generate an address error)	VMC-76
BPO expression	VMC-76
BPCO (Generate BPC instruction that will not generate an address error)	VMC-77
BPCO expression	VMC-77
BNO (Generate BN instruction that will not generate an address error)	VMC-77
BNO expression	VMC-77
DBNZO (Generate DBNZ instruction that will not generate an address error)	VMC-78
DBNZO expression	VMC-78
BEO (Generate BE instruction that will not generate an address error)	VMC-78
BEO expression	VMC-78
BNEO (Generate BNE instruction that will not generate an address error)	VMC-79
BNEO expression	VMC-79
 8. List File Format	 VMC-81
 9. Specifying Files for Linking	 VMC-85
File Name Specification	VMC-85
MS-DOS Version File Specification	VMC-85
UNIX Version File Specification	VMC-86
Specifying Parameters Through the Command line	VMC-87
Specifying Parameters in Response to Prompts	VMC-88
Default Responses	VMC-89
Files Referenced During Linking	VMC-89
 10. Specifying Linkage Loader Options	 VMC-91
Creating a HEX File for LC868000 Series External ROM	VMC-91
CSEG Loading Address Specification Method	VMC-91
DSEG Loading Address Specification Method	VMC-92
Enabling Duplicate Definition of DSEG Addresses	VMC-92
No Distinction Between Upper-Case and Lower-Case	VMC-92
Creating the Loading Map	VMC-92
Creating a Local Symbol List	VMC-93
Specifying Warning Messages Concerning Operand Data	VMC-94
CSEG FREE Block Optimized Loading	VMC-94
Specifying Symbol Sort Processing	VMC-95
 11. Object Placement	 VMC-97
 12. Errors	 VMC-101
Fatal Errors	VMC-101
Non-Fatal Errors	VMC-102

13. Program Startup	VMC-103
File Name Specification	VMC-103
MS-DOS Version File Specification	VMC-103
UNIX Version File Specification	VMC-104
Specifying Parameters Through the Command line	VMC-104
Option Specification	VMC-106
Command Line Execution Examples	VMC-106
Operation Using the Prompts	VMC-107
Prompt Line Expansion	VMC-107
Default Response	VMC-107
14. Errors	VMC-109
15. Cross-Reference List	VMC-111
16. Program Startup	VMC-113
File Name Specification	VMC-113
Parameter Specification Method	VMC-113
Option Specification	VMC-114
17. Errors	VMC-115
Fatal Errors	VMC-115
Visual Memory Unit (VMU) VMU-BIOS Specifications	VMB-i
1. VMU-BIOS Specifications	VMB-1
Outline	VMB-1
VMU Outline	VMB-1
System-BIOS Outline	VMB-1
Memory Space	VMB-2
System BIOS Functions	VMB-4
System BIOS Data and Memory Allocation	VMB-5
Program Layout	VMB-5
System programs	VMB-5
OS programs	VMB-6
Header	VMB-6
Subroutine Call Flow	VMB-6
Returning From User Program to Mode Selection Screen	VMB-8
VMU Initialization	VMB-9

Subroutine Description	VMB-11
Flash Memory Access Functions	VMB-11
Precautions for Using Flash Memory Access Subroutines	VMB-11
Flash Memory Page Data Readout	VMB-13
Writing to Flash Memory	VMB-15
Flash Memory Verify	VMB-17
Clock Function	VMB-19
Clock Countup Timer	VMB-19
Automatic low battery detection function	VMB-20
Automatic low battery detection flag	VMB-20
Visual Memory Unit (VMU) Sound Development Specifications	VMA-i
 1. VMU Sound Development Specifications	 VMA-1
VMU Sound Output Hardware Outline	VMA-1
Sound Output Principle	VMA-1
Timer 1 Outline	VMA-2
Timer 1 Block Configuration	VMA-2
Related Registers	VMA-3
Mode Setting	VMA-4
8-Bit Counter Mode	VMA-5
Output Waveform and Parameter Settings	VMA-5
8-Bit Counter Mode Setting	VMA-6
Frequency Response Characteristics	VMA-7
Table of Available Output Frequencies	VMA-8
Sample Program	VMA-13
Table of Defined Variables	VAP-1
VMU Mode Selection Operation Flow	VAP-2

Dreamcast VMU Specifications

1. VMU Specifications

1 Overview

This document describes the VMU, a peripheral device for the next-generation game system KATANA (Dreamcast).

1.1 VMU Overview

The VMU (Visual Memory Unit) is a memory cartridge that not only stores data, but also includes an LCD display that visually expresses that data.

The VMU connects to KATANA's (preliminary name) special controller, called "SEED" (preliminary name), and can be used to display subscreens during a game and as a memory card that stores game data files.

The VMU can be connected or disconnected while the game machine is on.

When not connected to a controller, the data files stored in the VMU's memory can be displayed and deleted. Files can also be copied from one VMU to another by connecting two VMUs to each other.

Furthermore, by downloading special executable files (programs) from KATANA, the VMU becomes a compact portable game player; two-player games are also possible.

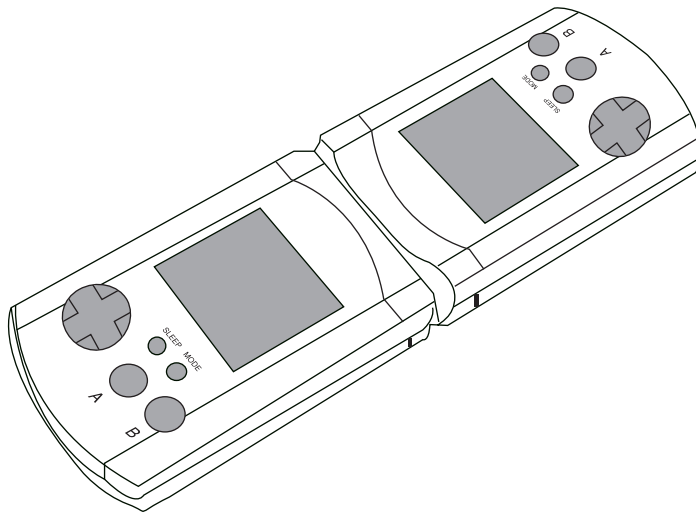


Figure 1.1 *Conceptual Image of the VMU*

In the top portion of Fig. 1.1, two VMUs are shown connected to each other as they exchange data.

1.2 VMU Configuration

This section describes the VMU configuration.

• Potato Chip (custom IC for the VMU)					
	Core CPU:	8 bits:	Instruction cycle time: When connected to game machine = 1[micro]s When operating on standalone basis = 183[micro]s Note: Operation on a standalone basis is extremely slow in order to minimize battery power consumption.		
	Memory:	Mask-ROM:	16Kbyte	System-BIOS IPL	
	:	Flash-EEPROM:	64K	Program code/data area	
	:	:	64K	Data area (of which 28K are reserved for the system)	
	:	RAM	512 bytes	General purposes (of which 256 bytes are reserved for the system)	
	:	:	512bytes	I/O mapping (can also be used as a Maple buffer)	
	:	LCD RAM	Bank 1	96 bytes	
			Bank 2	96 bytes	
			Bank 3	6 bytes (for icons; used by the system)	
	Serial I/F:	Uses the following interfaces exclusively:			
		Maple:	LM-Bus		
		Synchronous SIO:	Two 8-bit serial interfaces		
	Timer:	16bit	For Clock		
		16bit(or 8bit x2):	General purpose; of these, 8 bits are used exclusively for pulse generator output for alarms		
	I/O Port:	Input/output:	16 pins (buttons, serial interfaces)		
		Input:	4 pins (control pins)		
	LCD-Driver Controller:	Common:	33 pins		
		Segment:	48 pins		
• LCD:		LCD:	32 (V) x 48 (H) dots: Monochrome binary		
		Icons:	4 types (File, Game, Time, Attention: used by system)		
• Buzzer:		Voltage buzzer:	For alarms		
• Power supply:		Button batteries:	CR2032 x 2		
		External inputs:	+5V +3.3V		
		External outputs:	+3.3V		
• Buttons:		6 buttons:	Four-direction key, A button, B button, Mode button, Suspend button, SLEEP button		
• Communications connector:		14 pins:	Serial interface, power supply, control Connected to controller, another VMU, etc.		

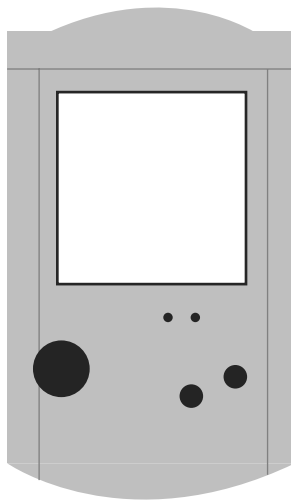


Figure 1.2 *External View (preliminary)*

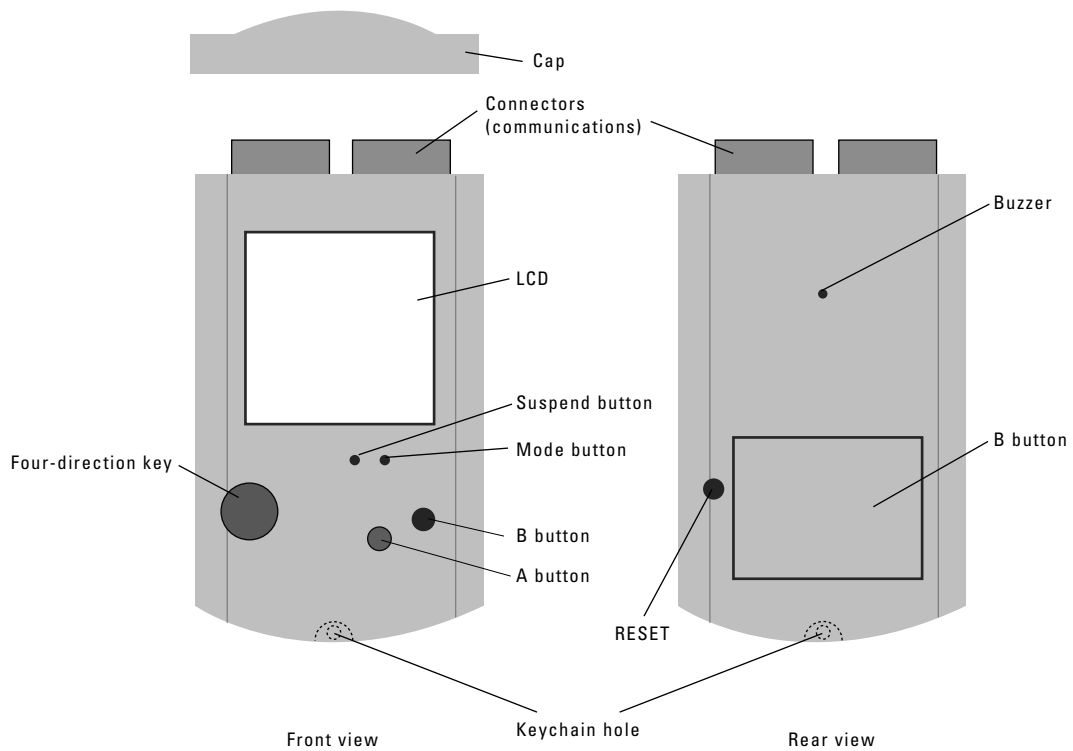


Figure 1.3 *External Appearance and configuration (preliminary)*

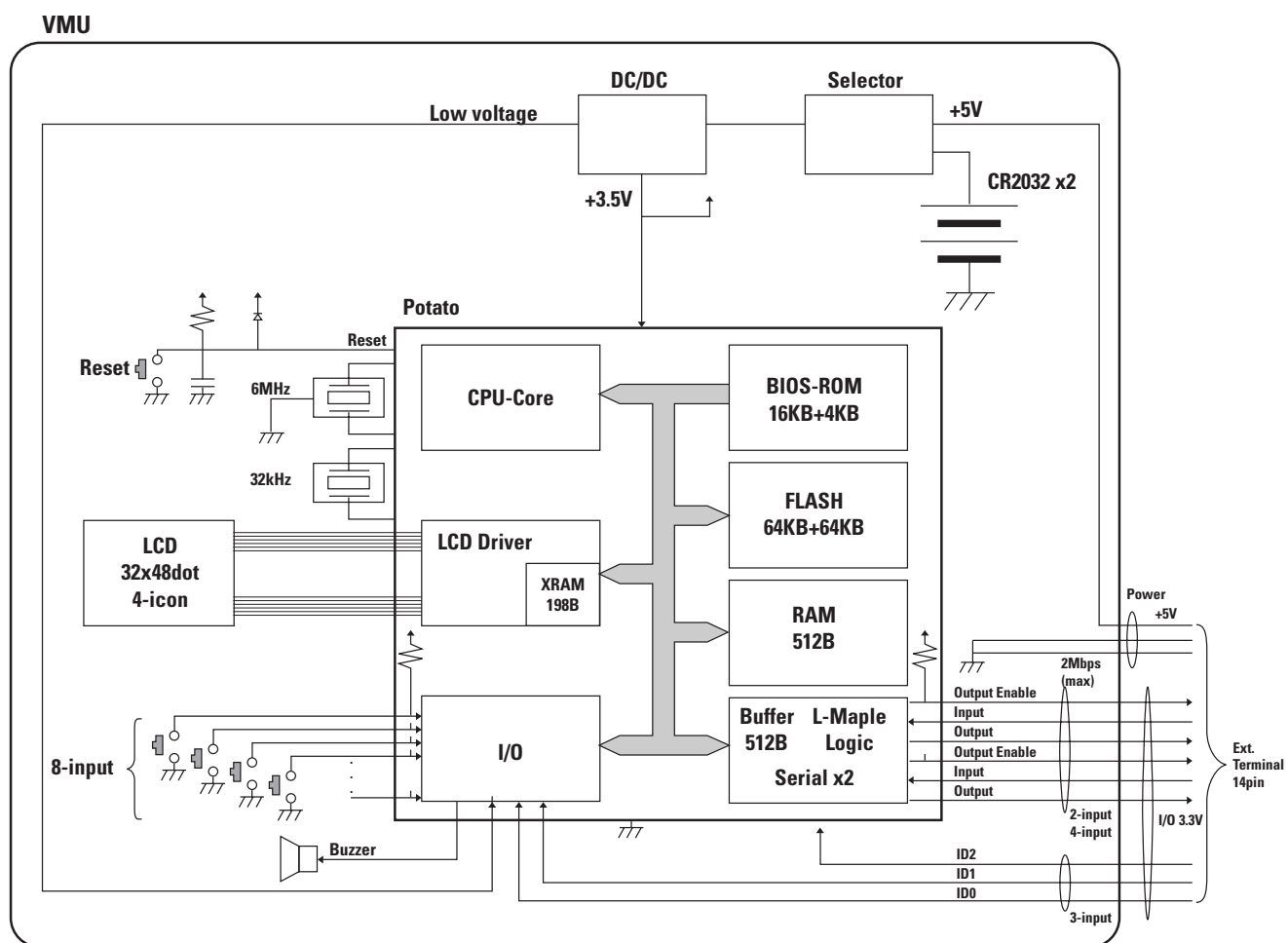


Figure 1.4 Block Diagram (preliminary)

1.3 VMU Functions

When connected to a game machine, the VMU conforms with the Maple Bus 1.0 Standard Specifications, and supports the following function types.

- 1 FT₁ Storage Function
- 2 FT₂ B/W LCD Function
- 3 FT₃ Timer Function

Accordingly, the Function Type (FT) is “00h-00h-00h-0Eh”. (FD1 = FT3, FD2 = FT2, FD3 = FT1)

For details, refer to the specifications for each function. An overview of the System-BIOS functions included in the VMU is provided below.

1) File management

This function manipulates and manages backup files and program files.

Files are managed in 1-block units (512 bytes), and reads and writes are also performed in block units. FAT operations and file information processing use subroutines in the System-BIOS. For details on file management methods, refer to section 3, “File Management.”

2) LCD display

When the VMU is connected to a game machine, this function only draws graphics (transferring screen image data).

This function conforms with the data format that is stipulated in the Maple Bus Function Type specifications, and sends graphics images from the game machine to the VMU in accordance with the VMU screen configuration, and then BIOS transfers the resulting image to the LCD display RAM (XRAM).

The amount of data required for one screen is 32 dots (V) x 48 dots (H) = 1536 bits = 192 bytes.

When the VMU is operating on a standalone basis, this function handles the drawing of graphics. The icons display the operation mode of the VMU.

File	File management
Game	Executable file initiation
Time	Time display
Attention	Memory access in progress

3) Executable file initiation

This function initiates execution of an executable file (program) that was downloaded from a game machine.

This function can only be executed while the VMU is operating on a standalone basis. A program can not be initiated while the VMU is connected to a game machine.

A number of functions that can be provided for executable files are System-BIOS subroutines and can be used by the executable file simply by calling the subroutine.

4) Communications

When the VMU is connected to a game machine, communications are handled according to the Maple Bus protocol.

When the VMU is operating on a standalone basis, the VMU supports 8-bit synchronous serial communications for exchanging data with another VMU.

This function is also provided as a subroutine for executable files. (Not finalized)

5) Clock

This function uses a timer to measure time.

This function is always operating, whether the VMU is connected to a game machine or is operating on a standalone basis.

6) Alarm

This function sounds a buzzer by means of a pulse generator. This function is also provided as a subroutine for executable files. (Not finalized)

This function conforms with the data format that is stipulated in the Maple Bus Function Type specifications, and when the VMU is connected to a game machine, this function allows the game machine to sound the buzzer.

7) Mode switching

When the VMU is connected to a game machine, the VMU operation mode can be changed by pressing the mode button.

The mode status is displayed by means of icons.

When the VMU is operating on a standalone basis, the Auto Power Off function can also be used.

8) Character font installation

8 dot (V) x 6 dot (H) alphabet, Katakana, and symbol fonts can be installed in the VMU. These fonts cannot be called and displayed from an executable file for the VMU that was downloaded from a game machine.

When the VMU is connected to a game machine and graphics are being displayed from the game machine side, fonts cannot be used.

Instead, transfer the screen image that is to be displayed as is.

Fonts can only be used by the System-BIOS.

2 Mode Settings

The operating mode of the VMU is determined by the connection status and the mode button.

Table 1.1 *Modes*

Connection Status	Mode Button (Icon Display)	Operating Mode
Connected to game machine	Off	System mode
	Attention	Flash access in progress
Standalone operation	Game	Executable file initiation
	File	File operations
	Time	Clock display
	Attention	Accessing flash memory

1) System mode

This mode is controlled by the System-BIOS' external control program.

This mode handles communications according to the Maple Bus protocol, memory management, LCD display, and timer management.

2) Game mode

In this mode, the System-BIOS initiates executable files in flash memory.

All processing is controlled by the executable file, except for the Maple Bus protocol.

Transitions from this mode to another mode are also controlled by the executable file.

To execute a mode, transmission, the executable file calls a subroutine from the System-BIOS.

At that point, all of the contents of RAM and the registers are saved to flash memory.

Note: This save operation requires approximately 8 seconds.

3) File mode

This mode is controlled by the System-BIOS' file control program.

This mode can display, copy, and delete files in flash memory through button operations.

Refer to other documents for details on the configuration and operation of the file management screen.

4) Time mode

This mode is controlled by the System-BIOS' timer program. This mode can display a digital clock (showing the hours, minutes, and seconds), and can be used to set the time. When the VMU returns from system mode, it enters this mode.

Transitions among the modes occur in response to changes in the connection status and the Mode button + Enter button being pressed.

However, Game mode can suppress changes in the connection status and the Mode button + Enter button being pressed. The mode cannot be changed while data is being written to the flash memory.

Attention is a warning indicator that lights for Read/Write while flash memory is being accessed.

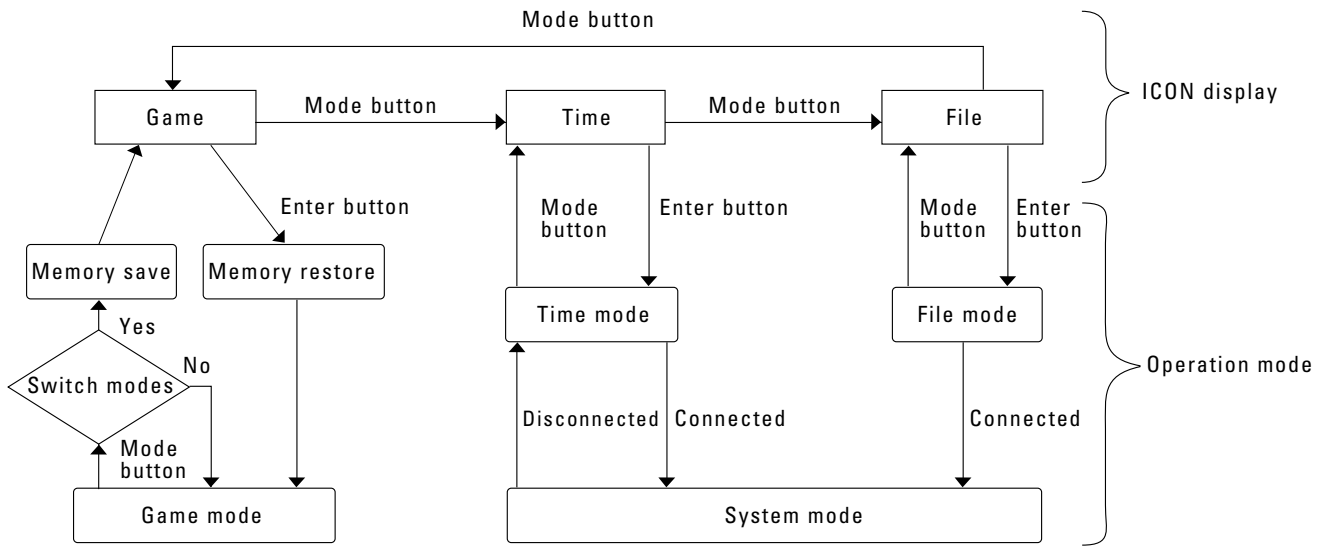


Figure 1.5 Mode Transitions

3 File Management

- File management in the VMU conforms with FT1: Storage Function in the Maple Bus 1.0 Function Type Specifications.
- The size of the VMU flash memory is 128K.
- The minimum read/write unit for a file is one block (512 bytes); the entire flash memory is divided into 256 blocks.

However, because 56 blocks are used as a system management area, the size of the area that can be used to store data is 200 blocks.

One executable file can exist in one partition, with a maximum size of 0080h blocks (64K: block numbers 0000h to 007Fh).

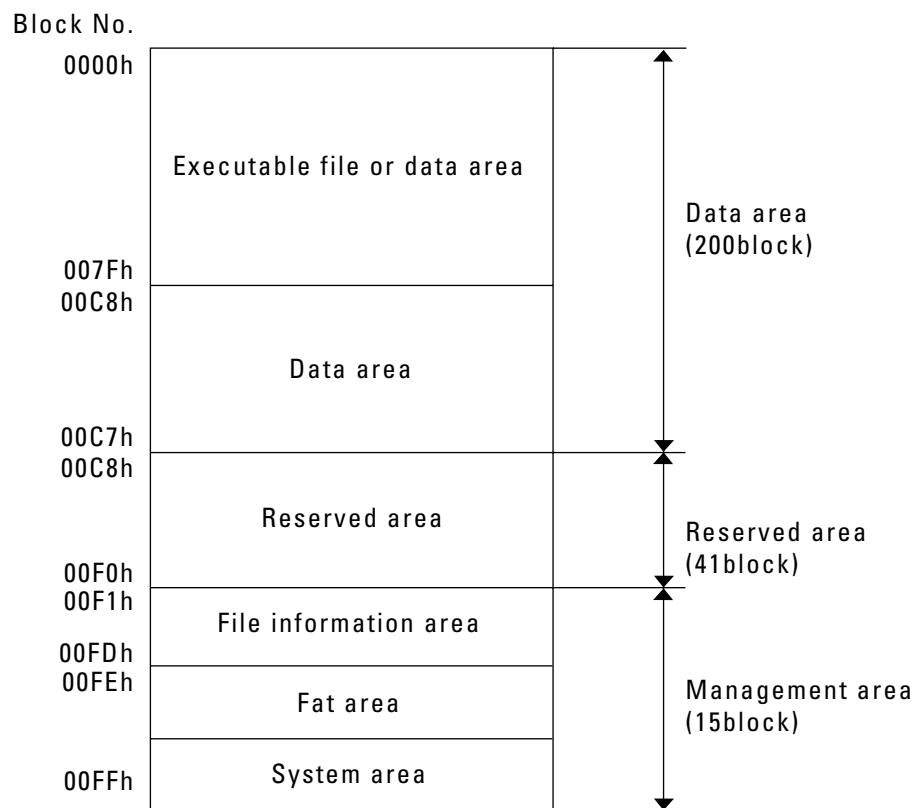


Figure 1.6 Memory Map

3.1 Management Area

- The 15 blocks at the top of memory (starting from block number 00FFh) are used for the management area.
- The management area is divided into three areas: the system area, the FAT area, and the file information area.
- The system area consists of one block, the FAT area consists of one block, and the file information consists of 13 blocks.
- The system area is write-protected, except during formatting.
- The FAT area has a chain structure in which every two bytes (16 bits) controls one block.
- The file information area allocates 32 bytes to each file, and can therefore manage a maximum of 200 files.
- There is only a root directory; no subdirectories are supported.
- File names consist of 12 bytes (ASCII codes representing up to 12 normal-width characters).

3.2 Data Area

- The data area, where data files can be stored, consists of 200 blocks, from block number 0000h to 00C7h.
- Data files are stored starting from 00C7h towards 0000h, while an executable file starts from 0000h.
- The areas from 0000h to 007Fh and from 0080h to 00FFh are controlled through bank switching; switching is performed by the System-BIOS automatically.
- Reading and writing flash memory must always be done by calling the System-BIOS subroutines.

3.3 Reserved Area

This area is used by the System-BIOS and in system mode.

4 LCD Display

- The LCD display in the VMU conforms with FT[2]: B/W LCD Function in the Maple Bus 1.0 Function Type Specifications.
- The LCD that is built into the VMU consists of a 32-dot (V) \times 48-dot (H) dot matrix display, and four icons that indicate the operating mode of the VMU.
- Drawing the LCD is accomplished by storing drawing data in the dedicated drawing RAM.

4.1 XRAM

The LCD's dedicated drawing RAM is called "XRAM."

XRAM consists of three banks; the first and second banks are open to executable files, while the third bank is used by the System-BIOS.

The first bank of XRAM corresponds to the upper half of the LCD (16×48 dots), and the second bank of XRAM corresponds to the lower half of the LCD (16×48 dots).

One dot on the LCD corresponds to one bit in XRAM. One byte of XRAM corresponds to 8 dots in a horizontal row on the LCD, and 6 bytes consist of one entire horizontal row on the LCD.

4.2 Screen Mode

When the VMU is connected to a game machine, the System-BIOS sends drawing data from the game machine directly to the XRAM as a graphics screen.

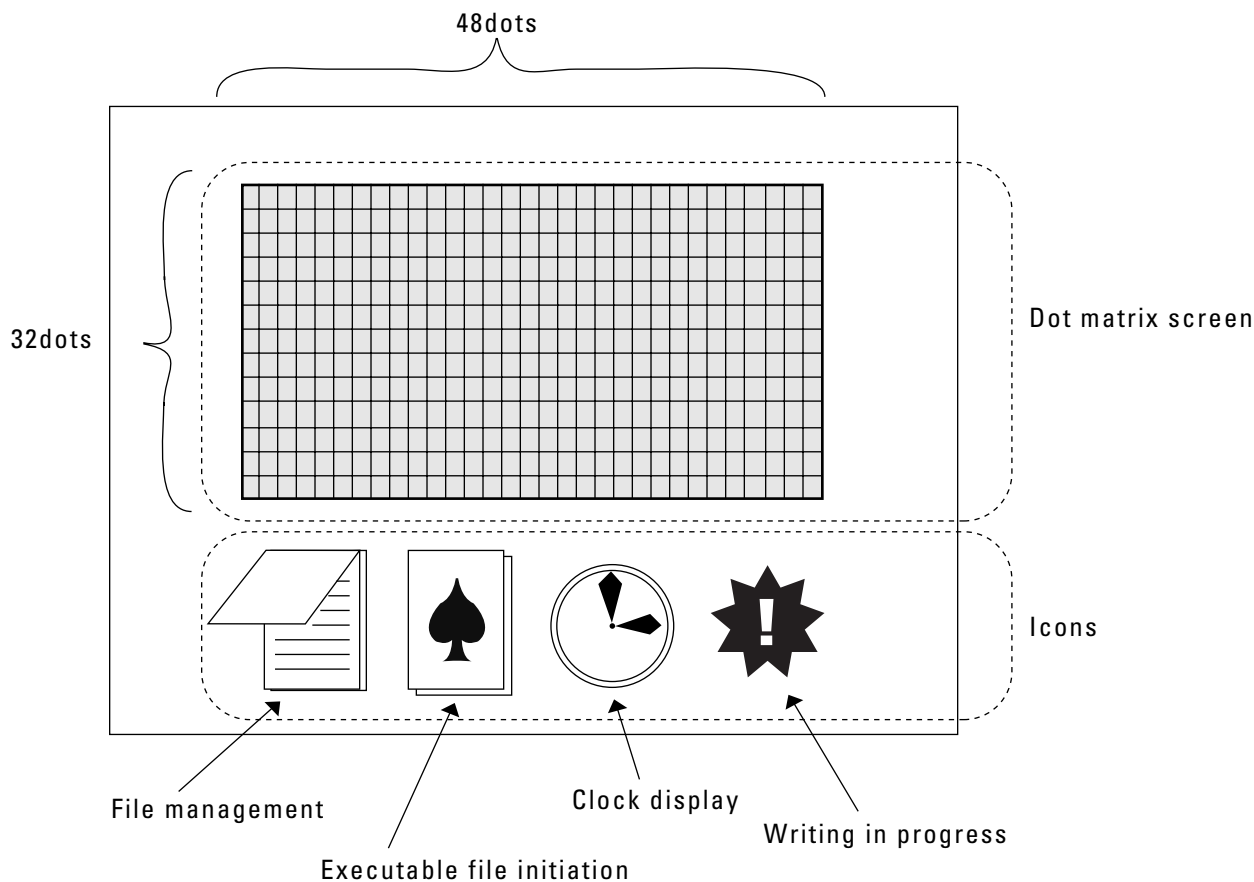
Therefore, when using the VMU's display as a game subscreen, etc., transfer the screen image as is to the VMU.

During standalone operation, the character font in the System-BIOS cannot be used for text display on a graphics screen.

For a graphics screen, write the screen image data as is to XRAM.

4.3 Icons

The System-BIOS uses the icons; use by an executable file is prohibited.



4.4 Screen Configuration

4.5 LCD Characteristics

The screen refresh concept for the LCD display differs from that for a TV.

Once data is transferred to XRAM, it is displayed on the LCD, but only after a delay due to the response speed of the LCD. When the LCD response is delayed, ghosting or flickering may occur, resulting in a display that is difficult to see. In addition, during standalone operation or when connected to a game machine, differences in the operating speeds result in different LCD display speeds. During standalone operation, the display speed is slower.

The recommended refresh rate for the VMU' LCD is 1Hz for standalone operation and 4Hz when connected to a game machine.

4.6 Miscellaneous

- There is no contrast adjustment or brightness adjustment for the LCD.
- There is no backlight for the LCD.
- It is not possible to incorporate a design (such as a picture, etc.) in the polarized panel (the back sheet) with a reflective panel that reflects the light in the LCD.

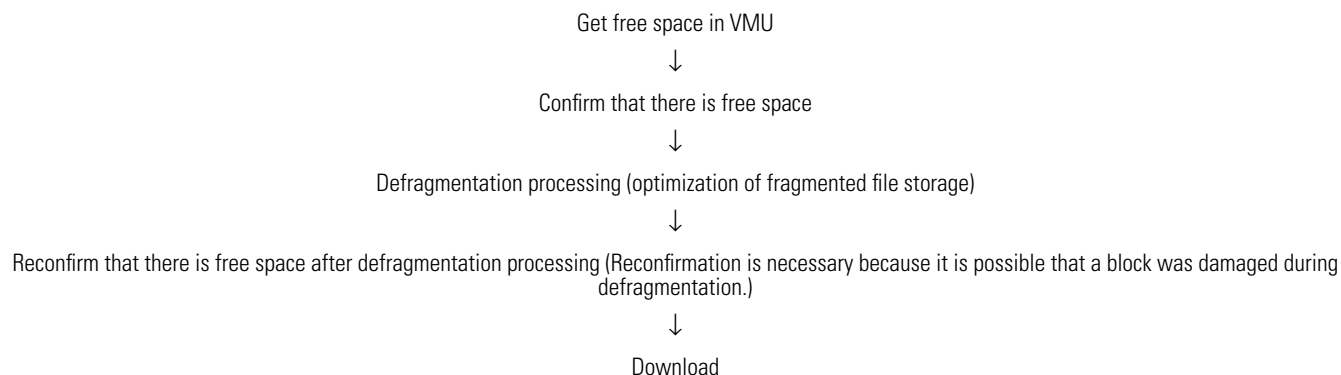
5 Executable File Initiation

- This function initiates an executable file that was downloaded from a game machine.
- The VMU can store and initiate only one executable file at a time.
- The System-BIOS includes subroutines that form that VMU functions. Of these subroutines, several are provided for executable files, and an executable file can call these subroutines.
- Program development of an executable file is performed using a VMU emulator (preliminary) that runs under Windows 95.

5.1 Downloading an Executable File

Executable files are stored in flash memory in the area consisting of block numbers 0000h to 007Fh, starting from the 0000h block. When an executable file is downloaded from a game machine application, confirm that there is contiguous free space starting from the 0000h block of the VMU. Even if the free space has been confirmed, it still will not be possible to download an executable file if there is any other file in the area where the executable file is to be stored (the area from block 0000h to the end of the executable file).

Game machine application processing is as described below:



5.2 File Size

- The maximum size of an executable file is 0080h blocks (64K).

5.3 Subroutine

A list of the available subroutines is shown below. (not finalized)

Each subroutine uses a RAM area (in the general-purpose RAM area) as a work area.

- | | |
|------------------------|---|
| 1) Data communications | :Performs synchronized serial communications. |
| 2) Alarm | :Sounds the buzzer. |
| 3) Flash memory write | :Writes flash memory. |
| 4) Flash memory read | : Reads flash memory. |

5.4 Interrupts

A list of external and internal interrupts is provided below. (planned)

Except for the Mode Change interrupt, these interrupts cannot be masked. (planned)

- 1) Low voltage interrupt
- 2) Timer interrupt
- 3) Mode Change interrupt (maskable)
- 4) SLEEP interrupt

5.5 RAM

The RAM areas that executable files can use are shown below.

General-purpose RAM:	000h to 0FFh (bank 1)
I/O mapping RAM:	000h to 1FFh (Set the address to the specified register and read/write one byte at a time.)
XRAM:	Bank 1, bank 2

5.6 Save Processing During Executable File Operations

Data on the midpoint status of an executable file and parameters for an executable file (such as a game) are saved by writing the data to an area within the executable file. When creating an executable file (such as a game), set aside an area within the file for this purpose. Because FAT processing, etc., is not possible due to the hardware design, such data cannot be saved in a separate file.

In order to link the game machine with an application and then use the saved data from an executable file (such as a game), load the executable file from the VMU to the game machine, and then read that portion of the file that contains the saved data.

5.7 Auto Power Off

- The `Auto Power Off` function puts the VMU into the SLEEP state if no buttons are pressed or no communications are received for two minutes.

This function can be enabled/disabled by executable files.

6 Communications Function

- The VMU is capable of conducting serial communications with other equipment.
- The VMU supports two serial communications protocols: the Maple Bus protocol and full-duplex synchronous serial communications.
- The System-BIOS switches between the Maple Bus protocol in system mode and synchronous serial communications in standalone operation mode.

6.1 Maple Bus Protocol

- When the VMU is connected to a game machine, the communications connector switches to the Maple Bus protocol side.
- The entire I/O mapping RAM becomes a transmission/receive buffer, and the synchronous serial side stops.
- The physical connection with the game machine is made through an LM-Bus connection, and the VMU becomes an expansion device.
- All processing is performed by the System-BIOS; this function is not accessible from an executable file.
- The transfer speed is 2Mbps.

6.2 Synchronous Serial Communications

- When the VMU is operating on a standalone basis, the communications connector switches to synchronous serial side, and the Maple Bus protocol side stops.
 - There are two synchronous serial interfaces, allowing full duplex communications with other devices.
 - Data is transferred one byte at a time, with a maximum transfer speed of 2.4Kbps. (not finalized)
- This function is available to executable files as a subroutine.

7 Clock Function

- The clock function in the VMU conforms with FT3: Timer Function in the Maple Bus 1.0 Function Type Specifications.
This function can measure time in 500ms units, using a 32KHz crystal resonator and a dedicated counter.
- The System-BIOS controls the clock function; an executable file can only read the clock function.

7.1 Settings

- On the setting screen, set the year, month, day, and time.
- When the VMU is connected to a game machine, the date and time can be set by the game machine through the Maple Bus protocol.

8 Alarm Function

- The alarm function in the VMU conforms with FT3: Timer Function in the Maple Bus 1.0 Function Type Specifications.

This function sounds the built-in voltage buzzer.

- Only one alarm can be sounded at one time.

The sound is generated by the pulse generator method; the frequency can be set over a range from 300Hz to 4KHz, and the duty ratio can be set as desired. (planned)

The volume cannot be adjusted. The sound can be turned on and off.

This function is made available for executable programs as a subroutine. (planned)

- When the VMU is connected to a game machine, the alarm function can be set by the game machine through the Maple Bus protocol.

9 SLEEP Function

In order to reduce power consumption when operating on a standalone basis, the VMU is equipped with a SLEEP function.

The VMU enters the SLEEP state either because the SLEEP button is pressed or because the Auto power Off function was triggered. (Refer to section 5.7, "Auto Power Off") To return from the SLEEP state, press the SLEEP button.

9.1 SLEEP Operation

When in Timer mode (clock display) or File mode (file management software), the LCD display shuts off and the VMU enters the idle state.

SLEEP processing in Game mode (after an executable file has been initiated is determined by the executable file. (We plan to indicate a recommended processing method.)

The contents of RAM and the registers are retained, except in Time mode. In SLEEP mode, all buttons are disabled except for the SLEEP button.

10 Buttons

Four-direction key: This key is used to move the cursor up, down, left, or right, and to scroll the screen.

A button: This button is used primarily to finalize selections.

B button: This button is used primarily to cancel selections.

Mode button: This button changes the mode during standalone operation. Each time this button is pressed, the mode changes according to the following cycle: File -> Game -> Time -> File -> Game ->...

SLEEP button: This button changes the mode to the SLEEP state during standalone operation.

Reset button: This button initiates a "power on" reset, which initializes the entire VMU unit (including the clock, etc.), except for the contents of flash memory.

11 Batteries

11.1 Battery Life

The VMU is equipped with two CR2032 batteries for standalone operation.

Battery life depends on the status of executable file operations.

If an executable file is continuously executed, with the LCD display on (refresh rate: 1Hz), no alarm outputs, no use of the communications function, no executable file save processing, and no use of the SLEEP function, the batteries should last for about one week.

The relationship between operational status and battery life is described below. Take battery life into consideration when creating executable files.

Flash memory reads:	This is the normal state of program execution.
LCD display updates:	Battery power consumption increases by a factor of 5 when overwriting XRAM as compared to when reading flash memory. Frequent screen updates have an effect on battery life.
Alarm output:	Consumes an extremely small amount of power.
Flash memory writes:	Consumes 25 times more battery power than when reading flash memory. Saving the operation status and similar processing should be performed as infrequently and in as small amounts as possible.
Data exchanges after an executable file has been initiated:	Such operations consume a tremendous amount of battery power. Simple parameter exchange could be used to reflect the development of game characters, for example.
File exchanges between two VMUs:	Copying an entire file consumes a tremendous amount of battery power. Because the receiving side in particular must write the data in flash memory, a large amount of battery power is consumed. In addition, the larger a file is, the longer the operation will take and the greater that the power consumption will be.

11.2 Processing When Battery Power Is Exhausted

The System-BIOS constantly monitors the battery voltage.

If the batteries are nearing the end of their life while in Game mode (while an executable file is being executed), the System-BIOS saves the contents of RAM and the registers. (planned to be implemented through the library, perhaps)

11.3 Battery Replacement

- The clock settings are initialized when the batteries are replaced.
- Any file that is stored in flash memory is retained.
- When replacing the batteries, always install two brand new CR2032 made by the same manufacturer.
- Make sure that the polarity (+/-) of the batteries is correct when you install them.

12 Postscript

The functions of the VMU are subject to change in whole or in part until the release of VMU Specifications Revision 1.0.

Visual Memory Unit (VMU)
Hardware Manual

1. Overview

1. General

The POTATO custom chip that is the core of the VMU (Visual Memory System), the memory system for our next-generation game machine, consists of a CPU core that operates with a minimum bus cycle time of 0.5[μs], 128K of flash EEPROM, 20K of ROM, 710 bytes of RAM, a dot-matrix LCD automatic display controller/driver, a 16-bit timer/counter/pulse generator (or a two-channel x 8-bit timer), a 16-bit timer (or a two-channel x 8-bit timer), a two-channel x 8-bit synchronous serial interface, a dedicated interface for the next-generation game machine, and an interrupt function with 13 sources and 10 vectors.

1.1 Features

- Flash EEPROM

65,536 x 8 bits:	Program/data area
65,536 x 8 bits:	Data area
- ROM

16,384 x 8 bits:	Program area
4096 x 8 bits:	BIOS program area
- RAM

256 x 8 bits ~ 2 banks:	Calculation area
198 x 8 bits:	Display area
256 x 8 bits x 2 banks*:	Work area

*When connected with the next-generation game machine, this area is used as a TX/RX buffer for the dedicated interface.

- Bus cycle time/ instruction cycle time

The bus cycle time indicates the ROM read time.

Bus cycle time	Instruction cycle time	System clock source	Oscillating frequency	Supply voltage	Miscellaneous
0.5É s	1.0É s	CF oscillation	6MHz	3.15_3.8Çu	OCR7=1 *1
3.8É s	7.5É s	Internal RC oscillation	800KHz	3.15_3.8Çu	OCR7=1 *1
91.5É s	183.0É s	Crystal (X'tal) oscillation	32KHz	3.15_3.8Çu	OCR7=1 *1

*1• OCR7: This is bit 7 of the Oscillation Control Register (OCR); this bit controls the system clock generation circuit and the cycle time. Refer to Chapter 4, section 4.2.4, "Related Registers," for further details.

OCR7 = 1: The cycle time is the system clock divided by 6.

- Ports

- Input/output ports: 2 ports (P1, P3)
- Input port: 1 port (P7)
- Segment output port for driving the LCD: 48 lines
- Common output port for driving the LCD: 33 lines

- LCD controller

- Display duty: 1/33 duty
- Display bias: 1/5 bias
- Liquid crystal instruction display: On/Off
- Clock for external voltage step-up (external step-up circuit)
- Graphic display: 1584-dot maximum display

- Serial interfaces

- 8-bit serial interface x 2 channels (synchronous)
- Built-in 8-bit baud rate generator (The baud rate generator is shared with a two-channel serial interface.)
- Dedicated next-generation game interface (start pattern/end pattern auto discrimination)

Note: The synchronous serial interface and the dedicated next-generation game interface cannot be used simultaneously.

- Timer

- Timer 0: 16-bit timer / counter
Built-in 2-bit prescaler + 8-bit programmable prescaler
- Timer 1: 16-bit timer / pulse generator
- Base timer: Clock selection function
32.768kHz crystal oscillation, system clock, or timer 0 is selected through the programmable prescaler output.
500ms overflow signal generation function for clock (when 32.768kHz crystal oscillation is selected)
Function that generates an overflow signal on every cycle of either 976[μs], 3.9ms, 15.6ms, or 62.5ms (when 32.768kHz crystal oscillation is selected)

- Interrupts

13 sources, 10 vectors

- (1) External interrupt INT0
- (2) External interrupt INT1
- (3) External interrupt INT2, timer counter T0L (timer 0, lower 8 bits)
- (4) External interrupt INT3, base timer
- (5) Timer / counter T0H (timer 0, upper 8 bits)
- (6) Timer T1L (timer 1, lower 8 bits), timer T1H (timer 1, upper 8 bits)
- (7) Serial interface 0 (SIO0)
- (8) Serial interface 1 (SIO1)
- (9) Dedicated next-generation game machine interface
- (10) Port 3

Built-in interrupt priority register

The microcontroller interrupts can be weighted as one of three levels: low level, high level, and highest level. The 11 interrupts sources from "external interrupt INT2, timer counter T0L (timer 0, lower 8 bits)" to "Port 3" can be specified as having an interrupt priority level of either "low level" or "high level." In addition, external interrupts INT0 and INT1 can be specified as having an interrupt priority level of either "low level" or "highest level."

- Subroutine stack level

A maximum of 128 levels (The stack is set in RAM.)

- Built-in fast multiplication and division instructions

16 bits x 8 bits (execution time: 7 instruction cycles)

16 bits ÷ 8 bits (execution time: 7 instruction cycles)

- Three types of oscillation circuits

RC oscillation circuit (built in): System clock (resistor R and capacitor C built in, no external circuits required)

CF oscillation circuit: System clock

Crystal oscillation circuit: Clock, system clock, LCD clock

- Standby functions

- HALT mode

- This mode halts instruction execution, and can be cancelled by either a reset or by the generation of an interrupt.

- HOLD mode

- This mode halts CF oscillation, RC oscillation, and crystal oscillation. There are three methods for canceling HOLD mode:

- (1) Input a low level signal to the reset pin.
 - (2) Input the specified level to either the P70/INT0 pin or the P71/INT1 pin.
 - (3) Input the port 3 interrupt condition.

- Flash EEPROM

- Data memory space: 128K bytes

- Data memory space overwriting: Used for BIOS program

- Overwriting block size: 128K bytes

- Erase/program voltage: 3.15 to 3.8V

- Number of times overwriting is possible: 50,000 times (writing FF and 00 once each) (Ta = 25°C, memory management by program)

- Program memory space: 64K bytes

- Switching between mask ROM/flash EEPROM program space

- Use CHANGE instruction; initially: Mask ROM

- Switching from flash EEPROM program space to mask ROM program space can be permitted/prohibited (EXT register)

1.2 System Block Diagram

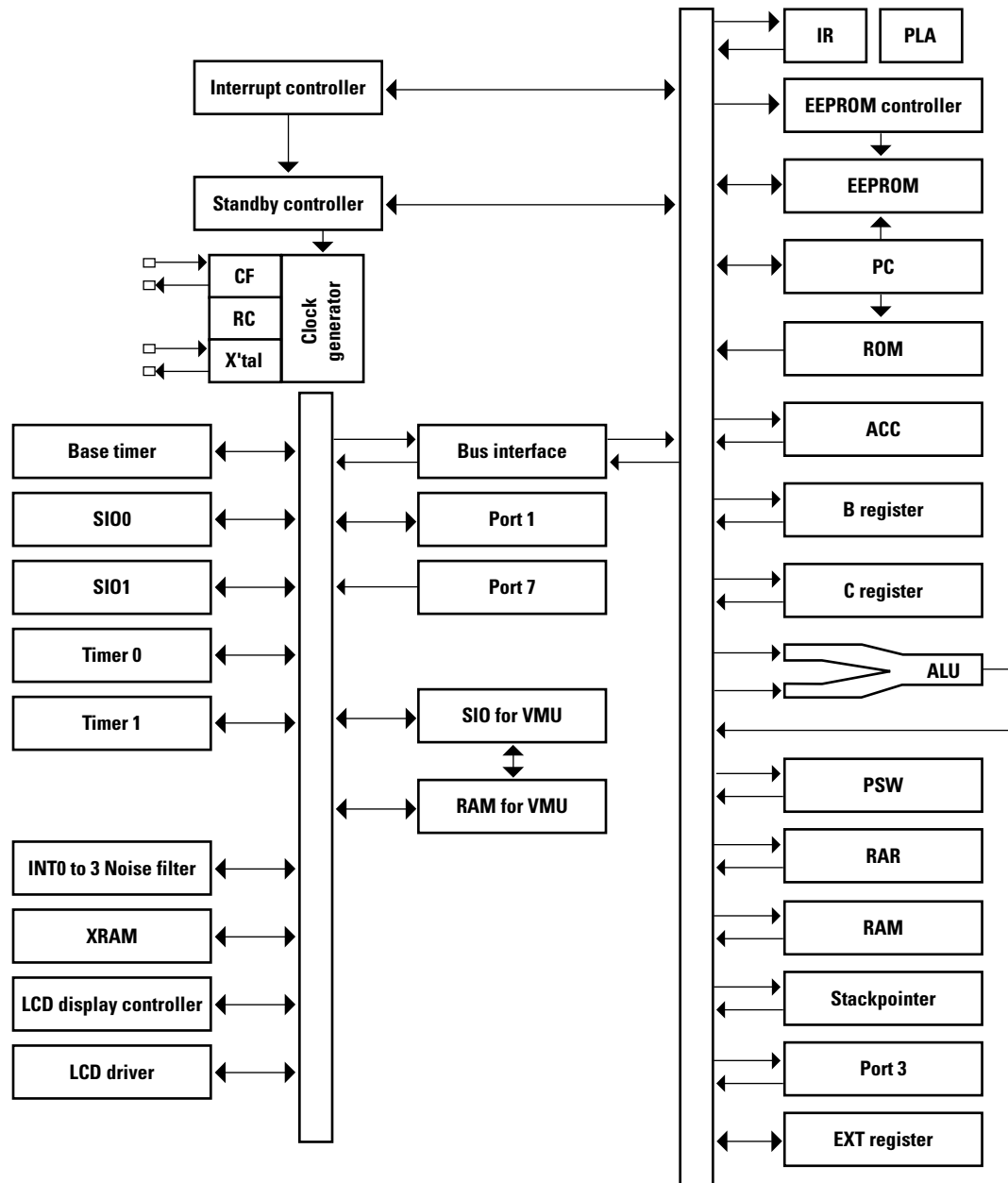


Figure 1.1 System Block Diagram

2. Internal System Configuration

1. Memory Space

POTATO has an internal memory space and a flash memory space.

The internal memory space includes a program memory space (64K) and a data memory space (512 bytes), while all 64K of program memory space (internal program ROM) can be accessed straight through by incrementing the pointer sequentially each time that a normal instruction is executed. Addresses 000 to 0FFH in the data memory space are allocated for 256 bytes of data memory (internal RAM). In addition, the 256 bytes from 100 to 1FFH are allocated for the Special Function Registers (SFR). Internal RAM consists of two banks, with the bank being specified by bit 1 (RAMBK0) of the Program Status Word (PSW) in the Special Function Registers (SFR). BANK 0 is also used as the stack area. The accumulator (ACC), PSW, timer, and input/output ports are allocated by the SFR, for a complete memory-mapped I/O configuration.

The flash memory space consists of a 128K space. This space consists of two 64K banks. Furthermore, only BANK 0 can be handled as a 64K external program memory, and a special macro instruction (CHANGE) is used to switch between internal and external programs. Writing data to the flash memory space must always be performed by calling a BIOS program subroutine.

The BIOS program memory space contains (as subroutines) a program for writing data to flash memory and checking the data that was written, and a preparation program for writing the data. This area is a BIOS-dedicated program area.

VMU game programs must be stored in the external program area (flash memory BANK 0).

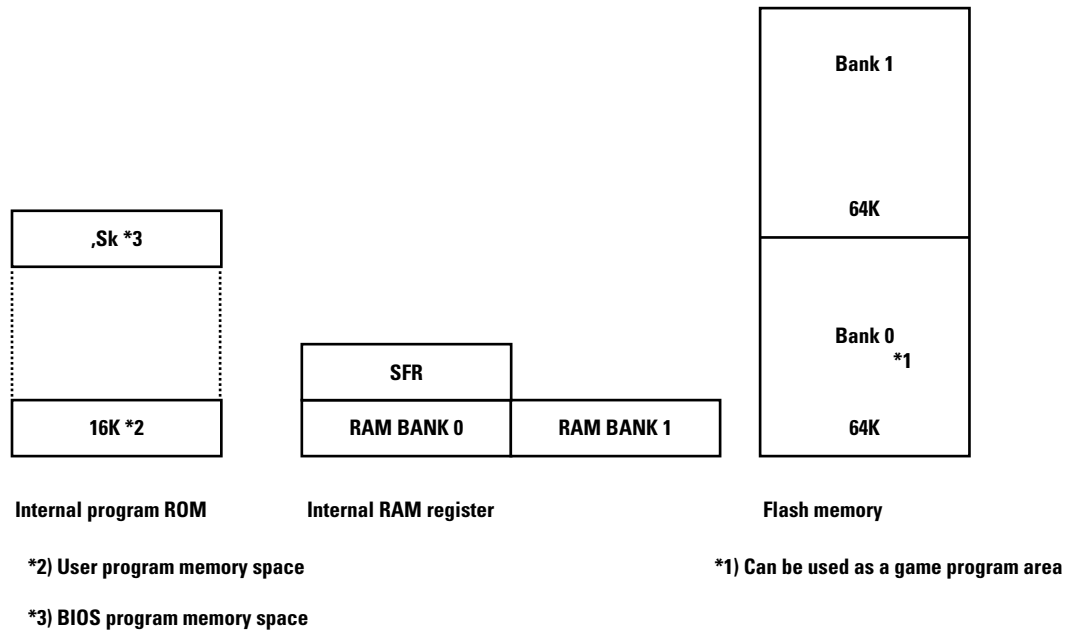


Figure 2.1 Memory Space

2. Program Counter (PC)

The Program Counter (PC) consists of 16 bits, and stores the program memory (ROM) address that contains the instruction that should be executed next. The processor executes the instructions in a program on the basis of the value in the PC. Normally, the PC is incremented for each instruction that is executed, but when a branch instruction or a subroutine instruction is executed, an interrupt is received, or a reset is executed, a value that is appropriate for that operation is set in the PC.

The data that is set in the PC for each operation is shown in Table on the next page.

Table 2.1 Program Counter Settings

Type of operation	Program counter value
Reset	0000H (internal program space)
External interrupt 0	0003H
External interrupt 1	000BH
External interrupt 2, timer counter T0L interrupt	0013H
External interrupt 3, base timer interrupt	001BH
Timer/counter T0H interrupt	0023H
Timer T1L, timer T1H interrupt	002BH
SIO0 interrupt	0033H
SIO1 interrupt	003BH

Type of operation			Program counter value
VMU SIO interrupt			0043H
Port 3 interrupt			004BH
Unconditional branching instructions	JMP	a12	PC15 to 12 = Current page PC11 to 00 = a12
	JMPF	a16	PC15 to 00 = a16
	BR	r8	(PC + 2) + r8[-128_ + 127]
	BRF	r16	(PC + 2) + r16[0~ + 65535]
Conditional branching instructions	BZ, BNZ, BP, BNE BPC, BN, DBNZ, BE		(PC + 2 or + 3) + r8[-128_ + 127]
CALL instructions	CALL	a12	PC15 to 12 = Current page PC11 to 00 = a12
	CALLF	a16	PC15 to 00 = a16
	CALLR	r16	(PC + 2) + r16[0~ + 65535]
Macro instructions	CHANGE label name (or address)		Value specified by label or address from a different program mode

Note:

- For the sake of convenience, each 4K of ROM space is called a "page."
- The "current page" is the page in the ROM space that includes the instruction that follows the instruction that is currently being executed.
- If an interrupt is generated while an internal program is running, a subroutine is called using the setting indicated in the above Table 2-2-1 in the internal program space. If an interrupt is generated while an external program is running, a subroutine is called in bank 0 of the external program space, with the lower 16 bits being the value in the above table. In a game program, some interrupt vectors can not be set as desired. it is always necessary to incorporate the specified program. Refer to Chapter 4, section 4.1, "Interrupt Function."

3. Internal Program ROM

The 64K program memory space includes 16K of user program memory and 4K of BIOS program memory (ROM). In addition, the 256 bytes from FF00H to FFFFH in the same space as the ROM are used as an option specification area for creating the mask version.

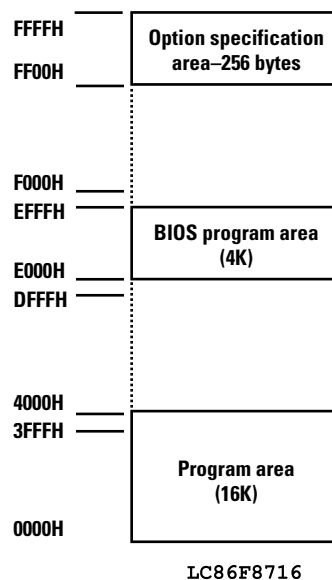


Figure 2.2 ROM Space

4. Internal Data Memory

POTATO has 1222 bytes of built-in data memory (RAM), which includes 198 bytes of XRAM and 512 bytes of VTRBF. In addition, the Special Function Registers (SFR) reside at 100H to 1FFH.

Table 2.2 Data Memory by Type (RAM)

Type	POTATO
RAM size	1222 bytes
XRAM	Bank 0 180H to 1FBH (96 bytes)
	Bank 1 180H to 1FBH (96 bytes)
	Bank 2 180H to 185H (6 bytes)
Main RAM	Bank 0 000H to 0FFH (256 bytes)
	Bank 1 000H to 0FFH (256 bytes)
VTRBF	166H (256 bytes x 2 banks)

The 16 byte area from 00H to 0FH in RAM consists of four banks of indirect address registers: @R0, @R1 (for RAM), @R2 and @R3 (for SFR), starting from the low address. The indirect address register bank that is used for addressing is specified by bits 3 and 4 (indirect address register bank flags: IRBK0, 1) of the Program Status Word (PSW). In addition, this 16-byte area can also be used as normal RAM.

The relationship between the indirect address registers and data memory is shown in Table below.

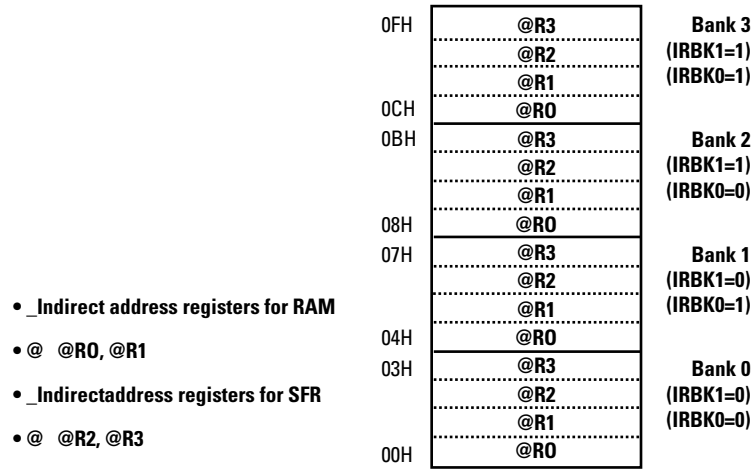


Figure 2.3 Arrangement of Indirect Address Registers

Table 2.3 Indirect Address Register Map

Indirect address register name	Function	Bank 0 (IRBK1=0) (IRBK0=0)	Bank 1 (IRBK1=0) (IRBK0=1)	Bank 2 (IRBK1=1) (IRBK0=0)	Bank 3 (IRBK1=1) (IRBK0=1)
@R0	RAM access	RAM 00H	RAM 04H	RAM 08H	RAM 0CH
@R1	RAM access	RAM 01H	RAM 05H	RAM 09H	RAM 0DH
@R2	SFR access	RAM 02H	RAM 06H	RAM 0AH	RAM 0EH
@R3	SFR access	RAM 03H	RAM 07H	RAM 0BH	RAM 0FH

Table below shows a data memory list. Refer to the respective items for details on the contents of each register.

Table 2.4 Data Memory Map

R=READ

X=Undefined

W=WRITE

H=Does not exist

Symbol	Address	R/W	Name	Initial value
RAM(BANK0)	000H-0FFH	R/W	Data memory	XXXXXXXX (retained after reset)
RAM(BANK1)	000H-0FFH	R/W	Data memory	XXXXXXXX (retained after reset)
ACC	100H	R/W	Accumulator	00000000
PSW	101H	R/W	Program Status Word	00000000
B	102H	R/W	B register	00000000
C	103H	R/W	C register	00000000
TRL	104H	R/W	Table reference register - low byte	00000000
TRH	105H	R/W	Table reference register - high byte	00000000
SP	106H	R/W	Stack pointer	XXXXXXXX
PCON	107H	R/W	Power control register	HHHHHH00
IE	108H	R/W	Master interrupt enable control register	0HHHHH00
IP	109H	R/W	Interrupt priority control register	00000000
EXT	10DH	R/W	External memory control register	HHHH0000
OCR	10EH	R/W	Oscillation control register	0H00HH00
T0CNT	110H	R/W	Timer 0 control register	00000000
T0PRR	111H	R/W	Timer 0 prescaler data	00000000
T0L	112H	R	Timer 0 low	00000000
T0LR	113H	R/W	Timer 0 low reload data	00000000
T0H	114H	R	Timer 0 high	00000000
T0HR	115H	R/W	Timer 0 high reload data	00000000
T1CNT	118H	R/W	Timer 1 control register	00000000
T1LC	11AH	R/W	Timer 1 low compare data	00000000
T1L	11BH	R	Timer 1 low	00000000

2. Internal System Configuration

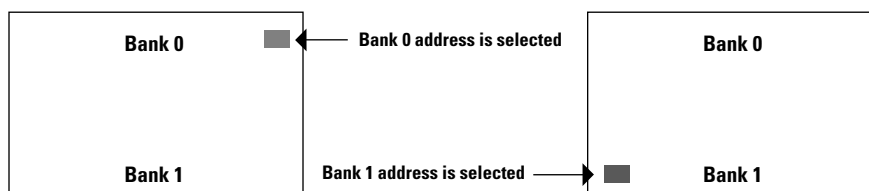
Symbol	Address	R/W	Name	Initial value
T1LR	11CH	W	Timer 1 low reload data	00000000
T1HC		R/W	Timer 1 high compare data	00000000
T1H	11DH	R	Timer 1 high	00000000
T1HR		W	Timer 1 high reload data	00000000
MCR	120H	W	Mode control register	00000000
STAD	122H	R/W	Start address register	00000000
CNR	123H	W	Character count register	H0000000
TDR	124H	W	Time interrupt register	HH000000
XBNK	125H	R/W	Bank address register	H0000000
VCCR	127H	W	LCD contrast control register	00000000
SCON0	130H	R/W	SIO0 control register	00H00000
SBUF0	131H	R/W	SIO0 buffer	00000000
SBR	132H	R/W	SIO baud rate generator	00000000
SCON1	134H	R/W	SIO1 control register	H0H00000
SBUF1	135H	R/W	SIO1 buffer	00000000
Symbol	Address	R/W	Name	Initial value
P1	144H	R/W	Port 1 latch	00000000
P1DDR	145H	W	Port 1 data direction register	00000000
P1FCR	146H	W	Port 1 function control register	00000000
P3	14CH	R/W	Port 3 latch	00000000
P3DDR	14D	W	Port 3 data direction register	00000000
P3INT	14EH	R/W	Port 3 interrupt control register	HHHHH000
P7	15CH	R	Port 7 latch	HHHHXXXX
I01CR	15DH	R/W	External interrupt 0, 1 control	00000000
I23CR	15EH	R/W	External interrupt 2, 3 control	00000000
ISL	15FH	R/W	Input signal selection	HH000000
VSEL	163H	R/W	Control register	HHH0HH00
VRMAD1	164H	R/W	System address register 1	00000000
VRMAD2	165H	R/W	System address register 2	HHHHHHH0

Symbol	Address	R/W	Name	Initial value
VTRBF	166H	R/W	TX/RX buffer	XXXXXXXX
BTCR	17FH	R/W	Base timer control	00000000
RAM(XRAM) (BANK0)	180H-1FBH	R/W	LCD memory	XXXXXXXX (retained after reset)
RAM(XRAM) (BANK1)	180H-1FBH	R/W		
RAM(XRAM) (BANK2)	180H-185H	R/W		

(1) Direct addressing mode

When executing an instruction such as: MOV #i8, d9

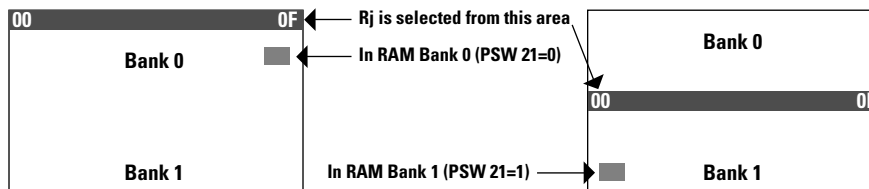
- ① In RAM BANK0 (PSW 21=0) ② RAM BANK1 (PSW 21=1)



(2) Indirect addressing mode

When executing an instruction such as: MOV #i8, @Rj

- ① RAM BANK0 (PSW 21=0) ② RAM BANK1 (PSW 21=1)

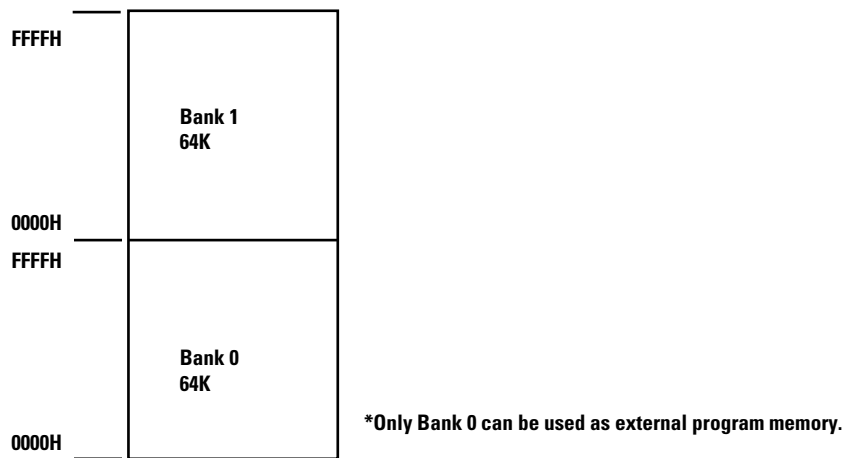


5. Flash Memory

POTATO has a 128K flash memory space.

The flash memory space consists of two 64K banks, and data can be written to and read from flash memory by referencing the BIOS program. In addition, the data in the ROM space in each bank can be referenced by using the ROM table reference instruction (LDC). Caution is required because the LDC instruction operates differently with an internal program as compared to with an external program. In addition, Bank 0 (only) can be used as a 64K external program memory space. The dedicated macro instruction CHANGE is used to switch between internal and external programs.

Flash memory size:	64K x2 banks
Banks:	Bank 0 and Bank 1
Addresses in each bank:	0000H to FFFFH



Writing and reading the flash memory space (including external program memory) is accomplished through the BIOS program. For details, refer to Chapter 3, section 3.10, "Flash EEPROM."

6. Accumulator

The accumulator is an 8-bit register that is used when performing arithmetic operations on data, transfers, input/output, and other processing. The accumulator is allocated in address 100H in the data memory space, and is initialized to 00H when a reset is executed.

- Accumulator (ACC)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ACC	100H	R/W	ACC7	ACC6	ACC5	ACC4	ACC3	ACC2	ACC1	ACC0
After reset			0	0	0	0	0	0	0	0

7. B Register, C Register

The B register and C register are 8-bit registers that are used in combination with the ACC to set data for arithmetic operations and to store the results of arithmetic operations when executing multiplication/division instructions.

These registers are allocated to addresses 102H and 103H in the data memory space, and are initialized to 00H when a reset is executed.

- B register (B)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
B	102H	R/W	B7	B6	B5	B4	B3	B2	B1	B0
After reset			0	0	0	0	0	0	0	0

- C register (C)

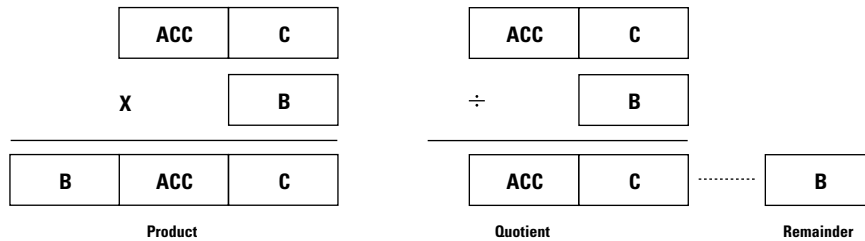
Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
C	103H	R/W	C7	C6	C5	C4	C3	C2	C1	C0
After reset			0	0	0	0	0	0	0	0

When performing multiplication, the multiplicand consists of 16 bits with the upper 8 bits stored in the ACC and the lower 8 bits stored in the C register, and the multiplier consists of 8 bits, stored in the B register.

The result of the operation (the product) consists of 24 bits, with the highest 8 bits stored in the B register the middle 8 bits stored in the ACC, and the lower 8 bits stored in the C register. In other words, $(ACC)(C) \times (B) = (B)(ACC)(C)$.

When performing division, the dividend consists of 16 bits with the upper 8 bits stored in the ACC and the lower 8 bits stored in the C register, and the divisor consists of 8 bits, stored in the B register.

The result of the operation (the quotient) consists of 16 bits, with the highest 8 bits stored in the ACC, the lower 8 bits stored in the C register and the remainder stored in the B register. In other words, $(ACC)(C) \div (B) = (ACC)(C) \text{ mod } (B)$.



8. Program Status Word

The Program Status Word (PSW) consists of flags that indicate the results of arithmetic operations, and flags that specify banks for data memory (RAM) and indirect address registers. The PSW is allocated to address 101H in the data memory space, and each bit is initialized to "0" when a reset is executed.

- Program Status Word (PSW)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PSW	101H	R/W	CY	AC	-	IRBK1	IRBK0	OV	RAMBK0	P
After reset			0	0	0	0	0	0	0	0

CY (bit 7):

Carry flag

CY is set (1) when there is a carry in the result of an arithmetic operation, or when a borrow is generated in a subtraction or compare operation; CY is reset (0) when neither of these events occurs. This bit is affected if a rotate instruction that includes CY is executed, and is reset (0) when a multiplication or division instruction is executed.

AC (bit 6):

Auxiliary carry flag

AC is set (1) when there is a carry from bit 3 of the ACC in the result of an arithmetic operation, or when a borrow is generated in a subtraction operation; CY is reset (0) when neither of these events occurs.

IRBK1 (bit 4):

Indirect address register bank flag 1

IRBK0 (bit 3):

Indirect address register bank flag 0

These bits specify one of the four banks that comprise the register group that is to be used as the indirect address register for indirect addressing instructions within each bank of data memory (RAM).

Bank	IRBK1	IRBK0
0	0	0
1	0	1

Bank	IRBK1	IRBK0
2	1	0
3	1	1

OV (bit 2):

Overflow flag

OV is set (1) if an overflow occurs and is reset (0) if it does not.

This bit is set (1) if the act of adding a negative number to a negative number or of subtracting a positive number from a negative number generates a positive number, or if the act of adding a positive number to a positive number or subtracting a negative number from a positive number results in a negative number. In all other cases, this bit is reset (0). This bit is also set (1) when the contents of the B register are not "0" after a multiplication or division operation, and is reset if the contents of the B register are "0."

RAMBK0 (bit 1):

Data memory (RAM) bank flag

This flag specifies the data memory (RAM) bank. When an instruction that accesses RAM is executed, the RAM address in the specified bank is accessed.

Bank	RAMBK0
0	0
1	1

P (bit 0):

This bit indicates the accumulator parity (ACC).

This bit is set (1) if the total number of bits that are set in the accumulator is odd, and reset (0) if the total number of bits that are set in the accumulator is even. This bit is a read-only bit. This bit cannot be written.

9. Stack Pointer

Bank 0 of data RAM is used as stack memory. The Stack Pointer (SP) is an 8-bit register that specifies an address in this stack area.

The SP is allocated in address 106H in the data memory space. The SP is incremented right before a save to the stack memory, and is decremented after data is returned from the stack memory.

When a reset is executed, the value of the SP is undefined, and must be initialized at the beginning of the BIOS program.

- Stack Pointer (SP)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SP	106H	R/W	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
After reset			X	X	X	X	X	X	X	X

- When executing a PUSH instruction, the SP is incremented and the data in data memory specified by the operand is pushed onto the stack. When executing a POP instruction, after the data is stored in data memory specified by the operand, the SP is decremented.
- Note that if a PUSH or POP instruction is executed after bank 1 is specified, the RAM in bank 0 specified by the Stack Pointer (SP) is still used as the stack memory. In addition, if the operand is a RAM address, bank 0 RAM, not bank 1 RAM, is accessed.
- When executing a CALL instruction, the SP is incremented, the lower 8 bits of the PC are saved in the stack, the SP is incremented again, and the upper 8 bits of the PC are saved. When the RET instruction is executed, the data specified by the SP is stored in the upper 8 bits of the PC, the SP is decremented, the data specified by the new value of the SP is stored in the lower 8 bits of the PC, and then the SP is decremented again.
- When an interrupt is accepted, the SP is incremented, the lower 8 bits of the PC are saved in the stack, the SP is incremented again, and the upper 8 bits of the PC are saved. When the RETI instruction is executed in order to return from the interrupt processing, the data specified by the SP is stored in the upper 8 bits of the PC, the SP is decremented, the data specified by the new value of the SP is stored in the lower 8 bits of the PC, and then the SP is decremented again.

10. The Table Reference Register (TRR)

The Table Reference Register (TRR) is a 16-bit register that specifies addresses for program memory (ROM) and flash memory (EEPROM). The low byte (TRL) is allocated to address 104H in the data memory space, and the high byte (TRH) is allocated to address 105H in the data memory space. This register is initialized to 00H when a reset is executed.

When executing the table reference instruction (LDC), the data stored in the TRR is added to the data stored in the ACC, and then the data that is read from the address indicated by the combined value is transferred to the ACC.

When a flash memory write/read is executed (through the BIOS program), the address indicated by the data stored in the TRR is referenced in the specified bank.

- Table Reference Register (Low) TRL

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TRL	104H	R/W	TRL7	TRL6	TRL5	TRL4	TRL3	TRL2	TRL1	TRL0
After reset			0	0	0	0	0	0	0	0

- Table Reference Register (High) TRH

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TRH	105H	R/W	TRH7	TRH6	TRH5	TRH4	TRH3	TRH2	TRH1	TRH0
After reset			0	0	0	0	0	0	0	0

11. CHANGE Instruction

Switching between the internal and external program is accomplished by executing the CHANGE instruction. When executed while currently running an internal program (game program), the mode changes to game program (internal program), and the program counter is set to the address that is specified by the label or address.

Format:

Change label name (or address)

Operation:

(1) Execution while in internal program mode

- The mode changes from internal program mode to game program mode.
- The program counter is set to the address in the game program specified by the label or address.

(2) Execution while in game program mode

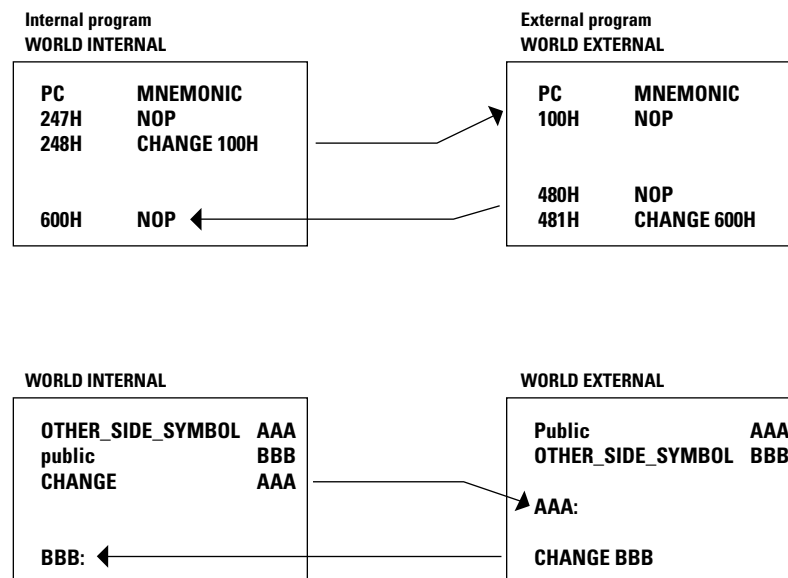
- The mode changes from game program mode to internal program mode.

However, if bit 1 (LDCEXT) of the external memory control register is set, executing the CHANGE instruction has no effect; the system remains in game program mode, and does not change to the internal program.

- The program counter is set to the address in the internal program specified by the label or address.

(3) The change in program mode occurs after a dedicated macro instruction is executed.

Example:



3. Peripheral System Configuration

1. Input/Output Ports

POTATO has three I/O ports, each allocated to their own address in the Special Function Registers (SFR); in addition, the input/output direction of ports 1 and 3 is determined by the corresponding data direction register (PnDDR). Port 1 is also used as a serial interface/next-generation game machine interface, and as a pulse generator output, and is controlled by a functional control register (P1FCR). In addition, port 7 is also provided as an input-only port.

When a reset is executed, all ports are set to input mode, and the port latch bits are set to "0."

The following Special Function Registers must be manipulated in order to use the input/output ports:

Port 1 (P1)	•P1	•P1DDR	•P1FCR	
Port 3 (P3)	•P3	•P3DDR	•P3INT	•EXT
Port 7 (P7)	•P7	(input only)		

Note:

- Caution is required when reading port data from an I/O port, because some instructions read the port latch, and some instructions read the data that is being applied to the port. The instructions listed below read the port latch data. Refer to the diagram below.
BPC, DBNZ, INC, DEC, SET1, CLR1, NOT1
-

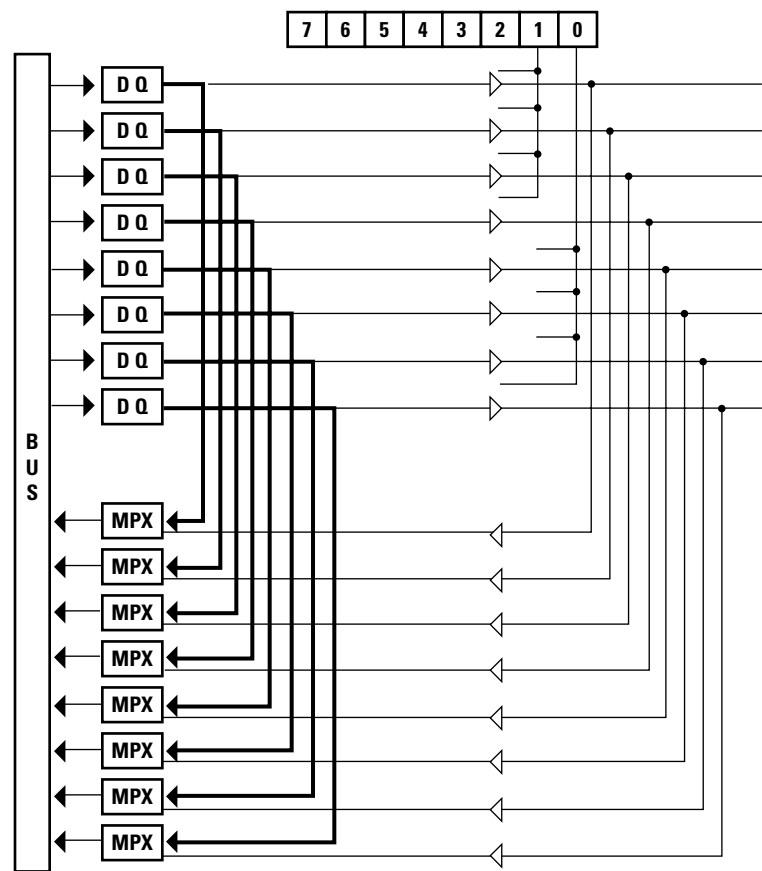


Figure 3.1 Data Path When Executing BPC, DBNZ, INC, DEC, SET1, CLR1, and NOT1

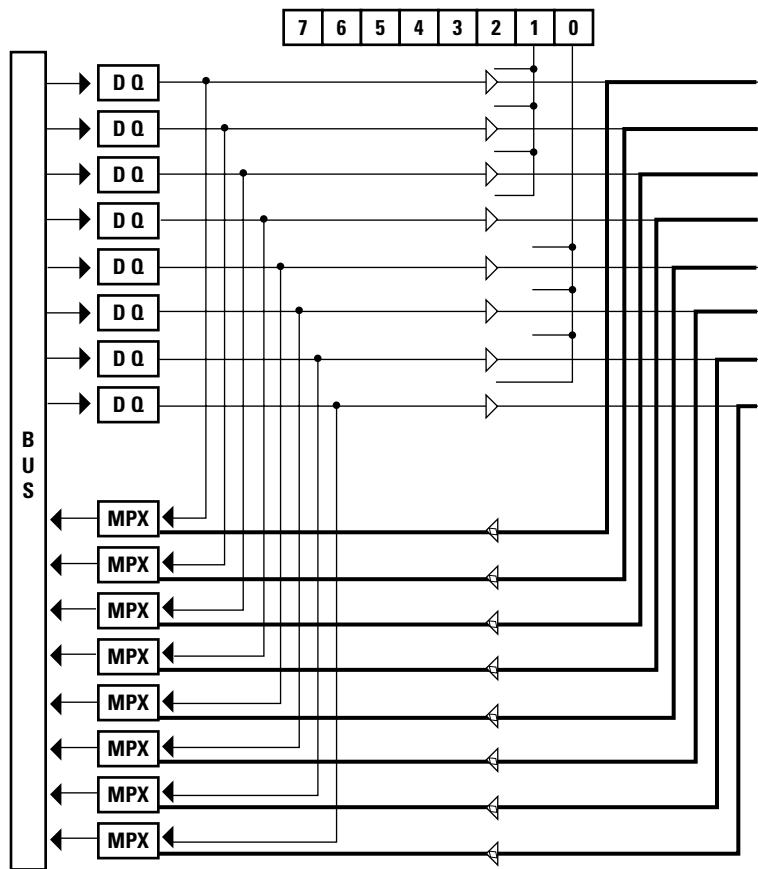


Figure 3.2 Data Path When Executing an Instruction Other Than BPC, DBNZ, INC, DEC, SET1, CLR1, and NOT1

1.1 Port 1

Port 1 can be used for input/output for the VMU serial interface, and for pulse generator output. Port 1 can also be used for input/output for the next-generation game machine interface. Only SIO (P10 to P15) and the pulse generator output (P17) can be used from a game program. Use only bit manipulation instructions to access this register. For details on the SIO output, refer to Chapter 3, section 3.5, "Serial Interface;" for details on the pulse generator output, refer to Chapter 3, section 3.3, "Timer 1 (T1)."

- Port 1 Latch (P1)

Port 1	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	P17	P16	P15	P14	P13	P12	P11	P10
Function	Pulse Output	TEST	SCK1	SB1	S01	SK0	SB0	S00

- Port 1 Data Direction Register (P1DDR)

The Port 1 Data Direction Register is a write-only register that corresponds to each data latch bit. It is essential to note that if a bit manipulation instruction, the INC instruction, the DEC instruction, or the DBNZ instruction is used on a write-only register, all bits other than the specified bit will be set to "1." The following instructions are used on P1DDR:

- MOV
- ST
- POP
- MOV @
- ST @

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1DDR	145H	W	P17DDR	P16DDR	P15DDR	P14DDR	P13DDR	P12DDR	P11DDR	P10DDR
After a reset			0	0	0	0	0	0	0	0

Bit name	Function
P17DDR (bit 7) to P10DDR (bit 0)	I/O control 0: Input mode 1: Output mode

P17DDR (bit 7): P17 I/O control

|

P10DDR (bit 0): P10 I/O control

These bits switch the data I/O direction for each bit of port 1 (P17 to P10) between output mode (1) and input mode (0). When a bit is set to "1," the corresponding bit from P17 to P10 enters output mode; when a bit is set to "0," the corresponding bit from P17 to P10 enters input mode.

Example: When P17DDR = 1, P17 is in output mode.

- Σ Port 1 Function Control Register (P1FCR)

The Port 1 Function Control Register is a write-only register. It is essential to note that if a bit manipulation instruction, the INC instruction, the DEC instruction, or the DBNZ instruction is used on a write-only register, all bits other than the specified bit will be set to "1." The following instructions are used on P1FCR:

- MOV
- ST
- POP
- MOV @
- ST @

3. Peripheral System Configuration

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PIFCR	146H	W	P17FCR	P16FCR	P15FCR	P14FCR	P13FCR	P12FCR	P11FCR	P10FCR
After a reset			0	0	0	0	0	0	0	0

Bit name	Function
P17FCR (bit 7)	P17 control function
	0: Port data (P17) output 1: PWM output
P16FCR (bit 6)	P16 control function
	0: Port data (P16) output 1: Buzzer (BUZ) output
P15FCR (bit 5)	P15 control function
	0: Port data (P15) output 1: Serial interface clock (SCK1) output
P14FCR (bit 4)	P14 control function
	0: Port data (P14) output 1: Serial interface data (SB1) input/output
P13FCR (bit 3)	P13 control function
	0: Port data (P13) output 1: Serial interface data (SO1) output
P12FCR (bit 2)	P12 control function
	0: Port data (P12) output 1: Serial interface clock (SCK0) output
P11FCR (bit 1)	P11 control function
	0: Port data (P11) output 1: Serial interface data (SB0) input/output
P10FCR (bit 0)	P10 control function
	0: Port data (P10) output 1: Serial interface data (SO0) output

P17FCR (bit 7):	<p>P17 function selection</p> <p>This pin selects either PWM (1) or port data (0) for the P17 output. When this bit is set to "1," P17 outputs the logical sum of the PWM signal and the port latch data. When this bit is set to "0," P17 outputs the port latch data.</p>
P16FCR (bit 6):	<p>P16 function selection</p> <p>This pin selects either the buzzer (1) or port data (0) for the P16 output. When this bit is set to "1," P16 outputs the logical sum of the buzzer signal and the port latch data. When this bit is set to "0," P16 outputs the port latch data.</p>
P15FCR (bit 5):	<p>P15 function selection</p> <p>This pin selects either the serial clock (1) or port data (0) for the P15 output. When this bit is set to "1," P15 outputs the logical sum of the serial interface clock (SCK1) and the port latch data. When this bit is set to "0," P15 outputs the port latch data.</p>
P14FCR (bit 4):	<p>P14 function selection</p> <p>This pin selects either serial data (1) or port data (0) for the P14 output. When this bit is set to "1," P14 outputs the logical sum of the serial interface data (SB1) and the port latch data. When this bit is set to "0," P14 outputs the port latch data.</p> <p>Note that serial interface data can always be input.</p>
P13FCR (bit 3):	<p>P13 function selection</p> <p>This pin selects either serial data (1) or port data (0) for the P13 output. When this bit is set to "1," P13 outputs the logical sum of the serial interface data (SO1) and the port latch data. When this bit is set to "0," P13 outputs the port latch data.</p>
P12FCR (bit 2):	<p>P12 function selection</p> <p>This pin selects either the serial clock (1) or port data (0) for the P12 output. When this bit is set to "1," P12 outputs the logical sum of the serial interface clock (SCK0) and the port latch data. When this bit is set to "0," P12 outputs the port latch data.</p>
P11FCR (bit 1):	<p>P11 function selection</p> <p>This pin selects either serial data (1) or port data (0) for the P11 input/output. When this bit is set to "1," P11 outputs the logical sum of the serial interface data (SB0) and the port latch data. In bus mode, however, this bit is an input/output (SB0). When this bit is set to "0," P11 outputs the port latch data.</p> <p>Note that serial interface data can always be input.</p>
P10FCR (bit 0):	<p>P10 function selection</p> <p>This pin selects either serial data (1) or port data (0) for the P13 output. When this bit is set to "1," P10 outputs the logical sum of the serial interface data (SO0) and the port latch data. When this bit is set to "0," P10 outputs the port latch data.</p>

Note: on Writing Game Programs for the VMU:

Game programs for the VMU must perform the following processing:

- When the VMU is operating on a standalone basis (SIO is not being used)

1. Monitor the 5V detection in port 7.
2. Once the 5V signal has been detected, change bits 2 and 5 of port 1 to port data output mode, and then output a "0" (zero) from each bits.

Once the above processing is completed, restore the settings that were in effect before the above processing was performed.

Note:

- When using one of Port 1's independent functions, it is necessary to set the port latch that corresponds to that function to "0." For example, when using PWM, set P17 = 0.
 - The port latch data is read by the following instructions: BPC, DBNZ, INC, DEC, SET1, CLR1 and NOT1. Other instructions read the data that is applied to the port.
-

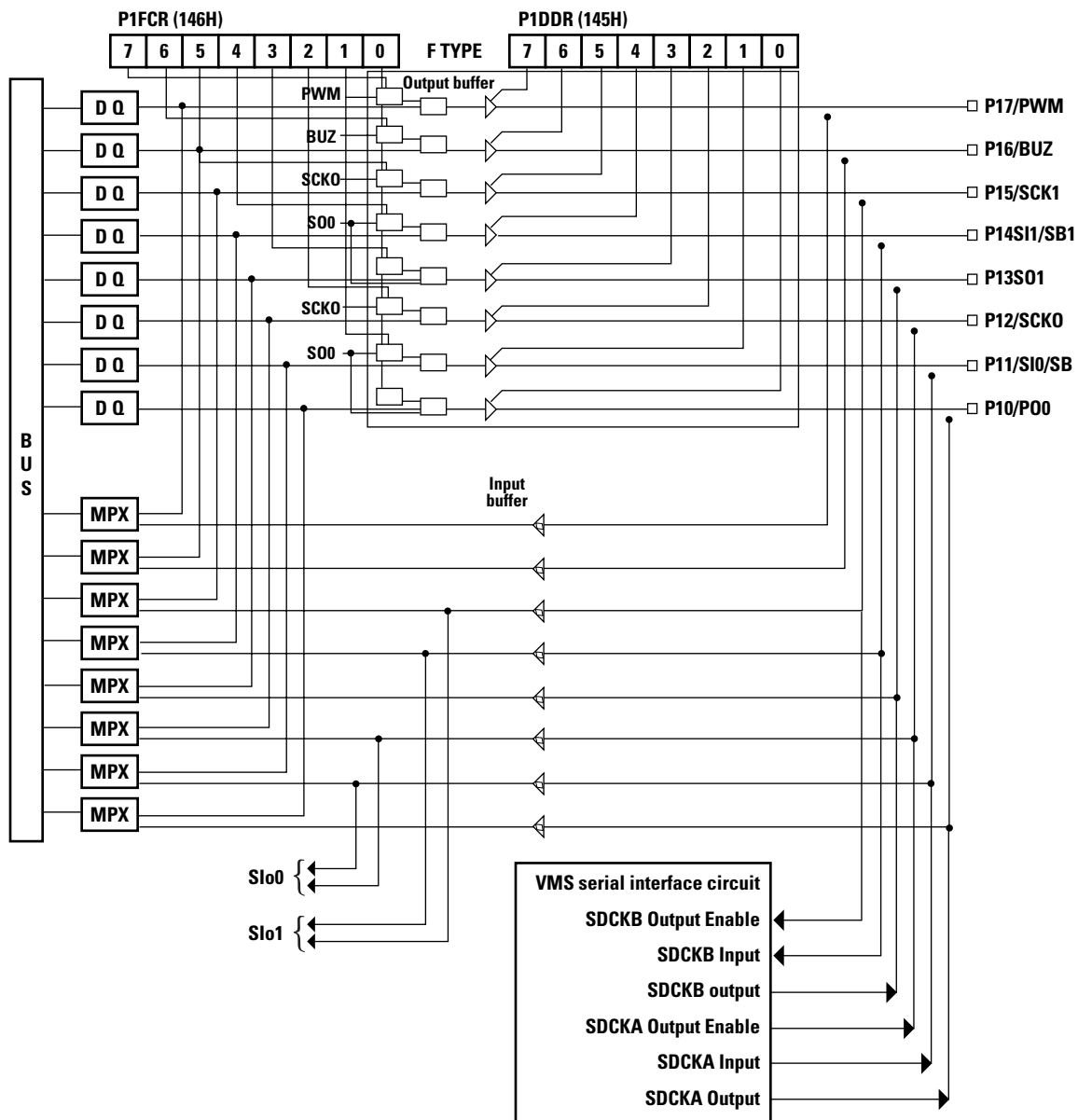


Figure 3.3 Port 1 Block Diagram

1.2 Port 3

Port 3 is an input-only port that is used for the VMU direction keys, buttons (A, B, MODE), and VMU SLEEP button.

- Port 3 Latch (P3)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P3	14CH	R/W	P37	P36	P35	P34	P33	P32	P31	P30
Function	SLEEP	MODE	Button B	Button A	RIGHT	LEFT	DOWN	UP		
After reset			0	0	0	0	0	0	0	0

Because bits 0 through 7 of port 3 are programmable and can be pulled up through software, it is necessary for the user program to substitute ones for P3. when a button is pressed, the corresponding bit goes to "0."

Note: Regarding the direction keys, design game programs so that opposing directions (up-down, left-right) cannot be pressed simultaneously.

- Port 3 Data Direction Register (P3DDR)14DH

The above register is set by the internal system program.

This register cannot be accessed from a game program.

- Port 3 Interrupt Control Register (P3INT)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P3INT	14EH	R/W	-	-	-	-	-	P32INT	P31INT	P30INT
After reset			H	H	H	H	H	0	0	0

Bit name	Function
P32INT (bit 2)	Port 3 interrupt control flag
	0: Prohibits cancellation of HOLD mode when an interrupt is generated by port 3. 1: Enables cancellation of HOLD mode when an interrupt is generated by port 3.
P31INT (bit 1)	Port 3 interrupt source flag
	0: No interrupt source 1: Interrupt source
P30INT (bit 0)	Port 3 interrupt request flag
	0: Interrupt request prohibited 1: Interrupt request enabled

P32INT (bit 2):	<p>Port 3 interrupt selection flag</p> <p>This flag selects whether port 3 is to be used for interrupt source generation or not.</p> <p>P32INT = 0: Does not generate interrupts on port 3.</p> <p>P32INT = 1: Generates interrupts on port 3.</p>
P31INT (bit 1):	<p>Port 3 interrupt source flag</p> <p>This flag has meaning only when the P32INT flag is set. This flag is used to monitor the presence/absence of port 3 interrupt sources.</p> <p>This flag is set (1) when an interrupt source is generated on port 3, and does not change if no interrupt source is generated. This flag must be reset by software.</p>
P30INT (bit 0):	<p>Port 3 interrupt request enable control</p> <p>This bit enables (1)/disables (0) interrupt requests due to the generation of an interrupt source on port 3.</p> <p>P30INT = 0: Disables interrupt requests on port 3.</p> <p>P30INT = 1: Enables interrupt requests to vector address 004BH when an interrupt source is generated on port 3 (when P31INT = 1).</p>

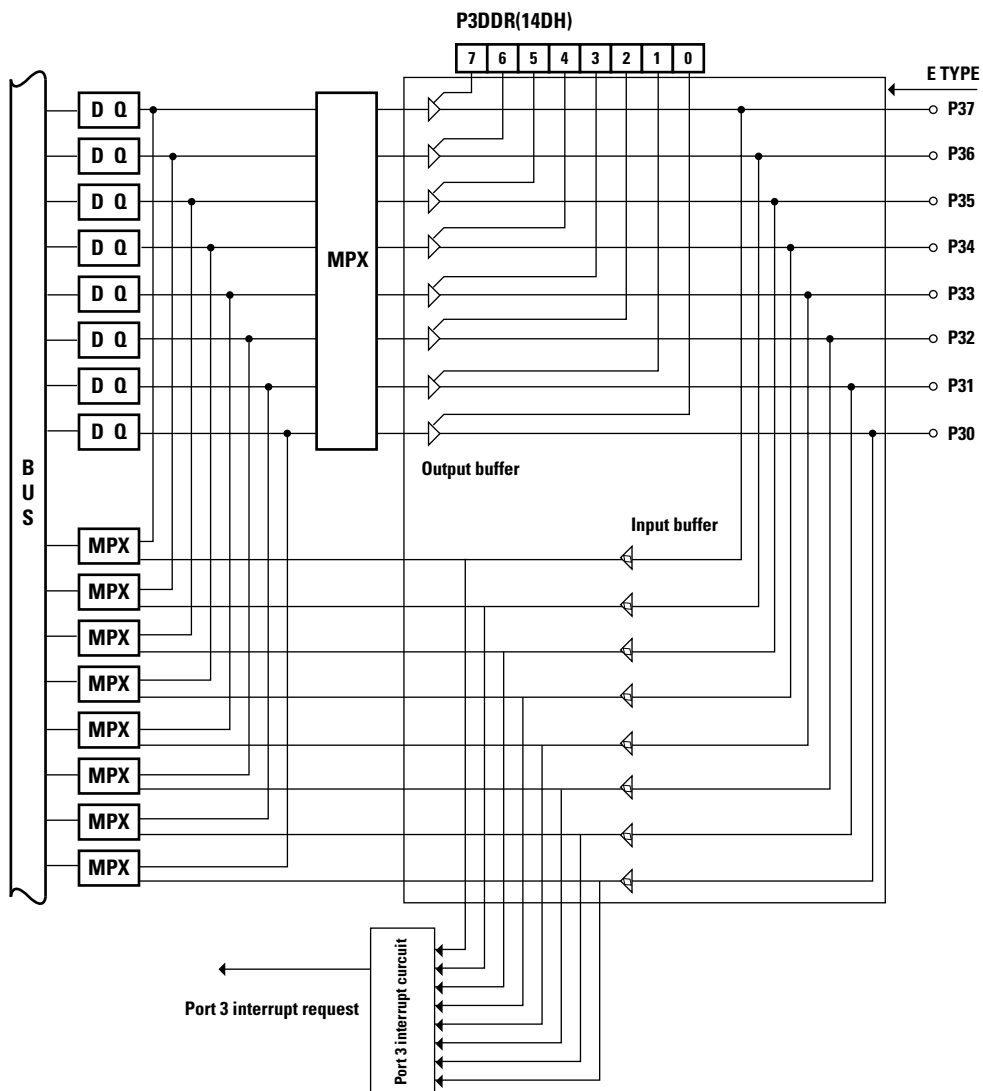


Figure 3.4 Port 3 Block Diagram

1.3 Port 7

Port 7 is an input-only port that is used for checking low voltage in the VMU and for checking the connection with the next-generation game machine. (Refer to Fig. 3-1-8.)

- Port 7 (P7)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P7	15CH	R	-	-	-	-	P73	P72	P71	P70
Function			-	-	-	-	ID1	ID0	Low voltage	5V detection
After reset			H	H	H	H	0	0	1	0

internal pull-up resistor transistors set the initial input state of bits 7 through 0 of port 7 to "1." When a button is pressed, the corresponding bit goes to "0."

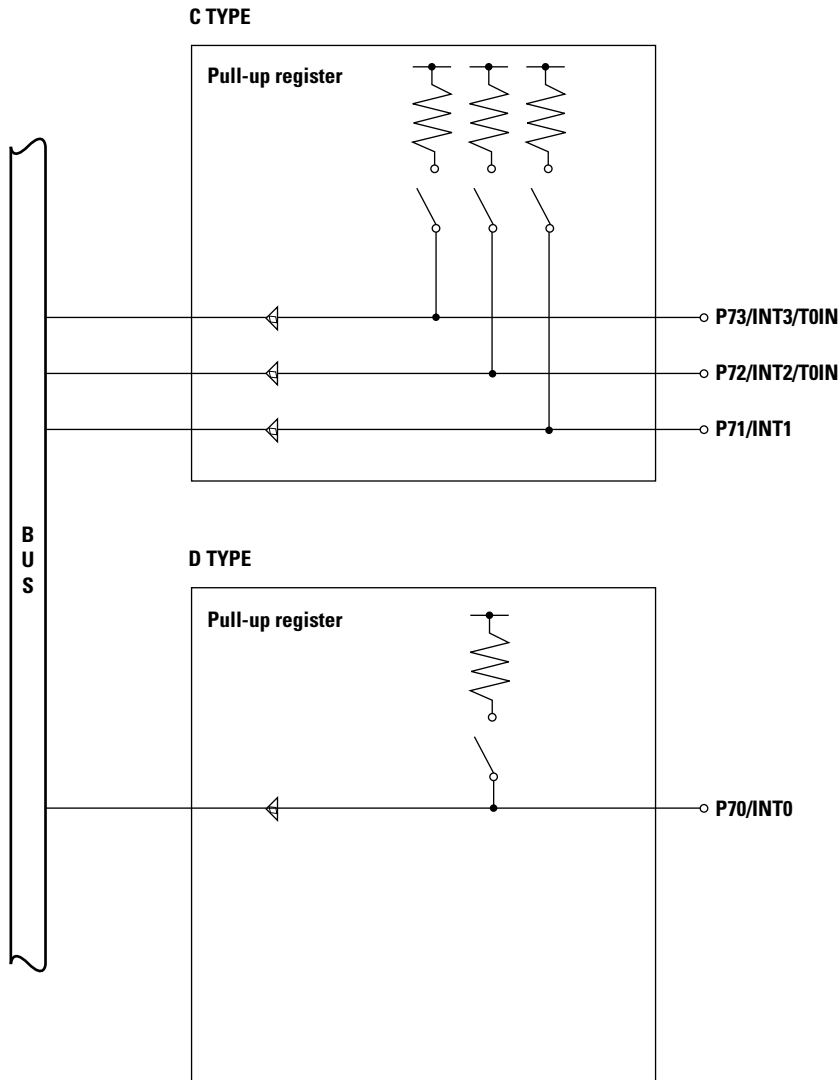


Figure 3.5 Port 7 Block Diagram

2. Timer/Counter 0 (T0)

2.1 Overview

POTATO's built-in Timer/Counter 0 (T0) is a 16-bit timer/counter that has the four functions listed below. In addition, the Timer 0 prescaler is an 8-bit prescaler.

Mode 0: 8-bit reload timer x 2 channels, with programmable prescaler

Mode 1: 8-bit reload timer with programmable prescaler + 8-bit reload counter

Mode 2: 16-bit reload timer with programmable prescaler

Mode 3: 16-bit reload counter

2.2 Functions

- 8-bit reload timer x 2 channels, with programmable prescaler (mode 0)

T0 operates as two independent 8-bit reload timers (T0H, T0L) according to the clock from the 10-bit/8-bit prescaler.

- 8-bit reload timer with programmable prescaler + 8-bit reload counter (mode 1)
- T0H operates as an 8-bit reload timer according to the clock from the 8-bit prescaler. T0L detects and counts external input signals from the P72/INT2/T0IN and P73/INT3/T0IN pins.
- 16-bit reload timer with programmable prescaler (mode 2)
- T0 operates as a 16-bit reload timer (T0H + T0L) according to the clock from the 10-bit/8-bit prescaler.
- 16-bit reload counter (mode 3)
- T0 operates as a 16-bit reload counter in which the overflow of T0L is used as the clock for T0H. T0L detects and counts external input signals from the P72/INT2/T0IN and P73/INT3/T0IN pins.
- Interrupt generation
- When the interrupt request enable bit is set, T0H and T0L interrupt requests are generated when the T0H and T0L registers overflow.
- The following Special Function Registers must be manipulated in order to control Timer/Counter 0 (T0).

• T0H	• T0HR
• T0L	• T0LR
• T0CNT	• T0PRR
• ISL	• I23CR

2.3 Circuit Configuration

The configuration of Timer/Counter 0 (T0) is shown in Fig. 3-2-1.

- Prescaler \neg

The 8-bit prescaler consists of an 8-bit programmable counter, and does not have a cycle clock divide-by-4 circuit.

The prescaler counts up while the system is running; "00" is set in the divide-by-4 circuit (in the case of the 10-bit prescaler) if the program has changed the prescaler data, and counting begins from the changed data. The cycle clock is a signal that is generated once per cycle while executing instructions and in HALT mode.

- Timer/Counter 0 Low (T0L) _

This is an 8-bit reload timer/counter that uses the prescaler output or a signal on an external pin as a clock. The TOLR data is reloaded upon a TOL overflow in modes 0 or 1 and upon a T0H overflow in modes 2 and 3; the TOLR data is also reloaded when TOLRUN (bit 6 of T0CNT) is reset and system operation is stopped.

- Timer/Counter 0 High (T0H) ®

This is an 8-bit reload timer that uses the prescaler output or T0L overflow as a clock. The Timer 0 High Reload (T0HR) register data is reloaded upon a T0H overflow; the TOHR data is also reloaded when TOHRUN (bit 7 of T0CNT) is reset and system operation is stopped.

- Timer/Counter 0 Control Register (T0CNT)

This register sets modes 0 through 3 for T0 and controls interrupts.

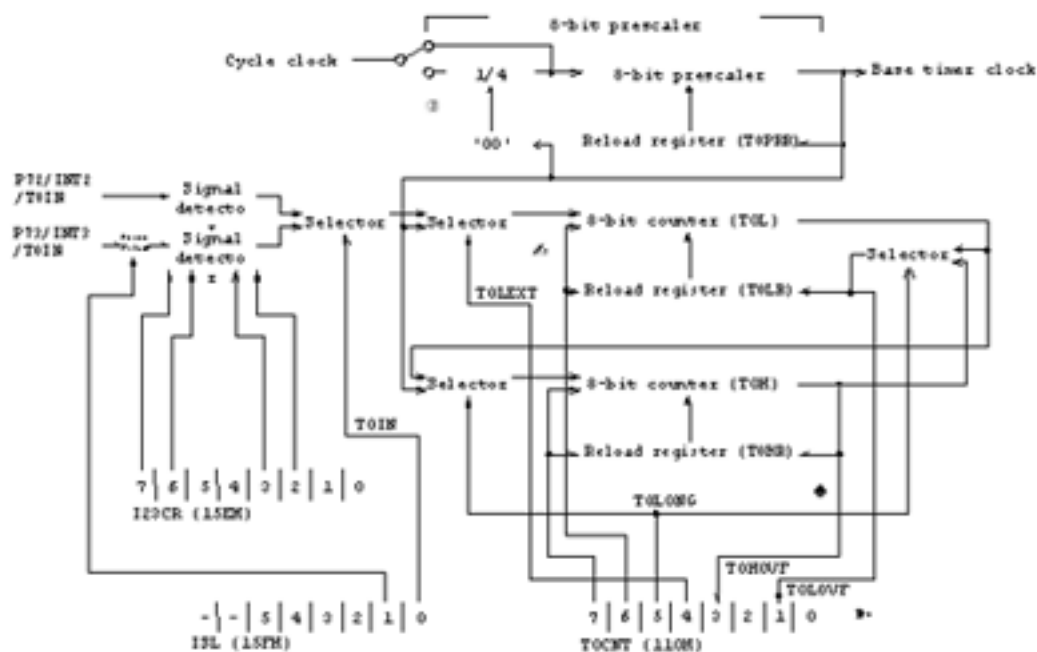


Figure 3.6 *Timer/Counter 0 Block Diagram*

2.4 Related Registers

- Timer/Counter 0 Control Register (T0CNT)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0CNT	14EH	R/W	TOHRUN	TOLRUN	TOLONG	TOLEXT	TOHOVF	TOHIE	TOLOVF	TOLIE
After reset			0	0	0	0	0	0	0	0

Bit name	Function
TOHRUN (bit 7)	TOH count control
	0: Count stop/data reload 1: Count start
TOLRUN (bit 6)	TOL count control
	0: Count stop/data reload 1: Count start
TOLONG (bit 5)	Timer/counter 0 bit length selection
	0: 8 bits 1: 16 bits
TOLEXT (bit 4)	TOL input clock selection
	0: Prescaler output 1: External pin input signal In the case of an external pin, specify the pin through the Input Signal Selection (ISL) register.
TOHOVF (bit 3)	TOH overflow flag
	0: No overflow flag 1: Overflow flag
TOHIE (bit 2)	TOH interrupt request enable
	0: Interrupt request disabled 1: Interrupt request enabled
TOLOVF (bit 1)	TOL overflow flag
	0: No overflow flag 1: Overflow flag
TOLIE (bit 0)	TOL interrupt request enable
	0: Interrupt request disabled 1: Interrupt request enabled

T0HRUN (bit 7):

T0H count control

This bit starts (1)/stops (0) Timer/Counter 0 High (T0H). When this bit is set to "1," the clock is input to T0H, which begins counting; when this bit is set to "0," the clock supplied to T0H is stopped and the reload data (T0HR) is transferred to T0H simultaneously.

T0LRUN (bit 6):

T0L count control

This bit starts (1)/stops (0) Timer/Counter 0 Low (T0L). When this bit is set to "1," the clock is input to T0L, which begins counting; when this bit is set to "0," the clock supplied to T0L is stopped and the reload data (T0LR) is transferred to T0L simultaneously.

T0LONG (bit 6):

Timer/Counter 0 bit length selection

This bit selects the T0 bit length as either 16 bits (1) or 8 bits (0). When this bit is set to "1," the bit length of Timer/Counter 0 is 16 bits; when this bit is set to "0," the bit length is set to 8 bits. Set this bit to "0" when using modes 0 and 1, and to "1" when using modes 2 and 3.

Mode	T0LONG	T0LEXT
0	0	0
1	0	1
2	1	0
3	1	1

T0LEXT (bit 4):

T0L input clock selection

This bit selects the T0L input clock as either external pin input signal (1) or as prescaler output (0). When this bit is set to "1," the input signal from the external input pin (either P72/INT2/T0IN or P73/INT3/T0IN) becomes the clock for T0L; when this bit is set to "0," the prescaler output becomes the clock. The Input Signal Selection (ISL) register selects either P72/INT2/T0IN or P73/INT3/T0IN as the external input pin.

T0HOVF (bit 3):

T0H overflow flag

This flag is set when an overflow occurs in T0H, and does not change when no overflow occurs. Therefore, this flag must be reset by software.

T0HIE (bit 2):

T0H interrupt request enable control

This bit enables (1)/disables (0) interrupt requests due to a T0H overflow. When this bit is set to "1," an interrupt request is generated to vector address 0023H in response to a T0H overflow; when this bit is set to "0," no interrupt request is generated.

T0LOVF (bit 1):	<p>T0L overflow flag</p> <p>This flag is set when an overflow occurs in T0L, and does not change when no overflow occurs. Therefore, this flag must be reset by software.</p> <p>In 16-bit mode, this flag is not set even if an overflow occurs in T0L; when an overflow occurs in T0H, this flag is set at the same time as T0HOVF.</p>
T0LIE (bit 0):	<p>T0L interrupt request enable control</p> <p>This bit enables (1)/disables (0) interrupt requests due to a T0L overflow. When this bit is set to "1," an interrupt request is generated to vector address 0013H in response to a T0L overflow; when this bit is set to "0," no interrupt request is generated.</p>

Note:

- T0HOVF and T0LOVF must be set to "0" by software.
 - When operating in 16-bit mode, set T0HRUN and T0LRUN to "1" simultaneously.
 - In 16-bit mode, T0HOVF and T0LOVF both go to "1" simultaneously.
-

- Input Signal Selection register (ISL)

This register is used to select the time constant for the noise elimination filter connected to the P73/INT3/T0IN pin, and to select the external signal input pin. This register cannot be accessed by game programs.

Use only bit manipulation instructions to access this register.

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ISL	15FH	R/W	-	-	ISL5	ISL4	ISL3	ISL2	ISL1	ISL0
After reset			H	H	0	0	0	0	0	0

Bit name	Function		
ISL5 (bit 5)	Base timer clock selection		
ISL4 (bit 4)	ISL5	ISL4	
	1	1	Timer/Counter T0 prescaler
	0	1	Cycle clock
	X	0	Subclock (crystal oscillator)
ISL3 (bit 3)	Buzzer output frequency selection		
	0: fBST/16		
	1: fBST/8		
ISL2 (bit 2)	Noise elimination filter time constant selection		
ISL1 (bit 1)	ISL2	ISL1	Time constant
	1	1	16Tcyc
	0	1	64Tcyc
	X	0	1Tcyc
ISL0 (bit 0)	T0 clock input pin selection		
	0: P72/INT2/T0IN pin		
	1: P73/INT3/T0IN pin		

ISL5 (bit 5): Base timer clock selection

ISL4 (bit 4):

These bits select the base timer input clock.

ISL5	ISL4	Base timer input clock
0	0	Fixed to subclock (crystal oscillator)

ISL3 (bit 3): Buzzer output frequency selection

This bit selects the buzzer output frequency as either fBST / 8 (1) or as fBST / 16 (0). When "1" is set, the signal that is output from the buzzer output pin (BUZ) has a frequency that is 1 / 8 that of the base timer input frequency; when "0" is set, the signal that is output has a frequency that is 1 / 16 that of the base timer input frequency.

ISL2 (bit 2): Noise elimination filter time constant selection

ISL1 (bit 1):

These bits select the noise elimination filter time constant.

ISL2	ISL1	Time constant
0	0	1Tcyc

The following table shows the signal and noise ranges for each time constant.

Time constant	Noise *1	Noise/signal *2	Signal *3
1Tcyc	< 1Tcyc	1Tcyc - 2Tcyc	2Tcyc <

- *1• A signal that does not meet the indicated time constant is deemed to be noise and is not accepted by the LSI.
- *2• A signal that falls within the indicated range for the time constant may be deemed to be noise and might not be accepted by the LSI.
- *3• A signal that exceeds the indicated time constant is deemed to be the correct signal and is accepted by the LSI.

ISL0 (bit 0):

T0 clock input pin selection

This bit selects the external signal input pin for the T0 as either P73/INT3/T0IN (1) or P72/INT2/T0IN (0). If "0" is set, P72/INT2/T0IN is selected as the external signal input pin.

• Timer 0 Prescaler Data Register (T0PRR)

The Timer 0 Prescaler Data Register sets the clock cycle for Timer/Counter 0; the clock cycle can be set to one of 256 levels through an 8-bit programmable counter.

For the 8-bit prescaler, the cycle clock signal is input directly. The clock cycle TPR for Timer/Counter 0 can be determined by setting the desired data in T0PRR (111H).

8-bit prescaler

$$:TPR = 1 \times Tcyc \times (256 - [T0PRR])(decimal)$$

Tcyc: Cycle clock cycle

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0PRR	111H	R/W	T0PRR7	T0PRR6	T0PRR5	T0PRR4	T0PRR3	T0PRR2	T0PRR1	T0PRR0
After reset			0	0	0	0	0	0	0	0

• Timer 0 Low register (T0L)

This is an 8-bit timer/counter. This timer/counter detects and counts either the clock from the prescaler or an external signal from P72/INT2/T0IN or P73/INT3/T0IN. When this register overflows, the T0L overflow flag is set.

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0L	112H	R	T0L7	T0L6	T0L5	T0L4	T0L3	T0L2	T0L1	T0L0
After reset			0	0	0	0	0	0	0	0

- Timer 0 Low Reload register (T0LR)

This is the reload register for Timer/Counter 0 Low (T0L). In 8-bit mode, the reload data is reloaded into T0L each time that T0L overflows and when T0LRUN = 0; in 16-bit mode, the reload data is reloaded into T0L each time that T0H overflows and when T0HRUN = 0.

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T0LR	113H	R/W	T0LR7	T0LR6	T0LR5	T0LR4	T0LR3	T0LR2	T0LR1	T0LR0
After reset			0	0	0	0	0	0	0	0

- Timer 0 High register (T0H)

This is an 8-bit timer/counter. This timer/counter detects and counts either the clock from the prescaler or T0L overflows. When this register overflows, the T0H overflow flag is set.

Symbol	Address	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T0H	114H	R	T0H7	T0H6	T0H5	T0H4	T0H3	T0H2	T0H1	T0H0
After reset			0	0	0	0	0	0	0	0

- Timer 0 High Reload register (T0HR)

This is the reload register for Timer/Counter 0 High (T0H). The reload data is reloaded into T0H each time that T0H overflows and when T0HRUN = 0.

Symbol	Address	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T0HR	115H	R/W	T0HR 7	T0HR6	T0HR5	T0HR4	T0HR3	T0HR2	T0HR1	T0HR0
After reset			0	0	0	0	0	0	0	0

- External Interrupt 2, 3 Control Register (I23CR)

This register sets external input signal detection and interrupts.

Regarding ISL0, refer to the "Input Signal Selection Register" (described later) for details.

3. Peripheral System Configuration

ISL0	I23CR7	I23CR6	I23CR3	I23CR2	External Signal Counting Condition
1	0	1	-	-	Count falling edge of P73/INT3/T0IN
1	1	0	-	-	Count rising edge of P73/INT3/T0IN
1	1	1	-	-	Count both edges of P73/INT3/T0IN
0	-	-	0	1	Count falling edge of P72/INT2/T0IN
0	-	-	1	0	Count rising edge of P72/INT2/T0IN
0	-	-	1	1	Count both edges of P72/INT2/T0IN
-	0	0	0	0	Do not count

Symbol	Address	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
I23CR	15EH	R/W	I23CR7	I23CR6	I23CR5	I23CR4	I23CR3	I23CR2	I23CR1	I23CR0
After reset			0	0	0	0	0	0	0	0

Bit name	Function
I23CR7 (bit 7)	INT3 rising edge detection control
	0: Do not detect 1: Detect
I23CR6 (bit 6)	INT3 falling edge detection control
	0: Do not detect 1: Detect
I23CR5 (bit 5)	INT3 interrupt source
	0: No interrupt source 1: Interrupt source
I23CR4 (bit 4)	INT3 interrupt request enable control
	0: Interrupt request disabled 1: Interrupt request enabled
I23CR3 (bit 3)	INT2 rising edge detection control
	0: Do not detect 1: Detect
I23CR2 (bit 2)	INT2 falling edge detection control
	0: Do not detect 1: Detect
I23CR1 (bit 1)	INT2 interrupt source
	0: No interrupt source 1: Interrupt source
I23CR0 (bit 0)	INT2 interrupt request enable control
	0: Interrupt request disabled 1: Interrupt request enabled

I23CR7 (bit 7):	<p>INT3 rising edge detection control</p> <p>This bit selects whether to detect (1) or not detect (0) the rising edge of the signal input to the P73/INT3/T0IN pin. When this bit is set to "1," I23CR5 is set (1) when the rising edge of the input signal is detected; if the INT3 interrupt request is enabled (I23CR4 = 1), then an interrupt is generated. When this bit is set to "0," the rising edge of the signal is not detected.</p>
I23CR6 (bit 6):	<p>INT3 falling edge detection control</p> <p>This bit selects whether to detect (1) or not detect (0) the falling edge of the signal input to the P73/INT3/T0IN pin. When this bit is set to "1," I23CR5 is set (1) when the falling edge of the input signal is detected; if the INT3 interrupt request is enabled (I23CR4 = 1), then an interrupt is generated. When this bit is set to "0," the falling edge of the signal is not detected.</p>
I23CR5 (bit 5):	<p>INT3 interrupt source</p> <p>This bit is set if the conditions specified by I23CR7 or I23CR6 are met; if the INT3 interrupt request is enabled, then control jumps to vector address 001BH and interrupt processing begins. The value of this bit does not change even when interrupt processing is completed. Therefore, this bit must be reset by software.</p>
I23CR4 (bit 4):	<p>INT3 interrupt request enable</p> <p>This bit enables (1) or disables (0) external interrupt 3 (INT3). When this bit is set to "1," and I23CR5 has been set, then INT3 interrupt processing is executed. When this bit is set to "0," interrupt processing is not executed.</p>
I23CR3 (bit 3):	<p>INT2 rising edge detection control</p> <p>This bit selects whether to detect (1) or not detect (0) the rising edge of the signal input to the P72/INT2/T0IN pin. When this bit is set to "1," I23CR1 is set (1) when the rising edge of the input signal is detected; if the INT2 interrupt request is enabled (I23CR0 = 1), then an interrupt is generated. When this bit is set to "0," the rising edge of the signal is not detected.</p>
I23CR2 (bit 2):	<p>INT2 falling edge detection control</p> <p>This bit selects whether to detect (1) or not detect (0) the falling edge of the signal input to the P72/INT2/T0IN pin. When this bit is set to "1," I23CR1 is set (1) when the falling edge of the input signal is detected; if the INT2 interrupt request is enabled (I23CR0 = 1), then an interrupt is generated. When this bit is set to "0," the falling edge of the signal is not detected.</p>
I23CR1 (bit 1):	<p>INT2 interrupt source</p> <p>This bit is set if the conditions specified by I23CR3 or I23CR2 are met; if the INT2 interrupt request is enabled, then control jumps to vector address 0013H and interrupt processing begins. The value of this bit does not change even when interrupt processing is completed. Therefore, this bit must be reset by software.</p>

I23CR0 (bit 0):

INT2 interrupt request enable control

This bit enables (1) or disables (0) external interrupt 2 (INT2). When this bit is set to "1," and I23CR1 has been set, then INT2 interrupt processing is executed. When this bit is set to "0," interrupt processing is not executed.

Note:

- Edge detection is not performed if I23CR7 and 6 or I23CR3 and 2 are both "0." If both of either pair are "1," then both edges are detected.
- Input from the P73/INT3/T0IN pin is connected to the noise filter.

2.5 Circuit Configuration and Description of Operation

- Timer 0 mode settings

Mode	T0LONG	T0LEXT
0	0	0
1	0	1
2	1	0
3	1	1

- Mode 0: 8-bit reload timer x 2 channels, with programmable prescaler

In mode 0, Timer 0 functions as two 8-bit reload timers. The relationship between the timer value and the reload register (T0LR) setting is as shown below.

Time until T0HOVF is set (1) (decimal) = (256 - T0HR setting) ¥ TPR

Time until T0LOVF is set (1) (decimal) = (256 - T0LR setting) ¥ TPR

TPR: Clock cycle from prescaler

Once the count control bit (T0HRUN, T0LRUN) is set, the counting operation starts. If the bit is reset, the counting operation stops, and the contents of the reload register (T0HR, T0LR) are transferred to the counter (T0H, T0L).

If Timer / Counter 0 (T0H, T0L) overflows, the overflow flag (T0HOVF, T0LOVF) is set, and the contents of the reload register (T0HR, T0LR) are transferred to the counter (T0H, T0L).

If both the overflow flag (T0HOVF, T0LOVF) and the interrupt request enable flag (T0HIE, T0LIE) are set, an interrupt request is sent to the interrupt control circuit.

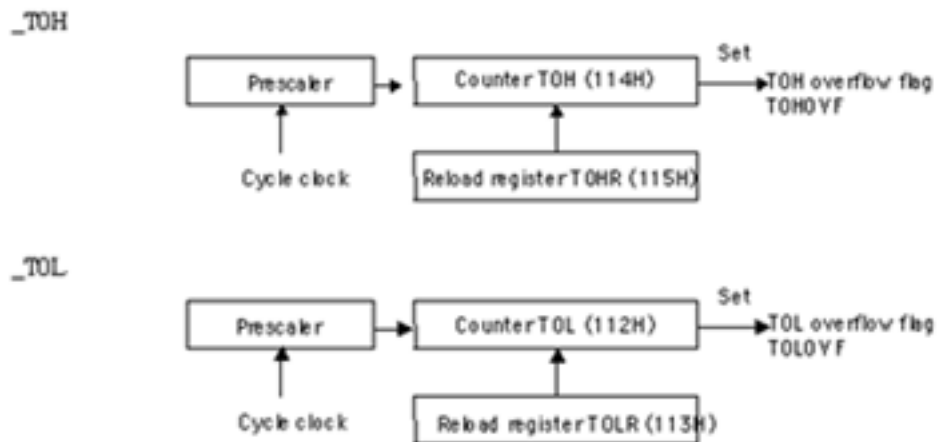
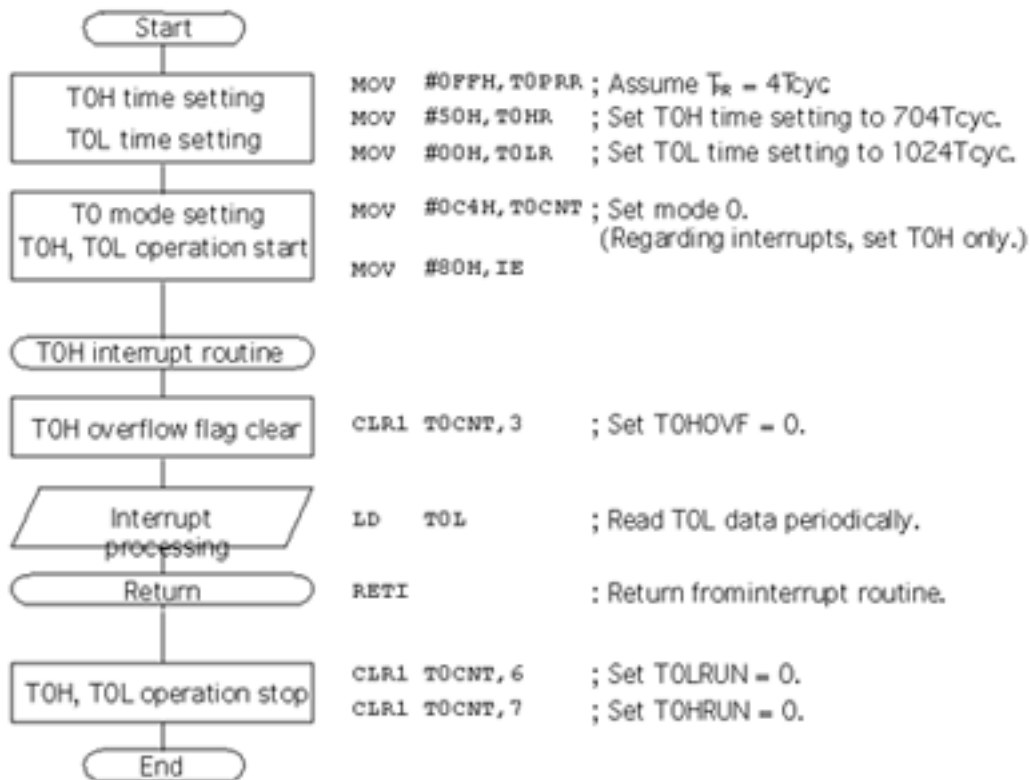


Figure 3.7 Circuit Configuration for Mode 0: 8-bit Reload Timer x 2 Channels

- Mode 0 Program Example



- Mode 1: 8-bit reload timer with programmable prescaler + 8-bit reload counter
- 8-bit reload timer

The upper 8 bits of Timer 0 (T0H) operate as an 8-bit reload timer. The relationship between the timer value and the reload register (T0HR) setting is as shown below.

$$\text{Time until T0HOVF is set (1) (decimal)} = (256 - \text{T0HR setting}) \times \text{TPR}$$

TPR: Cycle of clock from prescaler

The data in the reload register is loaded into the counter T0H at each interval at which T0HOVF is set. In addition, the timer operation continues until the T0H count control bit (T0HRUN) is reset. The operation method is the same as for mode 0.

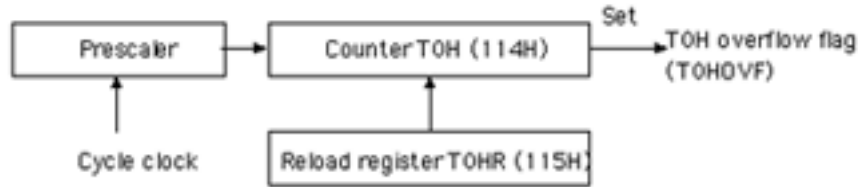


Figure 3.8 Block Diagram for Mode 1: 8-bit Reload Timer (T0H)

- 8-bit reload counter

The lower 8 bits of Timer 0 (T0L) are used to count the signals input through the external pin. The external signal is filtered through a noise filter. For details, refer to Chapter 3, section 3.2.4, "Related Registers and Input Signal Selection Register (ISL)."

The relationship between the counted value and the reload register (T0LR) setting is as follows:

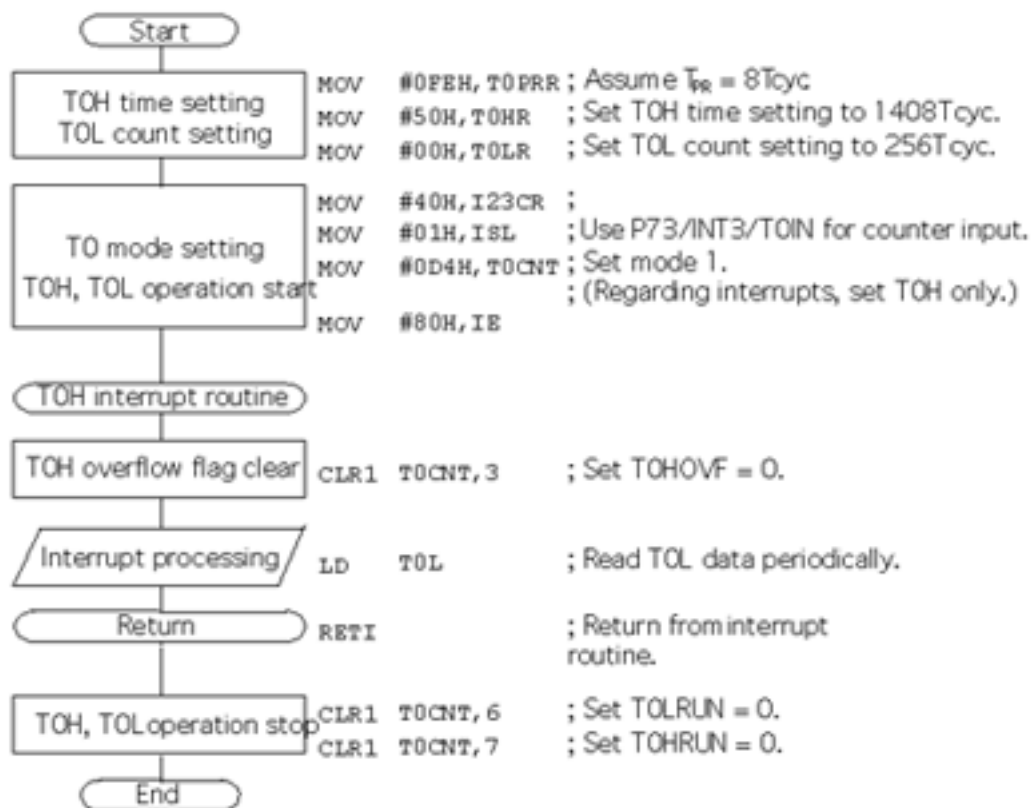
$$\text{Counted value until T0LOVF is set (1) (decimal)} = 256 - (\text{T0LR setting})$$

If the T0L overflow flag (T0LOVF) is set, the data in the reload register T0LR is transferred to the counter T0L. In addition, counting continues until the T0L count control bit (T0LRUN) is reset.



Figure 3.9 Block Diagram for Mode 1: 8-bit Reload Counter (T0L)

- Mode 1 Program Example



- Mode 2: 16-bit reload timer with programmable prescaler

Mode 2 connects T0H and T0L in series and uses them as a 16-bit timer.

To start the timer, set the counter control bits (T0HRUN and T0LRUN) for both T0H and T0L simultaneously.

The relationship between the timer value and the reload register (T0HR, T0LR) settings is as follows:

Time until T0HVOF is set (1) (decimal)

$$= (65,536 - 256 \times (\text{T0HR setting}) - (\text{T0LR setting})) \times \text{TPR}$$

TPR: Cycle of clock from prescaler

T0LOVF is set at the same time as T0HOVF, and each time that T0HOVF is generated, the reload data (T0LR, T0HR) is transferred to T0L and T0H, respectively. Timer operation continues until the count control bit is reset. The operation method is the same as for mode 0.

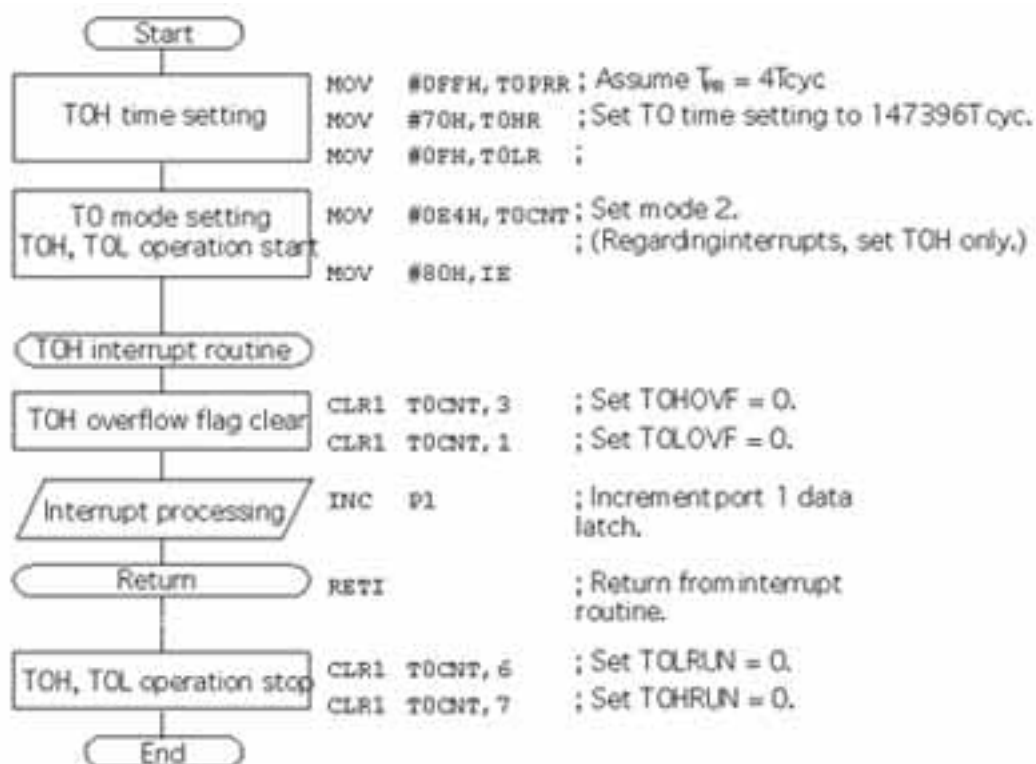
Read the Timer 0 (T0) data according to the following procedure:

T0L	LD	T0L	; Read T0L data (1).
↓	ST	020H	
T0H	LD	T0H	; Read T0H data.
↓	ST	021H	
T0L	LD	T0L	; Read T0L (2) data again.
↓	BP	T0L,7,DES	; When bit 7 of T0L (2) is "0" and
	BN	020H,7,DES	; bit 7 of T0L (1) is "1"...
	ST	020H	
T0H	LD	T0H	; Read T0H (2).
	ST	021H	
	DES:	_ next program	



Figure 3.10 Block Diagram for Mode 2: 16-bit Reload Timer

- Mode 2 Program Example



- Mode 3: 16-bit reload counter

Mode 3 connects T0H and T0L in series and uses them as a 16-bit counter. The clock is an external signal that is input from either P72/INT2/T0IN or P73/INT3/T0IN. The external input pin is selected through the special function register ISL. A noise elimination filter is connected to the P73/INT3/T0IN pin.

To start the counter, set the counter control bits (T0HRUN and T0LRUN) for both T0H and T0L simultaneously.

The relationship between the counted value and the reload register (T0HR, T0LR) settings is as follows:

Count until T0HVOF is set (1) (decimal)

$$= 65,536 - 256 \times (\text{T0HR setting}) - (\text{T0LR setting})$$

T0LOVF is set at the same time as T0HOVF, and each time that T0HOVF is generated, the reload data (T0LR, T0HR) is transferred to T0L and T0H, respectively. Timer operation continues until the count control bit is reset. The operation method is the same as for mode 0.

Read the Timer 0 (T0) data according to the following procedure:

T0L	LD	T0L	; Read T0L data (1).
↓	ST	020H	
T0H	LD	T0H	; Read T0H data.
↓	ST	021H	
T0L	LD	T0L	; Read T0L (2) data again.
↓	BP	T0L,7,DES	; When bit 7 of TOL (2) is "0" and
	BN	020H,7,DES	; bit 7 of TOL (1) is "1"...
	ST	020H	
T0H	LD	T0H	; Read T0H (2).
	ST	021H	
	DES:	_ next program	

- Mode 3 Program Example

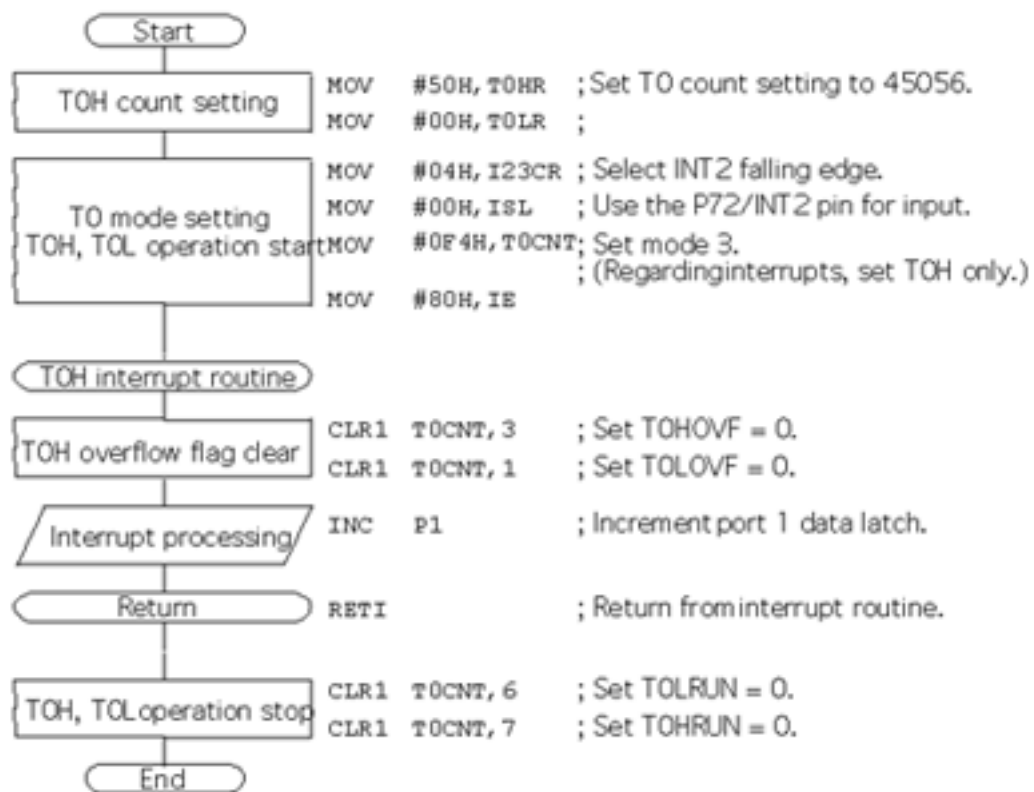


Figure 3.11 Block Diagram for Mode 3: 16-bit Reload Counter

3. Timer 1 (T1)

3.1 Overview

POTATO's built-in Timer 1 (T1) is a 16-bit timer that has the four functions listed below.

- Mode 0: 8-bit reload timer x 2 channels
- Mode 1: 8-bit reload timer + 8-bit reload counter
- Mode 2: 16-bit reload timer
- Mode 3: Variable bit length pulse generator (9 to 16 bits)

3.2 Functions

- 8-bit reload timer x 2 channels (mode 0)

T1 operates as two independent 8-bit reload timers (T1H, T1L), using as the clock the signal that is generated once each cycle (cycle clock) while executing an instruction.

- 8-bit reload timer + 8-bit reload counter (mode 1)

T1H operates as an 8-bit reload timer using the cycle clock. T1L operates as an 8-bit pulse generator. The pulse signal is output from the P17/pulse signal output pin.

- 16-bit reload timer (mode 2)

T0 operates as a 16-bit reload timer using the T1L overflow signal as the clock for T1H. The input clock for T1L is the cycle clock. Each time that T1L generates an overflow, the T1LR reload data is reloaded in T1L; the same applies to T1H.

Either the cycle clock or the cycle clock divided by 2 is used as the clock for T1L.

- Variable bit length pulse generator (9 to 16 bits) (mode 3)

T1L and T1H are used to generate a pulse signal of 9 to 16 bits. The pulse signal is output from the P17/pulse signal output pin.

Either the cycle clock or the cycle clock divided by 2 is used as the clock for T1L.

- Interrupt generation

When the interrupt request enable bit is set, T1H and T1L interrupt requests are generated when the T1H and T1L registers overflow.

The following Special Function Registers must be manipulated in order to control Timer 1 (T1).

<input type="checkbox"/> ET1H	<input type="checkbox"/> ET1HR	<input type="checkbox"/> ET1HC
<input type="checkbox"/> ET1L	<input type="checkbox"/> ET1LR	<input type="checkbox"/> ET1LC
<input type="checkbox"/> ET1CNT	<input type="checkbox"/> EP1DDR	<input type="checkbox"/> EP1FCR
<input type="checkbox"/> EP1		

3.3 Circuit Configuration

The configuration of Timer 1 (T1) is shown in Fig. 3-3-1.

- Timer 1 Low (T1L) ▸

This is an 8-bit reload timer that uses the cycle clock or the cycle clock divided by 2 as a clock. The T1LR data is reloaded upon a T1L overflow; the T1LR data is also transferred to T1L when T1LRUN (bit 6 of T1CNT) is set to "0."

- Timer 1 Low Compare Circuit (T1LC) ▸

This circuit consists of an 8-bit Timer 1 Low Compare Data register (T1LC) and an 8-bit data compare circuit. This circuit compares the data in T1L and T1LC.

- Timer 1 High (T1H) ®

This is an 8-bit reload timer that uses the cycle clock or the T1L overflow signal as a clock. The T1HR data is reloaded upon a T1H overflow; the T1HR data is reloaded even if T1HRUN (bit 7 of T1CNT) is reset.

- Timer 1 High Compare Circuit (T1HC) ▸

This circuit consists of an 8-bit Timer 1 High Compare Data register (T1HC) and an 8-bit data compare circuit. This circuit compares the data in T1H and T1HC.

- Timer 1 Control Register (T1CNT) °

This register sets the modes for T1 and controls interrupts.

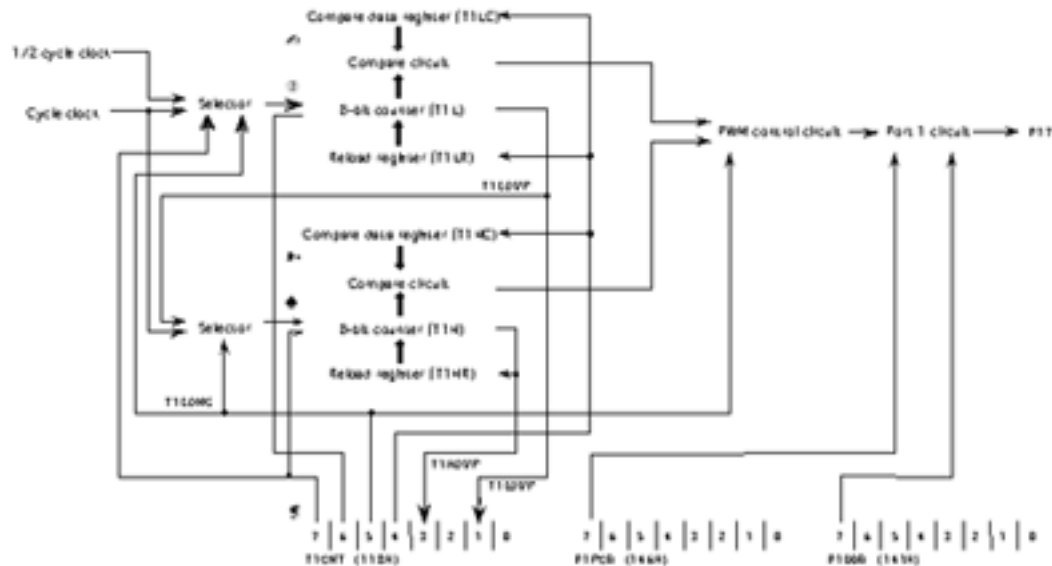


Figure 3.12 Timer 1 Block Diagram

3.4 Related Registers

- Timer 1 Control Register (T1CNT)

Symbol	Address	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
T1CNT	118H	R/W	T1HRUN	T1LRUN	T1LONG	ELDT1C	T1HOVF	T1HIE	T1LOVF	T1LIE
After reset			0	0	0	0	0	0	0	0

Bit name	Function
T1HRUN (bit 7)	T1H count control
	0: Count stop/data reload 1: Count start
T1LRUN (bit 6)	T1L count control
	0 Count stop/data reload 1: Count start
T1LONG (bit 5)	Timer 1 bit length selection
	0: 8 bits 1: 16 bits
ELDT1C (bit 4)	Pulse generator data update enable control
	0: Disabled 1: Enabled
T1HOVF (bit 3)	T1H overflow flag
	0: No overflow flag 1: Overflow flag
T1HIE (bit 2)	T1H interrupt request enable control
	0: Interrupt request disabled 1: Interrupt request enabled
T1LOVF (bit 1)	T1L overflow flag
	0: No overflow flag 1: Overflow flag
T1LIE (bit 0)	T1L interrupt request enable
	0: Interrupt request disabled 1: Interrupt request enabled

T1HRUN (bit 7):	<p>T1H count control</p> <p>This bit starts (1)/stops (0) Timer 1 High (T1H). When this bit is set to "1," the clock is input to T1H, which begins counting; when this bit is set to "0," the clock supplied to T1H is stopped and the reload data (T1HR) is transferred to T1H simultaneously.</p>
T1LRUN (bit 6):	<p>T1L count control</p> <p>This bit starts (1)/stops (0) Timer 1 Low (T1L). When this bit is set to "1," the clock is input to T1L, which begins counting; when this bit is set to "0," the clock supplied to T1L is stopped and the reload data (T1LR) is transferred to T1L simultaneously.</p>
T1LONG (bit 5):	<p>Timer 1 bit length selection</p> <p>This bit selects the T1 bit length as either 16 bits (1) or 8 bits (0). When this bit is set to "1," the bit length of Timer 1 is 16 bits; when this bit is set to "0," the bit length is set to 8 bits. Set this bit to "1" when using modes 2 and 3, and to "0" when using modes 0 and 1.</p>
ELDT1C (bit 4):	<p>Pulse generator data update enable control</p> <p>This bit enables (1) or disables (0) transfer of the compare data register (T1HC, T1LC) data that is used to generate the pulse signal to the compare circuit. When this bit is set to "1," the data is transferred to the compare circuit, and updated with the new pulse generator data; if this bit is set to "0," the data is not updated and the same pulse generator data is output.</p> <p>To update 16-bit data simultaneously, set this bit to "0," set the data for each 8 bits, and then set this bit to "1."</p>
T1HOVF (bit 3):	<p>T1H overflow flag</p> <p>This flag is set when an overflow occurs in T1H, and does not change when no overflow occurs. Therefore, this flag must be reset by software.</p>
T1HIE (bit 2):	<p>T1H interrupt request enable control</p> <p>This bit enables (1)/disables (0) interrupt requests due to a T1H overflow. When this bit is set to "1," an interrupt request is generated to vector address 002BH in response to a T1H overflow; when this bit is set to "0," no interrupt request is generated.</p>
T1LOVF (bit 1):	<p>T1L overflow flag</p> <p>This flag is set when an overflow occurs in T1L, and does not change when no overflow occurs. Therefore, this flag must be reset by software. T1LOVF is set whenever an overflow occurs in T1L, regardless of the bit length.</p>
T1LIE (bit 0):	<p>T1L interrupt request enable control</p> <p>This bit enables (1)/disables (0) interrupt requests due to a T1L overflow. When this bit is set to "1," an interrupt request is generated to vector address 002BH in response to a T1L overflow; when this bit is set to "0," no interrupt request is generated. In 16-bit mode, no interrupt request is generated in response to a T1L overflow.</p>

Note:

- T1HOVF and T1LOVF must be set to "0" by software.
- When operating in 16-bit mode, select either the cycle clock or the cycle clock divided by 2 as the clock. ("Ttc" is the clock cycle.)

Ttc= Tcyc ☐F T1HRUN=1☐CT1LRUN=1☐CT1LONG=1
Ttc= 1/2Tcyc ☐F T1HRUN=0☐CT1LRUN=1☐CT1LONG=1

- Timer 1 Low Register (T1L)

The Timer 1 Low register is an 8-bit timer. This register uses either the cycle clock or the cycle clock divided by 2 as its clock.

When T1L overflows, the T1LR data is transferred and the T1L overflow flag is set. Note that in modes 1 and 3, this register is used to generate pulse signals.

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1L	11BH	R	T1L7	T1L6	T1L5	T1L4	T1L3	T1L2	T1L1	T1L0
After reset			0	0	0	0	0	0	0	0

- Timer 1 Low Reload Register (T1LR)

The Timer 1 Low Reload register is the reload register for the Timer 1 Low (T1L) register.

Each time that T1L overflows and T1LRUN = 0, the reload data is reloaded into T1L. Note that in modes 1 and 3, this register is used to generate pulse signals.

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1LR	11BH	W	T1LR7	T1LR6	T1LR5	T1LR4	T1LR3	T1LR2	T1LR1	T1LR0
After reset			0	0	0	0	0	0	0	0

T1L and T1LR share the same address. T1L is a read-only register, and T1LR is a write-only register. It is essential to note that if a bit manipulation instruction, the INC instruction, the DEC instruction, or the DBNZ instruction is used to write data to the write-only register, bits other than the specified bits will be set. The following instructions are used with T1LR.

- MOV • MOV @
- ST • ST @
- POP

- Timer 1 Low Compare Data Register (T1LC)

This is the compare data register for the Timer 1 Low (T1L) register.

If ELDT1C (bit 4 of T1CNT) is set, the data that is set in this register is transferred to the pulse generator control circuit (compare circuit) the next time that T1L overflows (if T1LONG = 0) or the next time that T1H overflows (if T1LONG = 1). When T1LRUN = 0, the value of T1LC is always transferred to the pulse generator control circuit.

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1LC	11AH	R/W	T1LC7	T1LC6	T1LC5	T1LC4	T1LC3	T1LC2	T1LC1	T1LC0
After reset			0	0	0	0	0	0	0	0

- Timer 1 High Register (T1H)

The Timer 1 High register is an 8-bit timer. This register operates either according to the cycle clock or overflows in T1L (T1LOVF).

When T1H overflows, the T1H overflow flag is set. Note that in mode 3, this register is used to generate pulse signals.

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1H	11DH	R	T1H7	T1H6	T1H5	T1H4	T1H3	T1H2	T1H1	T1H0
After reset			0	0	0	0	0	0	0	0

- Timer 1 High Reload Register (T1HR)

The Timer 1 High Reload register is the reload register for the Timer 1 High (T1H) register.

Each time that T1H overflows and T1HRUN = 0, the reload data is reloaded into T1H. Note that in mode 3, this register is used to generate pulse signals.

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1HR	11DH	W	T1HR7	T1HR6	T1HR5	T1HR4	T1HR3	T1HR2	T1HR1	T1HR0
After reset			0	0	0	0	0	0	0	0

T1H and T1HR share the same address. T1H is a read-only register, and T1HR is a write-only register. It is essential to note that if a bit manipulation instruction, the INC instruction, the DEC instruction, or the DBNZ instruction is used to write data to the write-only register, bits other than the specified bits will be set. The following instructions are used with T1HR.

- MOV
- ST
- POP
- MOV @
- ST @

- Timer 1 High Compare Data Register (T1HC)

This is the compare data register for the Timer 1 High (T1H) register.

The data that is set in this register is transferred to the pulse generator control circuit (compare circuit) according to the same timing as T1LC.

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1HC	11CH	R/W	T1HC7	T1HC6	T1HC5	T1HC4	T1HC3	T1HC2	T1HC1	T1HC0
After reset			0	0	0	0	0	0	0	0

3.5 Circuit Configuration and Description of Operation

- Timer 1 Mode Settings

Mode	Clock Cycle	T1LONG	P17FCR	P17DDR	P17
0	Tcyc	0	0	X	X
1	Tcyc	0	1	1	0
2	Tcyc, 1/2Tcyc	1	0	X	X
3	Tcyc, 1/2Tcyc	1	1	1	0

- Mode 0: 8-bit reload timer x 2 channels

When in mode 0, Timer 1 functions as two 8-bit reload timers. The relationship between the timer values and the value set in the reload register (T1LR) is as shown below.

$$\text{Time until T1HOVF is set (1) (decimal)} = (256 - \text{T1HR setting}) \times \text{Tcyc}$$

$$\text{Time until T1LOVF is set (1) (decimal)} = (256 - \text{T1LR setting}) \times \text{Tcyc}$$

Tcyc: Cycle clock cycle

If the counter control bit (T1HRUN, T1LRUN) is set, the counting operation starts; if the bit is reset, the counting operation stops and the contents of the reload register (T1HR, T1LR) are transferred to the counter (T1H, T1L). If Timer 1 (T1H, T1L) overflows, the overflow flag (T1HOVF, T1LOVF) is set, and the contents of the reload register (T1HR, T1LR) are transferred to the counter (T1H, T1L).

In addition, if the overflow flag (T1HOVF, T1LOVF) and the interrupt request enable flag (T1HIE, T1LIE) are both set, then an interrupt request is sent to the interrupt control circuit.

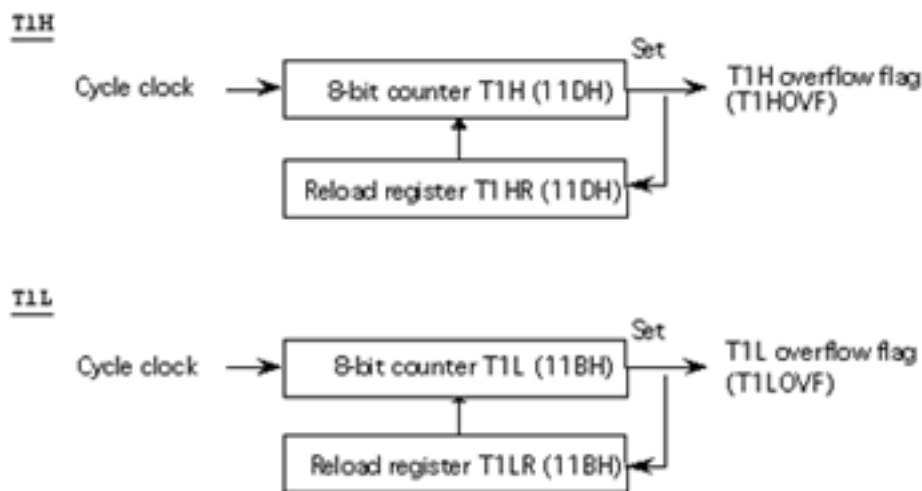
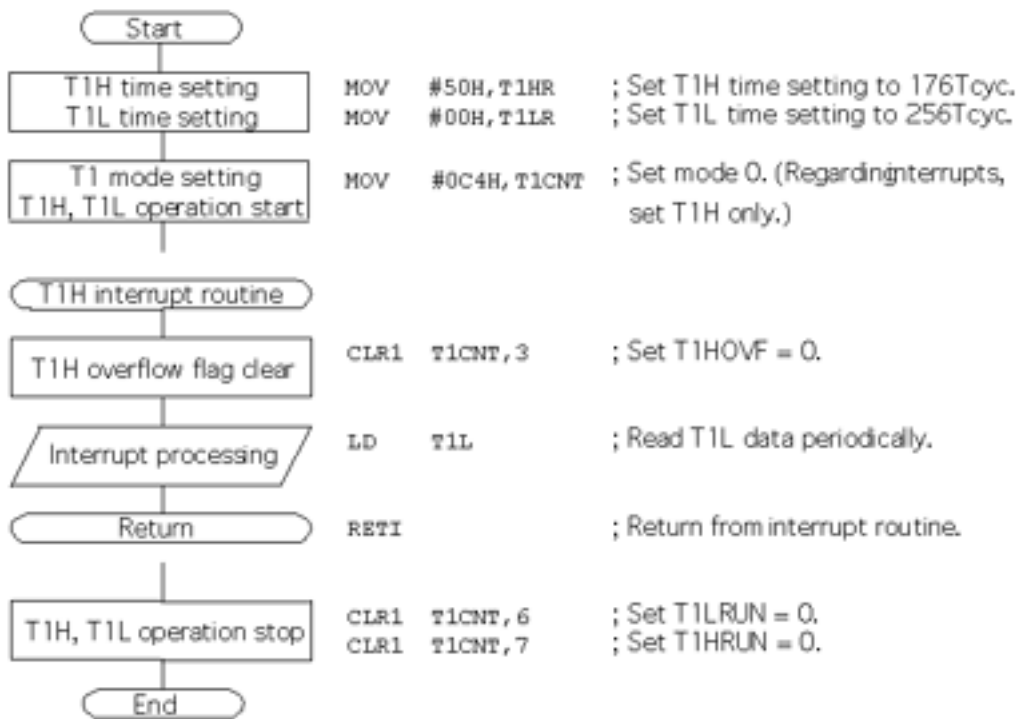


Figure 3.13 Circuit Configuration of 8-bit Reload Timer x 2 Channels

- Mode 0 Program Example



- Mode 1: 8-bit reload timer + 8-bit pulse generator
- 8-bit reload timer

The upper 8 bits of Timer 1 operate as an 8-bit reload timer. The relationship between the timer value and the reload register (T1HR) setting is as shown below.

$$\text{Time until T1HOVF is set (1) (decimal)} = (256 - \text{T1HR setting}) \times \text{Tcyc}$$

Tcyc : Cycle clock cycle

The data in the reload register is loaded into the counter T1H at each interval at which T1HOVF is set. In addition, the timer operation continues until the T1H count control bit (T1HRUN) is reset. The operation method is the same as for mode 0.

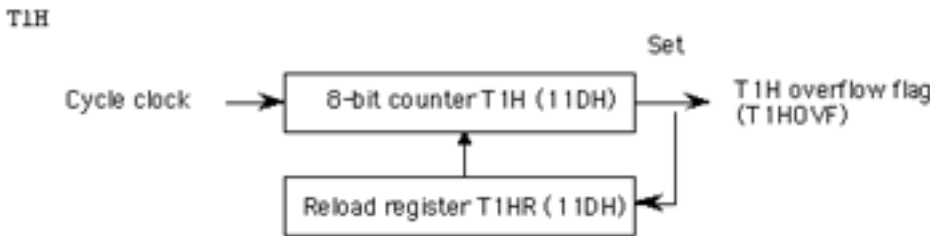


Figure 3.14 Block Diagram for Mode 1: 8-bit Reload Timer (T1H)

- 8-bit pulse generator

The compare circuit compares the value of T1L, which was counting according to the cycle clock starting from the reloaded value, with the value in the compare data register T1LC. This circuit outputs a "0" until the values match, at which point it outputs a "1;" this output continues until T1L overflows.

The pulse signal cycle is determined by the reload register T1LR. The relationship between the counter value and the pulse output waveform is shown in Fig. 3-4.

The pulse output waveform is determined by the value of the compare data register T1LC and the reload register T1LR. There is a delay in the pulse signal cycle from when the compare data register T1LC is overwritten until the pulse output according to that data is obtained.

Each time that T1L overflows, the T1L overflow flag (T1LOVF) is set. the relationship with the pulse output signal is as shown below.

Pulse output signal low level pulse width (decimal)

$$= (T1LC \text{ setting} - T1LR \text{ setting}) \times T_{cyc}$$

Pulse output signal cycle (decimal) = $(256 - T1LR \text{ setting}) \times T_{cyc}$

T_{cyc} : Cycle clock cycle

Note:

- Programs must be written in such a manner that $T1LC \geq T1LR$ is always true.
-

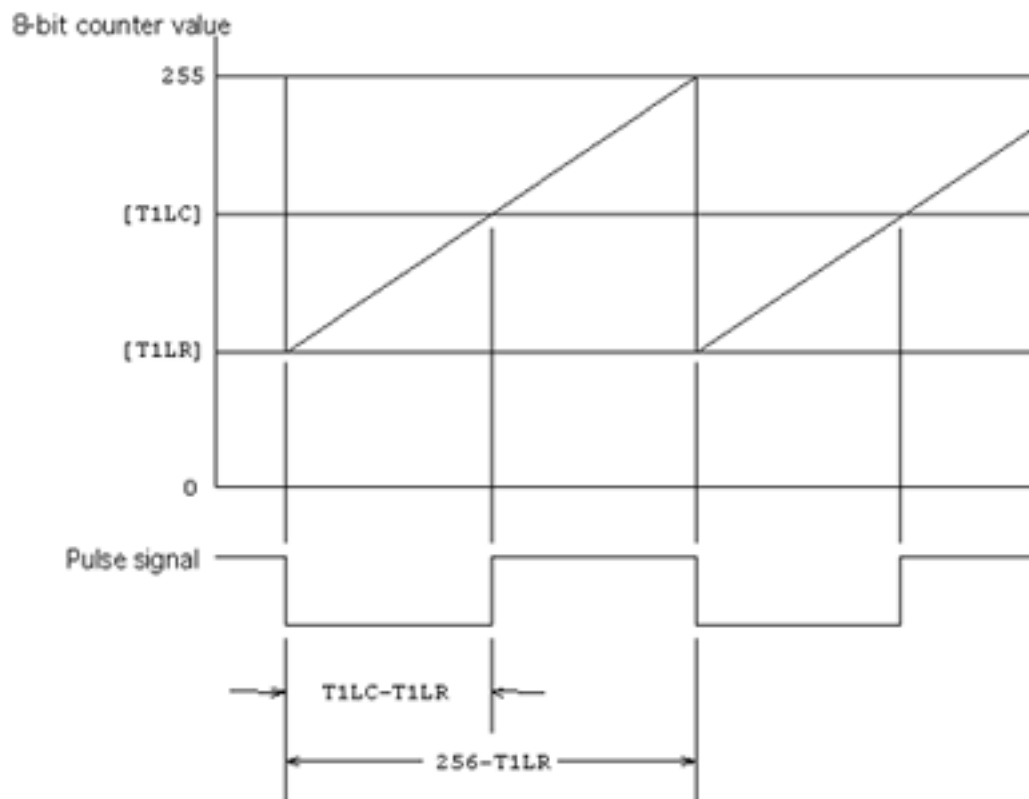


Figure 3.15 Relationship Between Counter Value and Pulse Generator Output Waveform

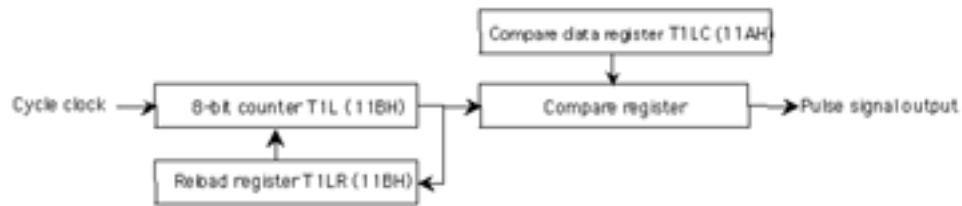
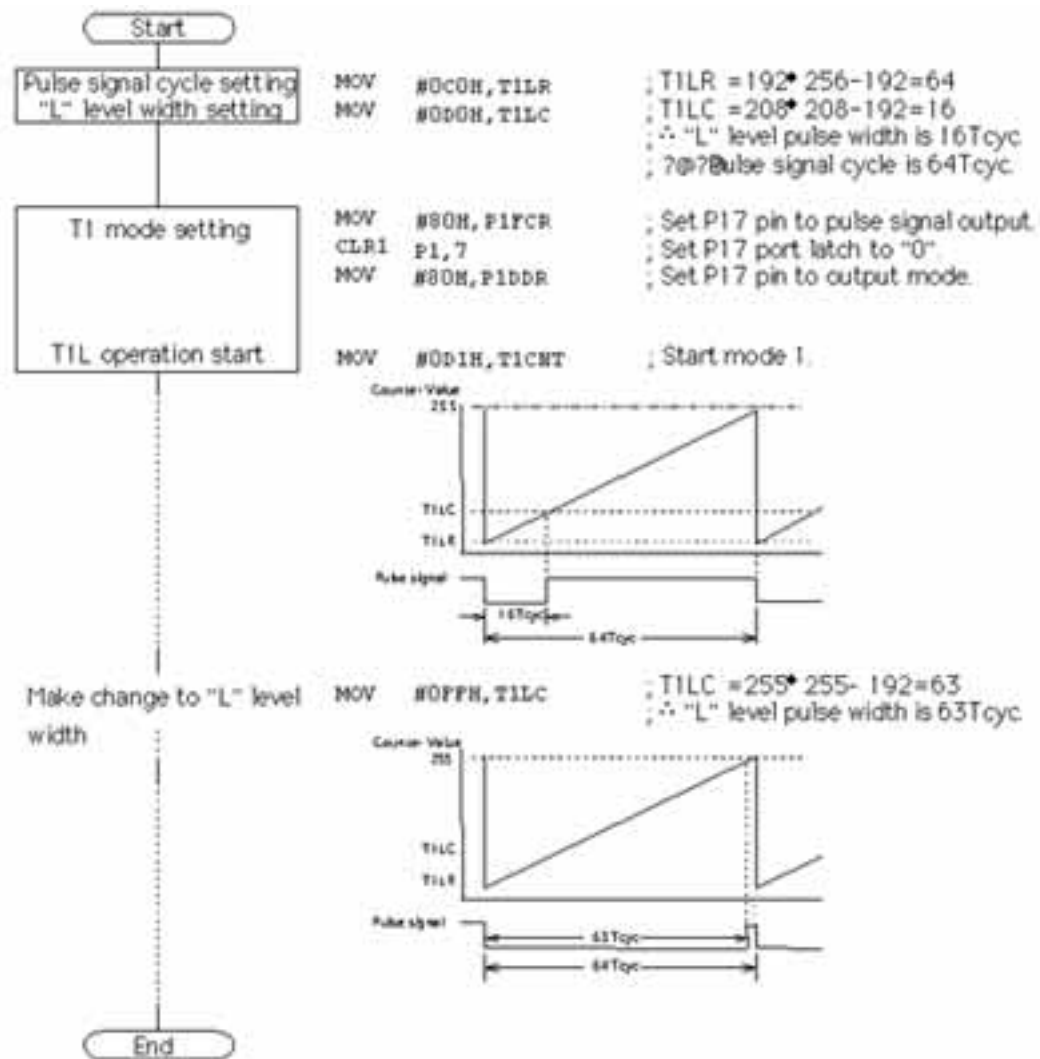


Figure 3.16 Block Diagram for Mode 1: 8-bit Pulse Generator

- Mode 1 (Pulse Output) Program Example



- Mode 2: 16-bit reload timer

In this mode, T1 operates as a 16-bit reload timer. To start the timer, set T1LRUN and T1LONG simultaneously. Use the MOV instruction to set these bits.

Either the cycle clock (Tcyc) or the cycle clock divided by 2 ($1/2T_{cyc}$) can be selected for the T1L clock cycle (Ttc). The settings are as shown below.

Ttc = Tcyc	: T1HRUN=1 □ CT1LRUN=1, T1LONG =1
Ttc = $1/2T_{cyc}$: T1HRUN=0 □ CT1LRUN=1, T1LONG =1

The relationship between the timer value and the value set in the reload registers (T1HR, T1LR) is as shown below. It is important to note that these relationships differ from those of Timer/Counter 0 (T0).

Time until T1HOVF is set (1) (decimal)

$$= (256 - T1HR \text{ setting}) \times (256 - T1LR \text{ setting}) \times Ttc$$

Time until T1LOVF is set (1) (decimal) = $(256 - T1LR \text{ setting}) \times Ttc$

Ttc: T1L clock cycle (Tcyc or $1/2 T_{cyc}$)

Each time that T1LOVF is generated, the reload data (T1LR) is transferred to T1L; each time that T1HOVF is generated, the reload data (T1HR) is transferred to T1H. Counting continues until the count control bit is reset. The operation method is the same as for mode 0.

Read the Timer 1 (T1) data according to the following procedure:

T1L	LD	T1L	; Read T1L data (1).
Ø	ST	020H	
T1H	LD	T1H	; Read T1H data.
Ø	ST	021H	
T1L	LD	T1L	; Read T1L (2) data again.
Ø	BP	T1L,7,DES	; When bit 7 of T1L (2) is "0" and
	BN	020H,7,DES	; bit 7 of T1L (1) is "1"...
	ST	020H	
T1H	LD	T1H	: Read T1H (2).
	ST	021H	

DES: · · next program

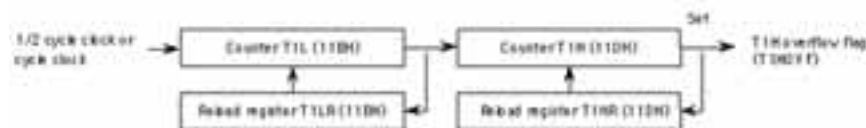
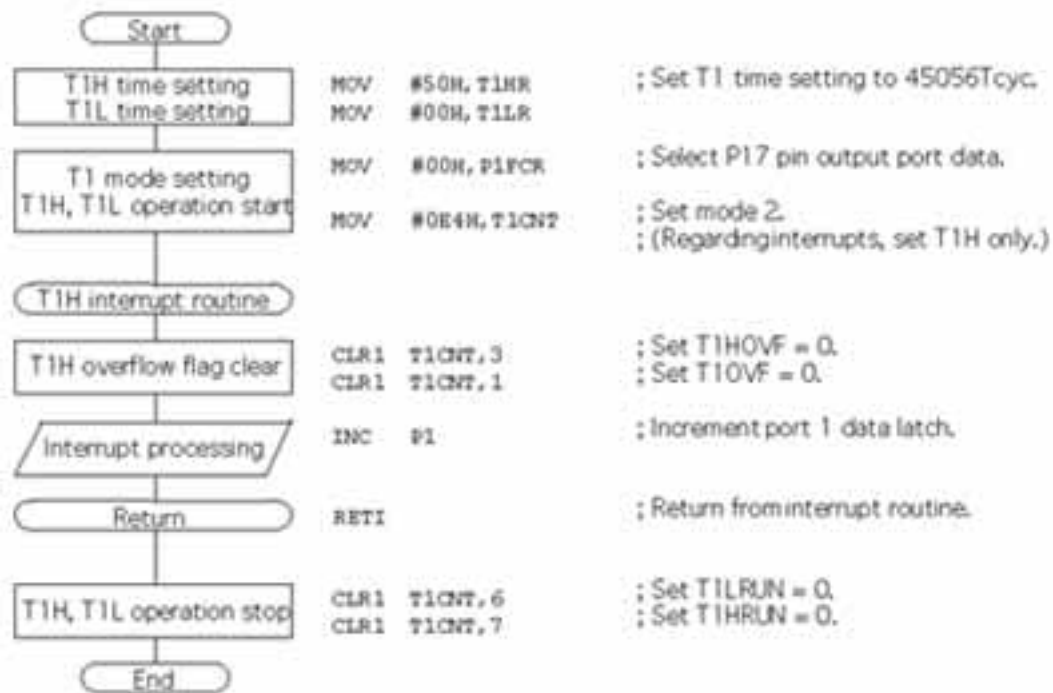


Figure 3.17 Block Diagram for Mode 2: 16-bit Reload Timer

- Mode 2 Program Example



- Mode 3: Variable bit length pulse generator (9 to 16 bits)

In mode 3, Timer 1 (T1L, T1H) functions as a variable bit length pulse generator. The length can vary from 9 to 16 bits, set by T1HR.

In order to run the pulse generator, select "16 bits" (T1LONG = 1) for the Timer 1 bit length and set the T1L count control bits (T1LRUN). If the length has been selected as 16 bits, then the control bit T1LRUN controls starting and stopping for all 16 bits. Use the MOV instruction in order to set the Timer 1 Control Register (T1CNT) bits simultaneously.

Either the cycle clock (Tcyc) or the cycle clock divided by 2 (1/2Tcyc) can be selected as the clock cycle (Ttc) for the pulse generator. The settings are as shown below.

Ttc = Tcyc : T1HRUN = 1, T1LRUN = 1, T1LONG = 1

Ttc = 1/2Tcyc : T1HRUN = 0, T1LRUN = 1, T1LONG = 1

Each time that T1L overflows, the T1L overflow flag (T1LOVF) is set; each time that T1H overflows, the T1H overflow flag (T1HOVF) is set. Counting continues until the count control bit is reset.

The relationship between the timer value and the value set in the reload registers (T1HR, T1LR) is as follows.

Time until T1HOVF is set (1) (decimal)

$$= (256 - \text{T1HR setting}) \times (256 - \text{T1LR setting}) \times \text{Ttc}$$

Time until T1LOVF is set (1) (decimal) = (256 - T1LR setting) × Ttc

Ttc: T1L clock cycle (Tcyc or 1/2 Tcyc)

The figure below shows an example of a signal that is output from the P17/pulse signal output pin in mode 3.

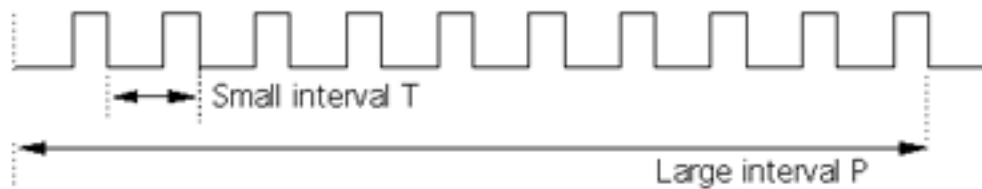


Figure 3.18 Mode 3 Pulse Signal Output Waveform

The output signal repeats large interval P, in which small interval T is repeated 256 times.

The number of times that small interval T is repeated is set by T1HR. The "L" level width in small interval T is set by T1LC in the same manner as in mode 1, with the minimum unit being Ttc. In addition, the total "L" level width [sigma]TL within the large interval P is set by T1LC and T1HC. In addition, the data that can be acquired by T1HC is limited by the value of T1HR.

For details on the relationship between the output waveform and T1HC and T1LC, refer to Appendix 1, "Variable Bit Length Pulse Generator."

The relationship between the pulse generator bit length and the values of T1LR and T1HR, and the values of T1LC and T1HC are shown in Table below. T1LR is set to 00H.

Table 2.5 Relationship Between Bit Length and T1H/L Register

Pulse	Pulse bit length setting (binary)		(deleted) "L" level pulse width setting (binary)	
Bit length	Value of T1HR	Value of T1LR	Value of T1LC (upper bits)	Value of T1HC (lower bits)
16	0000 0000	0000 0000	XXXX XXXX	XXXX XXXX
15	1000 0000	0000 0000	XXXX XXXX	XXXX XXX0
14	1100 0000	0000 0000	XXXX XXXX	XXXX XX00
13	1110 0000	0000 0000	XXXX XXXX	XXXX X000
12	1111 0000	0000 0000	XXXX XXXX	XXXX 0000
11	1111 1000	0000 0000	XXXX XXXX	XXX0 0000
10	1111 1100	0000 0000	XXXX XXXX	XX00 0000
9	1111 1110	0000 0000	XXXX XXXX	X000 0000

X: indicates valid bits.

For example, if the bit length is set to 16 bits, large interval P consists of 256 repetitions of small interval T:

$$TP = 256 \times T$$

Because small interval T is 256 times Ttc (1/2 or 1/1 of cycle clock), the following is true:

$$TP = 256 \times 256 \times Ttc = 65536 \times Ttc$$

The total "L" level cumulative pulse width ΣTL in large interval P is set by T1HC.

$$\Sigma TL = \Sigma [T1HC] \times Ttc$$

Because the "L" level of small interval T can be set by T1LC, the total "L" level interval width ΣTL becomes:

$$\Sigma TL = (256 \times [T1LC] \times [T1HC]) \times Ttc$$

When T1LC = 03H and T1HC = 0B4H, the following is true:

$$\Sigma TL = (256 \times 03 \times 180) \times Ttc = 948 \times Ttc$$

The "L" level ratio RL is:

$$RL = \Sigma TL / TP = 948 / 65536 @ 1.447\%$$

Furthermore, when T1LC = 0FFH and T1HC = 0FFH, the "L" level ratio becomes:

$$RL = \Sigma TL / TP = 65535 / 65536 @ 99.998\%$$

The relationship between the pulse bit length and the pulse width that can be set is shown below.

- Large interval P cycle TP

$$TP = 2^{[BIT]} \times Ttc$$

- Total "L" level pulse width ΣTL within large interval P

$$\Sigma TL = (2^{[BIT]} \times [T1LC] / 256 + [T1HC]) \times Ttc$$

* T1HC and T1LC are represented in decimal notation.

* [T1HC] is the valid bit value.

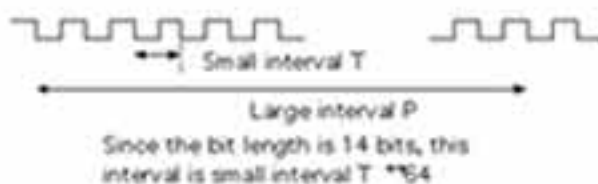
Table 2.6 Relationship Between Bit Length, Pulse Width and Precision

Bit length [BIT]	T1LC	T1HC	ΣTL	TP[Ttc]	Precision			
	min.	max.	min.	max.	min.	max.		
16	0	255	0	255	0	65535	65535	1/65535
15	0	255	0	127	0	32767	32767	1/32767
14	0	255	0	63	0	16383	16383	1/16383
13	0	255	0	31	0	8191	8191	1/8191
12	0	255	0	15	0	4095	4095	1/4095
11	0	255	0	7	0	2047	2047	1/2047
10	0	255	0	3	0	1023	1023	1/1023
9	0	255	0	1	0	511	511	1/511

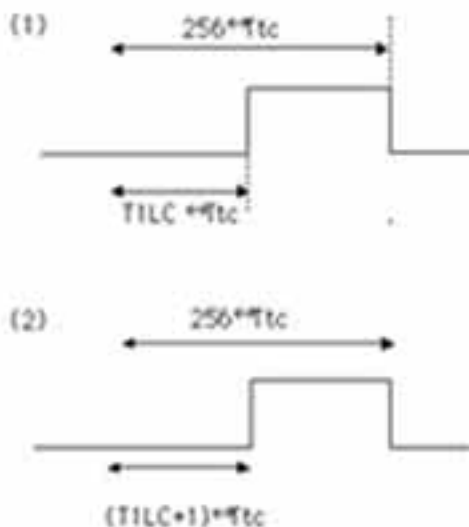
* T1HC indicates the value of the valid bits indicated by Table on previous page. for example, when the length is 11 bits, the bits from bit 7 to bit 5 are valid, so the maximum value is "7."

Example: Settings (in binary) when using the generator as a 14-bit pulse generator

- Value of T1HR: 1100 0000 B
- Value of T1LR: 0000 0000 B
- Values set in the 14 bits of the pulse generator



The following two types of pulse are output in small interval T. Pulse (1) is output (54 x T1HC) times in large interval P, and pulse 2 is output T1HC times.



For details on the relationship between the output waveform and T1HC and T1LC, refer to Appendix 1, "Variable Bit Length Pulse Generator."

Note:

- Follow the procedure described below when setting the "L" level pulse width.
 - ① Set the data update enable flag ELDT1C to "0."
 - ② Overwrite T1LC and T1HC.
 - ③ Set the data update enable flag ELDT1C to "1."
 - The delay after the values of T1HC and T1LC are overwritten until the waveform based on the new data is output is the time required for the maximum pulse after ELDT1C = "1" is set.
 - When using 16-bit mode, select either the cycle clock or the cycle clock divided by 2 for the clock.
- Ttc □ Tcyc : T1HRUN=1 □ CT1LRUN=1 □ CT1LONG=1
Ttc □ 1/2Tcyc : T1HRUN=0 □ CT1LRUN=1 □ CT1LONG=1

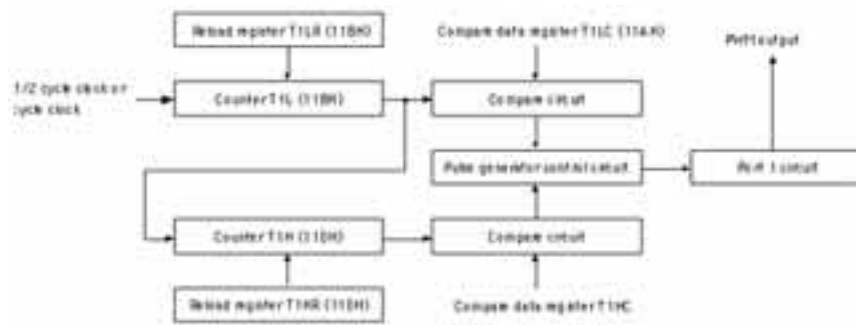
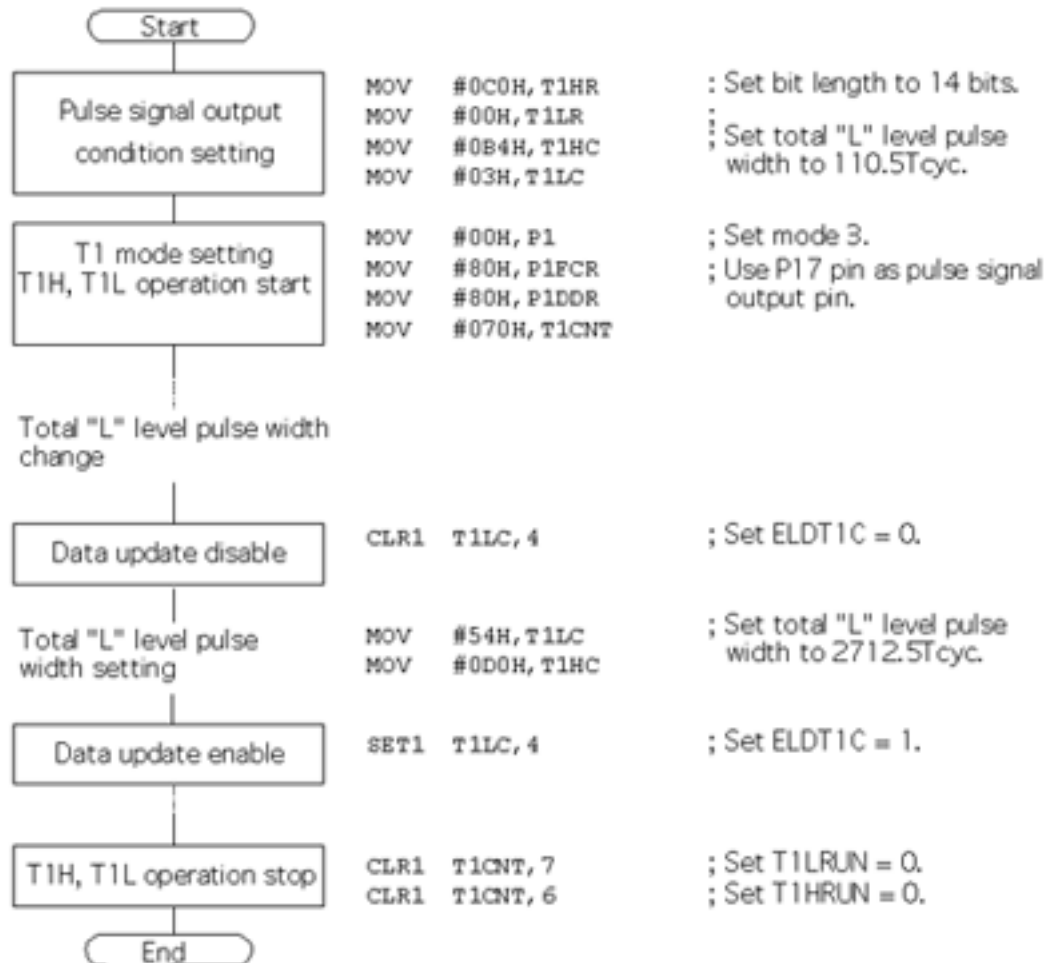


Figure 3.19 Block Diagram for Mode 3: Variable Bit Length Pulse Generator

- Mode 3 Program Example



4. Base Timer

4.1 Overview

POTATO's built-in base timer is a 14-bit binary up-counter that has the four functions listed below

- Clock timer
- 14-bit binary up counter
- Fast forward mode (when using a 6-bit base timer)

4.2 Function

- Clock timer

When the 32.768Khz sub-clock is used for the count clock for the base timer, time can be measured in 0.5-second intervals. The input signal is used to specify "sub clock" as the count clock for the base timer.

- 14-bit binary up counter

The 8-bit binary up counter and the 6-bit binary up counter can be used as a 14-bit binary up counter. These counters can be cleared by software.

- Fast forward mode (when using the 6-bit base timer)

If the 6-bit base timer is used, time can be measured in approximately 2ms intervals when the 32.768KHz sub-clock is used for the count clock. The bit length can be selected through the Base Timer Control Register (BTCCR).

- Interrupt generation

When the interrupt request enable bit is set and an interrupt request is generated from the base timer, an interrupt request to vector address 001BH is generated. There are two types of interrupt requests from the base timer: "base timer interrupt 0" and "base timer interrupt 1."

In order to control the base timer, it is necessary to manipulate the following Special Function Registers:

- | | |
|-------------------|---------------------|
| • BTCCR | • ISL |
| • P1DDR | • P1 |
| • P1FCR | |
| • Timer 0-related | • Interrupt related |

4.3 Circuit Configuration

The base timer configuration is shown in Fig. below.

- 8-bit binary up counter (1)

The input for this up counter is the signal selected by the Input Signal Select register (ISL).

This counter creates the 4KHz/2KHz buzzer output signal, and generates the base timer interrupt 1 source. The overflow event of this timer serves as a clock for the 6-bit binary counter.

- 6-bit binary up counter(2)

The input for this 6-bit up counter is the signal selected by the ISL Special Function Register, or the overflow signal from the 8-bit counter. This counter generates the base timer interrupt 0 and 1 sources. Switching the input clock is handled through the Base Timer Control Register (BTCR).

- Base timer input clock source(3)

The base timer input clock is selected through the Input Signal Select register (ISL) from among " cycle clock," "timer 0 prescaler," or "sub-clock."

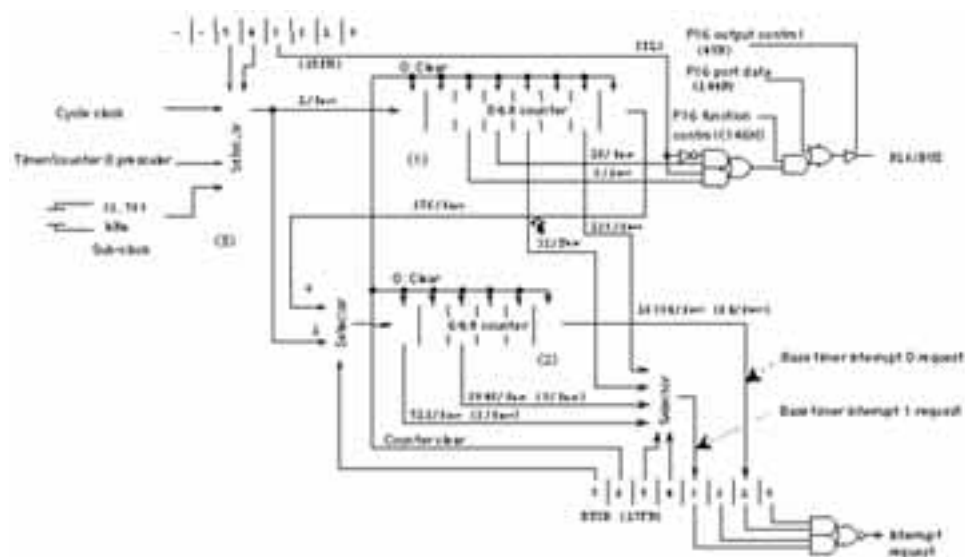


Figure 3.20 Base Timer Block Diagram

4.4 Related Registers

- Base Timer Control register (BTCR)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BTCR	17FH	R/W	BTCR7	BTCR6	BTCR5	BTCR4	BTCR3	BTCR2	BTCR1	BTCR0
After reset			0	0	0	0	0	0	0	0

*In the VMU, game software is prohibited from accessing BTVR7, BTCR6, and BTCR0. Bit manipulation instructions must always be used when accessing this register.

Bit name		Function		
BTCR7 (bit 7)	Base timer interrupt 0 cycle control			
	0:	16384/fBST		
BTCR6 (bit 6)	Base timer operation control			
	1: Base timer operation start			
BTCR5 (bit 5) BTCR4 (bit 4)	Base timer interrupt 1 cycle control			
	BTCR7	BTCR5	BTCR4	
	x	0	0	32/fBST
	x	0	1	128/fBST
	0	1	0	512/fBST
	0	1	1	2048/fBST
BTCR3 (bit 3)	Base timer interrupt 1 source			
	0: No interrupt source 1: Interrupt source			
BTCR2 (bit 2)	Base timer interrupt 1 request enable control			
	0: Interrupt request disable 1: Interrupt request enable			
BTCR1 (bit 1)	Base timer interrupt 0 source			
	0:No interrupt source 1:Interrupt source			
BTCR0 (bit 0)	Base timer interrupt 0 request enable control			
	1:Interrupt source			

BTCR7 (bit 7):	Base timer interrupt 0 cycle control	Fixed at "0"
	This bit selects either 64/fBST (1) or 16384f/BST (0) as the base timer interrupt 0 source generation cycle. When this bit is set to "0," the interrupt 0 source is generated by an overflow in the 14-bit counter, and the interrupt source generation interval is 16384/fBST. When using fast forward mode, set "1."	
BTVR6 (bit 6):	Base timer operation control	Fixed at "1"
	This bit starts (1)/stops (0) the base timer counting operation. When this bit is set to "1," the 14-bit counter counts up; when this bit is set to "0," the 14-bit counter is cleared and then stopped.	
BTCR5 (bit 5):	Base timer interrupt 1 cycle control	
BTCR4 (bit 4):	These bits select the base timer interrupt 1 source generation cycle.	

BTCR7	BTCR5	BTCR4	Base timer interrupt 1 cycle
x	0	0	32/fBST
x	0	1	128/fBST
0	1	0	512/fBST
0	1	1	2048/fBST

fBST: Input clock frequency

BTCR3 (bit 3):	Base timer interrupt 1 source flag	
This bit is set at each interval when the base timer 1 source is generated according to the settings of BTCR7, 5, and 4, and does not change when the source is not generated. Therefore, this bit must be reset by software.		
BTCR2 (bit 2):	Base timer interrupt 1 request enable control	
This bit enables (1) / disables (0) interrupt requests through base timer interrupt 1. If this bit is set to "1," then when the base timer 1 interrupt source is generated an interrupt request to vector address 001BH is generated; if this bit is set to "0," then no interrupt requests are generated.		
BTCR1 (bit 1):	Base timer interrupt 0 source flag	
This bit is set at each interval when the base timer 0 source is generated according to the setting of BTCR7, and does not change when the source is not generated. Therefore, this bit must be reset by software.		
BTCR0 (bit 0):	Base timer interrupt 0 request enable control	Fixed at "0"
This bit enables (1) / disables (0) interrupt requests through base timer interrupt 0. If this bit is set to "1," then when the base timer 0 interrupt source is generated an interrupt request to vector address 001BH is generated; if this bit is set to "0," then no interrupt requests are generated.		

Note:

- When BTCR7 and 5 = 1 (fast forward mode), do not select both the system clock and the base time clock simultaneously.
- When overwriting BTCR5 and 4, note that BTCR3 may be set to "1" as a result.
- If either the cycle clock or the sub-clock was selected as the base timer clock source, and then HOLD mode is set while the base timer is still running, then the base timer might miscount due to the effects of unstable oscillation that initially occurs when the main clock and sub-clock start to oscillate after HOLD mode is cancelled. When entering HOLD mode, therefore, stopping the base timer is recommended.

- Input Signal Select register (ISL)

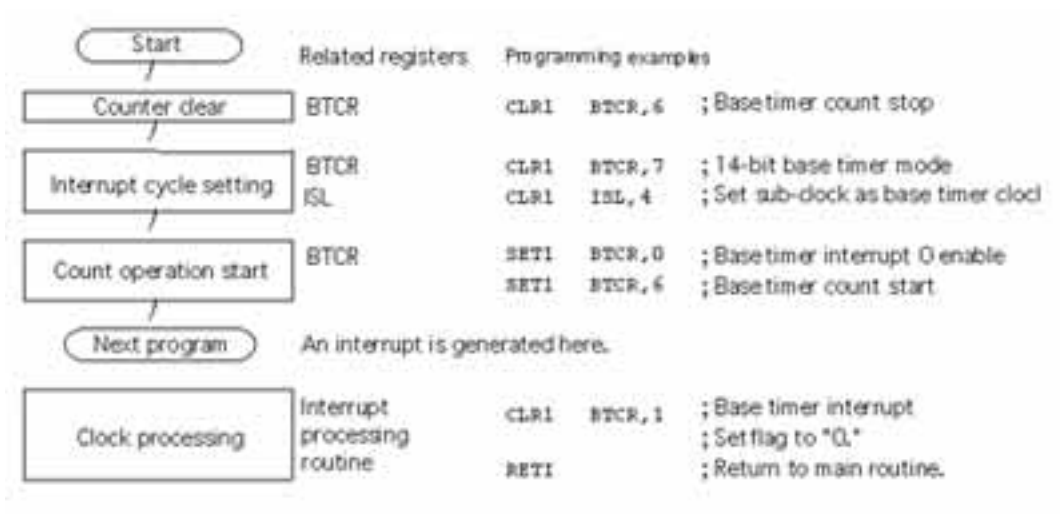
For details, refer to Chapter 3, section 3.2.4, "Input Signal Select Register."

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ISL	15FH	R/W	-	-	ISL5	ISL4	ISL3	ISL2	ISL1	ISL0
After reset			0	0	0	0	0	0	H	H

Bit name	Function		
ISL5 (bit 5)	Base timer clock selection		
ISL4 (bit 4)	ISL5	ISL4	
	X	0	Sub-clock (quartz oscillation) fixed
ISL3 (bit 3)	Buzzer output frequency selection		
	0: fBST/16 fixed		
ISL2 (bit 2)	Noise elimination filter time constant selection		
ISL1 (bit 1)	ISL2	ISL1	Time constant
	1	1	16Tcyc
	0	1	64Tcyc
	X	0	1Tcyc
ISL0 (bit 0)	T0 clock input pin selection		
	0: P72/INT2/TOIN pin 1: P73/INT3/TOIN pin		

4.5 Using the Base Timer

- Clock timer



5. Serial Interface

5.1 Overview

POTATO has a built-in 2-channel synchronous serial interface with an 8-bit data length that uses port 1. Because the next-generation game machine interface also uses port 1, use of the next-generation game machine interface must be prohibited by software if the synchronous serial interface is being used. The main functions of this interface are listed below.

- 2-channel synchronous serial interface
- Transfer clock selection function
- Serial interface SIO0 transfer clock polarity switching function
- LSB-first/MSB-first switching function
- Operation mode switching function
- Overrun detection function
- Transfer bit length control function

5.2 function

- 2-channel synchronous serial interface

Two serial interfaces are provided, with SIO0 using P10 through 12 as I/O pins and SIO1 using P13 through 15 as I/O pins.

Normally, in the VMU SIO0 is used as the master and SIO1 is used as the slave.

- Transfer clock selection function

One of the following three clocks can be selected. In addition, the polarity of the transfer clock for SIO0 only can be selected.

- Internal clock
- External clock
- Software clock

- Serial interface SIO0 transfer clock polarity switching function (bus can be supported)

The polarity of the transfer clock SCK0 for serial interface SIO0 can be switched.

- ① When operation is stopped, SCK0 = 1 and data output is maintained
- ② When operation is stopped, SCK0 = 0 and data output is bit 0 of SBUF0

- LSB-first/MSB-first switching function

It is possible to switch between starting transfers from the LSB or the MSB in data communications over the serial interface. This setting can be made separately for each channel.

- Overrun detection function

This function generates an error when a clock that exceeds 8 bits is received.

- Transfer bit length control function

This function selects whether to stop operation after 8 bits have been transferred, or to continue transfer operations after 8 bits have been transferred.

- Interrupt generation

When the interrupt request enable bit is set, the SIO0 and SIO1 interrupt requests are generated by overflows in an octal counter.

In order to control the serial interfaces, it is necessary to manipulate the following Special Function Registers:

- | | | |
|---------|---------|---------|
| • SCON0 | • SCON1 | • SBR |
| • SBUF0 | • SBUF1 | |
| • P1 | • P1DDR | • P1FCR |

5.3 Circuit Configuration

The configuration of the serial interfaces is shown in Fig. below.

- Shift register ①

This consists of an 8-bit shift register (SBUF0 and 1), and operates according to the specified clock.

- Octal counter ②

This counts the shift clock and detects the end of transfers.

- Baud rate generator ③

This consists of an 8-bit register (SBR) for setting data and an 8-bit reload counter. If "internal clock" is selected for the transfer clock, data transfers are executed according to the clock that is generated here. This baud rate generator is used for both SIO0 and SIO1.

- Polarity switching circuit

This circuit controls the polarity of the transfer clock before and after serial transfer.



5.4 Related Registers

- SIO0 control register (SCON0)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SCON0	130H	R/W	SCON07	SCON06	-	SCON04	SCON03	SCON02	SCON01	SCON00
After reset			H	0	0	0	0	0	0	0

Bit name	Function
SCON07 (bit 7)	Polarity control
	0: When operation is stopped, SCK0 = 1 and data output is maintained 1: When operation is stopped, SCK0 = 0 and data output is bit 0 of SBUF0
SCON06 (bit 6)	Overrun flag
	0: No overrun 1: Overrun
SCON04 (bit 4)	Transfer bit length control
	0: 8-bit transfer 1: Continuous transfer
SCON03 (bit 3)	Transfer control
	0: 0: LSB-first 1: MSB-first
SCON02 (bit 2)	LSB-/MSB-first select
	0: LSB-first 1: MSB-first
SCON01 (bit 1)	Serial transfer end flag
	0: Transfer in progress 1: Transfer completed
SCON00 (bit 0)	Interrupt request enable
	0: Interrupt request disabled 1: Interrupt request enabled

SCON07 (bit 7):	<p>SCK0 polarity control</p> <p>This bit controls the polarity of the transfer clock SCK0 that is used by SIO0. When "1" is set, SCK0 becomes "0" when operation of SIO0 has stopped, and bit 0 of SBUF0 is output. This mode permits bus support. When "0" is set, SCK0 becomes "1" when operation of SIO0 has stopped, and the last data that was transferred is maintained on the output.</p>
SCON06 (bit 6):	<p>Overrun flag</p> <p>This flag is used to detect a serial transfer error on SIO0. If a transfer clock is received (a falling edge is detected) after the transfer of 8 bits of data has been completed (SCON01 has been set to "1"), this flag is set. In addition, when executing a continuous transfer, the overrun flag is set after every eight bits. This bit is not reset automatically; it must be reset by software.</p>
SCON04 (bit 4):	<p>Transfer bit length control</p> <p>This bit selects the SIO0 transfer data bit length: continuous (1) or 8 bits (0). When this bit is set to "1," two or more 8-bit bytes of data can be sent consecutively. This flag does not change after a transfer; it must be reset by software. When this bit is set to "0," eight bits of data can be transferred.</p>
SCON03 (bit 3):	<p>SIO0 operation control</p> <p>This bit starts (1)/stops (0) SIO0 transfer. When this bit is set to "1," an 8-bit serial transfer on SIO0 starts; when the transfer is completed, this bit is reset. When this bit is set to "0," SIO0 operation stops.</p>
SCON02 (bit 2):	<p>LSB-/MSB-first select</p> <p>This bit selects whether to start the transfer of data from the MSB (1) or the LSB (0). When this bit is set to "1," the MSB is transferred first; when this bit is set to "0," the LSB is transferred first. This setting applies to both transmitting and receiving.</p>
SCON01 (bit 1):	<p>SIO0 transfer end flag</p> <p>This flag is used to detect the end of a serial transfer. This flag is set when an 8-bit serial transfer is completed. This bit is not reset automatically; it must be reset by software. If the falling edge of a transfer clock is detected while this bit is set to "1," the overrun flag is set.</p>
SCON00 (bit 0):	<p>SIO0 interrupt request enable control</p> <p>This bit enables (1)/disables (0) interrupt requests due to the end of a transfer on SIO0. When this bit is set to "1," an interrupt request to vector address 0033H is generated; when this bit is set to "0," no interrupt request is generated.</p>

Note:

- The transfer end flag is set to "1" when the transfer of 8 bits is completed, without regard for the transfer bit length setting. The overrun flag has no effect on the operation of the microcomputer.
-

- SIO1 control register (SCON1)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SCON1	134H	R/W	-	SCON16	-	SCON14	SCON13	SCON12	SCON11	SCON10
After reset			H	0	H	0	0	0	0	0

Bit name	Function
SCON16 (bit 6)	Overrun flag
	0: No overrun 1: Overrun
SCON14 (bit 4)	Transfer bit length control
	0: 8-bit transfer 1: Continuous transfer
SCON13 (bit 3)	Transfer control
	0: Transfer stop 1: Transfer start
SCON12 (bit 2)	LSB-/MSB-first select
	0: LSB-first 1: MSB-first
SCON11 (bit 1)	Serial transfer end flag
	0: Transfer in progress 1: Transfer completed
SCON10 (bit 0)	Interrupt request enable
	0: Interrupt request disabled 1: Interrupt request enabled

SCON16 (bit 6):	<p>Overrun flag</p> <p>This flag is used to detect a serial transfer error on SIO1. If a transfer clock is received (a falling edge is detected) after the transfer of 8 bits of data has been completed (SCON11 has been set to "1"), this flag is set. In addition, when executing a continuous transfer, the overrun flag is set after every eight bits. This bit is not reset automatically; it must be reset by software.</p>
SCON14 (bit 4):	<p>Transfer bit length control</p> <p>This bit selects the SIO1 transfer data bit length: continuous (1) or 8 bits (0). When this bit is set to "1," two or more 8-bit bytes of data can be sent consecutively. This flag does not change after a transfer; it must be reset by software. When this bit is set to "0," eight bits of data can be transferred. In this case, the transfer end flag (SCON11) is set when the transfer of 8 bits is completed.</p>
SCON13 (bit 3):	<p>SIO1 operation control</p> <p>This bit starts (1)/stops (0) SIO1 transfer. When this bit is set to "1," an 8-bit serial transfer on SIO1 starts; when the transfer is completed, this bit is reset. When this bit is set to "0," SIO1 operation stops.</p>
SCON12 (bit 2):	<p>LSB-/MSB-first select</p> <p>This bit selects whether to start the transfer of data from the MSB (1) or the LSB (0). When this bit is set to "1," the MSB is transferred first; when this bit is set to "0," the LSB is transferred first. This setting applies to both transmitting and receiving.</p>
SCON11 (bit 1):	<p>SIO1 transfer end flag</p> <p>This flag is used to detect the end of a serial transfer. This flag is set when an 8-bit serial transfer is completed. This bit is not reset automatically; it must be reset by software. If the falling edge of a transfer clock is detected while this bit is set to "1," the overrun flag is set.</p>
SCON10 (bit 0):	<p>SIO1 interrupt request enable control</p> <p>This bit enables (1)/disables (0) interrupt requests due to the end of a transfer on SIO1. When this bit is set to "1," an interrupt request to vector address 003bH is generated; when this bit is set to "0," no interrupt request is generated.</p>

Note:

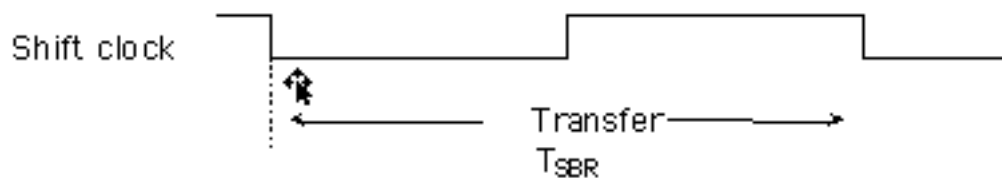
- The transfer end flag is set to "1" when the transfer of 8 bits is completed, without regard for the transfer bit length setting. The overrun flag has no effect on the operation of the microcomputer.
-

- Baud Rate Generator Register (SBR)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SBR	132H	R/W	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
After reset			0	0	0	0	0	0	0	0

This register sets the transfer rate when using the internal clock for the transfer clock. This value is shared by both SIO0 and SIO1. The transfer rate T_{SBR} is derived according to the following equation:

$$T_{SBR} = (256 - [SBR \text{ setting}]) \times 2 \times T_{Cyc} \text{ (} T_{Cyc}: \text{ Cycle clock cycle) }$$



- Serial Buffer 0 (SBUF0)

This register stores each 8 bits of data handled in a serial transfer on SIO0.

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SBUF0	131H	R/W	SBUF07	SBUF06	SBUF05	SBUF04	SBUF03	SBUF02	SBUF01	SBUF00
After reset			0	0	0	0	0	0	0	0

- Serial Buffer 1 (SBUF1)

This register stores each 8 bits of data handled in a serial transfer on SIO1.

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SBUF1	135H	R/W	SBUF17	SBUF16	SBUF15	SBUF14	SBUF13	SBUF12	SBUF11	SBUF10
After reset			0	0	0	0	0	0	0	0

- Next-generation game machine dedicated interface circuit

The separate function described above uses port 1 as the I/O port for the next-generation game machine dedicated interface. The next-generation game machine dedicated interface and the synchronous serial interface cannot be used simultaneously.

Note:

Note the following points when conducting serial communications through the serial interface:

1 Do not make settings concerning serial communications while the unit is in the reset state.

2 Confirm the connection between two VMU units before making settings concerning serial communications.

The connection between two VMU units can be confirmed through the value of port 7. When two VMU units are connected, the values of certain bits in port 7 are as follows:

PORT7 bit3 = '1'

PORT7 bit2 = '0'

3 When serial communications processing is completed, or if two VMU units are not connected, make the following settings:

SCON0 = 00h

SCON1 = 00h

P1FCR = 0BFh

P1DDR = 0A4h

The unit may not operate correctly if the settings for executing serial communications are made when two VMU units are not connected.

5.5 Serial Interface Operation

A serial transfer on a serial interface is initiated by setting the communications control bit (SCON03, SCON13) or the transfer bit length selection bit (SCON04, SCON14). There are two transfer modes:

- Normal mode

Two data lines and one clock line are used for data communications in this mode. SI is the data input line, and SO is the data output line. This mode is the general-purpose transfer method, and is suited for communications with a specific partner.

Use normal mode when connecting two VMU units.

- Bus mode (deleted)

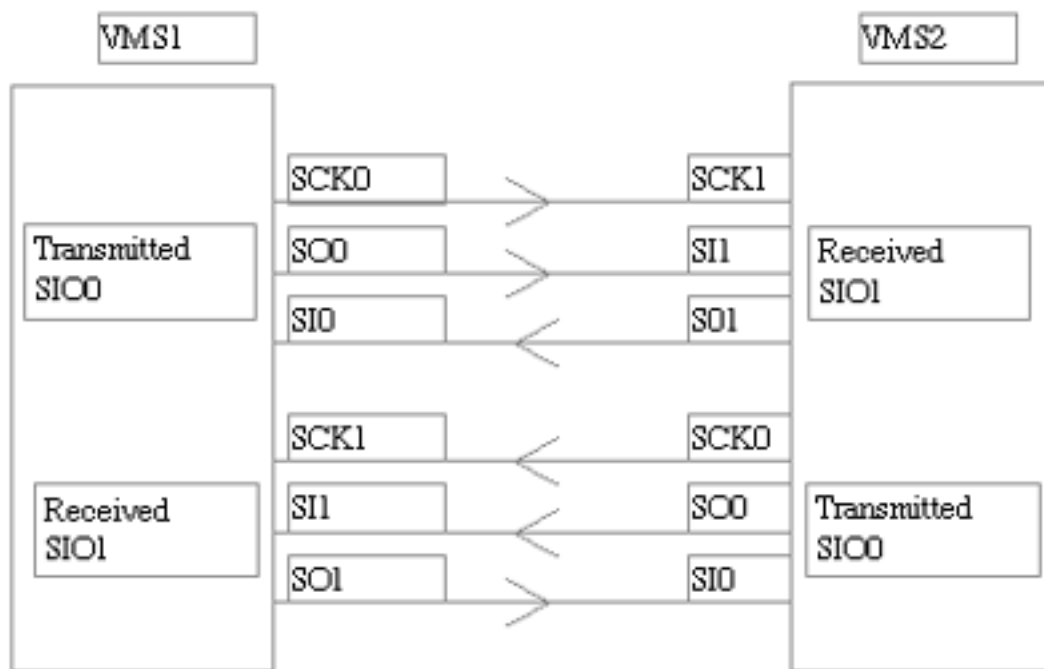


Figure 3.23 Connection between VMU Units

The transfer mode is set by manipulating the Special Function Registers associated with port 1. (Refer to Table 3-5-1.) In addition, SIO0 and SIO1 can each be set to either normal mode or bus mode independently.

- Serial transfer timing

The data in the shift register is shifted in synchronization with the falling edge of the serial clock SCK0 or SCK1, and is output on the SO0 or SO1 pin. The data that is input on the SIO or SI1 pin is loaded into the shift register at the rising edge of the serial clock.

5.6 Operation Mode Settings

- Normal mode

It is necessary to reset the port latch data corresponding to the output pin or the transfer clock when using the internal clock. The pins that are used in normal mode are shown in the table below.

Table 2.7 Pins Used in Normal Mode

	SI00	SI01
Input pin	P11/SI0/SB0	P14/SI1/SB1
Output pin	P10/S00(P11/SI0/SB0)	P13/S01(P14/SI1/SB1)
Transfer clock	P12/SCK0	P15/SCK1

Note:

Set SCKn high one Tcyc cycle before the start of transfer. If SCKn is set high less than one Tcyc cycle prior to the start of transfer, the correct data will not be output.

Table 2.8 Settings for Port 1 for SI00 (Special Function Registers)

Pin	Function	Special Function Register Value
P11/SI0/SB0 P10/S00	RX TX	P11DDR =0 P10 =0 P10DDR =1 P10FCR =1
P11/SI0/SB0 P10/S00	RX General-purpose I/O	P11DDR =0 P10FCR =0
	Internal clock	P12 =0 P12DDR =1 P12FCR =1

*For the software clock, the program writes "0" and "1" in alternation to a port (P12), and that output is used as the transfer clock.

Table 2.9 Settings for Port 1 for SIO1 (Special Function Registers)

Pin	Function	Special Function Register Value
P14/SI11/SB1	RX	P14DDR =0
P13/SO1	General-purpose I/O	P13FCR =0

*For the software clock, the program writes "0" and "1" in alternation to a port (P15), and that output is used as the transfer clock.

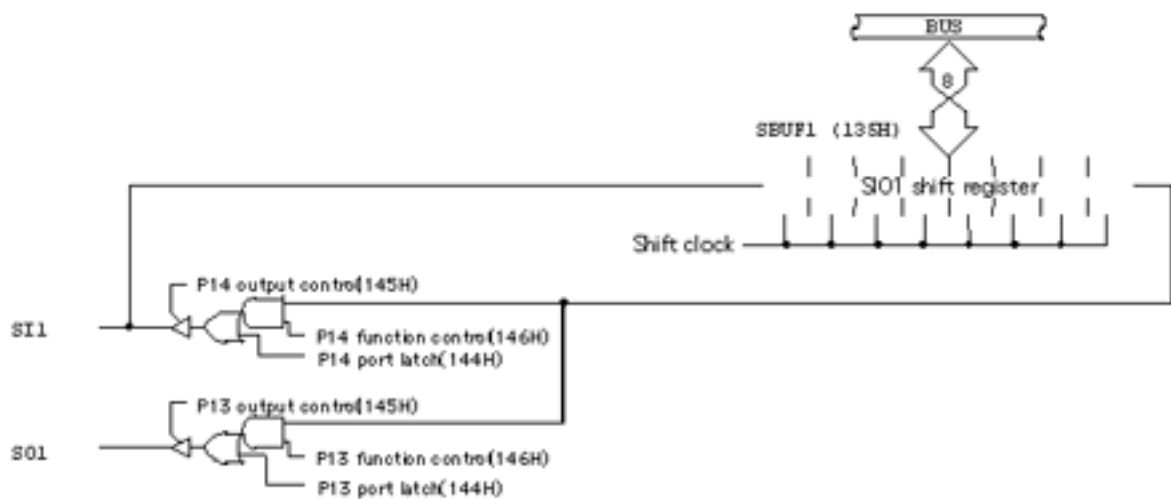


Figure 3.24 Signal Path in Normal Mode (Example for SIO1)

Note:

When setting Pn to output, set PnFCR to "1" before PnDDR. If PnDDR is set first, "0" might be output on Pn when PnDDR is set. This applies to both SIO0 and SIO1.

5.7 Serial Transfer Clock

The serial transfer clock uses the P12/SCK0 pin for SIO0, and the P15/SCK1 pin for SIO1. One of the following three types of serial transfer clock can be selected independently for SIO0 and one for SIO1 through the application circuit. In addition, in the case of SIO0 only, the polarity of the transfer clock can be switched.

- Internal clock
- External clock
- Software clock
- Internal clock

The transfer clock is generated by the serial transfer-dedicated baud rate generator (SBR) that is built into the LSI. This clock is shared by both SIO0 and SIO1. When running either or both of the serial interfaces according to the internal clock, it is necessary to operate the baud rate generator. In this case, the serial transfer clock is output from the clock pin (P12/SCK0, P15/SCK1) of the serial interface that is running according to the internal clock.

The relationship between the transfer rate and the baud rate generator setting is shown below. (The setting is made with a decimal value.)

$$TSBR = (256 - [\text{SBR setting}]) \times 2 \times T_{cyc} \quad (T_{cyc}: \text{Cycle clock cycle})$$

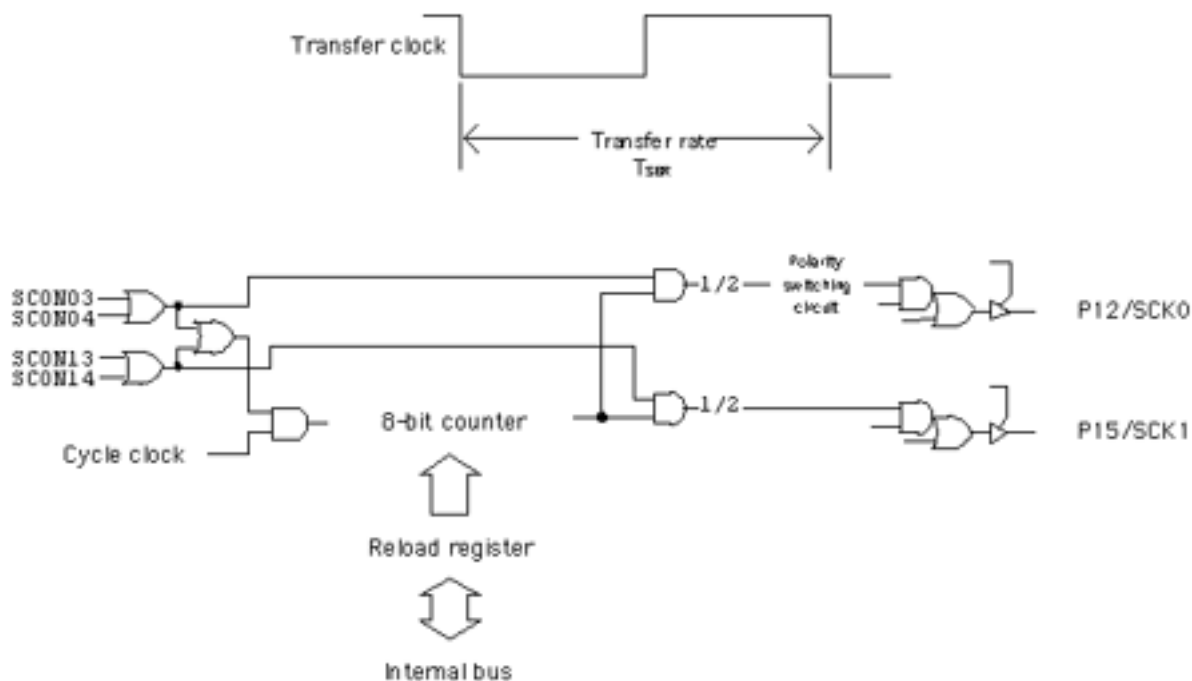


Figure 3.25 Baud Rate Generator Configuration Diagram

Note:

When setting Pn to output, set PnFCR to "1" before PnDDR. If PnDDR is set first, "0" might be output on Pn when PnDDR is set. This applies to both SIO0 and SIO1.

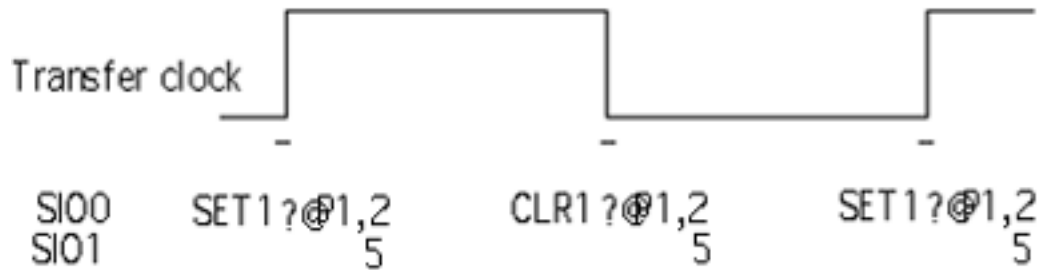
- External clock

Serial transfers are performed according to a clock that is input from outside of the LSI.

- Software clock

The program writes "0" and "1" in alternation to the port P12/SCK0 or P15/SCK1 pin, and that output is used as the serial transfer clock.

Clock generation example



When using these transfer clocks, it is necessary to set the status of the P12/SCK0 or P15/SCK1 pin.

Table 2.10 Transfer Clock Settings

Pin	Function	Special Function Register value
P12/SCK0	Internal clock	P12 =0 P12DDR =1 P12FCR =1
	External clock	P12DDR =0
	Software clock	P12 =0/1 P12DDR =1 P12FCR =0
P15/SCK1	Internal clock	P15 =0 P15DDR =1 P15FCR =1
	External clock	P15DDR =0
	Software clock	P15 =0/1 P15DDR =1 P15FCR =0

Note:

- At least 1/2 of a cycle is needed for the serial data and serial clock pulse width.
When using the sub-clock and external clock, caution is particularly essential. (When using a 32.768kHz crystal resonator for the sub-clock, the cycle clock cycle is 366[μs], so a pulse width of at least 183[μs] is required.)
- When outputting the serial clock from port 1, observe the following sequence when setting the port 1 registers. If this sequence is not observed, serial transfers will not be performed correctly.
 - (1) Set P1FCR.
 - (2) Set P1DDR.
 - (3) Set SCONn. (Set the transfer control bits.)

5.8 Serial Transfer Timing

In a serial transfer, the transfer clock SCK0 is output at high level (SCK0 = 1) before and after an operation is performed on the SIO0 (when SCON07 = 0) or SIO1 interface. In addition, the last data to be transferred is maintained on the output pin. (Refer to Fig. 3-5-7.) However, the transfer clock SCK0 is output at low level (SCK0 = 0) before and after an operation is performed on the SIO0 interface when SCON07 = 1. In addition, bit 0 (SBUF00) of Serial Buffer 0 (SBUF0) is output (and maintained at that level) on the output pin. (Refer to Fig. 3-5-8.) Note that it is not possible to switch the polarity of the SIO1 interface.

SIO0

SCON07=0	When operation is stopped, SCK0 = 1 and data output is maintained
SCON07=1	When operation is stopped, SCK0 = 0 and data output is bit 0 of SBUF0

SIO1

When operation is stopped, SCK1 = 1 and data output is maintained

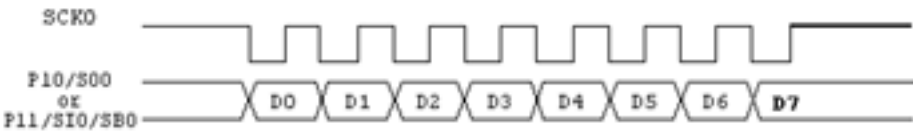


Figure 3.26 Transfer Clock and Output Data (1)

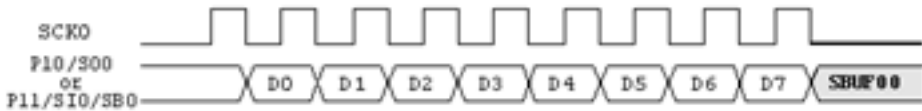


Figure 3.27 Transfer Clock and Output Data (2)

5.9 LSB-/MSB-first Switching Function

When reading or writing the serial transfer buffer, it is possible to reverse the sequence from LSB to MSB. This function can be used to switch between LSB-first and MSB-first. The switch is made through the Serial Transfer Control Register (SCON0, SCON1), and is made before reading or writing the serial transfer buffer. In addition, if the switch is made after reading or writing the serial transfer buffer, the transfer is made in the sequence that was used when the buffer was read or written.

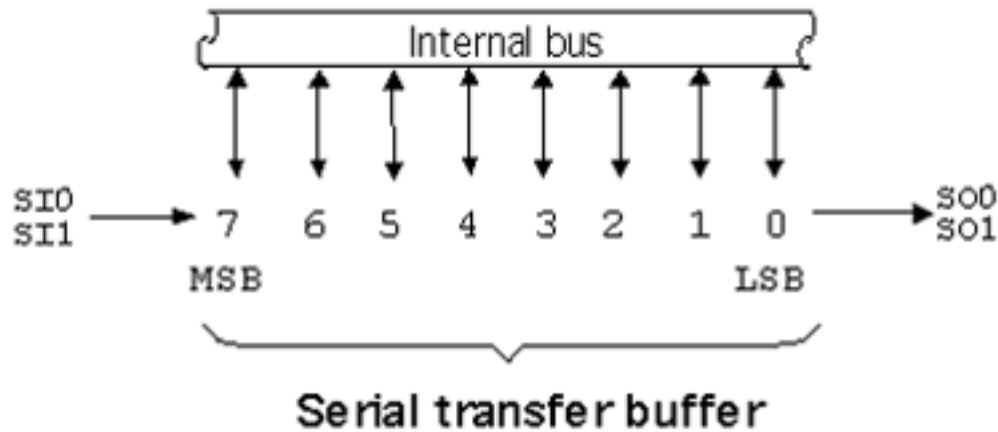


Figure 3.28 Correspondence Between the Serial Transfer Buffer and the Internal Bus When LSB-first Is Specified

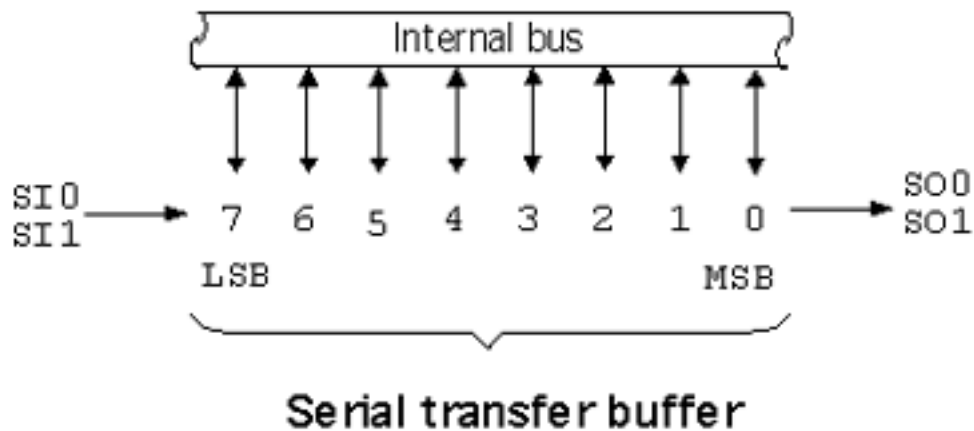


Figure 3.29 Correspondence Between the Serial Transfer Buffer and the Internal Bus When MSB-first Is Specified

Figs. below show the timing charts for LSB-first and MSB-first serial transfer transmission and reception using SIO0.

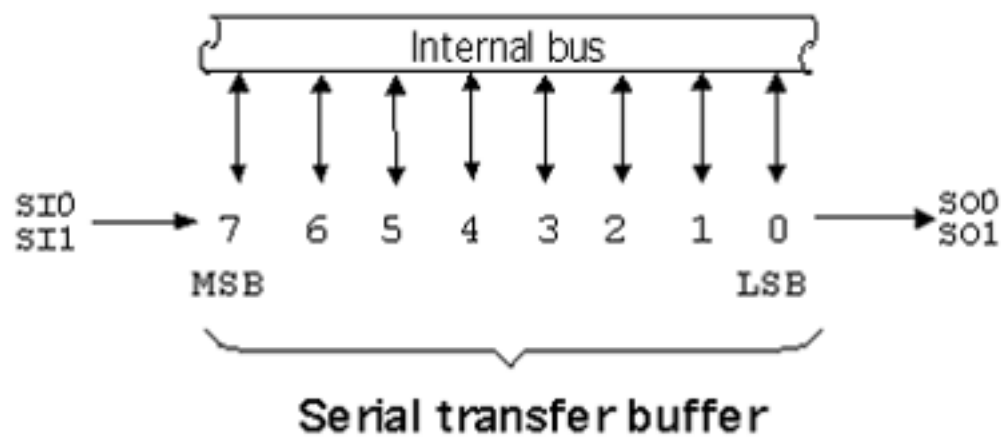


Figure 3.30 Correspondence Between the Serial Transfer Buffer and the Internal Bus When LSB-first Is Specified

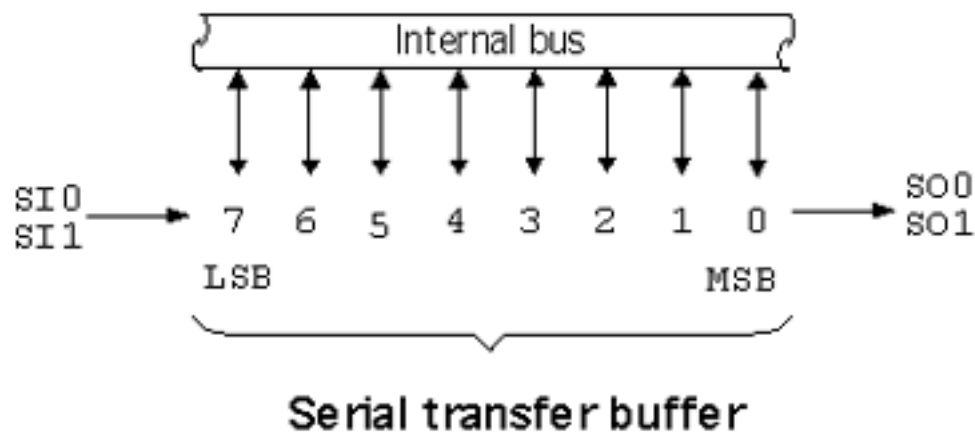


Figure 3.31 Correspondence Between the Serial Transfer Buffer and the Internal Bus When MSB-first Is Specified

5.10 Overrun Detection Function

The overrun detection function detects serial communication errors. When the interrupt source flag has been set (SCON01, SCON11), the overrun flag (SCON06, SCON16) is set at the falling edge of the transfer clock.

Fig. below shows the timing for normal communications and the timing when an overrun is generated. The interrupt source flag (SCON01, SCON11) is set at the rising edge of the transfer clock for the 8th bit of data. If, while in this state, the falling edge of the transfer clock is detected, the overrun detection flag is set. (Refer to the overrun generation timing chart.)

Note that the overrun flag has no effect on the operation of the microcomputer.

Note:

- Wait at least 1/2 of a transfer clock cycle after the interrupt source flag has been set to "1" before checking the overrun flag.
- Even if the transfer mode that is set will exceed 8 bits, the overrun detection function operates according to the same timing as for an 8-bit transfer.

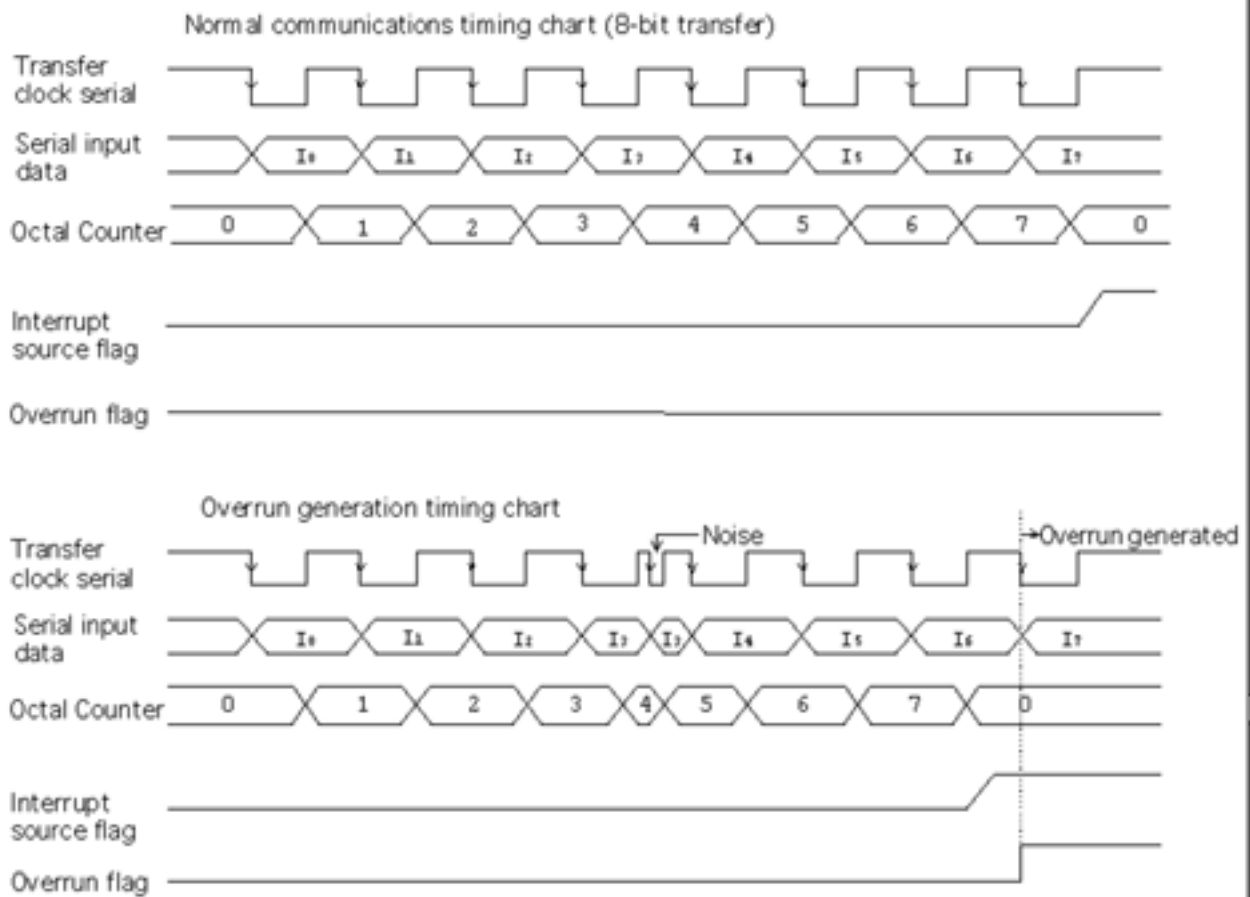


Figure 3.32 Correspondence Between the Serial Transfer Buffer and the Internal Bus When LSB-first Is Specified

5.11 Transfer Bit Length Control Function

When transferring more than 8-bits of serial data, set the transfer bit length control bit SCON04 or SCON14 (continuous transfer).

- Once SCON04 and SCON14 have been set, the serial transfer begins. These bits are not reset even after 8 bits have been transferred.
- The interrupt source flag is set according to the same timing as for an 8-bit transfer (after the completion of the transfer of 8 bits).
- The overrun detection bits SCON06 and SCON16 are set at the falling edge of the serial clock after the transfer of eight bits has been completed. (For the timing chart, refer to the section on the overrun detection function.)
- When the transfer bit length has been set to 8 bits, the transfer starts once the transfer control bit SCON03 or SCON13 is set. Once the transfer of 8 bits has been completed, the transfer control bit is reset. This causes the interrupt source flag SCON01 or SCON11 to be set. In addition, serial transfer stops automatically.
- When the transfer bit length has been set to "continuous transfer," the transfer starts once the transfer bit length control bit SCON04 or SCON14 is set, and continues until the bit is reset. The interrupt source flag is set after 8 bits have been transferred.

5.12 Program Examples

- SIO0 serial transfer (1) (transmission example)

Transfer conditions

- 8-bit transfer
- Transfer data: 038H (8 bits)
- MSB-first
- Falling edge output
- Normal mode
- Internal clock
- Baud rate: 25.6ms
- System clock: 32KHz crystal oscillating sub-clock

Working from the baud rate formula: $T[\text{SBR}] = (256 - [\text{SBR}]) \times 2 \times T_{\text{cyc}}$

$\backslash \quad [\text{SBR}] = 256 - T[\text{SBR}] / (2 \times T_{\text{cyc}})$

In this case, $T[\text{SBR}] = 25.6\text{ms}$, and $T_{\text{cyc}} = 366[\text{micro}]s$, so the value that is to be set in the Baud Rate Generator register (SBR) is determined as follows:

$[\text{SBR}] = 256 - 25600 / (2 \times 366)$

@ 221 (decimal) (approx.)

→ 0DDH (hexadecimal)

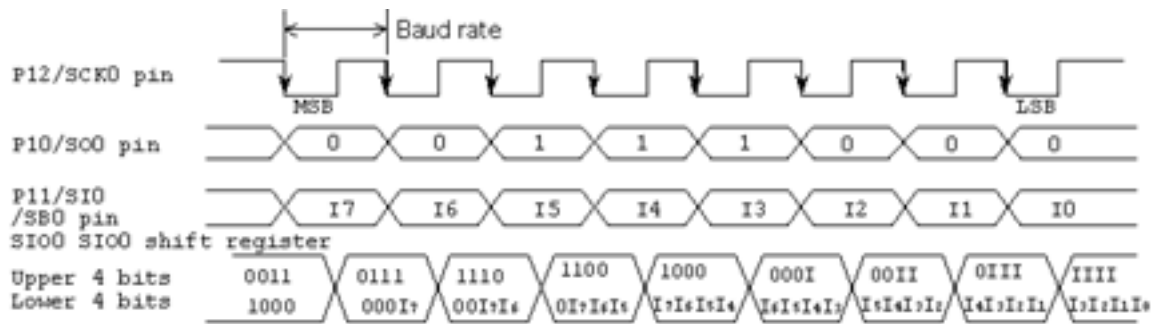
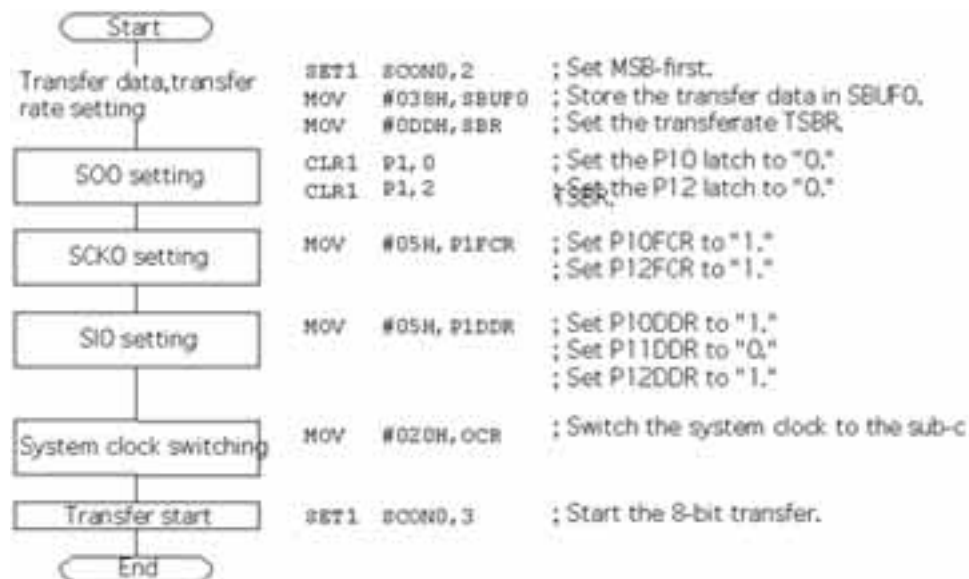


Figure 3.33 Timing for Serial Transfer (1)



• SIO1 serial transfer (2) (reception example)

Transfer conditions

- 16-bit transfer
- LSB-first
- Bus mode
- External clock
- Same data is output from SO1 as from SB1.
- The upper 8 bits of the data that is loaded is stored in RAM at address #031H, and the lower 8 bits are stored in RAM at address #030H.

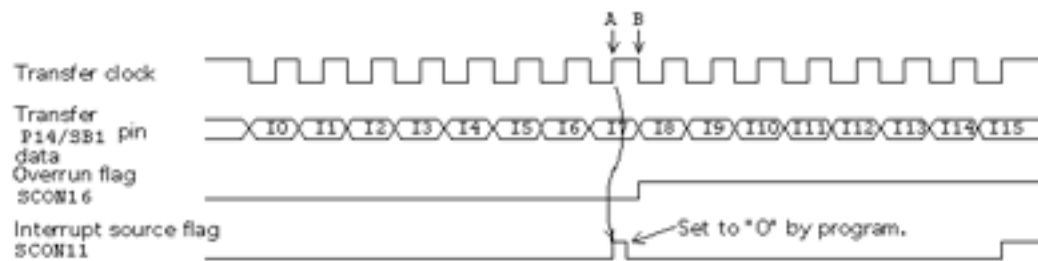


Figure 3.34 Timing for Serial Transfer (2)

Note:

- In this example, misoperation will result if there is a rising edge (B) on the transfer clock during the interval from the execution of the instruction following SELF0 to the execution of the SELF1 instruction. The transfer rate should allow enough time for the cycle clock cycle.
- Set SCKn high one Tcyc cycle before the start of transfer. If SCKn is set high less than one Tcyc cycle prior to the start of transfer, the correct data will not be output.
- When setting Pn to output, set PnFCR to "1" before PnDDR. If PnDDR is set first, "0" might be output on Pn when PnDDR is set. This applies to both SIO0 and SIO1.



6. Dot Matrix LCD Controller/Driver

6.1 Overview

The LCD controller/ driver automatically reads data that is stored in display RAM and generates the signals to drive the dot matrix LCD. The display mode is a graphics mode in which one bit of data in display RAM corresponds to one dot on the LCD.

The dot matrix LCD controller/ driver consists of the following circuit blocks:

- Display RAM
- Display controller register
- LCD power supply circuit

6.2 Functions

- Display duty:1/33 duty
- Display bias:1/5 bias
- Graphics display
- Liquid crystal instruction:

Display: ON/OFF

- Graphics display

1584 dots can be displayed (1 chip)

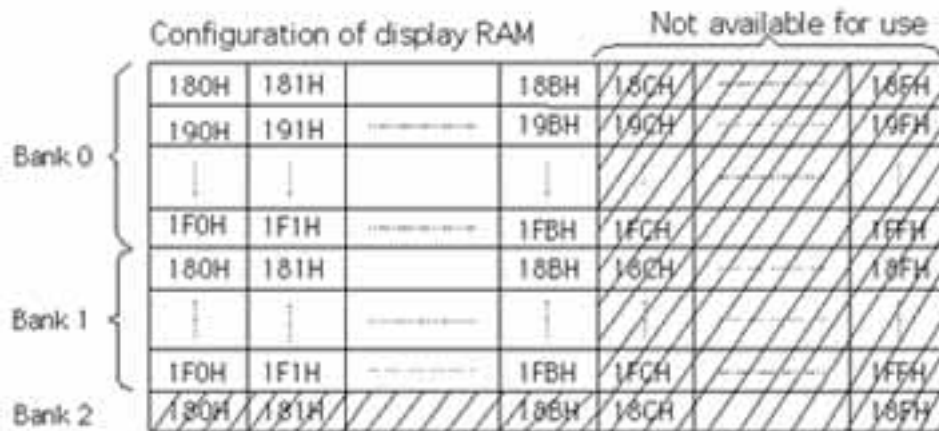
In order to control the liquid crystal display, it is necessary to manipulate the following Special Function Control Registers.

- MCR : LCD on/off control
- STAD : LCD start address control
- CNR : Horizontal byte count control
- TDR : Display duty control
- VCCR : LCD contrast control
- XBNK : Display RAM bank address control

6.3 Display RAM

Display RAM consists of three banks of 96 ¥ 8 bits of static RAM. The LCD controller / driver reads data that is stored in this display RAM and generates the signals to drive the dot matrix LCD. Before writing or reading data in display RAM, set the system clock to RC oscillation.

Symbol	Address	R/W	Name	Initial value	Bank
XRAM	180H - 1FBH	R/W	Display RAM	Undefined	Bank
	180H - 1FBH				Bank
	180H - 185H				Bank



6.4 Display Control Registers

- Mode Control Register (MCR)

This register controls the start / stop of LCD controller operation, cursor display, and the LCD clock division ratio. The Mode Control Register is a write-only register. It is important to note that if a bit manipulation instruction, an INC instruction, a DEC instruction, or a DBNZ instruction is used on a write-only register, bits other than the specified bits will be set. The following instructions are used with the MCR:

- MOV
- ST
- POP
- MOV @
- ST @

In addition, when accessing this register, bits 7 through 5 and bit 0 must be set to their fixed values.

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MCR	120H	W	MCR7	MCR6	MCR5	MCR4	MCR3	-	-	MCR0
After reset			0	0	0	0	0	0	0	0

Bit name	Function			
MCR7 (bit 7)	LCD clock division selection			
MCR6 (bit 6)	MCR7	MCR6	MCR5	Division ratio
MCR5 (bit 5)	0	0	0	1/1 * Always set MCR7 through MCR5 to "0."
MCR4 (bit 4)	LCD clock divide-by-2 circuit selection			
	0: Selects the signal selected by MCR7 through MCR5, divided by 2, as the LCD clock			
	1: Selects the signal selected by MCR7 through MCR5, as is, as the LCD clock. (Direct mode)			
MCR3 (bit 3)	LCD controller control			
	0: LCD controller stop			
	1: LCD controller start/continue			
MCR0 (bit 0)	Display mode selection			
	1: Graphics mode * Always set MCR4 = 1			

MCR7 (bit 7): LCD clock division selection

MCR6 (bit 6):

MCR5 (bit 5):

Always set MCR7 through MCR5 to "0."

MCR4 (bit 4): This bit controls whether or not to divide by 2 the LCD clock that was selected by MCR7 through MCR5.

The frame frequency is:

1/2 cycle (MCR4 = 0): 82.7Hz

1/1 cycle (MCR4 = 1): 165.5Hz

MCR3 (bit 3):

LCD controller control

This bit controls LCD controller operation start (1)/stop (0).

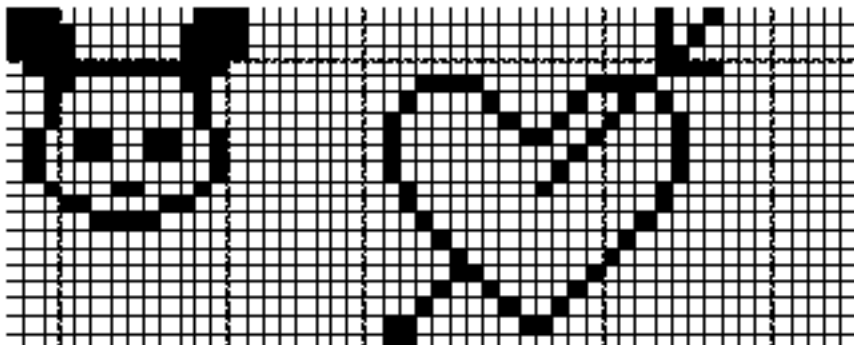
When this bit is set to "1," the LCD controller begins to operate. When this bit is set to "0," the LCD controller stops operating.

MCR0 (bit 0):

Display mode selection

Select graphics mode (1) for the display mode.

Graphics display: MCR0 = 1



- LCD display start address control register (STAD)

This register controls the LCD start address.

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STAD	122H	R/W	STAD7	STAD6	STAD5	STAD4	STAD3	STAD2	STAD1	STAD0
After reset			0	0	0	0	0	0	0	0

Bit name	Function								
STAD7 (bit 7) to STAD0 (bit 0)	LCD RAM display start address setting								
	STAD7	STAD6	STAD5	STAD4	STAD3	STAD2	STAD1	STAD0	Start address
	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	1
	↓	↓	1	1	1	1	1	1	↓
	1	1							255

STAD7 (bit 7):

LCD RAM display start address setting

•

STAD0 (bit 0):

These bits set the starting address of the display data for the LCD. (XRAM 180H is assumed as STAD = 00H.)

The data changes in two-byte units.

3. Peripheral System Configuration

Start address	XRAM address	STAD7	STAD6	STAD5	STAD4	STAD3	STAD2	STAD1	STAD0
0H	180H(Bank 0)	0	0	0	0	0	0	0	0
1H	182H(Bank 0)	0	0	0	0	0	0	0	1
2H	184H(Bank 0)	0	0	0	0	0	0	1	0
3H	186H(Bank 0)	0	0	0	0	0	0	1	1
4H	188H(Bank 0)	0	0	0	0	0	1	0	0
5H	18AH(Bank 0)	0	0	0	0	0	1	0	1
6H	Cannot be set	0	0	0	0	0	1	1	0
7H	Cannot be set	0	0	0	0	0	1	1	1
8H	190H(Bank 0)	0	0	0	0	1	0	0	0
9H	192H(Bank 0)	0	0	0	0	1	0	0	1
0AH	194H(Bank 0)	0	0	0	0	1	0	1	0
0BH	196H(Bank 0)	0	0	0	0	1	0	1	1
0CH	198H(Bank 0)	0	0	0	0	1	1	0	0
0DH	19AH(Bank 0)	0	0	0	0	1	1	0	1
0EH	Cannot be set	0	0	0	0	1	1	1	0
0FH	Cannot be set	0	0	0	0	1	1	1	1
10H	1A0H(Bank 0)	0	0	0	1	0	0	0	0
11H	1A2H(Bank 0)	0	0	0	1	0	0	0	1
3DH	1FAH(Bank 0)	0	0	1	1	1	1	0	1
3EH	Cannot be set	0	0	1	1	1	1	1	0
3FH	Cannot be set	0	0	1	1	1	1	1	1
40H	180H(Bank 1)	0	1	0	0	0	0	0	0
41H	182H(Bank 1)	0	1	0	0	0	0	0	1
7DH	1FAH(Bank 1)	0	1	1	1	1	1	0	1
7EH	Cannot be set	0	1	1	1	1	1	1	0
7FH	Cannot be set	0	1	1	1	1	1	1	1
80H	180H(Bank 2)	1	0	0	0	0	0	0	0
81H	182H(Bank 2)	1	0	0	0	0	0	0	1
82H	184H(Bank 2)	1	0	0	0	0	0	1	0
83H - FFH	Cannot be set								

As indicated above, some settings result in misoperation if they are set as the start address. xx6H, xx7H, xxEH, and xxFH cannot be set.

- Character Number Register (CNR) 123H

This register is set by an internal system program.

Game programs are prohibited from accessing this register.

- Time Division Register (TDR) 124H

This register is set by an internal system program.

Game programs are prohibited from accessing this register.

- Bank Address Register (XBNK)

This register controls the display RAM bank addresses.

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
XBNK	125H	R/W	-	-	-	-	-	-	XBNK1	XBNK0
After reset			H	H	H	H	0	0	H	H

Bit name	Function		
XRBK1 (bit 1) To XRBK0 (bit 0)	LCD display RAM start address setting		
	XRBK1	XRBK0	Bank address
	0	0	0
	0	1	1
	1	0	2
	1	1	Setting prohibited

XRBK1 (bit 1):

to

to XRBK0 (bit 0):

Display RAM bank address control

Data can be written to display RAM at the address specified by the Bank Address Register. Banks 0 and 1 of RAM data are 96 bytes each; game programs are allowed to access bank 0 and bank 1 RAM. (Only bit 0 can be set.)

Bank 2 contains only 6 bytes, and is used for icon display. The system program displays icons.

- LCD Contrast Control Register (VCCR)

This register turns the LCD display on and off. Note that there is no built-in contrast control circuit. The LCD Contrast Control Register is a write-only register. It is important to note that if a bit manipulation instruction, an INC instruction, a DEC instruction, or a DBNZ instruction is used on a write-only register, bits other than the specified bits will be set. The following instructions are used with the VCCR:

- MOV
- ST
- POP
- MOV @
- ST @

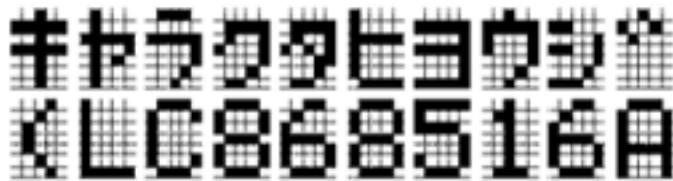
In addition, when accessing this register, bits 5 through 0 must be set to their fixed values.

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
VCCR	127H	W	VCCR7	VCCR6	VCCR5	VCCR4	VCCR3	VCCR2	VCCR1	VCCR0
After reset			0	0	0	0	0	0	0	0

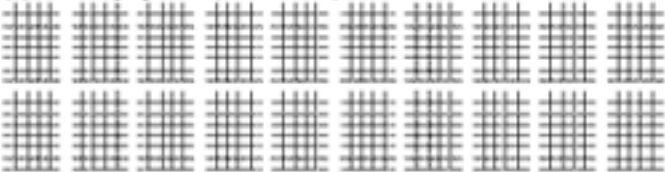
Bit name	Function	
VCCR7	(bit 7)	LCD display control
	0:	LCD display off
1:	LCD display on	
VCCR6	(bit 6)	LCD display RAM access control
	0:	Access from CPU to display RAM enabled
1:	Access from CPU to display RAM disabled	
VCCR	(bit 5)	
to		
VCCR0	(bit 0)	* Be certain to set VCCR5 through VCCR0 to "0."

VCCR7 (bit 7): LCD display control
This bit controls the LCD display: ON (1)/OFF (0).

(1) LCD display ON (VCCR7 = 1)



(2) LCD display OFF (VCCR7 = 0)



Set the LCD display to ON (VCCR7 = 1) after initiating LCD controller operation (MCR3 = 1).

- Setting sequence

1. MCR3=1
2. VCCR7=1

Be certain to always follows this sequence.

Conversely, when turning the LCD display OFF, make the settings in the following sequence:

1. VCCR7=0
2. MCR3=0

VCCR6 (bit 6):

LCD display RAM access control

When the sub-clock (crystal oscillation) has been set for the system clock and the LCD display is ON, be certain to disable access from the CPU to the LCD display RAM (VCCR6 = 1) after changing the system clock. In addition, enable access from the CPU to the LCD display RAM (VCCR6 = 0) when reading or writing the LCD display RAM, or when setting the system clock to CF oscillation or RC oscillation while the LCD is on.

Setting sequence

When changing the system clock from CF oscillation or RC oscillation to crystal oscillation while the LCD display is on:

VCCR6=1
OCR5=1,OCR4=0

When changing the system clock from crystal oscillation to CF oscillation or RC oscillation while the LCD display is on:

OCR5 = 0/1, OCR4 = 1 (CF oscillation), OCR5 = 0, OCR4 = 0 (RC oscillation)
VCCR6 = 0

VCCR5 (bit 5) through VCCR0 (bit 0):

|

Always set these bits to "0."

Note:

- When the LCD is on, set the VCCR last.
-

7. External Interrupt Function

7.1 Overview

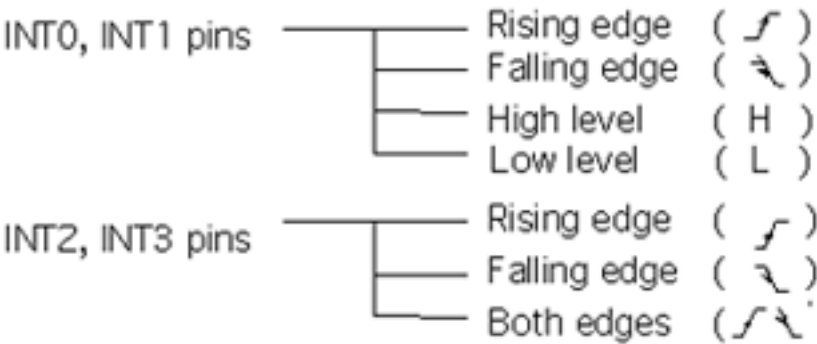
POTATO has a function that detects external input signals on the P70/INT0, P71/INT1, P72/INT2/T0IN, P73/INT3/T0IN pins and then generates interrupt requests to four vector addresses. The types of signals that are detected are selected by the program. P70 is used to detect the LM-BUS connection to VMU, and P71 is used to detect low voltage.

- Pins on which signals are detected and their corresponding vector addresses

Pin	Vector address	Pin	Vector address
P70/INT0	003H	P72/INT2/T0IN	013H
P71/INT1	00BH	P73/INT3/T0IN	01BH

- Signals that can be detected

The priority ranking of the INT0 and INT1 pin interrupts can be set to either "highest level" or "low level" through the Master Interrupt Enable Control Register (IE). If "highest level" is set, that interrupt processing can be executed, regardless of the master interrupt enable setting. The priority ranking of interrupts other than the INT0 and INT1 interrupts can be set to either "high level" or "low level" through the Interrupt Priority Ranking Control Register (IP). IN addition, a noise elimination filter with a switchable time constant is connected to the P73/INT3/T0IN pin.



- Detection of another VMU unit

The statuses of various ports when the unit is connected or not connected to another VMU unit are shown below.

P70	P72	P73	
When connected to a VMU unit	Çk	Çk	Çg
When not connected to a VMU unit	Çk	Çk	Çk

In order to use the external interrupt function, it is necessary to manipulate the following Special Function Registers:

- I01CR
- I23CR
- ISL
- IE

7.2 Circuit Configuration

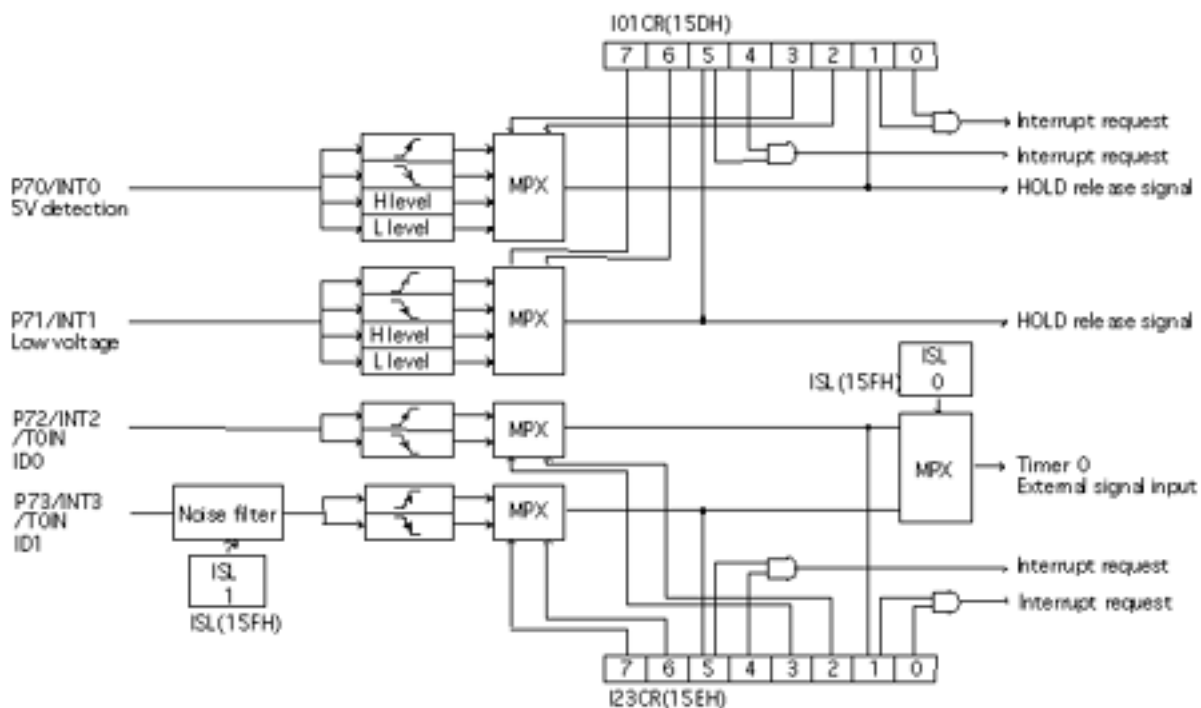


Figure 3.35 *External Interrupt Circuit Block Diagram*

7.3 Related Registers

- External Interrupt 0, 1 Control Register (I01CR)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
I01CR	15DH	R/W	I01CR7	I01CR6	I01CR5	I01CR4	I01CR3	I01CR2	I01CR1	I01CR0
After reset			0	0	0	0	0	0	0	0

Bit name	Function		
I01CR7 (bit 7)	INT1 detection level/edge selection		
I01CR6 (bit 6)	I01CR7	I01CR6	INT1 interrupt condition
	0	0	Falling edge detection
	0	1	L level detection
	1	0	Rising edge detection
	1	1	H level detection
I01CR5 (bit 5)	INT1 interrupt source		
	0: No interrupt source 1: Interrupt source		
I01CR4 (bit 4)	INT1 interrupt enable control		
	0: Interrupt disabled 1: Interrupt enabled		
I01CR3 (bit 3)	INT0 detection level/edge selection		
I01CR2 (bit 2)	I01CR3	I01CR2	INT0 interrupt condition
	0	0	Falling edge detection
	0	1	L level detection
	1	0	Rising edge detection
	1	1	H level detection
I01CR1 (bit 1)	INT0 interrupt source		
	0: No interrupt source 1: Interrupt source		
I01CR0 (bit 0)	INT0 interrupt enable control		
	0: Interrupt disabled 1: Interrupt enabled		

I01CR7 (bit 7): INT1 detection level/edge selection

I01CR6 (bit 6):

These bits select the INT1 interrupt condition for signals input on the P71/INT1 pin.

I01CR7	I01CR6	INT1 interrupt condition
0	0	Falling edge detection
0	1	L level detection
1	0	Rising edge detection
1	1	H level detection

I01CR5 (bit 5): INT1 interrupt source

This bit is set if the condition specified by bits I01CR7 and 6 is met. If INT1 interrupts are enabled (I01CR4 = 1), then control jumps to vector address 000BH and interrupt processing begins. This bit is not reset, even when interrupt processing is completed. Therefore, it is necessary for this bit to be reset by software.

I01CR4 (bit 4): INT1 interrupt enable control

This bit enables (1)/disables (0) the acceptance of external interrupt 1 (INT1). When this bit is set to "1," then when I01CR5 is set, INT1 interrupt processing is executed; when this bit is set to "0," interrupt processing is not executed.

I01CR3 (bit 3): INT0 detection level/edge selection

I01CR2 (bit 2):

These bits select the INT0 interrupt condition for signals input on the P70/INT0 pin.

I01CR3	I01CR2	INT0 interrupt condition
0	0	Falling edge detection
0	1	L level detection
1	0	Rising edge detection
1	1	H level detection

I01CR1 (bit 1): INT0 interrupt source

This bit is set if the condition specified by bits I01CR3 and 2 is met. If INT0 interrupts are enabled (I01CR0 = 1), then control jumps to vector address 0003H and interrupt processing begins. This bit is not reset, even when interrupt processing is completed. Therefore, it is necessary for this bit to be reset by software.

I01CR0 (bit 0):

INT0 interrupt enable control

This bit enables (1)/ disables (0) the acceptance of external interrupt 0 (INT0). When this bit is set to "1," then when I01CR1 is set, INT0 interrupt processing is executed; when this bit is set to "0," interrupt processing is not executed.

- External Interrupt 2, 3 Control Register (I23CR)

For details, refer to Chapter 3, section 3.2.4, "External Interrupt 2, 3 Control Register."

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
I23CR	15EH	R/W	I23CR7	I23CR6	I23CR5	I23CR4	I23CR3	I23CR2	I23CR1	I23CR0
After reset			0	0	0	0	0	0	0	0

Bit name	Function
I23CR7 (bit 7)	INT3 rising edge detection control
	0: Do not detect 1: Detect
I23CR6 (bit 6)	INT3 falling edge detection control
	0: Do not detect 1: Detect
I23CR5 (bit 5)	INT3 interrupt source
	0: No interrupt source 1: Interrupt source
I23CR4 (bit 4)	INT3 interrupt enable control
	0: Interrupt disabled 1: Interrupt enabled
I23CR3 (bit 3))	INT2 rising edge detection control
	0: Do not detect 1: Detect
I23CR2 (bit 2)	INT2 falling edge detection control
	0: Do not detect 1: Detect
I23CR1 (bit 1)	INT2 interrupt source
	0: No interrupt source 1: Interrupt source
I23CR0 (bit 0)	INT2 interrupt enable control
	0: Interrupt disabled 1: Interrupt enabled

- Input Signal Select Register (ISL)

For details, refer to Chapter 3, section 3.2.4, "Input Signal Selection Register."

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ISL	15FH	R/W	-	-	ISL5	ISL4	ISL3	ISL2	ISL1	ISL0
After reset			H	H	0	0	0	0	0	0

Bit name	Function		
ISL5 (bit 5) ISL4 (bit 4)	Base timer clock selection		
	ISL5	ISL4	
	1	1	Timer/counter T0 prescaler
	0	1	Cycle clock
ISL3 (bit 3)	X	0	Sub-clock (crystal oscillation)
	Buzzer output frequency selection		
	0: fBST/16		
	1: fBST/8		
ISL2 (bit 2)	Noise elimination filter time constant selection		
ISL1 (bit 1)	ISL2	ISL1	Time constant
	1	1	16Tcyc
	0	1	64Tcyc
	X	0	1Tcyc
ISL0 (bit 0)	T0 clock input pin selection		
	0: P72/INT2/T0IN pin		
	1: P73/INT3/T0IN pin		

- Master Interrupt Enable Control Register (IE)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IE	108H	R/W	IE7	-	-	-	-	-	IE1	IE0
After reset			H	H	0	0	0	0	0	0

Bit name	Function			
IE7 (bit 7)	Master interrupt enable control (high level, low level)			
	0: All interrupt requests disabled 1: All interrupt requests enabled			
IE1 (bit 1) IE0 (bit 0)	INT0, INT1 interrupt priority control			
	IE1	IE0	INT1 priority level	INT0 priority level
	0	0	Highest level	Highest level
	1	0	Low level	Highest level
	X	1	Low level	Low level

– IE7 (bit 7): Master interrupt enable control

This bit enables (1)/disables (0) the acceptance of all "high level" and "low level" interrupts. When this bit is set to "1," all interrupts for which interrupt requests have been generated are enabled; when this bit is set to "0," "high level" and "low level" interrupts are disabled.

IE1 (bit 1):

INT0, 1 interrupt priority control

IE0 (bit 0):

These bits set the priority level for external interrupts INT0 and 1.

IE1	IE0	INT1 priority level	INT0 priority level
0	0	Highest level	Highest level
1	0	Low level	Highest level
X	1	Low level	Low level

Note:

- Although "low level" priority for INT0 and 1 is controlled by IE7, "highest level" priority is not.
 - It is not possible to set just external interrupt INT1 alone to "highest level."
-

8. Port Interrupt Functions

8.1 Overview

In addition to its digital I/O function, port 3 can be used to generate interrupts or release HOLD mode. This function can be used to implement a "key-on wakeup" function that releases HOLD mode when a key switch is pressed.

A port interrupt can be implemented through port 3.

8.2 Function

In addition to its digital I/O function, port 3 also has the following functions:

- Generates an interrupt when it detects a low-level signal.
- Releases HOLD mode when it detects a low-level signal.

After HOLD mode is released, the internal RC oscillation is adopted for the system clock.

In order to use the port interrupt function, it is necessary to manipulate the Special Function Registers shown below.

For port 3 interrupt:

- P3
- P3DDR
- P3INT
- IE

8.3 Circuit Configuration

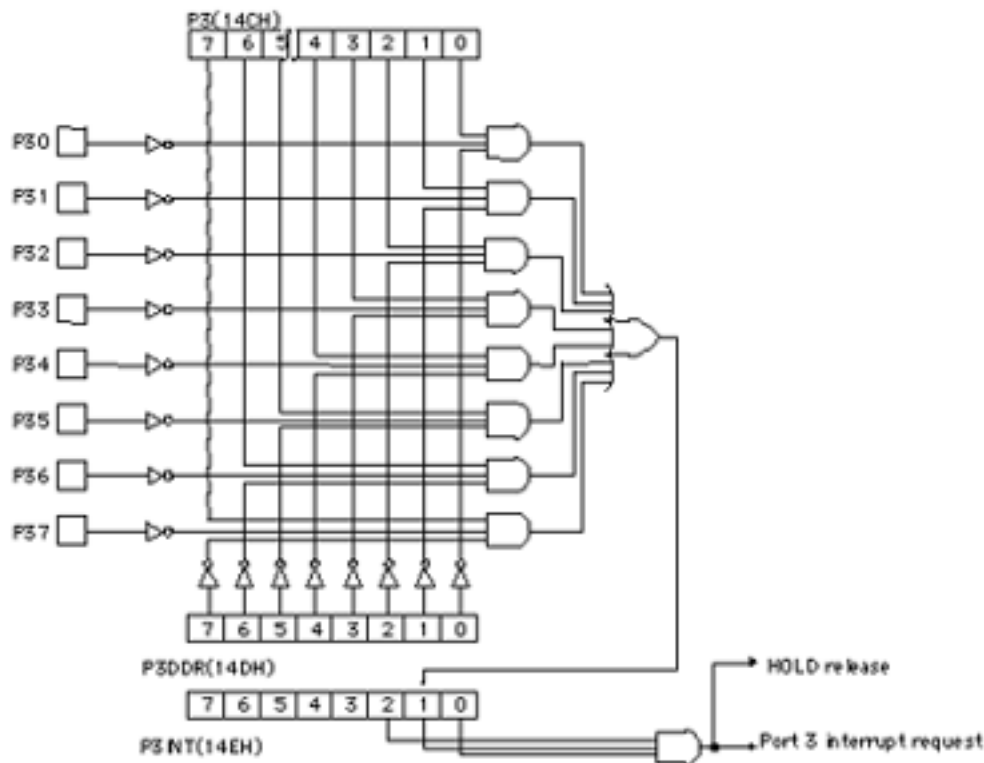


Figure 3.36 Port 3 Interrupt Circuit Block Diagram

8.4 Related Registers

- Port 3 Interrupt Control Register (P3INT)

For details, refer to Chapter 3, section 3.1.2, "Port 3 Interrupt Control Register."

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P3INT	14EH	R/W	-	-	-	-	-	P32INT	P31INT	P30INT
After reset			H	H	H	H	H	0	0	0

Bit name	Function
P32INT (bit 2)	Port 3 interrupt control flag
	0: Interrupts through port 3 and HOLD mode release through port 3 disabled. 1: Interrupts through port 3 and HOLD mode release through port 3 enabled.
P31INT (bit 1)	Port 3 interrupt source flag
	0: No interrupt source 1: Interrupt source
P30INT (bit 0)	Port 3 interrupt request enable
	0: Interrupt request enabled. 1: Interrupt request disabled.

- Master interrupt enable control register (IE)

For details, refer to chapter 3, section 3.8.3, "Master Interrupt Enable Control Register."

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IE	108H	R/W	IE7	-	-	-	-	-	IE1	IE0
After reset			0	H	H	H	H	H	0	0

Bit name	Function			
IE7 (bit 7)	Master interrupt enable control (high level, low level)			
	0: All interrupt requests disabled. 1: All interrupt requests enabled.			
IE1 (bit 1)	INT0, 1 interrupt priority control			
IE0 (bit 0)	IE1	IE0	INT1 priority level	INT0 priority level
	0	0	Highest level	Highest level
	1	0	Low level	Highest level
	X	1	Low level	Low level

8.5 Description of Operation

- Port 3 interrupts

Set bit 2 of the port 3 interrupt control register (P3INT) to "1." This selects the port 3 interrupt.

Through a Special Function Register, select the pin in port 3 (P37 through P30) on which the low-level signal is to be detected.

The following conditions must be met in order to accept a port 3 interrupt:

- The corresponding bit in the Port 3 Control Register (P3DDR) must be set to input mode.

$P3mDDR = 0$ ($m = 0$ to 7)

- The corresponding bit in the Port 3 Register (P3) must be set.

$P3n = 1$ ($n = 0$ to 7)

① If a low-level signal is detected, the interrupt source is set to "1." If the interrupt request enable flag has been set, an interrupt request is generated, and if the master interrupt enable flag has been set, control branches to vector address 004BH.

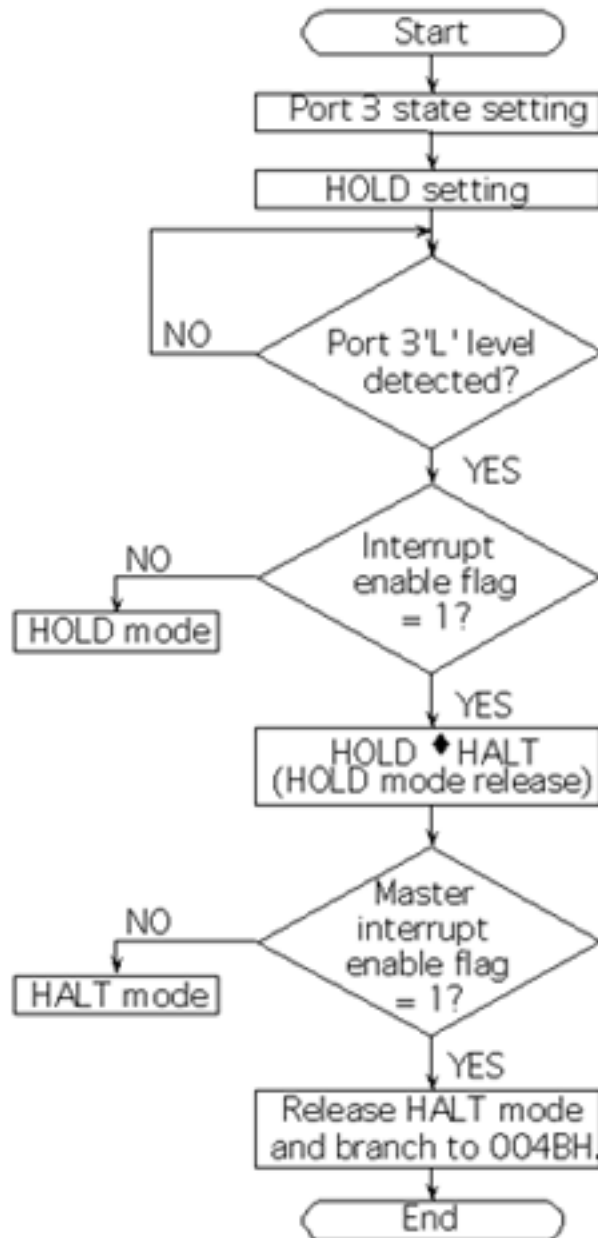
② If the two conditions described in item 2 above are met while in HALT mode, HALT mode is released and control branches to vector address 004BH.

③ If the two conditions described in item 2 above are met while in HOLD mode, HOLD mode is released and control branches to vector address 004BH. In this case, the internal RC oscillation is selected for the system clock.

8.6 State Transitions

- State transitions in HOLD mode

This flowchart applies to port 3 interrupts.

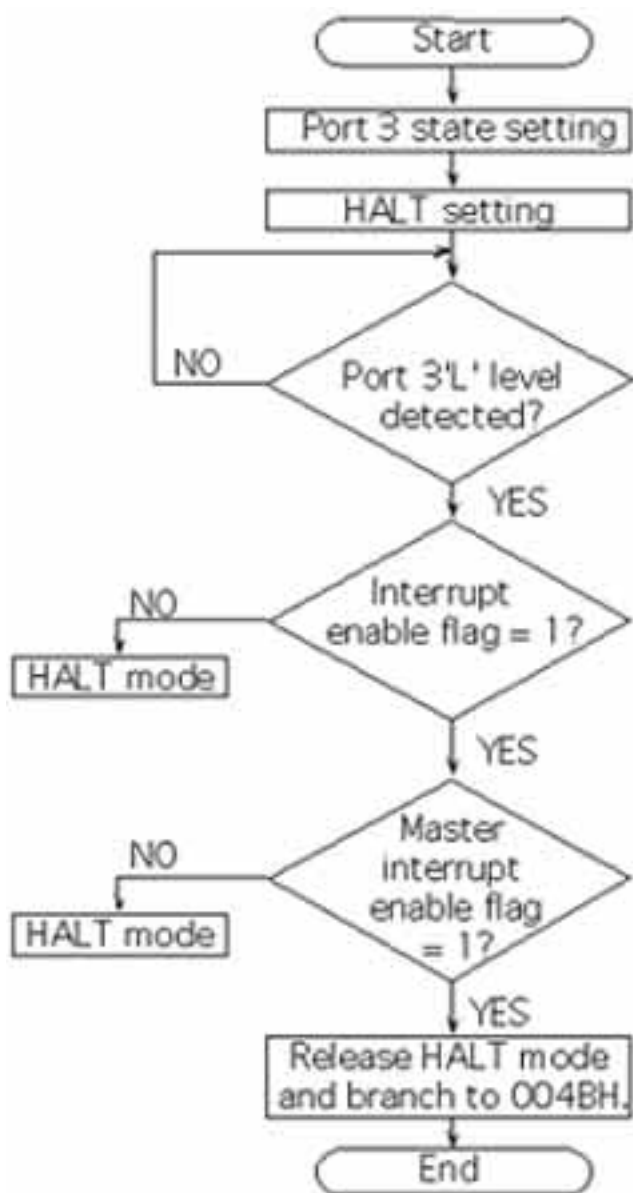


Note:

- When releasing HOLD mode through P3, set other individual interrupt request enable flags to "0."
-

- HALT mode in state transition

This flowchart applies to port 3 interrupts.

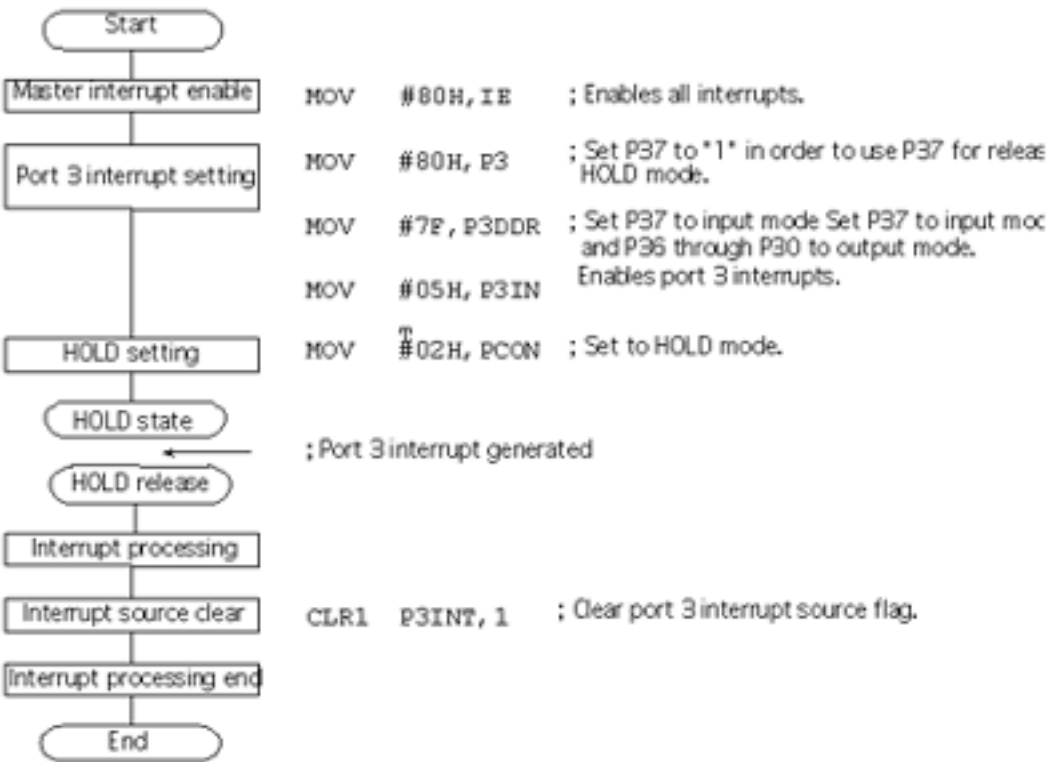


8.7 Program Example

This example applies to port 3 interrupts.

- Program

This program releases HOLD mode without branching to an interrupt routine when a low-level signal is detected on P37.



- Example application circuit

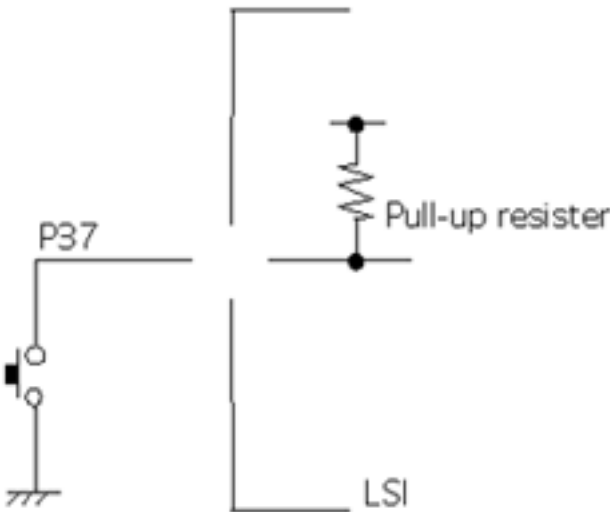


Figure 3.37 Application Circuit Example

9. VMU Work RAM

9.1 Overview

There are two banks of 256 bytes provided as communications buffers in the new-generation game machine dedicated interface. As long as a data transfer is not being performed on the LM-BUS, these buffers can also be accessed as work RAM.

Whether a data transfer is being conducted with the new generation game machine can be determined by referencing the ASEL flag in the VSEL register. (A data transfer is in progress when ASEL = "1.") Note that a normal data transfer cannot be guaranteed if this work RAM is accessed while a data transfer is being performed with a new generation game machine.

9.2 Work RAM Control Registers

VMU Control Register (VSEL)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
VSEL	163H	R/W	-	-	-	INCE	-	-	SIOSEL	ASEL
After reset			H	H	H	0	H	H	0	0

Game programs can access only bit 4. A bit manipulation instruction must be used.

INCE (bit 4):

VTRBF address counter automatic increment

This bit controls the automatic incrementing of the address counter when writing/reading VTRBF from the CPU side.

When this bit is set to "1," the address counter is automatically incremented by one after VTRBF has been accessed from the CPU side. When this bit is set to "0," the current address is saved after access.

SIOSEL (bit 1):

P1 port usage selection control

This bit controls the selection of whether the P1 port (P10 to P15) is to be used as a normal I/O port and as I/O pins for a synchronous serial interface, or as the new game machine dedicated interface. Always set this bit to "0" in game program mode.

ASEL (bit 0):

VTRBF address input select control

This bit controls the selection of access to VTRBF that is used as a buffer for the VMU and the new generation game machine dedicated interface.

Always set this bit to "0" in game program mode. If this bit is set to "1," access from the MPU to VTRBF is not possible.

- Work RAM access address (VRMAD1, 2)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
VRMAD1	164H	R/W	VRMAD7	VRMAD6	VRMAD5	VRMAD4	VRMAD3	VRMAD2	VRMAD1	VRMAD0
After reset			0	0	0	0	0	0	0	0

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
VRMAD2	165H	R/W	-	-	-	-	-	-	-	VRMAD8
After reset			H	H	H	H	H	H	H	0

These registers set the address that is to be accessed from the CPU side in work RAM (VTRBF).

VRMAD1 is the lower 8 bits of the address; VRMAD2 switches the bank.

When the VSEL bit is set to "1," VRMAD is incremented each time that VTRBF is accessed.

- Send/receive buffer (VTRBF)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
VTRBF	166H	R/W	VTRBF7	VTRBF6	VTRBF5	VTRBF4	VTRBF3	VTRBF2	VTRBF1	VTRBF0
After reset			0	0	0	0	0	0	0	0

This register is used to access the data in the address specified by VRMAD.

When the CPU writes to this register, the data is written to the address specified by VRMAD. when the CPU reads this register, it reads the data from the address that was specified by VRMAD.

If the VSEL bit is set to "1," VRMAD is incremented automatically each time that this register is accessed.

9.3 Accessing Work RAM

When accessing VMU work RAM, the address that is to be accessed is stored in the VRMAD1 and 2 registers. An application accesses work RAM by storing the address value in VRMAD1 and 2 and then reading or writing the VTRBF register.

It is important to note that the VRMAD1 and 2 registers have an auto-increment function. For details, refer to the next item.

9.4 Notes on Using the Address Register for Work RAM

Figure below illustrates an access to work RAM.

When a game program accesses work RAM, the address value is specified by the VRMAD1 and 2 registers.

It is essential to note that if the INCE flag in the VSEL register has been set to "1," the value in VRMAD is automatically incremented each time that game program work RAM is accessed. (No distinction is made between reads and writes.)

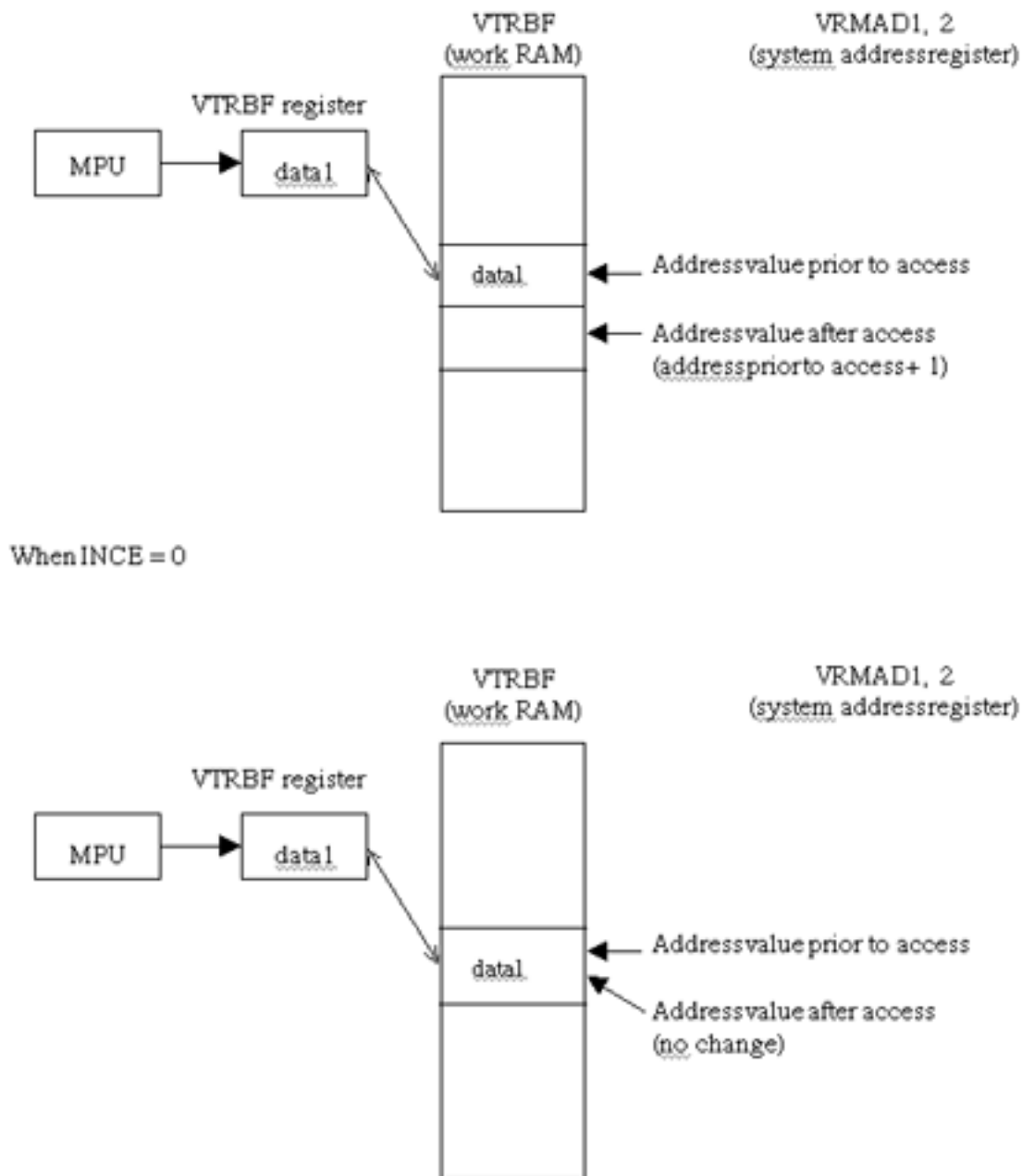


Figure 3.38 VMU Work RAM Access

10. Flash EEPROM

10.1 Overview

POTATO has an internal 128K flash EEPROM (Electrically Erasable Programmable ROM) that can operate with a single power supply.

10.2 Functions

- Programming and read operations possible with a single power supply
- Capacity: 131072 × 8 bits: Data area
- Programmable/erasable in block (page) units
block = 128 bytes (= 1 page)
- Number of times overwriting is possible
50,000 times/page (Ta = 25°C) (with memory management by program)
- Built-in step-up voltage circuit for writing
- Overwriting end detection function (detected within subroutine call in the internal BIOS program)
Toggle bit method
Data polling method
- Batch erase of software possible

10.3 Accessing the Data Area EEPROM

The data area can be written by using a PROM writer, and can be read and written by calling a subroutine in the microcontroller's internal BIOS program.

3.10.3.1 Reading/Writing Data in a Program

Loading the internal BIOS programs makes it possible to easily access the data area EEPROM.

BIOS programs that are used:

FM_WRT_EX	(writes data to the data area EEPROM)
FM_VRF_EX	(Verifies writing data in the data area EEPROM)
FM_PRD_EX	(reads the data in the data area EEPROM)

Data area EEPROM space: 128 × 8 bits × 1024 pages

When accessing (reading or writing) the data area EEPROM, load and use the target internal BIOS programs from within the program as subroutine calls ("callr" instruction and "callf" instruction).

- FM_WRT_EX

This subroutine is loaded in order to write data to the data area EEPROM.

Loading this subroutine writes one page (128 x 8 bits). The following settings must be made beforehand in order to load this subroutine.

(Items that should be set beforehand)

- | | |
|--------------------------------|---|
| address (17-bit specification) | <ol style="list-style-type: none">1. Specification of the EEPROM write start address (17-bit specification)<ol style="list-style-type: none">(1) Lower address (8 bits): Specify in 7FH in RAM (Bank-1).(2) Upper address (8 bits): Specify in 7EH in RAM (Bank-1).(3) Bank address (1 bit): Specify in bit 0 of 7DH in RAM (Bank-1).When setting the lower address, remember that it is not possible to write data so that it spans two pages.
Because this writing operation is performed one page (128 bytes) at a time, specify zeroes ("0") for bits 0 through 6 of the lower address.2. Setting the data that is to be written in EEPROM<ul style="list-style-type: none">• Set the data that is to be written in EEPROM in RAM at addresses 80H through 0FFH ahead of time.3. Specification of the method for detecting the end of the EEPROM writing operation<ul style="list-style-type: none">• There are two methods for detecting the end of the EEPROM writing operation; specify which of those methods is to be used.<ol style="list-style-type: none">(1) Toggle bit method: Bit 0 of 7CH in RAM (Bank-1) = 0(2) Data polling method: Bit 0 of 7CH in RAM (Bank-1) = 1• Specify which of the above two methods is to be used through bit 0 of 7CH in RAM. |
|--------------------------------|---|

- FM_VRF_EX

This subroutine is used to verify that the data that was written in the data area EEPROM was written correctly.

Loading this subroutine compares one page of data (128 x 8 bits) written in EEPROM with the original data (the data in addresses 80H through 0FFH in RAM (Bank-1)).

- When all 128 bytes are correct: Accumulator (ACC) = 00H
- When even one of the 128 bytes is correct: Accumulator (ACC) \neq 00H

This subroutine must be executed after the data has been completely written to EEPROM, but before the data in addresses 80H through 0FFH in ROM has been overwritten.

(Necessary data for loading)

1. Specification of the EEPROM read start address (17-bit specification)
 - (1) Lower address (8 bits): Specify in 7FH in RAM (Bank-1).
 - (2) Upper address (8 bits): Specify in 7EH in RAM (Bank-1).
 - (3) Bank address (1 bit): Specify in bit 0 of 7DH in RAM (Bank-1).

When setting the lower address, remember that it is not possible to read data spanning two pages.

2. EEPROM data and the comparison data

This subroutine verifies that the data that was written in EEPROM was written correctly. The data that is compared to the EEPROM data is the data in addresses 80H through 0FFH in RAM (Bank-1), starting from the data in address 80H.

3. End of the EEPROM reading operation

- (1) When comparison results do not match
- (2) When the lower 7 bits of the address are "7FH"

- FM_PRD_EX

This subroutine is used to read data from the data area EEPROM.

Loading this subroutine reads one page (128 x 8 bits). The following settings must be made beforehand in order to load this subroutine.

(Items that should be set beforehand)

1. Specification of the EEPROM read start address (17-bit specification)

- (1) Lower address (8 bits): Specify in 7FH in RAM (Bank-1).
- (2) Upper address (8 bits): Specify in 7EH in RAM (Bank-1).
- (3) Bank address (1 bit): Specify in bit 0 of 7DH in RAM (Bank-1).

When setting the lower address, remember that it is not possible to read data that spans two pages.

Because this reading operation is performed one page (128 bytes) at a time, specify zeroes ("0") for bits 0 through 6 of the lower address.

2. Data that was read from EEPROM

The data that is read from EEPROM is written in addresses 080H through 0FFH in RAM (Bank-1).

Note: When accessing the data area EEPROM, disable all interrupts before making subroutine calls of the microcomputer's internal OS program. In some cases interrupts cannot be accepted while accessing the data area EEPROM, which can lead to misoperation.

10.4 Accessing the Program Area EEPROM

The program area is created in the EEPROM. As a rule, when POTATO is shipped, the entire program area is filled with "FFH." The method provided for the user in order to write program data in the program area is described below.

10.5 Writing with a PROM Writer

A third-party PROM writer and Sega's Conversion Board (W86F8716Q) can be used to easily write the program area.

(Refer to Chapter 3, section 3.10.3.2, "Writing/Reading with a PROM Writer.")

Note that using this method after the board has been installed does carry a risk of causing problems with other circuits.

4. Control Functions

1. Interrupt Function

The interrupt function is used to temporarily interrupt the program that the microcomputer is currently executing and then execute another program in order to address an urgent need. POTATO includes circuits for generating 12 types of interrupt requests. In the VMU, some types of interrupt processing cannot be set as desired from within a game program. The interrupts are shown in Table on next page.

1.1 Types of Interrupts

Table 4.1 *List of interrupts*

Priority ranking	Interrupt type	Internal/external	Vector address	Interrupt request	Source flag	Enable flag	Register address	Priority ranking setting
1	External interrupt INT0	external	0003H	P70/INT0 pin event detection	I01CR1	I01CR0	15DH	Highest/low
2	External interrupt INT1	external	000BH	P71/INT1 pin event detection	I01CR5	I01CR4	15DH	
3	External interrupt INT2	external	0013H	P72/INT2 pin event detection	I23CR1	I23CR0	15EH	High/low
	Timer/counter TOL (lower 8 bits)	Internal		Timer/counter TOL lower 8 bits overflow	T0CNT1	T0CNT0	110H	
4	External interrupt INT3	external	001BH	P73/INT3 pin event detection	I23CR5	I23CR4	15EH	High/low
	Base timer	Internal		Base timer overflow	BTCR1	BTCR0	17FH	
					BTCR3	BTCR2		
5	Timer/counter T0H (upper 8 bits)	Internal	0023H	Timer/counter T0H upper 8 bits overflow	T0CNT3	T0CNT2	110H	High/low
6	Timer T1	Internal	002BH	Timer T1L overflow	T1CNT1	T1CNT0	118H	High/low
				Timer T1H overflow	T1CNT3	T1CNT2		
7	SI00	Internal	0033H	SI00 end detection	SCON01	SCON00	130H	High/low
8	SI01	Internal	003BH	SI01 end detection	SCON11	SCON10	134H	High/low
9	VMU interrupt	Internal	0043H	VMU communications reception end detection	RFB	RFBENA	160H/161H	High/low
10	Port 3 interrupt (P32INT = 1)	external	004BH	Port 3 I low level detection	P31INT	P30INT	14EH	High/low

Note:

- The "priority ranking" indicates the order of priority given to interrupts when multiple interrupts are generated simultaneously. However, the priority ranking changes when specified in the Interrupt Priority Control Register (IP).

1.2 Interrupt Function Operation

- If an interrupt is generated from an interrupt request source that is shown in Table 4-1-1, the corresponding interrupt request flag is set.

If the interrupt request enable flag that corresponds to the interrupt request source is set, the microcomputer's interrupt control circuit is notified of the interrupt request.

- The interrupt control circuit accepts the interrupt according to the priority ranking rules.

Interrupts can have a priority of either "highest level," "high level," or "low level;" in order to enable a high level or low level interrupt, it is necessary to set the master interrupt enable flag (IE7) in addition to the individual interrupt enable flags. IE7 controls high level and low level interrupts. In addition, if "highest level" has been set for INT0 or INT1 by the interrupt priority control flags (IE1, 0), interrupt processing is executed regardless of the master interrupt enable flag.

- The interrupt sources with an interrupt priority ranking from 3 to 9 can be specified as having either "high level" or "low level" interrupt priority according to the Interrupt Priority Control Register (IP).
- If an interrupt is generated, then after execution of the instruction that is currently being executed is completed, the interrupt control circuit automatically stores the contents of the program counter (PC) in the stack (in RAM), and then the microcomputer executes the interrupt service program. Because the program counter data uses two bytes of the stack, the stack pointer (SP) is incremented by 2. After control returns from the interrupt service program, the SP is decremented by 2.
- After executing the interrupt service program, the microcomputer executes the RETI instruction in order to resume execution of the original program.
- Up to three interrupts can be nested.
- Interrupt request flag acceptance processing is not performed while executing the RETI instruction, while executing any instructions (such as MOV or ST) that write to the Special Function Registers listed below, or while writing to the data area EEPROM:
 - IE
 - IP
 - PCON
 - EXT



In order to use the interrupt function, it is necessary to manipulate the following Special Function Registers:

- IE
- IP
- SP (because it is undefined after a reset)
- Special Function Registers in the function block that accepts interrupts

1.3 Circuit Configuration

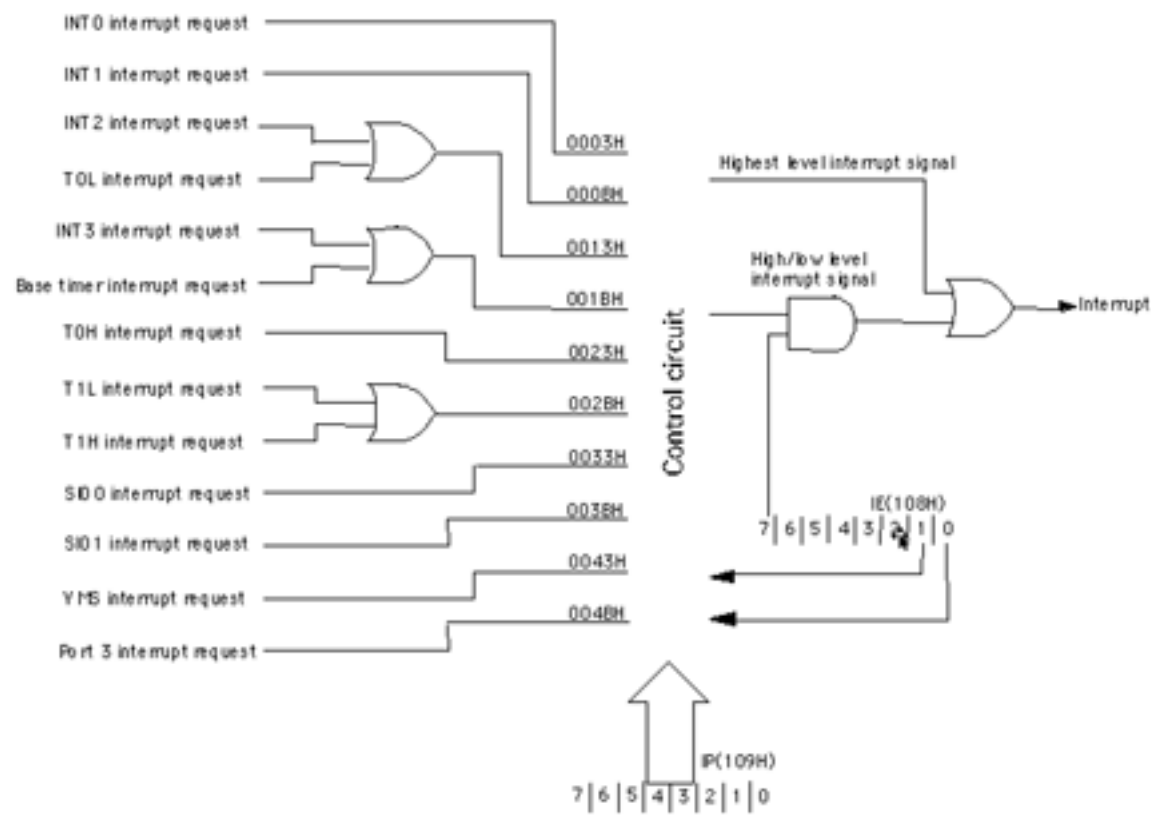


Figure 4.1 Interrupt Function Block Diagram

1.4 Related Registers

- Master Interrupt Enable Control Register (IE)
For details, refer to Chapter 3, section 3.7.3, "Master Interrupt Enable Control Register."

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IE	108H	R/W	IE7	-	-	-	-	-	IE1	IE0
After reset			0	H	H	H	H	H	0	0

Bit name	Function			
IE7 (bit 7)	Master interrupt enable control (high level, low level)			
	0: All interrupt requests disabled 1: All interrupt requests enabled			
IE1 (bit 1) IE0 (bit 0)	INT0, INT1 interrupt priority control			
	IE1	IE0	INT1 priority level	INT0 priority level
	0	0	Highest level	Highest level
	1	0	Low level	Highest level
	X	1	Low level	Low level

• Interrupt Priority Ranking Control Register (IP)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IP	109H	R/W	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0
After reset			0	-	0	0	0	0	0	0

Bit name	
IP7 (bit 7)	Port 3 interrupt priority level setting
	0: Low level
	1: High level
IP5 (bit 5)	SI01 interrupt priority level setting
	0: Low level
	1: High level
IP4 (bit 4)	SI00 interrupt priority level setting
	0: Low level
	1: High level
IP3 (bit 3)	T1 priority level setting
	0: Low level
	1: High level
IP2 (bit 2)	TOH priority level setting
	0: Low level
	1: High level
IP1 (bit 1)	INT3 and base timer interrupt priority level setting
	0: Low level
	1: High level
IP0 (bit 0)	INT2 and TOL interrupt priority level setting
	0: Low level
	1: High level

IP7 (bit 7):	<p>Port 3 interrupt priority level setting</p> <p>This bit selects either "high" (1) or "low" (0) for the port 3 interrupt priority level. When this bit is set to "1," the priority level for this interrupt is set to "high level," giving this interrupt higher priority than low level INT0 and INT1 interrupts (IE0 = 1). When this bit is set to "0," the priority level for this interrupt is set to "low level."</p>
IP5 (bit 5):	<p>SIO1 interrupt priority level setting</p> <p>This bit selects either "high" (1) or "low" (0) for the SIO1 interrupt priority level. When this bit is set to "1," the priority level for this interrupt is set to "high level," giving this interrupt higher priority than low level INT0 and INT1 interrupts (IE0 = 1). When this bit is set to "0," the priority level for this interrupt is set to "low level."</p>
IP4 (bit 4):	<p>SIO0 interrupt priority level setting</p> <p>This bit selects either "high" (1) or "low" (0) for the SIO0 interrupt priority level. When this bit is set to "1," the priority level for this interrupt is set to "high level," giving this interrupt higher priority than low level INT0 and INT1 interrupts (IE0 = 1). When this bit is set to "0," the priority level for this interrupt is set to "low level."</p>
IP3 (bit 3):	<p>T1 priority level setting</p> <p>This bit selects either "high" (1) or "low" (0) for the T1 interrupt priority level. When this bit is set to "1," the priority level for this interrupt is set to "high level," giving this interrupt higher priority than low level INT0 and INT1 interrupts (IE0 = 1). When this bit is set to "0," the priority level for this interrupt is set to "low level."</p>
IP2 (bit 2):	<p>T0H priority level setting</p> <p>This bit selects either "high" (1) or "low" (0) for the T0H interrupt priority level. When this bit is set to "1," the priority level for this interrupt is set to "high level," giving this interrupt higher priority than low level INT0 and INT1 interrupts (IE0 = 1). When this bit is set to "0," the priority level for this interrupt is set to "low level."</p>
IP1 (bit 1):	<p>INT3 and base timer interrupt priority level setting</p> <p>This bit selects either "high" (1) or "low" (0) for the INT3/base timer interrupt priority level. When this bit is set to "1," the priority level for this interrupt is set to "high level," giving this interrupt higher priority than low level INT0 and INT1 interrupts (IE0 = 1). When this bit is set to "0," the priority level for this interrupt is set to "low level."</p>
IP0 (bit 0):	<p>INT2 and T0L interrupt priority level setting</p> <p>This bit selects either "high" (1) or "low" (0) for the INT2/T0L interrupt priority level. When this bit is set to "1," the priority level for this interrupt is set to "high level," giving this interrupt higher priority than low level INT0 and INT1 interrupts (IE0 = 1). When this bit is set to "0," the priority level for this interrupt is set to "low level."</p>

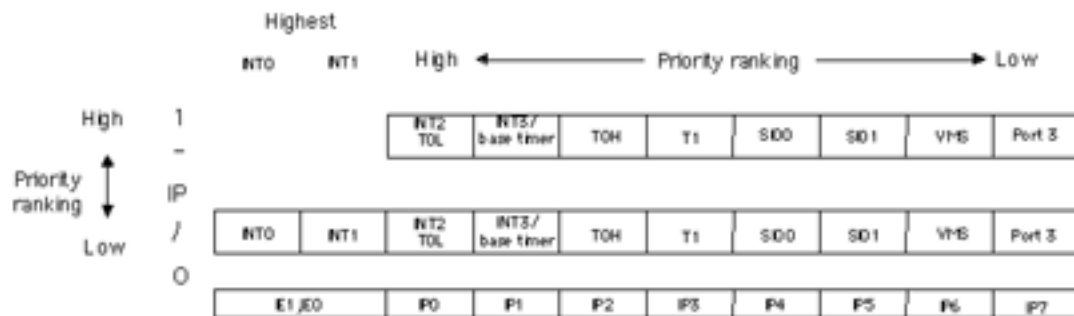
1.5 Interrupt Priority Ranking

The priority ranking of the interrupt levels is as follows:

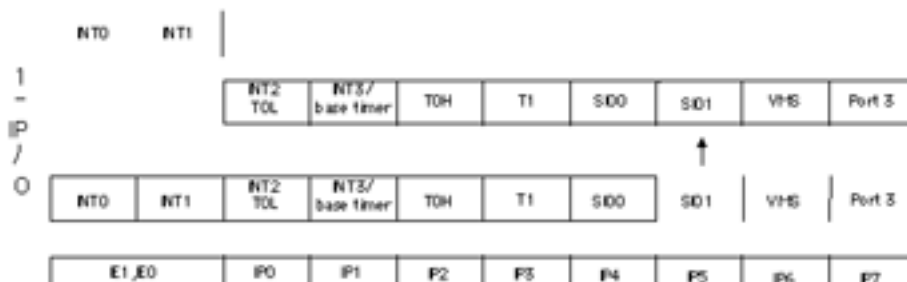
Highest level > high level > low level

The priority ranking of multiple interrupt sources of the same priority ranking level that are generated simultaneously is as listed in Table 4-1-1. In addition, the overlapping interrupt control circuit controls overlapping interrupts, permitting nesting of "low level" Æ "high level" Æ "highest level" interrupt routines.

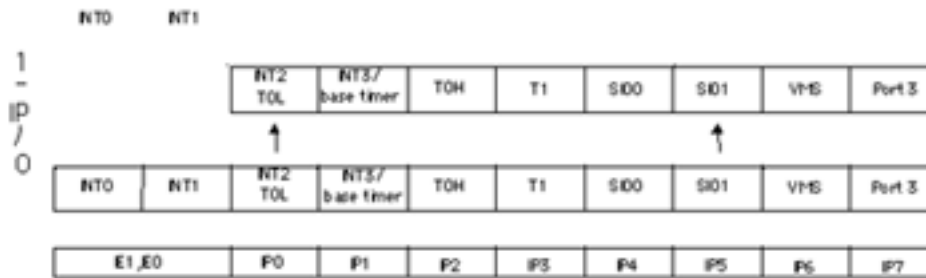
Highest level:	External interrupts INT0 and INT1 (when set to "highest level") This level is not controlled by the mask interrupt enable flag (IE7).
High level:	Those interrupt sources other than INT0 and INT1 that correspond to the bits that are set in the interrupt priority control register (IP). This level is controlled by the mask interrupt enable flag (IE7).
Low level:	Those interrupt sources INT0 and INT1 for which "low level" is set in IE0 or IE1, and those interrupt sources other than INT0 and INT1 that correspond to the bits that are reset (0) in the interrupt priority control register (IP). This level is controlled by the mask interrupt enable flag (IE7).



- To give the SIO1 end interrupt higher priority than the INT0 interrupt, set IE0 to "1" and IP5 to "1." (IE0 = 1, IP = 001 00000B)



- To give the SIO1 end interrupt priority between the INT2 interrupt and the INT0 interrupt, set IE0 to "1" and IP5 and IP0 to "1." (IE0 = 1, IP = 00100001B)



- Notes concerning overlapping interrupts
- When a low-level interrupt request is generated while executing the service program for a high-level interrupt, the low-level interrupt is accepted after one instruction is executed after the end of the service program for the high-level interrupt.
- When an interrupt request of the same level as an interrupt request for which a service program is already being executed is generated, that second interrupt request is not accepted.

2. System Clock Generation Function

2.1 Overview

POTATO has three internal oscillation circuits for use as system clock generation circuits: the main clock oscillation circuit, the sub-clock oscillation circuit, and the RC oscillation circuit. Of these, the RC oscillation circuit has an internal resistor (R) and capacitor (C), and does not require any external circuitry. The selection of one of these three clocks as the system clock is made through software.

Note that, in actual practice, battery consumption is high when the main clock oscillation circuit and the RC oscillation circuit are used, so select the sub-clock as the system clock whenever the other circuits are not needed.

2.2 Functions

- This function generates the system clock, which is the foundation of the execution of instructions by the microcomputer.
- One of two clocks (sub-clock oscillation or RC oscillation) can be selected as the system clock through software. Game programs should not use the main clock oscillation.
- This function generates the base timer clock.
- Main clock oscillation and RC oscillation can be halted by software instructions.
(This makes it possible to conserve battery power.)
- This function generates system clock 1 (S1), which is the foundation for operation of circuit blocks that still operate in HALT mode, and system clock 2 (S2), which is the foundation for operation of circuit blocks that stop operating in HALT mode.
- In HOLD mode, main clock oscillation, sub-clock oscillation, and RC oscillation are all stopped.

In order to control the system clock, it is necessary to manipulate the following Special Function Registers:

- OCR
- PCON

2.3 Circuit Configuration

- Main clock oscillation circuit..... ▴

This circuit is made to oscillate by connecting a ceramic oscillation circuit to the CF1 and CF2 pins. If the main clock is not to be used, connect CF1 to VDD and leave the CF2 pin open.

- Sub-clock oscillation circuit..... ▾

This circuit is made to oscillate by connecting a crystal oscillation circuit (32.768kHz typ.) to the XT1 and XT2 pins.

If the sub-clock is not to be used, connect XT1 to VDD and leave the XT2 pin open.

- Internal RC oscillation circuit..... ®

This circuit is made to oscillate by a resistor (R) and capacitor (C) that are built into the microcomputer. After a reset or the release of HOLD, the system runs according to this clock.

- System clock selector..... ¯

Bits 4 and 5 of the Oscillation Control Register (OCR) are used to select either the sub-clock oscillation circuit or the RC oscillation circuit as the system clock source. Game programs should not use the main clock oscillation.

- System clock generation circuit..... °

System clocks 1 and 2 are generated from the clock source that was selected by the system clock selector. System clock 1 (S1) runs when executing instructions and when in HALT mode. System clock 2 (S2) runs when executing instructions. When in HOLD mode, both S1 and S2 stop.

- Oscillation Control Register (OCR)..... ±

This register controls the start and stop of oscillation by the main clock oscillation circuit and the RC oscillation circuit, switches the system clock source, and controls the cycle time.

- Power Control Register (PCON)..... ▾

This register sets the standby state (HOLD / HALT mode).

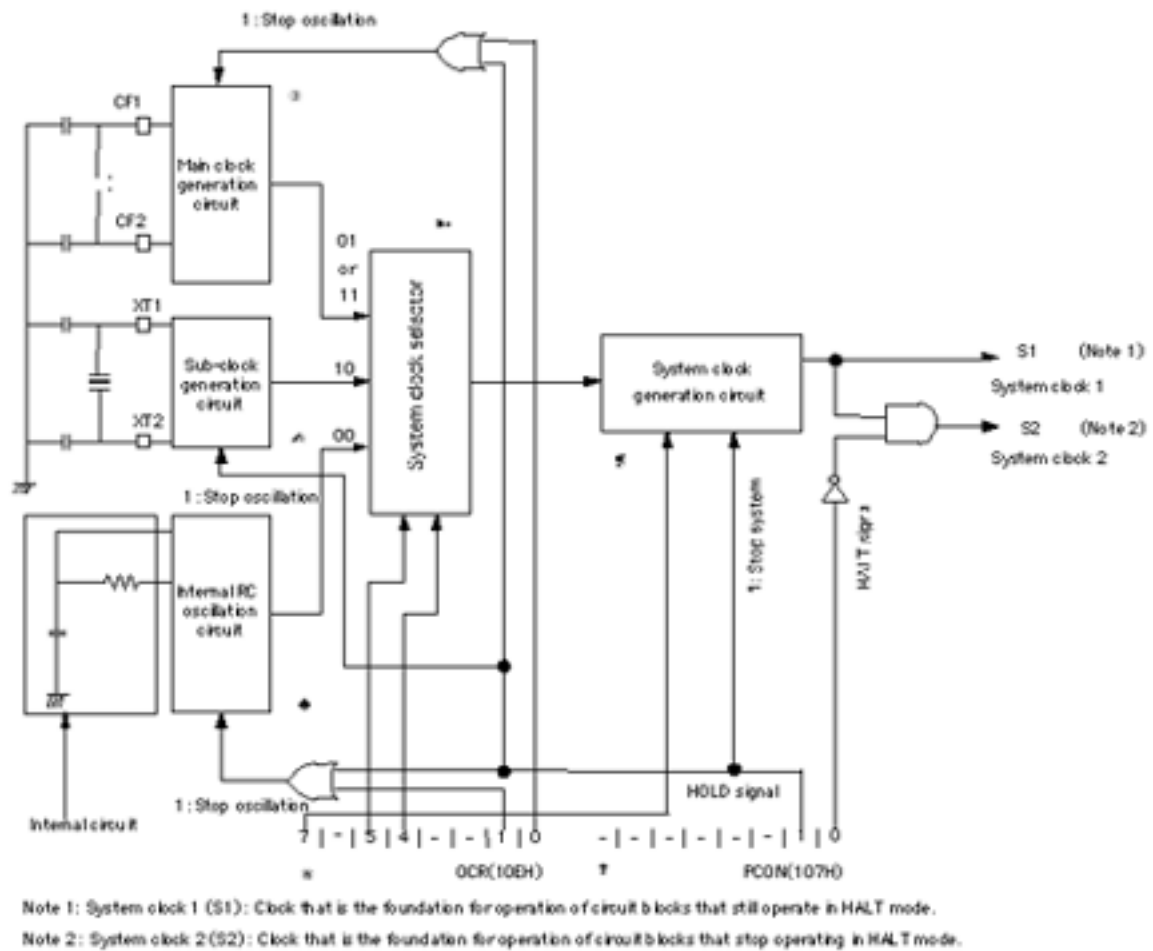


Figure 4.2 System Clock Generation Circuit Block Diagram

- Status of each block during reset, HALT, and HOLD

Table 4.2 Status of Each Block During Standby

Block	State		
	During reset	During HALT	During HOLD
Main clock oscillation circuit	Oscillates	Status when power is suddenly applied	Stopped
Internal RC oscillation circuit	Oscillates	Status when power is suddenly applied	Stopped
Sub-clock oscillation circuit	Stopped	Status when power is suddenly applied	Stopped
System clock oscillation circuit	Running	Running	Stopped

Note:

- After a reset or after HOLD is released, the internal RC oscillation clock is automatically selected as the system clock.

2.4 Related Registers

- Oscillation Control Register

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OCR	10EH	R/W	OCR7	-	OCR5	OCR4	-	-	OCR1	OCR0
After reset			0	H	0-	0	H	H	0	0

Bit name	Function		
OCR7 (bit 7)	System clock generation circuit control		
	0: Cycle time source is 1/12 of the oscillating frequency 1: Cycle time source is 1/6 of the oscillating frequency		
OCR5 (bit 5)	System clock selection		
OCR4 (bit 4)	OCR5	OCR4	System clock
	0	0	Internal RC oscillation
	0	1	Do not use
	1	0	Sub-clock (crystal oscillation)
	1	1	After reset or HOLD release: RC oscillation
OCR1 (bit 1)	Internal RC oscillation circuit control		
	0: Internal RC oscillation circuit operation start/in progress 1: Internal RC oscillation circuit stopped		
OCR0 (bit 0)	Main clock oscillation circuit control		
	0: Main clock oscillation circuit operation start/in progress 1: Main clock oscillation circuit stopped		

OCR7 (bit 7):

System clock generation circuit control

This bit controls whether the cycle time is to be 1 / 12 of the source oscillation frequency, or 1 / 6. When this bit is set to "1," the cycle time is implemented as 1 / 6 of the source oscillation frequency; when this bit is set to "0," the cycle time is implemented as 1 / 12 of the source oscillation frequency.

In the VMU, this bit should be set as shown below:

* Be sure to set '1' when using the sub-clock.

System clock	OCR7
Main clock (CF oscillation)	OCR7=1
Internal RC oscillation	OCR7=0/1
Sub-clock (crystal oscillation)	OCR7=1

OCR5 (bit 5):System clock selection

OCR4 (bit 4):These bits select the system clock. After a reset or after the release of HOLD mode, internal RC oscillation is selected automatically.

OCR5	OCR4	System clock
0	0	Internal RC oscillation
0	1	Do not use
1	0	Sub-clock (crystal oscillation)
1	1	Do not use

OCR1 (bit 1):

Internal RC oscillation circuit control

This bit stops (1)/starts the internal RC oscillation circuit. When this bit is set to "1," the internal RC oscillation circuit stops; when this bit is set to "0," the internal RC oscillation circuit starts or continues to run.

OCR0 (bit 0):

Main clock oscillation circuit control

This bit stops (1)/starts the main clock oscillation circuit. The main clock is not used by the VMU, so always set this bit to '1'.

Note:

- An adequate amount of time must be provided when starting the oscillation of the main clock. When the main clock is stopped, the RC clock is running, the RC (or sub-) clock is selected as the system clock, and you wish to switch the system clock to the main clock, start the main clock first and then wait an adequate amount of time before actually switching the clock.
- An adequate amount of time at least equal to the time required for starting the main clock (200us) must be provided when starting the oscillation of the sub-clock (200us). When switching the system clock source from RC oscillation to the sub-clock after releasing a reset or releasing HOLD mode, start sub-clock oscillation first and then wait an adequate amount of time before actually switching the clock.

- Power Control Register (PCON)

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PCON	107H	R/W	-	-	-	-	-	-	PCON1	PCON0
After reset			H	H	H	H	H	H	0	0

Bit name	Function
PCON1 (bit 1)	HOLD mode control
	0: 1: Set HOLD mode
PCON0 (bit 0)	HALT mode control
	0: 1: Set HALT mode

PCON1 (bit 1):

HOLD mode control

This bit selects the standby state. When this bit is set to "1," the microcomputer enters HOLD mode and, once all oscillation circuits have stopped, the system stops. When HOLD mode is released, this bit is automatically reset. Note that setting this bit to "0" does not change the standby state.

There are three methods for releasing HOLD mode:

- Reset
- Applying the specified level to the P70/INT0 or P71/INT1 pin
- Port 3 interrupt source

PCON0 (bit 0):

HALT mode control

This bit selects the standby state. When this bit is set to "1," the microcomputer enters HALT mode, the program stops at the address where the HALT was executed, and the oscillation circuits maintain their current state. HALT mode can be released an interrupt.

When HALT mode is released, this bit is automatically reset. Note that setting this bit to "0" does not change the standby state.

When HALT mode is in effect, system clock 2 (S2) stops.

2.5 System Clock Operation Mode

There are three system clocks:

- Internal RC oscillation clock

After a reset, when the power is turned on, or when HOLD mode is released, this clock is set as the system clock. Even if there are no external oscillation circuits, the microcomputer runs using just this clock.

- Main clock

The unit enters fast processing mode whenever the main clock is used, but this increases battery consumption by a factor of ten compared to internal RC oscillation. The main clock should not be used in game programs.

- Sub-clock

This is a slow processing mode that is used in order to reduce current consumption and make backup power last longer.

When operating in sub-clock mode, the main clock and the internal RC oscillation clock can be stopped in order to further reduce current consumption.

Fig. below shows the state transition diagram for the microcomputer when it enters HALT or HOLD mode.

It is important to note that if the main clock or the sub-clock is specified as the system clock in an application circuit that does not have an external main clock oscillation circuit or sub-clock oscillation circuit, the microcomputer will cease to operate.

RC OSC	: Internal RC oscillation circuit	MAIN	: Main clock oscillating frequency
MAIN OSC	: Main clock oscillation circuit	SUB	: Sub-clock oscillating frequency
SUB OSC	: Sub-clock oscillation circuit	PCON0	: Power control register bit 0 (HALT control)
Oscillating	: Oscillating state	PCON1	: Power control register bit 1 (HOLD control)
Stopped	: Stopped state	OCR0	: Oscillation control register bit 0
S1	: System clock 1	OCR1	: Oscillation control register bit 1
S2	: System clock 2	OCR4	: Oscillation control register bit 4
RC	: Internal RC oscillating frequency	OCR5	: Oscillation control register bit 5

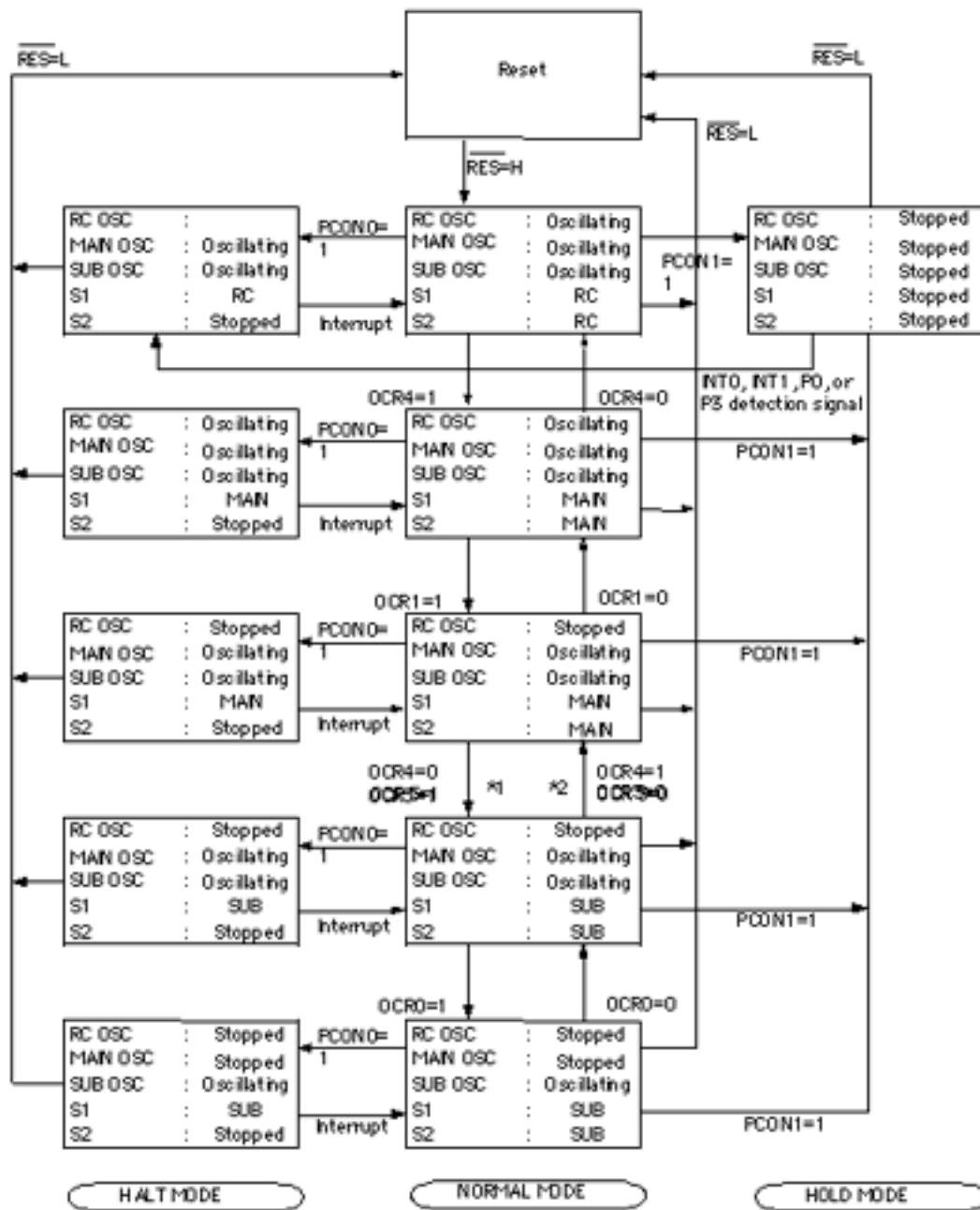


Figure 4.3 Clock Operation Mode Transition Diagram

*1) Before switching the system clock to the sub-clock, allow enough time to ensure that the oscillation of the sub-clock has stabilized. For details on the oscillation stabilization time for the sub-clock (32.768kHz crystal oscillation), refer to the most recent "Semiconductor News."

*2) Before switching the system clock to the main clock, allow enough time to ensure that the oscillation of the main clock has stabilized. For details on the oscillation stabilization time for the main clock, refer to the most recent "Semiconductor News."

3. Standby function

3.1 Overview

POTATO has two standby modes (HALT and HOLD) that are designed to reduce current consumption during a loss of power or while a program is in a standby state.

The microcomputer ceases operations while in the standby state.

3.2 4.3.2. Related Registers

- Power Control Register (PCON)

* For details, refer to chapter 4, section 4.2.4, "Related Registers."

Symbol	Address	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PCON	107H	R/W	-	-	-	-	-	-	PCON1	PCON0
After reset			H	H	H	H	H	H	0	0

Bit name	Function
PCON1 bit 1)	HOLD mode control
	0: 1: et HOLD mode
PCON0 bit 0)	HALT mode control
	0: 1: et HALT mode

3.3 Operating Statuses When in Standby

Table 4.3 *erating Status of Each Block When in Standby*

Item	HALT mode	HOLD mode	
Setting method		PCON0 = 1	PCON1 = 1
Oscillation circuit	Main	Oscillation continues *1	Oscillation stops
	Internal RC		
	Sub-clock	Operation continues	
Internal clock	S1	Operation continues	Operation stops
	S2	Operation stops	Operation stops
CPU		Operation stops	Operation stops
I/O port		Retains data from directly prior to entering HALT mode	Retains data from directly prior to entering HALT mode
RAM		Retains data from directly prior to entering HALT mode	Retains data from directly prior to entering HALT mode
Base timer		Operation continues	Operation stops
Timer 0		Operation continues	Operation stops
Timer 1		Operation continues	Operation stops
Serial communications		Operation continues	Operation stops
Interrupt circuit		Operation continues	Operation stops
LCD display controller		Operation continues	Operation stops
Remote control communications circuit		Operation continues	Operation stops
Watchdog timer		Operation continues, or stops	Operation stops
Release sources		Reset Acceptance of interrupt request	Reset P70/INT0 pin, P71/INT1 pin Port 3 pin

*1) When the sub-clock is the system clock, oscillation can be stopped by a program.

(Internal RC: OCR1 = 1; Main: OCR0 = 1)

3.4 HALT Mode

HALT mode stops program execution while each of the oscillation circuits (main clock, sub-clock and internal RC) continue to operate.

Power consumption can be reduced through intermittent operation of the system by repeatedly setting HALT mode and then having it released in response to an interrupt.

- Setting HALT mode

HALT mode is set by setting bit 0 (PCON0) of the power control register.

- Releasing HALT mode

HALT mode is released in one of two ways: "release by reset" and "release upon acceptance of an interrupt request."

Release by reset

If a low-level signal is applied to the RES pin, HALT mode is released and the microcomputer enters the reset state. When the RES pin is returned to the high level, operation is identical to a start after a normal reset.

Release upon acceptance of an interrupt request

If an interrupt source is generated while the master interrupt enable flag (IE7) and the interrupt request enable flag are both set, an interrupt request is generated and HALT mode is released simultaneously. Subsequently, the microcomputer enters the interrupt processing routine.

However, if the system is in HALT mode and executing an interrupt service program, and the interrupt source that was generated has a priority level that is either the same level as the program that is being executed or lower, then that interrupt is not accepted.

Note:

- If external interrupt INT0 or INT1 is set to "highest level," then that interrupt is not affected by the master interrupt enable flag.
 - The interrupt level of a HALT release source should be higher than the interrupt level in effect when the system entered HALT mode.
-

Table 4.4 HALT Release Source Interrupt Level

Interrupt level when HALT was entered	Interrupt level of HALT release source
Normal level	Low level, high level or highest level
Low level	High level or highest level
High level	Highest level
Highest level	(Cannot be released by an interrupt.)
Normal level: State when no interrupt is in effect	

3.5 HOLD Mode

HOLD mode stops each of the oscillation circuits (main clock, sub-clock and internal RC). HOLD mode can be set in order to maintain data while minimizing current consumption.

- Setting HOLD mode

HOLD mode is set by setting bit 1 (PCON1) of the Power Control register (PCON).

- Releasing HOLD mode

HOLD mode is released in one of three ways: release by reset; release through P70/INT0 level detection or P71/INT1 level detection; or port 3 low level detection.

Release by reset

If a low-level signal is applied to the RES pin, HOLD mode is released and the microcomputer enters the reset state. When the RES pin is returned to the high level, operation is identical to a start after a normal reset.

Release through P70/INT0 level detection or P71/INT1 level detection

If the set level is detected on the P70/INT0 or P71/INT1 pin, HOLD mode is released, and the system enters HALT mode. In this case, if the interrupt request flag for either external interrupt INT0 or external interrupt INT1 is set, the microcomputer enters the corresponding interrupt processing routine; if the interrupt enable flag for the external interrupt has not been set, the system continues in HALT mode. HALT mode is released in the same fashion as described in Chapter 4, section 4.3.3, "Release upon acceptance of an interrupt request." In addition, before setting HOLD mode, it is necessary to set the External Interrupt 0 and 1 Control Register (IO1CR) so that the level (whether high level or low level) is set. That level is detected on the P70/INT0 and P71/INT1 pins.

It is not possible to release HOLD mode with the edge detection setting.

For details on the level detection conditions, refer to Chapter 3, section 3.8, "External Interrupt Function."

Release through port 3 low level detection

- P32INT = 1: Port 3 interrupt and HOLD mode release function

* P32INT = 1 must be set.

Release through port 3 low level detection

When the port 3 interrupt request enable flag is set, and a low-level signal is detected on port 3, the interrupt request flag is set, HOLD mode is released, and the system enters HALT mode. In this case, if the master interrupt enable flag has been set, HALT mode is released and the microcomputer enters the interrupt processing routine. If the master interrupt enable flag is reset (0), the system remains in HALT mode.

Note: When releasing HOLD through port 3, disable any interrupts caused by a source other than port 3.

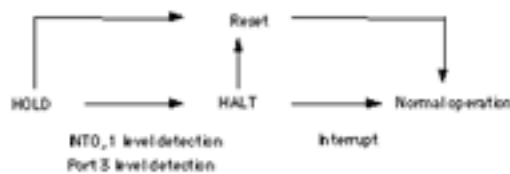


Figure 4.4 Standby Function State Transition Diagram

4. Reset Function

4.1 Overview

The reset function initializes the microcomputer when the power is turned on or while the microcomputer is running.

4.2 Function

The microcomputer is equipped with the following two functions.

- External reset function through RES pin

A reset can definitely be initiated by applying a low-level signal to the RES pin for at least 200[μs]. However, it is important to note that applying a low-level signal for a shorter duration may also initiate a reset.

If a suitable time constant is connected to the RES pin externally, the RES pin can also be used to initiate the power-on reset.

The reset circuit configuration is shown in Fig. above.

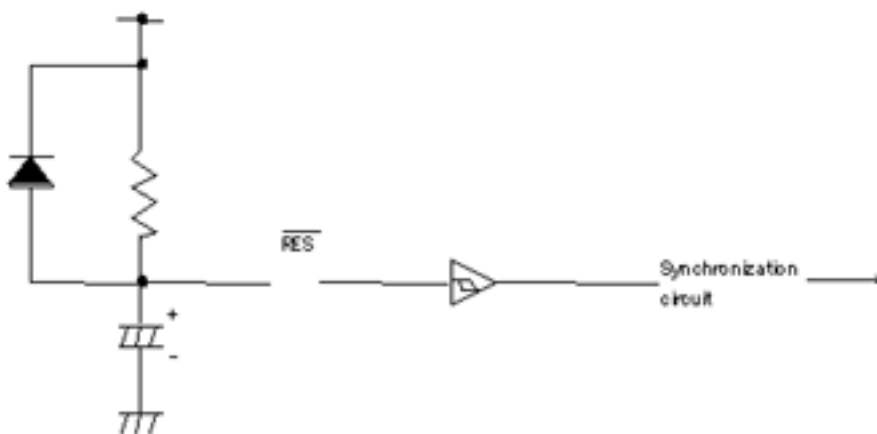


Figure 4.5 Reset Circuit Block Diagram

4.3 Hardware Status During a Reset

If a reset is generated through the RES pin, all of the hardware is initialized according to the reset signal, which is synchronized with the system clock.

Once a reset is initiated, the hardware is initialized immediately, even in the case of a power-on reset, because the system clock switches to internal RC oscillation. After waiting in order to allow the main clock oscillation to stabilize, the system clock switches to the main clock.

During a reset, the Program Counter is initialized to 0000H. For the initial values of the Special Function Registers, refer to Table 4-4-1, "Data Memory/Register Map."

The contents of data RAM, the stack pointer, and LCD RAM are maintained. Caution is required after a power-on reset, however, because these contents are undefined.

Table 4.5 Initial Values of Each Special Function Register

Symbol	Address	R/W	Name	Initial value
RAM(BANK0)	000H-0FFH	R/W	Data memory	XXXXXXXX (retained after reset)
RAM(BANK1)	000H-0FFH	R/W	Data memory	XXXXXXXX (retained after reset)
ACC	100H	R/W	Accumulator	00000000
PSW	101H	R/W	Program Status Word	00H00000
B	102H	R/W	B register	00000000
C	103H	R/W	C register	00000000
TRL	104H	R/W	Table Reference Register lower byte	00000000
TRH	105H	R/W	Table Reference Register upper byte	00000000
SP	106H	R/W	Stack Pointer	XXXXXXXX
PCON	107H	R/W	Power Control Register	HHHHHH00
IE	108H	R/W	Master Interrupt Enable Control Register	0HHHHH00
IP	109H	R/W	Interrupt Priority Ranking Control Register	00000000
EXT	10DH	R/W	External Memory Control Register	HHHH0000
OCR	10EH	R/W	Oscillation Control Register	0H00HH00
TOCNT	110H	R/W	Timer 0 Control Register	00000000
TOPRR	111H	R/W	Timer 0 Prescaler Data	00000000
T0L	112H	R	Timer 0 Lower	00000000
T0LR	113H	R/W	Timer 0 Lower Reload Data	00000000
T0H	114H	R	Timer 0 Upper	00000000
T0HR	115H	R/W	Timer 0 Upper Reload Data	00000000
T1CNT	118H	R/W	Timer 1 Control Register	00000000
T1LC	11AH	R/W	Timer 1 Lower Compare Data	00000000
T1L	11BH	R	Timer 1 Lower	00000000
T1LR		W	Timer 1 Lower Reload Data	00000000
T1HC	11CH	R/W	Timer 1 Upper Compare Data	00000000
T1H	11DH	R	Timer 1 Upper	00000000
T1HR		W	Timer 1 Upper Reload Data	00000000
MCR	120H	W	Mode Control Register	00000000
STAD	122H	R/W	Start Address Register	00000000

4. Control Functions

Symbol	Address	R/W	Name	Initial value
CNR	123H	W	Character Count Register	H0000000
TDR	124H	W	Time Division Register	HH000000
XBNK	125H	R/W	Bank Address Register	HHHHHH00
VCCR	127H	W	LCD Contrast Control Register	00000000
SCON0	130H	R/W	SIO0 Control Register	00H00000
SBUF0	131H	R/W	SIO0 Buffer	00000000
SBR	132H	R/W	SIO Baud Rate Generator	00000000
SCON1	134H	R/W	SIO1 Control Register	H0H00000
SBUF1	135H	R/W	SIO1 Buffer	00000000
Deleted	Deleted	Deleted	Deleted	Deleted
Deleted	Deleted	Deleted	Deleted	Deleted
P1	144H	R/W	Port 1 latch	00000000
P1DDR	145H	W	Port 1 Data Direction Register	00000000
P1FCR	146H	W	Port 1 Function Control Register	00000000
P3	14CH	R/W	Port 3 latch	00000000
P3DDR	14DH	W	Port 3 Data Direction Register	00000000
P3INT	14EH	R/W	Port 3 Interrupt Control Register	HHHHH000
P7	15CH	R	Port 7 latch	HHHHXXXX
I01CR	15DH	R/W	External Interrupt 0, 1 Control	00000000
I23CR	15EH	R/W	External Interrupt 2, 3 Control	00000000
ISL	15FH	R/W	Input Signal Select	HH000000
VSEL	163H	R/W	Control Register	HHH0HH00
VRMAD1	164H	R/W	System Address Register 1	00000000
VRMAD2	165H	R/W	System Address Register 2	HHHHHHH0
VTRBF	166H	R/W	TX/RX Buffer	XXXXXXXX
BTCCR	17FH	R/W	Base Timer Control	00000000
RAM(XRAM) (BANK0)	180H-1FBH	R/W	LCD display memory	XXXXXXXX (retained after reset)
RAM(XRAM) (BANK1)	180H-1FBH	R/W		
RAM(XRAM) (BANK2)	180H-185H	R/W		

5. Instructions

1. Instruction Overview

The POTATO instruction set includes 70 instructions.

These encompass 45 opcodes, which are grouped into the following eight types of functions:

• Arithmetic operation instructions	ADD,ADDC,SUB,SUBC,INC,DEC,MUL,DIV
• Boolean operation instructions	AND,OR,XOR,ROL,ROLC,ROR,RORC
• Data transfer instructions	LD,ST,MOV,LDC,PUSH,POP,XCH
• Jump instructions	JMP,JMPF,BR,BRF
• Conditional branching instructions	BZ,BNZ,BP,BPC,BN,DBNZ,BE,BNE
• Subroutine instructions	CALL,CALLF,CALLR,RET,RETI
• Bit manipulation instructions	CLR1,SET1,NOT1
• Miscellaneous instruction	NOP
• Macro instruction	CHANGE

1.1 Arithmetic Operation Instructions

The arithmetic operation instructions primarily use the accumulator, and include the four basic arithmetic operations as well as increment and decrement. The results of one of the four basic arithmetic operations are set in CY, AC, and OV.

- CY (Carry Flag)

Operation instruction	Operation result	CY
When an addition instruction was executed	When a carry is generated from bit 7 (MSB)	1
	When no carry is generated from bit 7 (MSB)	0
When a subtraction or compare instruction was executed	When a borrow is required for bit 7 (MSB)	1
	When a borrow is not required for bit 7 (MSB)	0
When a multiplication instruction was executed	•	0

- AC (auxiliary carry flag)

Operation instruction	Operation result	AC
When an addition instruction was executed	When a carry is generated from bit 3	1
	When no carry is generated from bit 3	0
When a subtraction instruction was executed	When a borrow is required for bit 3	1
	When a borrow is not required for bit 3	0

- OV (Overflow flag)

Operation instruction	Operation result	OV
When an addition or multiplication instruction was executed	When a carry is generated from bit 7, and no carry is generated from bit 6	1
	When a carry is generated from bit 6, and no carry is generated from bit 7	1
	When an overflow error was generated while executing an addition or subtraction instruction involving signed variables	1
	All other cases	0
When a multiplication instruction was executed	When the product is 256 or higher	1
	When the product is 255 or lower	0
When a division instruction was executed	When an attempt was made to divide by zero	1
	When dividing by any other number	0

1.2 Logical Operation Instructions

The Boolean operation instructions are used to perform Boolean operations and to rotate bits. CY is also affected after executing the RORC or ROLC instruction.

1.3 Data Transfer Instructions

The data transfer instructions are used to write, read, save and replace data in data memory (RAM), the Special Function Registers (SFR), external data ROM, and external RAM.

1.4 Jump Instructions

Jump instructions unconditionally transfer control to the target instruction.

1.5 Conditional Branching Instructions

A conditional branching instruction determines whether a condition that is specified by the instruction is met (true) or not (false), and then, if the evaluation is "true," branches to the target address. If the evaluation is "false," the instruction does not branch; instead, execution continues with the next instruction.

The BE and BNE instructions branch on the basis of a comparison of two 8-bit data bytes. CY is set or reset by these instructions, according to the results of the comparison.

Operand				Carry flag (CY)
	#i8,r8	d9,r8	@Rj,#i8,r8	
Relationship	#i8>(ACC)	(d9)>(ACC)	#i8>((Rj))	1
	#i8=(ACC)	(d9)=(ACC)	#i8=((Rj))	0
	#i8<(ACC)	(d9)<(ACC)	#i8<((Rj))	0

1.6 Subroutine Instructions

Subroutine instructions branch unconditionally and are used to transfer control to the target instruction. The address of the instruction is stored in the stack so that, after branching a return instruction (RET, RETI) can be used to return control to the instruction that follows the CALL instruction. The stack is located in data memory (RAM), and is pointed at by the Stack Pointer (SP). it is necessary to allocate an area in RAM for use by the stack according to the nesting level of the subroutine.

1.7 Bit Manipulation Instructions

The bit manipulation instructions are used to manipulate individual bits in specified contents of data memory (RAM) or Special Function Registers (SFR).

1.8 Miscellaneous Instruction

The NOP instruction consumes one machine cycle without doing anything.

1.9 Macro Instruction

This is POTATO's own standard macro instruction. This macro instruction switches between internal program and external program execution.

1.10 Addressing

There are several addressing methods that are used for addressing program memory (ROM), data memory (RAM), and the Special Function Registers (SFR).

1.11 Program Memory (ROM) Addressing

Jump instructions, branching instructions, and subroutine instructions specify the destination address in program ROM as part of the instruction code. In this case, the address is specified by one of the following addressing methods:

- r8 (8-bit relative addressing)

This form of addressing permits jumps (branching) to an address within -128 to +127 addresses of the starting address of the next instruction that follows the instruction that is currently being executed. The jump is expressed through signed 8-bit data.

[80H to 7FH: -128 to +127]

- r16 (16-bit relative addressing)

This form of addressing permits jumps anywhere within the 64K program ROM space.

The address is expressed through unsigned 16-bit data.

[0000H to FFFFH: +0 to +65535]

- a12 (12-bit relative addressing)

This form of addressing leaves as is the bits PC15 through PC12 (which represent the current page) of the starting address (represented by PC15 through PC00) of the next instruction that follows the instruction that is currently being executed, and replaces the bits PC11 through PC00 with 12-bit addressing data [000H to FFFH].

This form of addressing permits jumps within a page (PC15 to PC12).

Note:

- Note that, in the above instance, the "current page" will be different from the page where the JMP instruction or CALL instruction is located if the JMP instruction or CALL instruction is located at the end of a page.
-

- a16 (16-bit absolute addressing)

This form of addressing permits jumps anywhere within the 64K program ROM space.

The address is expressed through 16-bit data.

[0000H to FFFFH: 0 to 65535]

- Table jumps

It is possible to execute a jump by setting the destination address in the stack, and then forcibly loading that value into the Program Counter (PC) by means of the RET instruction.

Refer to example 1. In line 1, the Stack Pointer (SP) is set to 09H.

If the RET instruction is executed at this point, a jump will be executed to the address where address 08H in RAM is the upper byte and address 07H in RAM is the lower byte. Therefore, the jump destination address is set in lines 2 and 3.

Because the jump destination in this example is PC = 0C13H, line 2 sets the lower byte, 13H, and line 3 sets the upper byte, 0CH. When the RET instruction in line 4 is executed, the SP is set to 07H and control jumps to 0C13H. However, because Example 1 requires the SP value to be known ahead of time, normally a PUSH instruction is used as shown in Example 2.

Example 1:

```
MOV    #09H,SP
MOV    #13H,07H
MOV    #0CH,08H
RET
```

Example 2:

```
MOV    #13H,ACC
PUSH   ACC
MOV    #0CH,ACC
PUSH   ACC
RET
```

In Example 3, the program branches to one of 128 addresses, ranging from 00H to 7FH, depending on the data in address 70H in RAM

Lines 1 and 2 set the lower byte of the address, while line 4 sets the upper byte of the address. When the RET instruction in the line 6 is executed, the program branches to the jump table in lines 7 and 8, and then jumps to the branching destination indicated in those lines.

This technique is called a "table jump," and is an effective tool for branching to multiple addresses conditionally.

Example 3:

```
A0:    LD          070H
        ROL
        ADD        #LOW(A1)
        PUSH      ACC
        MOV        #HIGH(A1),ACC
        PUSH      ACC
        RET

;
        ORG        0C00H
A1:    JMP         B00    Jump table
        Ø
        JMP         B7F

;
B00:    XXXXXX
```

1.12 Data Memory (RAM) and Special Function Register (SFR) Addressing

- d9 (direct addressing)

This form of addressing directly specifies an address in RAM or an SFR with 9 bits (d8 through d0).

Addresses 000H through 0FFH.....Specifies RAM.

Addresses 100H through 1FFH.....Specifies SFR.

- b3 (bit addressing)

This form of addressing uses 3-bit bit addressing data in combination with d9 (direct addressing) to specify a specific bit in RAM or SFR.

	MSB						LSB
	2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹
Bit addressing data -----	7	6	5	4	3	2	1
	(11)	(10)	(01)	(00)	(11)	(10)	(01)

- @Rj(indirect addressing)

Indirect addressing specifies an address in RAM or SFR by setting in a specific address in RAM (called the "indirect address register") the address in RAM or SFR that is actually to be accessed, and then accessing the indirect address register. The indirect address registers are labelled @R0, @R1, @R2, and @R3. A specific indirect address register (one of the four from @R0 to @R3) is specified through the 2-bit indirect addressing data (j1 and j0). As shown in the table below, four indirect address banks are allocated in the first 16 bytes (00H through 0FH) of each RAM bank. The RAM bank is set by RAMBK0 (bit 1 of the PSW). The indirect address register bank is set by IRBK1, 0 (bits 4 and 3 of the PSW). Note that when executing an indirect addressing instruction, RAM in the RAM bank that is set by IRBK1 and 0 and by RAMBK0 is used for the indirect address register and the RAM that is specified by the indirect address register. During a reset, both IRBK0 and IRBK1 are set to "0" and RAMBK0 is set to "0" as well, so the respective absolute addresses of @R0, @R1, @R2, and @R3 are 00H, 01H, 02H, and 03H in RAM bank 0.

Indirect address registers @R3, @R2, @R1, and @R0

Indirect addressing data (j1, j0) (11) (10) (01) (00)

Table 5.6 Indirect Addressing Register Map

Indirect address register name	Function	Bank 0 (IRBK1=0) (IRBK0=0)	Bank 1 (IRBK1=0) (IRBK0)	Bank 2 (IRBK1=1) (IRBK0=0)	Bank 3 (IRBK1=1) (IRBK0=1)
@R0	RAM access	RAM 00H	RAM 04H	RAM 08H	RAM 0CH
@R1	RAM access	RAM 01H	RAM 05H	RAM 09H	RAM 0DH
@R2	SFR access	RAM 02H	RAM 06H	RAM 0AH	RAM 0EH
@R3	SFR access	RAM 03H	RAM 07H	RAM 0BH	RAM 0FH

- Example of using indirect addressing

This example illustrates an operation using the indirect address register.

Refer to Example 1. Line 2 sets the immediate data 68H in RAM at address 00H. If RAM (address 00H) is used as an indirect address register, address 68H in RAM will be accessed. For example, line 3 sets the immediate data 10H in the address that is specified by the indirect address register (@R0), which is address 68H in RAM.

Line 5 adds the contents of the address that is specified by the indirect address register (@R0), which is address 68H in RAM, and the Accumulator (ACC).

Example 1:

```
MOV    #055H,ACC
MOV    #068H,00H
MOV    #010H,@R0
ADD    #015H
ADD    @R0
```



In the following example, an SFR is specified through indirect addressing.

Refer to Example 2. Lines 1 and 2 clear bits 4 and 3 of the PSW and set up RAM addresses 00H through 03H as the indirect address registers. Line 4 sets the immediate data 02H in address 02H in RAM. If address 02H in RAM is used as the indirect address register, address 02H in RAM is accessed. For example, in line 5 the immediate data 12H is set in the SFR (address 02H: B register) that is specified by the indirect address register @R2.

Line 6 increments the B register, which is accessed through indirect addressing again.

Example 2:

```
CLR1      PSW,4
CLR1      PSW,3
MOV       #0ACH,ACC
MOV       #002H,02H
MOV       #012H,@R2
INC       @R2
```

The following example changes the bank through the PSW and then specifies an SFR through indirect addressing.

Refer to Example 3. Lines 1 and 2 set the bank to "2" in the PSW and set up RAM addresses 08H through 0BH as the indirect address registers. Line 4 sets the immediate data 02H in address 0BH in RAM. If address 0BH in RAM is used as the indirect address register, address 02H in RAM is accessed. For example, in line 5 the immediate data 12H is set in the SFR (address 02H: B register) that is specified by the indirect address register @R2.

Line 6 increments the B register, which is accessed through indirect addressing again.

Example 3:

```
SET1      PSW,4
CLR1      PSW,3
MOV       #0ACH,ACC
MOV       #002H,0BH
MOV       #012H,@R2
INC       @R2
```

2. Arithmetic Operation Instructions

ADD #i8 (ADD immediate data to accumulator)

Instruction code	1 0 0 0 0 0 1 i7i6i5i4i3i2i1i081H
Number of bytes	2
Number of cycles	1
Function	(ACC) ← (ACC) + #i8
Affected flags	CY • CAC • COV
Interrupt acceptance	Permitted

Description:

This instruction adds the immediate data (i7 to i0) to the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example:

		ACC	CY	AC	OV
MOV	#055H,ACC	55H	-	-	-
ADD	#013H	68H	0	0	0
ADD	#00AH	72H	0	1	0
ADD	#00FH	81H	0	1	1
ADD	#080H	01H	1	0	1

ADD d9 (ADD direct byte to accumulator)

Instruction code	1 0 0 0 0 1 d8 d7d6d5d4d3d2d1d082H~83H
Number of bytes	2
Number of cycles	1
Function	(ACC) • ©(ACC) + (d9)
Affected flags	CY • CAC • COV
Interrupt acceptance	Permitted

Description:

This instruction adds the contents of data memory (RAM) or a Special Function Register (SFR), as specified by d8 to d0, to the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1:

		ACC	RAM 23H	CY	AC	OV
MOV	#055H,ACC	55H	-	-	-	-
MOV	#068H,023H	55H	68H	-	-	-
ADD	#00CH	61H	68H	0	1	0
ADD	023H	C9H	68H	0	0	1

Example 2:

		ACC	B	CY	AC	OV
MOV	#070H,ACC	70H	-	-	-	-
MOV	#095H,B	70H	95H	-	-	-
ADD	#002H	72H	95H	0	0	0
ADD	B	07H	95H	1	0	0

ADD @Rj (ADD indirect byte to accumulator)

Instruction code	1 0 0 0 1j1j084H~87H
Number of bytes	1
Number of cycles	1
Function	(ACC) ← (ACC) + ((Rj)) j=0,1,2,3
Affected flags	CY • CAC • COV
Interrupt acceptance	Permitted

Description:

This instruction adds the contents of data memory (RAM) or a Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0, to the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1:

		ACC	RAM 00H	RAM 68H	CY	AC	OV
MOV	#055H,ACC	55H	-	-	-	-	-
MOV	#068H,000H	55H	68H	-	-	-	-
MOV	#010H,@R0	55H	68H	10H	-	-	-
ADD	#015H	6AH	68H	10H	0	0	0
ADD	@R0	7AH	68H	10H	0	0	0

Example 2:

		ACC	RAM 02H	TRL	CY	AC	OV
MOV	#0AAH,ACC	AAH	-	-	-	-	-
MOV	#004H,002H	AAH	04H	-	-	-	-
MOV	#055H,@R2	AAH	04H	55H	-	-	-
ADD	#001H	ABH	04H	55H	0	0	0
ADD	@R2	00H	04H	55H	1	1	0

ADDC #i8 (ADD immediate data and carry flag to accumulator)

Instruction code	1 0 0 1 0 0 0 1 i7i6i5i4i3i2i1i091H
Number of bytes	2
Number of cycles	1
Function	$(ACC) \leftarrow (ACC) + (CY) + \#i8$
Affected flags	$CY \bullet CAC \bullet COV$
Interrupt acceptance	Permitted

Description:

This instruction adds the carry flag (CY) and the immediate data (i7 to i0) to the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example:

		ACC	CY	AC	OV
MOV	#055H,ACC	55H	-	-	-
ADD	#013H	68H	0	0	0
ADDC	#00AH	72H	0	1	0
ADDC	#00FH	81H	0	1	1
ADDC	#080H	01H	1	0	1
ADDC	#001H	03H	0	0	0

ADDC d9 (ADD direct byte and carry flag to accumulator)

Instruction code	1 0 0 1 0 0 1d8 d7d6d5d4d3d2d1d092H~93H
Number of bytes	2
Number of cycles	1
Function	$(ACC) \bullet \odot (ACC) + (CY) + (d9)$
Affected flags	$CY \bullet CAC \bullet COV$
Interrupt acceptance	Permitted

Description:

This instruction adds the carry flag (CY) and the contents of data memory (RAM) or a Special Function Register (SFR), as specified by d8 to d0, to the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1:

		ACC	RAM 23H	CY	AC	OV
MOV	#055H,ACC	55H	-	-	-	-
MOV	#068H,023H	55H	68H	-	-	-
ADD	#00CH	61H	68H	0	1	0
ADDC	023H	C9H	68H	0	0	1
SET1	PSW,7	C9H	68H	1	0	1
ADDC	023H	32H	68H	1	1	0

Example 2:

		ACC	B	CY	AC	OV
MOV	#070H,ACC	70H	-	-	-	-
MOV	#095H,B	70H	95H	-	-	-
ADD	#002H	72H	95H	0	0	0
ADDC	B	07H	95H	1	0	0
ADDC	B	9DH	95H	0	0	0

ADDC @Rj (ADD indirect byte and carry flag to accumulator)

Instruction code	1 0 0 1 0 1j1j094H~97H
Number of bytes	1
Number of cycles	1
Function	$(ACC) \leftarrow (ACC) + (CY) + ((Rj))$ j=0,1,2,3
Affected flags	CY • CAC • COV
Interrupt acceptance	Permitted

Description:

This instruction adds the carry flag (CY) and the contents of data memory (RAM) or a Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0, to the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1:

		ACC	RAM 00H	RAM 68H	CY	AC	OV
MOV	#055H,ACC	55H	-	-	-	-	-
MOV	#068H,000H	55H	68H	-	-	-	-
MOV	#010H,@R0	55H	68H	10H	-	-	-
ADD	#015H	6AH	68H	10H	0	0	0
ADDC	@R0	7AH	68H	10H	0	0	0
SET1	PSW,7	7AH	68H	10H	1	0	0
ADDC	@R0	8BH	68H	10H	0	0	1

Example 2:

		ACC	RAM 02H	TRL	CY	AC	OV
MOV	#0AAH,ACC	AAH	-	-	-	-	-
MOV	#004H,002H	AAH	04H	-	-	-	-
MOV	#055H,@R2	AAH	04H	55H	-	-	-
ADD	#001H	ABH	04H	55H	0	0	0
ADDC	@R2	00H	04H	55H	1	1	0
ADDC	@R2	56H	04H	55H	0	0	0

SUB #i8 (Subtract immediate data from accumulator)

Instruction code	1 0 1 0 0 0 1 i7i6i5i4i3i2i1i0A1H
Number of bytes	2
Number of cycles	1
Function	(ACC) ← (ACC) - #i8
Affected flags	CY • CAC • COV
Interrupt acceptance	Permitted

Description:

This instruction subtracts the immediate data (i7 to i0) from the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example:

		ACC	CY	AC	OV
MOV	#055H,ACC	55H	-	-	-
SUB	#013H	42H	0	0	0
SUB	#003H	3FH	0	1	0
SUB	#03FH	00H	0	0	0
SUB	#002H	FEH	1	1	0

SUB d9 (Subtract direct byte from accumulator)

Instruction code	1 0 1 0 0 0 1d8 d7d6d5d4d3d2d1d0A2H~A3H
Number of bytes	2
Number of cycles	1
Function	(ACC) ← (ACC) - (d9)
Affected flags	CY • CAC • COV
Interrupt acceptance	Permitted

Description:

This instruction subtracts the contents of data memory (RAM) or a Special Function Register (SFR), as specified by d8 to d0, from the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1

		ACC	RAM 23H	CY	AC	OV
MOV	#055H,ACC	55H	-	-	-	-
MOV	#068H,023H	55H	68H	-	-	-
SUB	#00CH	49H	68H	0	1	0
SUB	023H	E1H	68H	1	0	0

Example 2:

		ACC	RAM	CY	AC	OV
MOV	#080H,ACC	80H	-	-	-	-
MOV	#095H,B	80H	95H	-	-	-
SUB	#002H	7EH	95H	0	1	1
SUB	B	E9H	95H	1	0	1

SUB @Rj (Subtract indirect byte from accumulator)

Instruction code	1 0 1 0 0 1 j1 j0 A4H~A7H
Number of bytes	1
Number of cycles	1
Function	(ACC) ← (ACC) - ((Rj)) j=0,1,2,3
Affected flags	CY • CAC • COV
Interrupt acceptance	Permitted

Description:

This instruction subtracts the contents of data memory (RAM) or a Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0, from the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1:

		ACC	RAM 00H	RAM 68H	CY	AC	OV
MOV	#055H,ACC	55H	-	-	-	-	-
MOV	#068H,00H	55H	68H	-	-	-	-
MOV	#010H,@R0	55H	68H	10H	-	-	-
SUB	#016H	3FH	68H	10H	0	1	0
SUB	@R0	2FH	68H	10H	0	0	0

Example 2:

		ACC	RAM 02H	TRL	CY	AC	OV
MOV	#0AAH,ACC	AAH	-	-	-	-	-
MOV	#004H,002H	AAH	04H	-	-	-	-
MOV	#0AAH,@R2	AAH	04H	AAH	-	-	-
SUB	#001H	A9H	04H	AAH	0	0	0
SUB	@R2	FFH	04H	AAH	1	1	0

SUBC #i8 (Subtract immediate data and carry flag from accumulator)

Instruction code	1 0 1 1 0 0 0 1 i7i6i5i4i3i2i1i0B1H
Number of bytes	2
Number of cycles	1
Function	(ACC) ← (ACC) - (CY) - #i8
Affected flags	CY • CAC • COV
Interrupt acceptance	Permitted

Description:

This instruction subtracts the carry flag (CY) and the immediate data (i7 to i0) from the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example:

		ACC	CY	AC	OV
MOV	#055H,ACC	55H	-	-	-
SUB	#013H	42H	0	0	0
SUBC	#003H	3FH	0	1	0
SUBC	#03FH	00H	0	0	0
SUBC	#002H	FEH	1	1	0
SUBC	#03EH	BFH	0	1	0

SUBC d9 (Subtract direct byte and carry flag from accumulator)

Instruction code	1 0 1 1 0 0 1d8 d7d6d5d4d3d2d1d0B2H~B3H
Number of bytes	2
Number of cycles	1
Function	(ACC)(ACC) - (CY) - (d9)
Affected flags	CY•CAC•COV
Interrupt acceptance	Permitted

Description:

This instruction subtracts the carry flag (CY) and the contents of data memory (RAM) or a Special Function Register (SFR), as specified by d8 to d0, from the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1:

		ACC	RAM 23H	CY	AC	OV
MOV	#055H,ACC	55H	—	-	-	-
MOV	#068H,023H	55H	68H	-	-	-
SUB	#00CH	49H	68H	0	1	0
SUBC	023H	E1H	68H	1	0	0
SUBC	023H	78H	68H	0	1	1

Example 2:

		ACC	B	CY	AC	OV
MOV	#080H,ACC	80H	-	-	-	-
MOV	#095H,B	80H	95H	-	-	-
SUB	#002H	7EH	95H	0	1	1
SUBC	B	E9H	95H	1	0	1
SUBC	B	53H	95H	0	0	1

SUBC @Rj (Subtract indirect byte and carry flag from accumulator)

Instruction code	1 0 1 1 0 1j1j0B4H~B7H
Number of bytes	1
Number of cycles	1
Function	$(ACC) \leftarrow (ACC) - (CY) - ((Rj)) \quad j=0,1,2,3$
Affected flags	CY•CAC•COV
Interrupt acceptance	Permitted

Description:

This instruction subtracts the carry flag (CY) and the contents of data memory (RAM) or a Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0, from the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1:

		ACC	RAM 00H	RAM 68H	CY	AC	OV
MOV	#055H,ACC	55H	-	-	-	-	-
MOV	#068H,00H	55H	68H	-	-	-	-
MOV	#040H,@R0	55H	68H	40H	-	-	-
SUB	#016H	3FH	68H	40H	0	1	0
SUBC	@R0	FFH	68H	40H	1	0	0
SUBC	@R0	BEH	68H	40H	0	0	0

Example 2:

		ACC	RAM 02H	TRL	CY	AC	OV
MOV	#0AAH,ACC	AAH	-	-	-	-	-
MOV	#004H,002H	AAH	04H	-	-	-	-
MOV	#0AAH,@R2	AAH	04H	AAH	-	-	-
SUB	#001H	A9H	04H	AAH	0	0	0
SUBC	@R2	FFH	04H	AAH	1	1	0
SUBC	@R2	54H	04H	AAH	0	0	0

INC d9 (Increment direct byte)

Instruction code	0 1 1 0 0 0 1d8 d7d6d5d4d3d2d1d062H~63H
Number of bytes	2
Number of cycles	1
Function	(d9) ← (d9)+1
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction increments the contents of data memory (RAM) or a Special Function Register (SFR), as specified by d8 to d0.

Example 1:

```
MOV    #0FDH,ACC
INC    ACC
INC    ACC
INC    ACC
INC    ACC
```

ACC
FDH
FEH
FFH
00H
01H

Example 2:

```
MOV    #0FDH,07FH
INC    07FH
INC    07FH
INC    07FH
INC    07FH
```

RAM 7FH
FDH
FEH
FFH
00H
01H

Note:

- CY, AC, and OV do not change.
 - When this instruction is applied to one of the ports P0 through P5, the port latch of that port is selected; the external signal that is applied to that port is not selected.
- Furthermore, applying this instruction to port P7 does not change its status.ÅB

INC @Rj (Increment indirect byte)

Instruction code	0 1 1 0 0 1j1j064H~67H
Number of bytes	1
Number of cycles	1
Function	$((Rj)) \leftarrow ((Rj)) + 1 \quad j=0,1,2,3$
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction increments the contents of data memory (RAM) or a Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0.

Example 1:

		ACC	RAM 03H
MOV	#000H,003H	-	00H
MOV	#0FDH,@R3	FDH	00H
INC	@R3	FEH	00H
INC	@R3	FFH	00H
INC	@R3	00H	00H

Example 2:

		RAM 7FH	RAM 01H
MOV	#07FH,001H	-	7FH
MOV	#0FDH,@R1	FDH	7FH
INC	@R1	FEH	7FH
INC	@R1	FFH	7FH
INC	@R1	00H	7FH

Note:

- CY, AC, and OV do not change.
 - When this instruction is applied to one of the ports P0 through P5, the port latch of that port is selected; the external signal that is applied to that port is not selected.
- Furthermore, applying this instruction to port P7 does not change its status.
-

DEC d9 (Decrement direct byte)

Instruction code	0 1 1 1 0 0 1 d8 d7d6d5d4d3d2d1d072H~73H
Number of bytes	2
Number of cycles	1
Function	(d9) ← (d9) - 1
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction decrements the contents of data memory (RAM) or a Special Function Register (SFR), as specified by d8 through d0.

Example 1:

```
MOV    #002H,ACC
DEC    ACC
DEC    ACC
DEC    ACC
DEC    ACC
```

ACC
02H
01H
00H
FFH
FEH

Example 2:

```
MOV    #002H,07FH
DEC    07FH
DEC    07FH
DEC    07FH
DEC    07FH
```

RAM 7FH
02H
01H
00H
FFH
FEH

Note:

- CY, AC, and OV do not change.
 - When this instruction is applied to one of the ports P0 through P5, the port latch of that port is selected; the external signal that is applied to that port is not selected.
- Furthermore, applying this instruction to port P7 does not change its status.

DEC @Rj (Decrement indirect byte)

Instruction code	0 1 1 1 0 1j1j074H~77H
Number of bytes	1
Number of cycles	1
Function	$((Rj)) \leftarrow ((Rj)) - 1 \quad j=0,1,2,3$
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction decrements the contents of data memory (RAM) or a Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0.

Example 1:

		ACC	RAM 02H
MOV	#000H,002H	-	00H
MOV	#002H,@R2	02H	00H
DEC	@R2	01H	00H
DEC	@R2	00H	00H
DEC	@R2	FFH	00H

Example 2:

		RAM 7FH	RAM 00H
MOV	#07FH,000H	-	7FH
MOV	#002H,@R0	02H	7FH
DEC	@R0	01H	7FH
DEC	@R0	00H	7FH
DEC	@R0	FFH	7FH

Note:

- CY, AC, and OV do not change.
 - When this instruction is applied to one of the ports P0 through P5, the port latch of that port is selected; the external signal that is applied to that port is not selected.
- Furthermore, applying this instruction to port P7 does not change its status.
-

MUL (Multiply accumulator and c register times b register)

Instruction code	0 0 1 1 0 0 0 030H
Number of bytes	1
Number of cycles	7
Function	(B)(ACC)(C)“(ACC)(C) x (B)
Affected flags	CY•COV
Interrupt acceptance	Permitted after 7th cycle

Description:

This instruction multiplies the unsigned 16-bit data that is represented by the Accumulator (ACC) and the C register (C) by the unsigned 8-bit data that is represented by the B register (B). Of the 24-bit result of the operation, the lower 8 bits are sent to C, the middle 8 bits are sent to the ACC, and the upper 8 bits are sent to B.

After the operation, if B is "0" then the overflow flag (OV) is reset, and if B is not "0" then OV is set; the carry flag (CY) is always reset.

Example 1:

		ACC	C	B	CY	AC	OV
MOV	#0C4H,PSW	-	-	-	1	1	1
MOV	#011H,ACC	11H	-	-	1	1	1
MOV	#023H,C	11H	23H	-	1	1	1
MOV	#052H,B	11H	23H	52H	1	1	1
MUL		7DH	36H	05H	0	1	1

Example 2:

		ACC	C	B	CY	AC	OV
MOV	#0C4H,PSW	-	-	-	1	1	1
MOV	#007H,ACC	07H	-	-	1	1	1
MOV	#005H,C	07H	05H	-	1	1	1
MOV	#010H,B	07H	05H	10H	1	1	1
MUL		70H	50H	00H	0	1	0

DIV (Divide accumulator and c register by b register)

Instruction code 0100000040H	
Number of bytes	1
Number of cycles	7
Function	(ACC)(C),mod(B)“(ACC)(C) ÷ (B)
Affected flags	CY•COV
Interrupt acceptance	Permitted after 7th cycle

Description:

This instruction divides the 16-bit data that is represented by the Accumulator (ACC) (the upper byte) and the C register (C) (the lower byte) by the contents of the B register (B) (unsigned 8-bit data). The upper byte of the resulting quotient is sent to the ACC while the lower byte is sent to C; the remainder is sent to B. After the operation, if B is "0" then the overflow flag (OV) is set, and if B is not "0" then OV is reset; the carry flag (CY) is always reset.

Example 1:

		ACC	C	B	CY	AC	OV
MOV	#0C4H,PSW	-	-	-	1	1	1
MOV	#078H,ACC	79H	-	-	1	1	1
MOV	#005H,C	79H	05H	-	1	1	1
MOV	#007H,B	79H	05H	07H	1	1	1
DIV		11H	49H	06H	0	1	0

Example 2:

		ACC	C	B	CY	AC	OV	
MOV	#0C0H,PSW	-	-	-	1	1	0	
MOV	#007H,ACC	07H	-	-	1	1	0	
MOV	#010H,C	07H	10H	-	1	1	0	
MOV	#000H,B	07H	10H	00H	1	1	0	
DIV		FFH	10H	00H	0	1	1	Error

3. Logical Operation Instructions

AND #i8 (AND immediate data to accumulator)

Instruction code	1 1 1 0 0 0 1 i7i6i5i4i3i2i1i0E1H
Number of bytes	2
Number of cycles	1
Function	(ACC) ← (ACC) ∧ #i8
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction ANDs the immediate data (i7 to i0) with the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1:

```
MOV      #0FFH,ACC
AND      #0FAH
AND      #0AFH
AND      #00FH
AND      #0F0H
```

ACC
FFH
FAH
AAH
0AH
00H

Example 2:

```
MOV      #0FFH,ACC
AND      #0FEH
AND      #0FDH
AND      #0FBH
AND      #0F7H
AND      #0EFH
AND      #0DFH
AND      #0BFH
AND      #07FH
```

ACC
FFH
FEH
FCH
F8H
F0H
E0H
C9H
80H
00H

AND d9 (AND direct byte to accumulator)

Instruction code	1 1 1 0 0 0 1d8 d7d6d5d4d3d2d1d0E2H~E3H
Number of bytes	2
Number of cycles	1
Function	(ACC)←(ACC)^(d9)
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction ANDs the contents of data memory (RAM) or a Special Function Register (SFR), as specified by d8 to d0, with the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1:

		ACC	RAM 23H
MOV	#0FFH,ACC	FFH	-
MOV	#055H,023H	FFH	55H
AND	023H	55H	55H
MOV	#0AAH,023H	55H	AAH
AND	023H	00H	AAH

Example 2:

		ACC	B
MOV	#0FFH,ACC	FFH	-
MOV	#0FEH,B	FFH	FEH
AND	B	FEH	FEH
MOV	#0FDH,B	FEH	FDH
AND	B	FCH	FDH
MOV	#0FBH,B	FCH	FBH
AND	B	F8H	FBH
MOV	#0F7H,B	F8H	F7H
AND	B	F0H	F7H

AND @Rj (AND indirect byte to accumulator)

Instruction code	1 1 1 0 0 1 j1 j0 E4H~E7H
Number of bytes	1
Number of cycles	1
Function	$(ACC) \leftarrow (ACC) \wedge ((Rj))$ j=0,1,2,3
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction ANDs the contents of data memory (RAM) or a Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0, with the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1:

		ACC	RAM 00H	RAM 68H
MOV	#0FFH,ACC	FFH	-	-
MOV	#068H,000H	FFH	68H	-
MOV	#0F0H,@R0	FFH	68H	F0H
AND	@R0	F0H	68H	F0H
MOV	#00FH,@R0	F0H	68H	0FH
AND	@R0	00H	68H	0FH

Example 2:

		ACC	RAM 02H	TRL
MOV	#0FFH,ACC	FFH	-	-
MOV	#004H,002H	FFH	04H	-
MOV	#0EFH,@R2	FFH	04H	EFH
AND	@R2	EFH	04H	EFH
MOV	#0DFH,@R2	EFH	04H	DFH
AND	@R2	CFH	04H	DFH

OR #i8 (OR immediate data to accumulator)

Instruction code	1 1 0 1 0 0 0 1 i7i6i5i4i3i2i1i0D1H
Number of bytes	2
Number of cycles	1
Function	(ACC) ← (ACC) ∨ #i8
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction ORs the immediate data (i7 to i0) with the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1:

		ACC
MOV	#000H,ACC	00H
OR	#003H	03H
OR	#00CH	0FH
OR	#030H	3FH
OR	#0C0H	FFH

Example 2:

		ACC
MOV	#000H,ACC	00H
OR	#001H	01H
OR	#002H	03H
OR	#004H	07H
OR	#008H	0FH
OR	#010H	1FH
OR	#020H	3FH
OR	#040H	7FH
OR	#080H	FFH

OR d9 (OR direct byte to accumulator)

Instruction code	1 1 0 1 0 0 1 d8 d7d6d5d4d3d2d1d0D2H~D3H
Number of bytes	2
Number of cycles	1
Function	(ACC) ← (ACC) ∨ (d9)
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction ORs the contents of data memory (RAM) or a Special Function Register (SFR), as specified by d8 to d0, with the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1:

		ACC	RAM 23H
MOV	#000H,ACC	00H	-
MOV	#055H,023H	00H	55H
OR	023H	55H	55H
MOV	#0AAH,023H	55H	AAH
OR	023H	FFH	AAH

Example 2:

		ACC	B
MOV	#000H,ACC	00H	-
MOV	#001H,B	00H	01H
OR	B	01H	01H
MOV	#002H,B	01H	02H
OR	B	03H	02H
MOV	#004H,B	03H	04H
OR	B	07H	04H
MOV	#008H,B	07H	08H
OR	B	0FH	08H

OR @Rj (OR indirect byte to accumulator)

Instruction code	1 1 0 1 0 1j1j0D4H~D7H
Number of bytes	1
Number of cycles	1
Function	$(ACC) \leftarrow (ACC) \vee ((Rj))$ j=0,1,2,3
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction ORs the contents of data memory (RAM) or a Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0, with the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1:

		ACC	RAM 00H	RAM 68H
MOV	#000H,ACC	00H	-	-
MOV	#068H,000H	00H	68H	-
MOV	#0F0H,@R0	00H	68H	F0H
OR	@R0	F0H	68H	F0H
MOV	#000FH,@R0	F0H	68H	0FH
OR	@R0	FFH	68H	0FH

Example 2:

		ACC 02H	RAM	TRL
MOV	#0AAH,ACC	AAH	-	-
MOV	#004H,002H	AAH	04H	-
MOV	#005H,@R2	AAH	04H	05H
OR	@R2	AFH	04H	05H
MOV	#050H,@R2	AFH	04H	50H
OR	@R2	FFH	04H	50H

XOR #i8 (XOR immediate data to accumulator)

Instruction code	1 1 0 0 0 1 j 1 j 0 C 4 H ~ C 7 H
Number of bytes	1
Number of cycles	1
Function	$(ACC) \leftarrow (ACC) \nabla ((Rj)) \quad j=0,1,2,3$
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction XORs the immediate data (i7 to i0) with the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1:

```

MOV      #000H,ACC
XOR      #00FH
XOR      #0F0H
XOR      #00FH
XOR      #0F0H

```

ACC
00H
0FH
FFH
F0H
00H

Example 2:

```

MOV      #000H,ACC
XOR      #001H
XOR      #002H
XOR      #004H
XOR      #008H
XOR      #008H
XOR      #004H
XOR      #002H
XOR      #001H

```

ACC
00H
01H
03H
07H
0FH
07H
03H
01H
00H

XOR d9 (XOR direct byte to accumulator)

Instruction code	1 1 1 0 0 1 d8 Å d7 d6 d5 d4 d3 d2 d1 d0 F2H~F3H
Number of bytes	2
Number of cycles	1
Function	(ACC) ← (ACC) (d9)
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction XORs the contents of data memory (RAM) or a Special Function Register (SFR), as specified by d8 to d0, with the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1:

		ACC	RAM 23H
MOV	#000H,ACC	00H	-
MOV	#055H,023H	00H	55H
XOR	023H	55H	55H
MOV	#0FFH,023H	55H	FFH
XOR	023H	AAH	FFH

Example 2:

		ACC	B
MOV	#0FFH,ACC	FFH	-
MOV	#010H,B	FFH	10H
XOR	B	EFH	10H
MOV	#020H,B	EFH	20H
XOR	B	CFH	20H
MOV	#040H,B	CFH	40H
XOR	B	8FH	40H
MOV	#080H,B	8FH	80H
XOR	B	0FH	80H

XOR @Rj (XOR indirect byte to accumulator)

Instruction code	1 1 1 0 1 j1 j0 F4H~F7H
Number of bytes	1
Number of cycles	1
Function	(ACC) ← (ACC) ⊕ ((Rj)) j=0,1,2,3
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction XORs the contents of data memory (RAM) or a Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0, with the contents of the Accumulator (ACC), and then sends the result to the ACC.

Example 1:

		ACC	RAM 01H	RAM 68H
MOV	#000H,ACC	00H	-	-
MOV	#068H,001H	00H	68H	-
MOV	#0F0H,@R1	00H	68H	F0H
XOR	@R1	F0H	68H	F0H
MOV	#0FFH,@R1	F0H	68H	FFH
XOR	@R1	0FH	68H	FFH

Example 2:

		ACC	RAM 03H	TRL
MOV	#0AAH,ACC	AAH	-	-
MOV	#004H,003H	AAH	04H	-
MOV	#0FFH,@R3	AAH	04H	FFH
XOR	@R3	55H	04H	FFH
XOR	@R3	AAH	04H	FFH
XOR	@R3	55H	04H	FFH
XOR	@R3	AAH	04H	FFH

ROL (Rotate accumulator left)

Instruction code 1 1 1 0 0 0 0 0E0H	
Number of bytes	1
Number of cycles	1
Function	A7←A6←A5←A4←A3←A2←A1←A0"
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction rotates the 8-bit data that is stored in the Accumulator (ACC) one bit to the left. Accordingly, the data in bit 7 of the ACC moves to bit 0.

Example 1:

		ACC	
MOV	#01H,ACC	01H	0000 0001B
ROL		02H	0000 0010B
ROL		04H	0000 0100B
ROL		08H	0000 1000B
ROL		10H	0001 0000B
ROL		20H	0010 0000B
ROL		40H	0100 0000B
ROL		80H	1000 0000B
ROL		01H	0000 0001B
MOV	#55H,ACC	55H	0101 0101B
ROL		AAH	1010 1010B
ROL		55H	0101 0101B
ROL		AAH	1010 1010B
ROL		55H	0101 0101B

ROL (Rotate accumulator left through the carry flag)

Instruction code	1 1 1 1 0 0 0 0F0H
Number of bytes	1
Number of cycles	1
Function	$A7 \leftarrow A6 \leftarrow A5 \leftarrow A4 \leftarrow A3 \leftarrow A2 \leftarrow A1 \leftarrow A0 \leftarrow CY \leftarrow$
Affected flags	CY
Interrupt acceptance	Permitted

Description:

This instruction rotates the 8-bit data that is stored in the Accumulator (ACC), including the carry flag (CY), one bit to the left. Accordingly, the data in bit 7 of the ACC moves to CY, and the contents in CY move to bit 0.

Example 1:

		ACC		CY
MOV	#01H,ACC	01H	0000 0001B	-
SET1	PSW,7	01H	0000 0001B	1
ROL		03H	0000 0011B	0
ROL		06H	0000 0110B	0
ROL		0CH	0000 1100B	0
ROL		11H	0001 1000B	0
ROL		30H	0011 0000B	0
ROL		60H	0110 0000B	0
ROL		C0H	1100 0000B	0
ROL		80H	1000 0000B	1
ROL		01H	0000 0001B	1
MOV	#55H,ACC	55H	0101 0101B	1
ROL		ABH	1010 1011B	0
ROL		56H	0101 0110B	1
ROL		ADH	1010 1101B	0

ROR (Rotate accumulator right)

Instruction code 1 1 0 0 0 0 0 0 C0H	
Number of bytes	1
Number of cycles	1
Function	→A7→A6→A5→A4→A3→A2→A1→A0
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction rotates the 8-bit data that is stored in the Accumulator (ACC) one bit to the right. Accordingly, the data in bit 0 of the ACC moves to bit 7.

Example 1:

		ACC	
MOV	#01H,ACC	01H	0000 0001B
ROR		80H	1000 0000B
ROR		40H	0100 0000B
ROR		20H	0010 0000B
ROR		10H	0001 0000B
ROR		08H	0000 1000B
ROR		04H	0000 0100B
ROR		02H	0000 0010B
ROR		01H	0000 0001B
MOV	#51H,ACC	51H	0101 0001B
ROR		A8H	1010 1000B
ROR		54H	0101 0100B
ROR		2AH	0010 1010B
ROR		15H	0001 0101B

RORC (Rotate accumulator right through the carry flag)

Instruction code 1 1 0 1 0 0 0 0D0H	
Number of bytes	1
Number of cycles	1
Function	→CY→A7→A6→A5→A4→A3→A2→A1→A0
Affected flags	CY
Interrupt acceptance	Permitted

Description:

This instruction rotates the 8-bit data that is stored in the Accumulator (ACC), including the carry flag (CY), one bit to the right. Accordingly, the data in bit 0 of the ACC moves to CY, and the contents in CY move to bit 7.

Example 1:

		ACC		CY
MOV	#01H,ACC	01H	0000 0001B	-
SET1	PSW,7	01H	0000 0001B	1
RORC		80H	1000 0000B	1
RORC		C0H	1100 0000B	0
RORC		60H	0110 0000B	0
RORC		30H	0011 0000B	0
RORC		18H	0001 1000B	0
RORC		0CH	0000 1100B	0
RORC		06H	0000 0110B	0
RORC		03H	0000 0011B	0
RORC		01H	0000 0001B	1
MOV	#55H,ACC	55H	0101 0101B	1
RORC		AAH	1010 1010B	1
RORC		D5H	1101 0101B	0
RORC		6AH	0110 1010B	1

4. Data Transfer Instructions

LD d9 (Load direct byte to accumulator)

Instruction code	0 0 0 0 0 1d8 d8d7d6d5d4d3d2d1d0 02H~03H
Number of bytes	2
Number of cycles	1
Function	(ACC)←(d9)
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction transfers the contents of data memory (RAM) or a Special Function Register (SFR), as specified by d8 to d0, to the Accumulator (ACC).

Example 1:

		ACC	RAM 70H	RAM 71H
MOV	#0FF,ACC	FFH	-	-
MOV	#055H,070H	FFH	55H	-
MOV	#0AAH,071H	FFH	55H	AAH
LD	070H	55H	55H	AAH
LD	071H	AAH	55H	AAH

Example 2:

		ACC	B	SP
MOV	#0FF,ACC	FFH	-	-
MOV	#0F0H,B	FFH	F0H	-
MOV	#00FH,SP	FFH	F0H	0FH
LD	B	F0H	F0H	0FH
LD	SP	0FH	F0H	0FH
LD	B	F0H	F0H	0FH

LD•@Rj•(Load indirect byte to accumulator)

Instruction code	0 0 0 0 1 j1 j0 04H~07H
Number of bytes	1
Number of cycles	1
Function	(ACC)←((Rj)) j=0,1,2,3
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction transfers the contents of data memory (RAM) or a Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0, to the Accumulator (ACC).

Example 1:

		ACC	RAM 00H	RAM 01H	RAM 70H	RAM 7FH
MOV	#0FFH,ACC	FFH	-	-	-	-
MOV	#070H,000H	FFH	70H	-	-	-
MOV	#07FH,001H	FFH	70H	7FH	-	-
MOV	#0F0H,@R0	FFH	70H	7FH	F0H	-
MOV	#00FH,@R1	FFH	70H	7FH	F0H	0FH
LD	@R0	F0H	70H	7FH	F0H	0FH
LD	@R1	0FH	70H	7FH	F0H	0FH

Example 2:

		ACC	RAM 02H	RAM 03H	B 102H	C 103H
MOV	#0FF,ACC	FFH	-	-	-	-
MOV	#004H,002H	FFH	04H	-	-	-
MOV	#005H,003H	FFH	04H	05H	-	-
MOV	#0AAH,@R2	FFH	04H	05H	AAH	-
MOV	#055H,@R3	FFH	04H	05H	AAH	55H
LD	@R2	AAH	04H	05H	AAH	55H
LD	@R3	55H	04H	05H	AAH	55H

ST d9 (Store direct byte to accumulator)

Instruction code		0 0 0 1 0 0 1d8 d7d6d5d4d3d2d1d0 12H~13H
Number of bytes		2
Number of cycles		1
Function		(d9)←(ACC)
Affected flags		
Interrupt acceptance		Permitted

Description:

This instruction transfers the contents of the Accumulator (ACC) to data memory (RAM) or a Special Function Register (SFR), as specified by d8 to d0.

Example 1:

		ACC	RAM 70H	RAM 71H
MOV	#0FFH,ACC	FFH	-	-
MOV	#055H,070H	FFH	55H	-
MOV	#0AAH,071H	FFH	55H	AAH
ST	070H	FFH	FFH	AAH
MOV	#000H,ACC	00H	FFH	AAH
ST	071H	00H	FFH	00H

Example 2:

		ACC	B	SP
MOV	#012H,ACC	12H	-	-
MOV	#0F0H,B	12H	F0H	-
MOV	#00FH,SP	12H	F0H	0FH
ST	B	12H	12H	0FH
MOV	#034H,ACC	34H	12H	0FH
ST	SP	34H	12H	34H
ST	B	34H	34H	34H

ST @Rj (Store indirect byte to accumulator)

Instruction code	0 0 0 1 0 1 j1 j0 14H~17H
Number of bytes	1
Number of cycles	1
Function	((Rj)) ← (ACC) j=0,1,2,3
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction transfers the contents of the Accumulator (ACC) to data memory (RAM) or a Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0.

Example 1:

		ACC	RAM 00H	RAM 01H	RAM 70H	RAM 7FH
MOV	#0FFH,ACC	FFH	-	-	-	-
MOV	#070H,000H	FFH	70H	-	-	-
MOV	#07FH,001H	FFH	70H	7FH	-	-
MOV	#0F0H,@R0	FFH	70H	7FH	F0H	-
MOV	#00FH,@R1	FFH	70H	7FH	F0H	0FH
ST	@R0	FFH	70H	7FH	FFH	0FH
ST	@R1	FFH	70H	7FH	FFH	FFH

Example 2:

		ACC	RAM 02H	RAM 03H	TRL 104H	TRH 105H
MOV	#000H,ACC	00H	-	-	-	-
MOV	#004H,002H	00H	04H	-	-	-
MOV	#005H,003H	00H	04H	05H	-	-
MOV	#0AAH,@R2	00H	04H	05H	AAH	-
MOV	#055H,@R3	00H	04H	05H	AAH	55H
ST	@R2	00H	04H	05H	00H	55H
ST	@R3	00H	04H	05H	00H	00H

MOV #i8,d9 (Move immediate data to direct byte)

Instruction code	0 0 1 0 0 0 1 d8 d7 d6 d5 d4 d3 d2 d1 d0 i7 i6 i5 i4 i3 i2 i1 i0 22H~23H
Number of bytes	3
Number of cycles	2
Function	(d9) ← #i8
Affected flags	
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction transfers immediate data (i7 to i0) to data memory (RAM) or a Special Function Register (SFR), as specified by d8 to d0.

Example 1:

		RAM 00H	RAM 01H	RAM 02H	RAM 03H
MOV	#0FFH,000H	FFH	-	-	-
MOV	#0FEH,001H	FFH	FEH	-	-
MOV	#0FDH,002H	FFH	FEH	FDH	-
MOV	#0FCH,003H	FFH	FEH	FDH	FCH
MOV	#0FBH,003H	FFH	FEH	FDH	FBH
MOV	#0FAH,002H	FFH	FEH	FAH	FBH
MOV	#0F9H,001H	FFH	F9H	FAH	FBH
MOV	#0F8H,000H	F8H	F9H	FAH	FBH

Example 2:

		ACC	B	TRL
MOV	#0FFH,100H	FFH	-	-
MOV	#0FEH,102H	FFH	FEH	-
MOV	#0FDH,104H	FFH	FEH	FDH
MOV	#0FAH,104H	FFH	FEH	FAH
MOV	#0F9H,102H	FFH	F9H	FAH
MOV	#0F8H,100H	F8H	F9H	FAH

MOV #i8,@Rj (Move immediate data to indirect byte)

Instruction code	0 0 1 0 0 1 j1 j0 i7 i6 i5 i4 i3 i2 i1 i0 24H~27H
Number of bytes	2
Number of cycles	1
Function	((Rj)) ← #i8 j=0,1,2,3
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction transfers immediate data (i7 to i0) to data memory (RAM) or a Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0.

Example 1:

		RAM 00H	RAM 01H	RAM 7EH	RAM 7FH
MOV	#07FH,000H	7FH	-	-	-
MOV	#07EH,001H	7FH	7EH	-	-
MOV	#0FDH,@R0	7FH	7EH	-	FDH
MOV	#0FCH,@R1	7FH	7EH	FCH	FDH
MOV	#0FBH,@R0	7FH	7EH	FCH	FBH
MOV	#0FAH,@R1	7FH	7EH	FAH	FBH
MOV	#0F9H,@R0	7FH	7EH	FAH	F9H
MOV	#0F8H,@R1	7FH	7EH	F8H	F9H

Example 2:

		RAM 02H	RAM 03H	ACC 100H	B 102H
MOV	#000H,002H	00H	-	-	-
MOV	#002H,003H	00H	02H	-	-
MOV	#0FDH,@R2	00H	02H	FDH	-
MOV	#0FCH,@R3	00H	02H	FDH	FCH
MOV	#0FBH,@R2	00H	02H	FBH	FCH
MOV	#0FAH,@R3	00H	02H	FBH	FAH

LDC (Load code byte relative to TRR to accumulator)

Instruction code 1 1 0 0 0 0 1 C1H	
Number of bytes	1
Number of cycles	2
Function	(ACC) ← (BNK)((TRR)+(ACC)) [ROM]
Affected flags	
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction transfers contents of the address in program memory (ROM) that is specified by the sum of the contents of the Table Reference Register (TRR) and the contents of the Accumulator (ACC), to the ACC. The ROM data that is referenced during internal program operation and external program operation differs. During internal program operation, internal ROM is referenced; during external program operation, BANK0 of external ROM is referenced.

The LDC instruction cannot reference BANK1 of external ROM.

Example:

		ACC	TRR		TRR +ACC
			TRH	TRL	
MOV	#001H,TRH	-	01H	-	-
MOV	#023H,TRL	-	01H	23H	-
MOV	#000H,ACC	00H	01H	23H	0123H
LDC		30H	01H	23H	0153H
MOV	#001H,ACC	01H	01H	23H	0124H
LDC		FFH	01H	23H	0222H
MOV	#002H,ACC	02H	01H	23H	0125H
LDC		57H	01H	23H	017AH
MOV	#003H,ACC	03H	01H	23H	0126H
LDC		EAH	01H	23H	020DH

PC	ROM
0123H	30H
0124H	FFH
0125H	57H
0126H	EAH

PUSH d9 (Push direct byte to stack)

Instruction code	0 1 1 0 0 0 0d8 d7d6d5d4d3d2d1d0 60H~61H
Number of bytes	2
Number of cycles	2
Function	(SP) ← (SP)+1, ((SP)) ← (d9)
Affected flags	
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Stack Pointer (SP), and then transfers the contents of data memory (RAM) or a Special Function Register (SFR), as specified by d8 to d0, to the address in RAM specified by the SP.

Example:

		ACC	B	RAM 00H	SP	RAM 20H	RAM 21H	RAM 22H
MOV	#0AAH,ACC	AAH	-	-	-	-	-	-
MOV	#055H,B	AAH	55H	-	-	-	-	-
MOV	#012H,000H	AAH	55H	12H	-	-	-	-
MOV	#01FH,SP	AAH	55H	12H	1FH	-	-	-
PUSH	ACC	AAH	55H	12H	20H	AAH	-	-
PUSH	B	AAH	55H	12H	21H	AAH	55H	-
PUSH	000H	AAH	55H	12H	22H	AAH	55H	12H
POP	B	AAH	12H	12H	21H	AAH	55H	12H
POP	ACC	55H	12H	12H	20H	AAH	55H	12H
POP	000H	55H	12H	AAH	1FH	AAH	55H	12H

POP d9 (Pop direct byte from stack)

Instruction code		0 1 1 0 0 0 d8 d7d6d5d4d3d2d1d0 70H~71H
Number of bytes		2
Number of cycles		2
Function		(d9) ← ((SP)), (SP) ← (SP) - 1
Affected flags		
Interrupt acceptance		Permitted at 2nd cycle

Description:

This instruction transfers the contents of the address in RAM specified by the Stack Pointer (SP) to data memory (RAM) or a Special Function Register (SFR), as specified by d8 to d0, and then decrements the SP.

Example:

		ACC	B	TRL	SP	RAM 20H	RAM 21H	RAM 22H
MOV	#0AAH,ACC	AAH	-	-	-	-	-	-
MOV	#055H,B	AAH	55H	-	-	-	-	-
MOV	#012H,TRL	AAH	55H	12H	-	-	-	-
MOV	#01FH,SP	AAH	55H	12H	1FH	-	-	-
PUSH	ACC	AAH	55H	12H	20H	AAH	-	-
PUSH	B	AAH	55H	12H	21H	AAH	55H	-
PUSH	TRL	AAH	55H	12H	22H	AAH	55H	12H
POP	B	AAH	12H	12H	21H	AAH	55H	12H
POP	ACC	55H	12H	12H	20H	AAH	55H	12H
POP	TRL	55H	12H	AAH	1FH	AAH	55H	12H

XCH d9 (Exchange direct byte with accumulator)

Instruction code	1 1 0 0 0 1 d8 d7d6d5d4d3d2d1d0 C2H~C3H
Number of bytes	2
Number of cycles	1
Function	(ACC)↔(d9)
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction exchanges the contents of the Accumulator (ACC) with the contents of data memory (RAM) or a Special Function Register (SFR), as specified by d8 to d0.

Example 1:

```
MOV    #0FFH,ACC
MOV    #055H,023H
XCH    023H
XCH    023H
XCH    023H
XCH    023H
```

ACC	RAM 23H
FFH	-
FFH	55H
55H	FFH
FFH	55H
55H	FFH
FFH	55H

Example 2:

```
MOV    #0FFH,ACC
MOV    #0FEH,B
XCH    B
XCH    B
XCH    B
XCH    B
```

ACC	B
FFH	-
FFH	FEH
FEH	FFH
FFH	FEH
FEH	FFH
FFH	FEH

XCH @Rj (Exchange indirect byte with accumulator)

Instruction code	1 1 0 0 0 1j1j0 C4H~C7H
Number of bytes	1
Number of cycles	1
Function	(ACC) \longleftrightarrow ((Rj)) j=0,1,2,3
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction exchanges the contents of the Accumulator (ACC) with the contents of data memory (RAM) or a Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0.

Example 1:

		ACC	RAM 01H	RAM 68H
MOV	#0FFH,ACC	FFH	-	-
MOV	#068H,001H	FFH	68H	-
MOV	#0F0H,@R1	FFH	68H	F0H
XCH	@R1	F0H	68H	FFH
XCH	@R1	FFH	68H	F0H
XCH	@R1	F0H	68H	FFH
XCH	@R1	FFH	68H	F0H

Example 2:

		ACC	RAM 03H	TRL
MOV	#0AAH,ACC	AAH	-	-
MOV	#004H,003H	AAH	04H	-
MOV	#055H,@R3	AAH	04H	55H
XCH	@R3	55H	04H	AAH
XCH	@R3	AAH	04H	55H
XCH	@R3	55H	04H	AAH

5. Jump Instructions

JMP a12 (Jump near absolute address)

Instruction code	0 0 1 a11 1 a10 a9 a8 a7 a6 a5 a4 a3 a2 a1 a0 28H~2FH, 38H~3FH
Number of bytes	2
Number of cycles	2
Function	$(PC) \leftarrow (PC) + 2, (PC11 \sim 00) \leftarrow a12$
Affected flags	
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) twice, and then transfers the data a11 through a0 to bits 11 through 00 of the PC.

Example 1:

The value of label LA is 0F0EH.

```

                NOP
                NOP
                JMP      LA
LA:             INC      ACC
                ROR

```

PC	Instruction code
0FFBH	00H
0FFCH	00H
0FFDH	3F0EH
0F0EH	6300H
0F10H	C0H

Example 2:

The value of label LA is 1F0EH.

```

                NOP
                NOP
                JMP      LA
LA:             INC      ACC
                ROR

```

PC	Instruction code
0FFCH	00H
0FFDH	00H
0FFEH	3F0EH
1F0EH	6300H
1F10H	C0H

JMPF a16 (Jump far absolute address)

Instruction code 0 0 1 0 0 0 1 a15a14a13a12a11a10a9a8 a7a6a5a4a3a2a1a0 21H	
Number of bytes	3
Number of cycles	2
Function	(PC)←a16
Affected flags	
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction transfers the data a15 through a0 to the Program Counter (PC).

Example 1:

The value of label LA is 0F0EH.

```

NOP
NOP
JMPF      LA
LA:      INC      ACC
ROR
```

PC	Instruction code
OFFAH	00H
OFFBH	00H
OFFCH	210F0EH
0F0EH	6300H
0F10H	C0H

Example 2:

The value of label LA is 0F0EH.

```

NOP
NOP
JMPF      LA
LA:      INC      ACC
ROR
```

PC	Instruction code
OFFCH	00H
OFFDH	00H
OFFEH	210F0EH
0F0EH	6300H
0F10H	C0H

BR r8 (Branch near relative address)

Instruction code	0 0 0 0 0 0 1 r7r6r5r4r3r2r1r0 01H
Number of bytes	2
Number of cycles	2
Function	$(PC) \leftarrow (PC) + 2, (PC) \leftarrow (PC) + r8$
Affected flags	
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) twice, adds the data r7 through r0 to the PC, and then transfers that result to the PC.

Example 1:

The value of label LA is 0F5FH.

```

                NOP
                NOP
                BR      LA
LA:             INC     ACC
                ROR

```

PC	Instruction code
0F1CH	00H
0F1DH	00H
0F1EH	013FH
0F5FH	6300H
0F61H	C0H

Example 2:

The value of label LA is 1F0EH.

```

                NOP
                NOP
                INC     ACC
                ROR
                NOP
                NOP
                BR      LA

```

PC	Instruction code
1F0CH	00H
1F0DH	00H
1F0EH	6300H
1F10H	C0H
1F11H	00H
1F12H	00H
1F13H	01F9H

BRF r16 (Branch far relative address)

Instruction code 0 0 0 1 0 0 0 1 r7r6r5r4r3r2r1r0 r15r14r13r12r11r10r9r8r11H	
Number of bytes	3
Number of cycles	4
Function	$(PC) \leftarrow (PC) + 3, (PC) \leftarrow (PC) - 1 + r16$
Affected flags	
Interrupt acceptance	Permitted at 4th cycle

Description:

This instruction increments the Program Counter (PC) three times, then decrements the PC, adds the data r15 through r0 to the PC, and then transfers that result to the PC.

Example 1:

The value of label LA is 105FH.

```

NOP
NOP
BRF      LA
LA:      INC      ACC
        ROR
```

PC	Instruction code
0F1CH	00H
0F1DH	00H
0F1EH	113F01H
105FH	6300H
1061H	C0H

Example 2:

The value of label LA is 1F0EH.

```

NOP
NOP
LA:      INC      ACC
        ROR
        NOP
        NOP
        BRF      LA
```

PC	Instruction code
1FFCH	00H
1FFDH	00H
1F0EH	6300H
1F10H	C0H
1F11H	00H
1F12H	00H
1F13H	11F8FFH

6. Conditional Branching Instructions

BZ r8 (Branch near relative address if accumulator is zero)

Instruction code	1 0 0 0 0 0 0 r7r6r5r4r3r2r1r080H
Number of bytes	2
Number of cycles	2
Function	$(PC) \leftarrow (PC) + 2$, if $(ACC) = 0$ then $(PC) \leftarrow (PC) + r8$
Affected flags	
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) twice, and then, if the Accumulator (ACC) is zero, adds the data r7 through r0 to the PC, and transfers that result to the PC. If the ACC is not zero, the next instruction is executed.

Example 1:

			PC	Instruction code	ACC
	MOV	#000H,ACC	0F1BH	230000H	00H
	BZ	LA	0F1EH	803FH	00H
LA:	INC	ACC	0F5FH	6300H	01H
	ROR		0F61H	C0H	80H

- Because $ACC = "0"$ when the BZ instruction is executed, the program branches to the label LA.

Example 2:

			PC	Instruction code	ACC
	MOV	#001H,ACC	0F1BH	230001H	01H
	BZ	LA	0F1EH	803FH	01H
	DEC	ACC	0F20H	7300H	00H
	ROR		0F22H	C0H	00H
LA:	INC	ACC			

- Because $ACC \neq "0"$ when the BZ instruction is executed, the program simply executes the next instruction.

BNZ r8 (Branch near relative address if accumulator is not zero)

Instruction code 1 0 0 1 0 0 0 0 r7r6r5r4r3r2r1r0 90H	
Number of bytes	2
Number of cycles	2
Function	$(PC) \leftarrow (PC) + 2$, if $(ACC) \neq 0$ then $(PC) \leftarrow (PC) + r8$
Affected flags	
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) twice, and then, if the Accumulator (ACC) is not zero, adds the data r7 through r0 to the PC, and transfers that result to the PC. If the ACC is zero, the next instruction is executed.

Example 1:

			PC	Instruction code	ACC
	MOV	#001H,ACC	0F1BH	230001H	01H
BNZ	LA	0F1EH	903FH	01H	
LA:	INC	ACC	0F5FH	6300H	02H
	ROR		0F61H	C0H	01H

- Because $ACC \neq 0$ when the BNZ instruction is executed, the program branches to the label LA.

Example 2:

			PC	Instruction code	ACC
	MOV	#000H,ACC	0F1BH	230000H	00H
	BNZ	LA	0F1EH	903FH	00H
	DEC	ACC	0F20H	7300H	FFH
	ROR		0F22H	C0H	FFH
LA:	INC	ACC			

- Because $ACC = 0$ when the BNZ instruction is executed, the program simply executes the next instruction.

BP d9,b3,r8 (Branch near relative address if direct bit is positive)

Instruction code	0 1 1d8 1b2b1b0 d7d6d5d4d3d2d1d0 r7r6r5r4r3r2r1r068H~6FH,78H~7FH
Number of bytes	3
Number of cycles	2
Function	$(PC) \leftarrow (PC) + 3$, if (d9,b3) = 1 then $(PC) \leftarrow (PC) + r8$
Affected flags	
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) three times, and then, if the bit specified by b2 to b0 of the address in data memory (RAM) or the Special Function Register (SFR) that is specified by d8 to d0 is set (1), this instruction adds the data r7 through r0 to the PC, and transfers that result to the PC. If the bit specified by b2 to b0 of the address in data memory (RAM) or the Special Function Register that is specified by d8 to d0 is reset (0), the next instruction is executed.

Example 1:

			PC	Instruction code	B
	MOV	#001H,B	0F1AH	230201H	01H
	BP	B,0,LA	0F1DH	78023FH	01H
LA:	INC	B	0F5FH	6302H	02H
	NOP		0F61H	00H	02H

- Because bit 0 of B is "1" when the BP instruction is executed, the program branches to the label LA.

Example 2:

			PC	Instruction code	ACC
	MOV	#080H,ACC	0F1AH	230080H	80H
	BP	ACC,0,LA	0F1DH	78003FH	80H
	DEC	ACC	0F20H	7300H	7FH
	ROR		0F22H	C0H	BFH
LA:	INC	ACC			

- Because bit 0 of the ACC is "0" when the BP instruction is executed, the program simply executes the next instruction.

BPC d9,b3,r8 (Branch near relative address if direct bit is positive, and clear)

Instruction code	0 1 0d8 1b2b1b0 d7d6d5d4d3d2d1d0 r7r6r5r4r3r2r1r048H~4FH,58H~5FH
Number of bytes	3
Number of cycles	2
Function	$(PC) \leftarrow (PC) + 3$, if (d9,b3) = 1 then $(PC) \leftarrow (PC) + r8$, (d9,b3) = 0
Affected flags	
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) three times, and then, if the bit specified by b2 to b0 of the address in data memory (RAM) or the Special Function Register (SFR) that is specified by d8 to d0 is set (1), this instruction resets that bit, then adds the data r7 through r0 to the PC, and transfers that result to the PC. If the bit specified by b2 to b0 of the address in data memory (RAM) or the Special Function Register that is specified by d8 to d0 is reset (0), the next instruction is executed.

Example 1:

			PC	Instruction code	B
	MOV	#003H,B	0F1AH	230203H	03H
	BPC	B,0,LA	0F1DH	58023FH	02H
LA:	INC	B	0F5FH	6302H	03H
	NOP		0F61H	00H	03H

- Because bit 0 of B is "1" when the BPC instruction is executed, the program branches to the label LA.

Example 2:

			PC	Instruction code	ACC
	MOV	#080H,ACC	0F1AH	230080H	80H
	BPC	ACC,0,LA	0F1DH	58003FH	80H
	DEC	ACC	0F20H	7300H	7FH
	ROR		0F22H	C0H	BFH
LA:	INC	ACC			

- Because bit 0 of the ACC is "0" when the BPC instruction is executed, the program simply executes the next instruction.

Note:

- When this instruction is applied to one of the ports P0, P1, P2, P3, P4, or P5, the port latch of that port is selected; the external signal that is applied to that port is not selected. Furthermore, applying this instruction to port P7 does not change its status.
-

BN d9,b3,r8 (Branch near relative address if direct bit is negative)

Instruction code	1 0 0 d8 1 b2b1b0 d7d6d5d4d3d2d1d0 r7r6r5r4r3r2r1r0 88H~8FH,98H~9FH
Number of bytes	3
Number of cycles	2
Function	$(PC) \leftarrow (PC) + 3$, if (d9,b3) = 0 then $(PC) \leftarrow (PC) + r8$
Affected flags	
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) three times, and then, if the bit specified by b2 to b0 of the address in data memory (RAM) or the Special Function Register (SFR) that is specified by d8 to d0 is reset (0), this instruction adds the data r7 through r0 to the PC, and transfers that result to the PC. If the bit specified by b2 to b0 of the address in data memory (RAM) or the Special Function Register that is specified by d8 to d0 is set (1), the next instruction is executed.

Example 1:

			PC	Instruction code	B
	MOV	#0FEH,B	0F1AH	2302FEH	FEH
	BN	B,0,LA	0F1DH	98023FH	FEH
LA:	INC	B	0F5FH	6302H	FFH
	NOP		0F61H	00H	FFH

- Because bit 0 of B is "0" when the BN instruction is executed, the program branches to the label LA.

Example 2:

			PC	Instruction code	ACC
	MOV	#001H,ACC	0F1AH	230001H	01H
	BN	ACC,0,LA	0F1DH	98003FH	01H
	DEC	ACC	0F20H	7300H	00H
	ROR		0F22H	C0H	00H
LA:	INC	ACC			

- Because bit 0 of the ACC is "1" when the BN instruction is executed, the program simply executes the next instruction.

DBNZ d9,r8 (Decrement direct byte and branch near relative address if direct byte is not zero)

Instruction code	0 1 0 1 0 0 1d8 d7d6d5d4d3d2d1d0 r7r6r5r4r3r2r1r0 52H~53H
Number of bytes	3
Number of cycles	2
Function	$(PC) \leftarrow (PC) + 3$, $(d9) = (d9) - 1$, if $(d9) \neq 0$ then $(PC) \leftarrow (PC) + r8$
Affected flags	
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) three times, and then decrements the address in data memory (RAM) or the Special Function Register (SFR) that is specified by d8 to d0. If the contents of the decremented RAM address or SFR are not zero, this instruction adds the data r7 through r0 to the PC, and transfers that result to the PC. If the contents of the decremented RAM address or Special Function Register are zero, the next instruction is executed.

Example 1:

			PC	Instruction code	B
	MOV	#002H,B	0F1AH	230202H	02H
	DBNZ	B,LA	0F1DH	53023FH	01H
LA:	INC	B	0F5FH	6302H	02H
	NOP		0F61H	00H	02H

- Because B \neq "0" after being decremented when the DBNZ instruction is executed, the program branches to the label LA.

Example 2:

			PC	Instruction code	ACC
	MOV	#001H,ACC	0F1AH	230001H	01H
	DBNZ	ACC,LA	0F1DH	53003FH	00H
	DEC	ACC	0F20H	7300H	FFH
	ROR		0F22H	C0H	FFH
LA:	INC	ACC			

- Because ACC = "0" after being decremented when the DBNZ instruction is executed, the program simply executes the next instruction.

Note:

- When this instruction is applied to one of the ports P0, P1, P2, P3, P4, or P5, the port latch of that port is selected; the external signal that is applied to that port is not selected. Furthermore, applying this instruction to port P7 does not change its status.
-

DBNZ @Rj,r8 (Decrement indirect byte and branch near relative address if indirect byte is not zero)

Instruction code	0 1 0 1 1 j1 j0 r7 r6 r5 r4 r3 r2 r1 r0 54H~57H
Number of bytes	2
Number of cycles	2
Function	$(PC) \leftarrow (PC) + 2, ((Rj)) = ((Rj)) - 1,$
if $((Rj)) \neq 0$ then $(PC) \leftarrow (PC) + r8 \quad j = 0, 1, 2, 3$	
Affected flags	
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) twice, and then decrements the address in data memory (RAM) or the Special Function Register (SFR) that is specified by the indirect address register that is specified by j1 and j0. If the contents of the decremented RAM address or SFR are not zero, this instruction adds the data r7 through r0 to the PC, and transfers that result to the PC. If the contents of the decremented RAM address or Special Function Register are zero, the next instruction is executed.

Example 1:

			PC	Instruction code	B	RAM 03H
LA:	MOV	#002H,B	0F18H	230202H	02H	-
	MOV	#002H,003H	0F1BH	220302H	02H	02H
	DBNZ	@R3,LA	0F1EH	573FH	01H	02H
	INC	B	0F5FH	6302H	02H	02H

- Because B \neq "0" after being decremented when the DBNZ instruction is executed, the program branches to the label LA.

Example 2:

Example 2:

			PC	Instruction code	ACC	RAM 03H
	MOV	#001H,ACC	0F18H	230001H	01H	-
	MOV	#000H,003H	0F1BH	220300H	01H	00H
	DBNZ	@R3,LA	0F1EH	573FH	00H	00H
	DEC	ACC	0F20H	7300H	FFH	00H
LA:	INC	ACC				

- Because ACC = "0" after being decremented when the DBNZ instruction is executed, the program simply executes the next instruction.

Note:

- When this instruction is applied to one of the ports P0, P1, P2, P3, P4, or P5, the port latch of that port is selected; the external signal that is applied to that port is not selected. Furthermore, applying this instruction to port P7 does not change its status.

BE #i8,r8 (Compare immediate data to accumulator and branch near relative address if equal)

Instruction code	0 0 1 1 0 0 0 1 i7i6i5i4i3i2i1i0 r7r6r5r4r3r2r1r0 31H
Number of bytes	3
Number of cycles	2
Function	(PC) \leftarrow (PC) + 3, if (ACC) = #i8 then (PC) \leftarrow (PC) + r8
Affected flags	CY
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) three times, and then compares the immediate data (i7 to i0) with the contents of the Accumulator (ACC). If the compared data are the same, this instruction adds the data r7 through r0 to the PC, and then transfers that result to the PC. If the data are not the same, the next instruction is executed.

Furthermore, if the ACC is less than the immediate data, the carry flag (CY) is set; if the ACC is greater than or equal to the immediate data, the carry flag (CY) is reset.

$ACC < \#i8 \rightarrow CY=1$

$ACC \geq \#i8 \rightarrow CY=0$

Example 1:

			PC	Instruction code	ACC	CY
	MOV	#002H,ACC	0F1AH	230002H	02H	-
	BE	#002H,LA	0F1DH	31023FH	02H	0
LA:	INC	ACC	0F5FH	6300H	03H	0

- Because $ACC = 02H$ when the BE instruction is executed, CY is reset and the program branches to the label LA.

Example 2:

			PC	Instruction code	ACC	CY
	MOV	#003H,ACC	0F1AH	230003H	03H	-
	BE	#004H,LA	0F1DH	31043FH	03H	1
	DEC	ACC	0F20H	7300H	02H	1
LA:	INC	ACC				

- Because $ACC < 04H$ when the BE instruction is executed, CY is set and the program executes the next instruction.

BE d9,r8 (Compare direct byte to accumulator and branch near relative address if equal)

Instruction code	0 0 1 1 0 0 1d8 d7d6d5d4d3d2d1d0 r7r6r5r4r3r2r1r0 32H~33H
Number of bytes	3
Number of cycles	2
Function	(PC) \leftarrow (PC) + 3, if (ACC) = (d9) then (PC) \leftarrow (PC) + r8
Affected flags	CY
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) three times, and then compares the contents of data memory (RAM) or a Special Function Register (SFR), as specified by d8 to d0, with the contents of the Accumulator (ACC). If the compared data are the same, this instruction adds the data r7 through r0 to the PC, and then transfers that result to the PC. If the data are not the same, the next instruction is executed.

Furthermore, if the ACC is less than the contents of data memory (RAM) or the Special Function Register (SFR), the carry flag (CY) is set; if the ACC is greater than or equal to the contents of data memory (RAM) or the Special Function Register (SFR), the carry flag (CY) is reset.

ACC < d9 (RAM or SFR) \rightarrow CY=1

ACC > d9 (RAM or SFR) \rightarrow CY=0

Example 1:

			PC	Instruction code	ACC	B	CY
	MOV	#002H,ACC	0F17H	230002H	02H	-	-
	MOV	#002H,B	0F1AH	230202H	02H	02H	-
	BE	B,LA	0F1DH	33023FH	02H	02H	0
LA:	INC	ACC	0F5FH	6300H	03H	02H	0

- Because ACC = B when the BE instruction is executed, CY is reset and the program branches to the label LA.

Example 2:

			PC	Instruction code	ACC	B	CY
	MOV	#003H,ACC	0F17H	230003H	03H	-	-
	MOV	#0F2H,B	0F1AH	2302F2H	03H	F2H	-
	BE	B,LA	0F1DH	33023FH	03H	F2H	1
	DEC	ACC	0F20H	7300H	02H	F2H	1
LA:	INC	ACC					

- Because ACC < B when the BE instruction is executed, CY is set and the program executes the next instruction.

BE @Rj, #i8, r8 (Compare immediate data to indirect byte and branch near relative address if equal)

Instruction code	0 0 1 1 0 1 j1 j0 i7 i6 i5 i4 i3 i2 i1 i0 r7 r6 r5 r4 r3 r2 r1 r0 34H~37H
Number of bytes	3
Number of cycles	2
Function	(PC) \leftarrow (PC) + 3, if ((Rj)) = # i8 then (PC) \leftarrow (PC) + r8 j = 0,1,2,3
Affected flags	CY
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) three times, and then compares the contents of data memory (RAM) or a Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0, with the immediate data (i7 to i0). If the compared data are the same, this instruction adds the data r7 through r0 to the PC, and then transfers that result to the PC. If the data are not the same, the next instruction is executed.

Furthermore, if the contents of data memory (RAM) or the Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0, is less than the immediate data (i7 to i0), the carry flag (CY) is set; if the contents of data memory (RAM) or the Special Function Register (SFR) are greater than or equal to the immediate data (i7 to i0), the carry flag (CY) is reset.

@Rj < #i8 \rightarrow CY=1

@Rj > #i8 \rightarrow CY=0

Example 1:

			PC	Instruction code	B	RAM 03H	CY
	MOV	#005H, B	0F17H	230205H	05H	-	-
	MOV	#002H, 003H	0F1AH	220302H	05H	02H	-
	BE	@R3, #5H, LA	0F1DH	37053FH	05H	02H	0
LA:	INC	B	0F5FH	6302H	06H	02H	0

- Because B = 05H when the BE instruction is executed, CY is reset and the program branches to the label LA.

Example 2:

			PC	Instruction code	ACC	RAM 02H	CY
	MOV	#003H, ACC	0F17H	230003H	03H	-	-
	MOV	#000H, 002H	0F1AH	220200H	03H	00H	-
	BE	@R2, #9H, LA	0F1DH	36093FH	03H	00H	1
	DEC	ACC	0F20H	7300H	02H	00H	1
LA:	INC	ACC					

- Because ACC < 09H when the BE instruction is executed, CY is set and the program executes the next instruction.

BNE #i8,r8 (Compare immediate data to accumulator and branch near relative address if not equal)

Instruction code	0 1 0 0 0 0 1 i7i6i5i4i3i2i1i0 r7r6r5r4r3r2r1r0 41H
Number of bytes	3
Number of cycles	2
Function	(PC) \leftarrow (PC) + 3, if (ACC) \neq #i8 then (PC) \leftarrow (PC) + r8
Affected flags	CY
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) three times, and then compares the immediate data (i7 to i0) with the contents of the Accumulator (ACC). If the compared data are not the same, this instruction adds the data r7 through r0 to the PC, and then transfers that result to the PC. If the data are the same, the next instruction is executed. Furthermore, if the ACC is less than the immediate data, the carry flag (CY) is set; if the ACC is greater than or equal to the immediate data, the carry flag (CY) is reset.

ACC < #i8 \rightarrow CY=1

ACC > #i8 \rightarrow CY=0

Example 1:

			PC	Instruction code	ACC	CY
	MOV	#002H,ACC	0F1AH	230002H	02H	-
	BNE	#000H,LA	0F1DH	41003FH	02H	0
LA:	INC	ACC	0F5FH	6300H	03H	0

- Because ACC > 00H when the BNE instruction is executed, CY is reset and the program branches to the label LA.

Example 2:

			PC	Instruction code	ACC	CY
	MOV	#003H,ACC	0F1AH	230003H	03H	-
	BNE	#003H,LA	0F1DH	41033FH	03H	0
	DEC	ACC	0F20H	7300H	02H	0
LA:	INC	ACC				

- Because ACC = 03H when the BNE instruction is executed, CY is set and the program executes the next instruction.

BNE d9,r8 (Compare direct byte to accumulator and branch near relative address if not equal)

Instruction code	0 1 0 0 0 1 d8 d7d6d5d4d3d2d1d0 r7r6r5r4r3r2r1r0 42H~43H
Number of bytes	3
Number of cycles	2
Function	(PC) \leftarrow (PC) + 3, if (ACC) \neq (d9) then (PC) \leftarrow (PC) + r8
Affected flags	CY
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) three times, and then compares the contents of data memory (RAM) or a Special Function Register (SFR), as specified by d8 to d0, with the contents of the Accumulator (ACC). If the compared data are not the same, this instruction adds the data r7 through r0 to the PC, and then transfers that result to the PC. If the data are the same, the next instruction is executed.

Furthermore, if the ACC is less than the contents of data memory (RAM) or the Special Function Register (SFR), the carry flag (CY) is set; if the ACC is greater than or equal to the contents of data memory (RAM) or the Special Function Register (SFR), the carry flag (CY) is reset.

$ACC < d9(RAM \text{ or } SFR) \rightarrow CY=1$

$ACC > d9(RAM \text{ or } SFR) \rightarrow CY=0$

Example 1:

			PC	Instruction code	ACC	B	CY
	MOV	#002H,ACC	0F17H	230002H	02H	-	-
	MON	#003H,B	0F1AH	230203H	02H	03H	-
	BNE	B,LA	0F1DH	43023FH	02H	03H	1
LA:	INC	ACC	0F5FH	6300H	03H	03H	1

- Because $ACC < B$ when the BNE instruction is executed, CY is set and the program branches to the label LA.

Example 2:

			PC	Instruction code	ACC	B	CY
	MOV	#0F2H,ACC	0F17H	2300F2H	F2H	-	-
	MOV	#0F2H,B	0F1AH	2302F2H	F2H	F2H	-
	BNE	B,LA	0F1DH	43023FH	F2H	F2H	0
	DEC	ACC	0F20H	7300H	F1H	F2H	0
LA:	INC	ACC					

- Because $ACC = B$ when the BNE instruction is executed, CY is reset and the program executes the next instruction.

BNE @Rj, #i8, r8 (Compare immediate data to indirect byte and branch near relative address if not equal)

Instruction code	0 1 0 0 1 j1 j0 i7 i6 i5 i4 i3 i2 i1 i0 r7 r6 r5 r4 r3 r2 r1 r0 44H~47H
Number of bytes	3
Number of cycles	2
Function	$(PC) \leftarrow (PC) + 3$, if $((Rj)) \neq i8$ then $(PC) \leftarrow (PC) + r8$ $j = 0, 1, 2, 3$
Affected flags	CY
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) three times, and then compares the contents of data memory (RAM) or a Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0, with the immediate data (i7 to i0). If the compared data are not the same, this instruction adds the data r7 through r0 to the PC, and then transfers that result to the PC. If the data are the same, the next instruction is executed.

Furthermore, if the contents of data memory (RAM) or the Special Function Register (SFR), as specified by the indirect address register that is specified by j1 and j0, is less than the immediate data (i7 to i0), the carry flag (CY) is set; if the contents of data memory (RAM) or the Special Function Register (SFR) are greater than or equal to the immediate data (i7 to i0), the carry flag (CY) is reset.

@Rj < #i8 → CY=1

@Rj > #i8 → CY=0

Example 1:

			PC	Instruction code	ACC	B	CY
	MOV	#002H, ACC	0F17H	230002H	02H	-	-
	MON	#003H, B	0F1AH	230203H	02H	03H	-
	BNE	B, LA	0F1DH	43023FH	02H	03H	1
LA:	INC	ACC	0F5FH	6300H	03H	03H	1

- Because B < 08H when the BNE instruction is executed, CY is set and the program branches to the label LA.

Example 2:

			PC	Instruction code	ACC	RAM 02H	CY
	MOV	#003H, ACC	0F17H	230003H	03H	-	-
	MOV	#000H, 002H	0F1AH	220200H	03H	00H	-
	BNE	@R2, #3H, LA	0F1DH	46033FH	03H	00H	0
	DEC	ACC	0F20H	7300H	02H	00H	0
LA:	INC	ACC					

- Because ACC = 03H when the BNE instruction is executed, CY is reset and the program executes the next instruction.

7. Subroutine Instructions

CALL a12 (Near absolute subroutine call)

Instruction code	0 0 0a11 1a10a9a8 a7a6a5a4a3a2a1a0 08H~0FH,18H~1FH
Number of bytes	2
Number of cycles	2
Function	$(PC) \leftarrow (PC) + 2, (SP) \leftarrow (SP) + 1, ((SP)) \leftarrow (PC7 \sim 0), (SP) \leftarrow (SP) + 1, ((SP)) \leftarrow (PC15 \sim 8), (PC11 \sim 0) \leftarrow a12$
Affected flags	
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) twice, increments the Stack Pointer (SP), and then stores the lower byte of the PC in the address in data memory (RAM) that is specified by the SP. This instruction then increments the Stack Pointer (SP) again, and stores the upper byte of the PC in the address in RAM specified by the SP. Finally, this instruction then transfers the data a11 through a0 to bits 11 through 00 of the PC.

Example 1:

The value of label LA is 0F0EH.

		PC	Instruction code	SP	RAM 20H	RAM 21H
	MOV	#01FH,SP	0FFAH	23061FH	1FH	-
	CALL	LA	0FFDH	1F0EH	21H	FFH
LA:	INC	ACC	0F0EH	6300H	21H	FFH
	RET		0F10H	A0H	1FH	FFH
	NOP		0FFFH	00H	1FH	FFH

Example 2:

The value of label LA is 1F0EH.

		PC	Instruction code	SP	RAM 20H	RAM 21H
	MOV	#01FH,SP	0FFBH	23061FH	1FH	-
	CALL	LA	0FFEh	1F0EH	21H	00H
LA:	INC	ACC	1F0EH	6300H	21H	00H
	RET	1F10H	A0H	1FH	00H	10H
	INC	ACC	1000H	6300H	1FH	00H

CALLF a16 (Far absolute subroutine call)

Instruction code	0 0 1 0 0 0 0 a15a14a13a12a11a10a9a8a7a6a5a4a3a2a1a0 20H
Number of bytes	3
Number of cycles	2
Function	$(PC) \leftarrow (PC) + 3, (SP) \leftarrow (SP) + 1, ((SP)) \leftarrow (PC7-0), (SP) \leftarrow (SP) + 1, ((SP)) \leftarrow (PC15-8), (PC) \leftarrow a16$
Affected flags	
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction increments the Program Counter (PC) three times, increments the Stack Pointer (SP), and then stores the lower byte of the PC in the address in data memory (RAM) that is specified by the SP. This instruction then increments the Stack Pointer (SP) again, and stores the upper byte of the PC in the address in RAM specified by the SP. Finally, this instruction then transfers the data a15 through a0 to bits 15 through 00 of the PC.

Example 1:

The value of label LA is 0F0EH.

			PC	Instruction code	SP	RAM 20H	RAM 21H
	MOV	#01FH,SP	0FF9H	23061FH	1FH	-	-
	CALLF	LA	0FFCH	200F0EH	21H	FFH	0FH
LA:	INC	ACC	0F0EH	6300H	21H	FFH	0FH
	RET		0F10H	A0H	1FH	FFH	0FH
	NOP		0FFFH	00H	1FH	FFH	0FH

Example 2:

The value of label LA is 0F0EH.

			PC	Instruction code	SP	RAM 20H	RAM 21H
	MOV	#01FH,SP	0FFAH	23061FH	1FH	-	-
	CALLF	LA	0FFDH	200F0EH	21H	00H	10H
LA:	INC	ACC	0F0EH	6300H	21H	00H	10H
	RET		0F10H	A0H	1FH	00H	10H
	INC	ACC	1000H	6300H	1FH	00H	10H

CALLR r16 (Far relative subroutine call)

Instruction code		00010000 r7r6r5r4r3r2r1r0 r15r14r13r12r11r10r9r8 10H
Number of bytes		3
Number of cycles		4
Function		$(PC) \leftarrow (PC) + 3, (SP) \leftarrow (SP) + 1, ((SP)) \leftarrow (PC7-0), (SP) \leftarrow (SP) + 1, ((SP)) \leftarrow (PC15-8), (PC) \leftarrow (PC) - 1 + r16$
Affected flags		
Interrupt acceptance		Permitted at 4th cycle

Description:

This instruction increments the Program Counter (PC) three times, increments the Stack Pointer (SP), and then stores the lower byte of the PC in the address in data memory (RAM) that is specified by the SP. This instruction then increments the Stack Pointer (SP) again, and stores the upper byte of the PC in the address in RAM specified by the SP. Finally, this instruction then decrements the PC, adds the data r15 through r0 to the contents of the PC, and transfers the result to the PC.

Example 1:

The value of label LA is 1100H.

			PC	Instruction code	SP	RAM 20H	RAM 21H
	MOV	#01FH,SP	0FF9H	23061FH	1FH	-	-
	CALLR	LA	0FFCH	100201H	21H	FFH	0FH
LA:	INC	ACC	1100H	6300H	21H	FFH	0FH
	RET		1102H	A0H	1FH	FFH	0FH
	NOP		0FFFH	00H	1FH	FFH	0FH

Example 2:

The value of label LA is 1100H.

			PC	Instruction code	SP	RAM 20H	RAM 21H
	MOV	#01FH,SP	0FFCH	23061FH	1FH	-	-
	CALLR	LA	0FFDH	100101H	21H	00H	10H
LA:	INC	ACC	1100H	6300H	21H	00H	10H
	RET		1102H	A0H	1FH	00H	10H
	INC	ACC	1000H	6300H	1FH	00H	10H

RET (Return for subroutine)

Instruction code 1 0 1 0 0 0 0 0 A0H	
Number of bytes	1
Number of cycles	2
Function	(PC15~8) $\leftarrow ((SP)), (SP) \leftarrow (SP) - 1, (PC7~0) \leftarrow ((SP)), (SP) \leftarrow (SP) - 1$
Affected flags	
Interrupt acceptance	Permitted at 2nd cycle

Description:

This instruction transfers the contents of the address in data memory (RAM) that is specified by the Stack Pointer (SP) to the upper byte of the Program Counter (PC). This instruction then decrements the SP, transfers the contents of the address in RAM that is specified by the SP to the lower byte of the Program Counter (PC), and then decrements the SP again.

Example 1:

The value of label LA is 0F0EH.

			PC	Instruction code	SP	RAM 20H	RAM 21H
LA:	MOV	#01FH,SP	0FF9H	23061FH	1FH	-	-
	CALLF	LA	0FFCH	200F0EH	21H	FFH	0FH
	INC	ACC	0F0EH	6300H	21H	FFH	0FH
	RET		0F10H	A0H	1FH	FFH	0FH
	NOP		0FFFH	00H	1FH	FFH	0FH

Example 2:

The value of label LA is 0F0EH.

			PC	Instruction code	SP	RAM 20H	RAM 21H
LA:	MOV	#01FH,SP	0FFAH	23061FH	1FH	-	-
	CALLF	LA	0FFDH	200F0EH	21H	00H	10H
	INC	ACC	0F0EH	6300H	21H	00H	10H
	RET		0F10H	A0H	1FH	00H	10H
	INC	ACC	1000H	6300H	1FH	00H	10H

RETI(Return for interrupt)

Instruction code 1 0 1 1 0 0 0 0 B0H	
Number of bytes	1
Number of cycles	2
Function	(PC15~8) \leftarrow ((SP)), (SP) \leftarrow (SP) - 1, (PC7~0) \leftarrow ((SP)), (SP) \leftarrow (SP) - 1
Affected flags	
Interrupt acceptance	Not permitted

Description:

This instruction transfers the contents of the address in data memory (RAM) that is specified by the Stack Pointer (SP) to the upper byte of the Program Counter (PC). This instruction then decrements the SP, transfers the contents of the address in RAM that is specified by the SP to the lower byte of the Program Counter (PC), decrements the SP again, and then restarts the interrupt acceptance function that was disabled when an interrupt was accepted.

Example 1:

			PC	Instruction code	\leftarrow External interrupt 0 generated
NOP			0FFAH	00H	
NOP			0FFBH	00H	
MOV	#001H,ACC		0FFCH	230001H	
INC	ACC		0003H	6300H	
RET1			0005H	B0H	
NOP			0FFFH	00H	

Example 2:

			PC	Instruction code	\leftarrow External interrupt 1 generated
NOP			0FFCH	00H	
MOV	#00EH,B		0FFDH	23020EH	
INC	ACC		0013H	6300H	
RET1			0015H	B0H	
INC	ACC		1000H	6300H	

8. Bit Manipulation Instructions

CLR1 d9,b3 (Clear direct bit)

Instruction code	1 1 0d8 1b2b1b0 d7d6d5d4d3d2d1d0 C8H~CFH,D8H~DFH
Number of bytes	2
Number of cycles	1
Function	(d9,b3) ← 0
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction resets the bit specified by b2 to b0 of the address in data memory (RAM) or the Special Function Register that is specified by d8 to d0.

Example 1:

		ACC		
MOV	#001H,ACC	01H	0000	0001B
CLR1	ACC,0	00H	0000	0000B

Example 2:

		RAM 7FH		
MOV	#001H,07FH	01H	0000	0001B
CLR1	07FH,0	00H	0000	0000B

Note:

- When this instruction is applied to one of the ports P1 or P3, the port latch of that port is selected; the external signal that is applied to that port is not selected. Furthermore, applying this instruction to port P7 does not change its status.

SET1 d9,b3 (Set direct bit)

Instruction code	1 1 1d8 1b2b1b0 d7d6d5d4d3d2d1d0 E8H~EFH, F8H~FFH
Number of bytes	2
Number of cycles	1
Function	(d9,b3) ← 1
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction sets the bit specified by b2 to b0 of the address in data memory (RAM) or the Special Function Register that is specified by d8 to d0.

Example 1:

		ACC		
MOV	#000H,ACC	00H	0000	0000B
SET1	ACC,7	80H	1000	0000B

Example 2:

		RAM 7FH		
MOV	#001H,07FH	01H	0000	0001B
SET1	07FH,6	41H	0100	0001B

Note:

- When this instruction is applied to one of the ports P1 or P3, the port latch of that port is selected; the external signal that is applied to that port is not selected. Furthermore, applying this instruction to port P7 does not change its status.
-

NOT1 d9,b3 (Not direct bit)

Instruction code	1 0 1d8 1b2b1b0 d7d6d5d4d3d2d1d0 A8H~AFH, B8H~BFH
Number of bytes	2
Number of cycles	1
Function	(d9,b3) \leftarrow (d9,b3)
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction inverts the bit specified by b2 to b0 of the address in data memory (RAM) or the Special Function Register that is specified by d8 to d0.

Example 1:

		ACC		
MOV	#000H,ACC	00H	0000	0000B
NOT1	ACC,7	80H	1000	0000B
NOT1	ACC,7	00H	0000	0000B

Example 2:

		RAM 7FH		
MOV	#001H,07FH	01H	0000	0001B
NOT1	07FH,6	41H	0100	0001B
NOT1	07FH,6	01H	0000	0001B

Note:

- When this instruction is applied to one of the ports P1 or P3, the port latch of that port is selected; the external signal that is applied to that port is not selected. Furthermore, applying this instruction to port P7 does not change its status.

9. Miscellaneous Instruction

NOP (No operation)

Instruction code	0 0 0 0 0 0 0 0 0 H
Number of bytes	1
Number of cycles	1
Function	
Affected flags	
Interrupt acceptance	Permitted

Description:

This instruction consumes one machine cycle.

10. Macro Instruction

CHANGE label name (or address)

Description:

This is POTATO's own macro instruction.

(1) When executed in internal program mode

- Switches from internal program mode → external program mode
- The program counter is set to the external program address specified by the label or address.

(2) When executed in external program mode

- Switches from external program mode → internal program mode (when LDCEXT = 0)
- The program counter is set to the internal program address specified by the label or address.

(Note:Even if the CHANGE instruction is executed in external program mode when LDCEXT = 1, the system does not enter internal program mode. In actuality, it jumps to the address that was specified by the CHANGE instruction in the external program.)

(3) The program mode switch is made after executing any special macro instructions.

(4) Interrupts are not accepted while this macro is executing.

Visual Memory Unit (VMU) Programming Manual

1. Environment Variables

1. Environment Variables for the L86K Series

The L86K series development support tools use the environment variables described below.

PATH: This variable defines the search path. This is defined by a format that is added to the previously defined PATH.

Name	File that is searched for
M86K	Searches for the reserved word file m86krsvd.rwd in the directory defined by PATH.
L86K	Searches for the reserved word definition symbol file lc86k.lib in the directory defined by PATH.
CGR86K	Searches for the default character generator data file (DEFAULT.CDF for the LC864000 Series, DEFAULT.GGR in all other cases) in the directory defined by PATH.

CHIPNAME: This variable defines the name of the chip (or series) that is the target of processing.

Name	Description
M86K	Defines the name of the chip that is the target of the assembly operation. Any CHIP pseudo instructions (refer to Part 2, "Assembler," Chapter 6, "Pseudo Instructions," under the item "CHIP Pseudo Instructions") that are written in the source program are ignored. This environment variable is referenced when assembling a source program that contains no chip pseudo instructions. If the ROM size field in the chip name (the last two digits in the chip name) is "00," the ROM size is not checked, and the assembly operation proceeds on the assumption that there are 64K of ROM.
SU86K	Defines the name of the chip for which option data is to be created. The ROM size field in the chip name (the last two digits in the chip name) is ignored.
CGR86K	Defines the name of the chip for which a character generator data file is to be created. The ROM size field in the chip name (the last two digits in the chip name) is ignored.

When the last two digits in the chip name for each series are "00," that name does not exist as an actual chip. Such chip names are defined for convenience sake as having the largest ROM size in their respective series. Using such a chip name for assembly is valuable when a user wants to know the size of a user-created program.

M86KRSVDFILE: This variable defines the name of the reserved word file.

Name	File that is searched for
M86K	Defines the file name and the directory where the reserved word file is stored. If this environment variable is not defined, m86krsvd.rwd is used as the default file name, and m86krsvd.rwd is searched for in the sequence described in PATH.

M86KWORKFILE: This variable defines the work file name.

Name	File that is searched for
M86K	If the work memory that is dynamically allocated by M86K while the assembly operation is in progress is too large to fit in main memory, or if there is no EMS memory, or if all of the EMS memory has been used, this variable specifies the name of a work file that can be used as a type of extended memory. The name can be specified with a drive name and a path name (for the MS-DOS version), or a path name (for the UNIX version).

TMP: This variable defines the directory where the work file is stored.

Name	File that is searched for
M86K	If it is necessary for M86K to create a work file (refer to the item, "M86KWORKFILE") and the environment variable M86KWORKFILE is not defined, a work file is created in the directory specified by this environment variable. If this environment variable is not defined, the work file is created in the current directory. In all of these cases, however, the file name is fixed to m86kwork.tmp.

1.1 Setting the Environment Variables (MS-DOS Version)

The SET command is used to set the environment variables in MS-DOS. For details on the SET command, refer to the MS-DOS manual.

Example: Setting the default chip name to LC866200

```
A> SET CHIPNAME=LC866200
```

1.2 Setting the Environment Variables (UNIX Version)

The setenv command is used to set the environment variables in UNIX. For details on the setenv command, refer to the UNIX manual.

Example: Setting the default chip name to LC866200 host% setenv CHIPNAME LC866200

Note: UNIX versions are provided for the following tools only: M86K (assembler), L86K (linkage loader) and LIB86K (library manager).

2. File Specification for the Assembler

There are two methods for starting up M86K and passing the necessary data to M86K.

- 1) Passing all of the information to M86K through the command line
- 2) Passing all of the information in response to the prompts that are displayed by M86K

Regardless of the method that was used to start up M86K, it can be forcibly terminated by either pressing CTRL+C (by holding down the CTRL key while pressing the C key) or pressing the STOP key.

1. File Name Specification

1.1 MS-DOS Version File Specification

Upper-case and lower-case letters can be used in any combination in a file name that is specified in the command line when starting up M86K, or in a file name that is given in response to the M86K prompts. For example, the following three file names are all equivalent:

```
sample.asm
SAmpLe.ASM
SAMPLE.asm
```

In addition, when a file name is specified with no extension, M86K uses the following default file name extensions.

File format	Default extension
Source file	.ASM
Object file	.OBJ
List file	.LST
Cross-reference file	.CRF
Error file	.ERR

1.2 UNIX Version File Specification

A distinction is made between upper-case and lower-case letters used in a file name that is specified in the command line when starting up M86K, or in a file name that is given in response to the M86K command prompts. For example, the following three file names are all different:

```
sample.asm  
SAmple.ASM  
SAMPLE.asM
```

In addition, when a file name is specified with no extension, M86K uses the following default file name extensions.

File format	Default extension
Source file	.asm
Object file	.obj
List file	.lst
Cross-reference file	.crf
Error file	.err

2. Specifying Parameters through the Command Line

M86K	[option]	[source],	[object],	[list],	[cross],	[error]
------	----------	-----------	-----------	---------	----------	---------

1) option field

Specify the assembler options that are described in Chapter 2. When specifying options, they must be specified ahead of all of the other fields.

2) source field

Specify the name of the source file that is to be assembled. If the file name extension is omitted from the specification, the default extension ".ASM" is assumed when the file is searched for. If the file name is specified with an extension, that extension is given priority. In either case, the drive name and path name (in the case of the MS-DOS version) or the path name (in the case of the UNIX version) can also be specified.

3) object field

Specify the name of the object file that is to be produced as a result of the assembly operation. The drive name and path name (in the case of the MS-DOS version) or the path name (in the case of the UNIX version) can also be specified. If this entire file name is omitted, the source file name is used, except that the file extension is changed to ".OBJ". If a file with the same file name already exists, the existing file is overwritten.

4) list field

Specify the name of the file to which the assembly results listing is output. The drive name and path name (in the case of the MS-DOS version) or the path name (in the case of the UNIX version) can also be specified. If this entire file name is omitted, no list file is created. If a file with the same file name already exists, the existing file is overwritten.

5) cross field

Specify the file name of the cross-reference list for symbols in the source file that was the target of the assembly operation. The drive name and path name (in the case of the MS-DOS version) or the path name (in the case of the UNIX version) can also be specified. If this entire file name is omitted, no symbol cross-reference list file is created. If a file with the same file name already exists, the existing file is overwritten.

6) error field

Specify the name of the file in which the error messages that were detected as a result of the assembly operation are to be stored. The drive name and path name (in the case of the MS-DOS version) or the path name (in the case of the UNIX version) can also be specified. If this entire file name is omitted, no error file is created. If a file with the same file name already exists, the existing file is overwritten.

Example:

```
A> M86K MAIN.ASM,MAIN, ,TEST.CXXØ
```

The assembly operation starts, using the file MAIN.ASM (which resides in the current directory) as the source file. The object file is written to MAIN.OBJ, no list is generated, and the cross-reference list is written in TEST.CXX.

3. Specifying Parameters in Response to Prompts

When starting up the assembler, input the command without specifying a file name. Afterwards, input each of the file names in response to the prompts that are output by the assembler.

```
prompt M86K [option]Ø
SANYO (R) LC86K series Macro Assembler Version X.XX
Copyright (c) SANYO Electric Co., Ltd. 1989-1995. All rights reserved.
Source filename[.ASM]:
Object filename[.OBJ]:
Source listing [NUL.LST]:
Cross reference[NUL.CRF]:
Error messages [NUL.ERR]:
```

1) option field

Specify the assembler options described in Chapter 2.

2) Source filename

Specify the name of the source file that is to be assembled. If the file name extension is omitted from the specification, the default extension ".ASM" (".asm" for UNIX) is assumed when the file is searched for. If the file name is specified with an extension, that extension is given priority. In either case, the drive name and path name (in the case of the MS-DOS version) or the path name (in the case of the UNIX version) can also be specified. This file name can not be omitted. If only the Return key is pressed, the assembler will prompt the user to input the name of the source file again. In order to interrupt the input operation, either press CTRL+C (by holding down the CTRL key while pressing the C key) or press the STOP key; doing so will terminate M86K.

3) Object Filename

Specify the name of the object file that is to be produced as a result of the assembly operation. The drive name and path name (in the case of the MS-DOS version) or the path name (in the case of the UNIX version) can also be specified. If this file name is omitted (i.e., only the Return key is pressed), the source file name is used, except that the file extension is changed to ".OBJ" (".obj" in the case of UNIX). If a file with the same file name already exists, the existing file is overwritten.

4) List Filename

Specify the name of the file to which the assembly results listing is output. The drive name and path name (in the case of the MS-DOS version) or the path name (in the case of the UNIX version) can also be specified. If this file name is omitted (i.e., only the Return key is pressed), no list file is created. If a file with the same file name already exists, the existing file is overwritten.

5) Cross reference

Specify the file name of the cross-reference list for symbols in the source file that was the target of the assembly operation. The drive name and path name (in the case of the MS-DOS version) or the path name (in the case of the UNIX version) can also be specified. If this entire file name is omitted (i.e., only the Return key is pressed), no symbol cross-reference list file is created. If a file with the same file name already exists, the existing file is overwritten.

6) Error messages

Specify the name of the file in which the error messages that were detected as a result of the assembly operation are to be stored. The drive name and path name (in the case of the MS-DOS version) or the path name (in the case of the UNIX version) can also be specified. If this entire file name is omitted (i.e., only the Return key is pressed), no error file is created. If a file with the same file name already exists, the existing file is overwritten.

3. Assembler Option Specification

This chapter explains how to use the assembler options to specify and control the operation of M86K. In the MS-DOS version, all options begin with the assembler option characters "-" or "/", and in the UNIX version all options begin with the assembler option character "-". In either case, no distinction is made between upper- and lower-case letters for the option specification letter. For example, "-I" and "-i" are interpreted as having the same meaning.

1. Specification for Upper- & Lower-case Letters in Identifiers

Option

-I

If this switch is specified, the assembler makes no distinction between upper-case and lower-case letters in user-defined identifiers (labels, macro names, symbols). If this switch is not specified, a distinction is made between the upper- and lower-case forms of each letter. the effect of this switch is limited to user-defined identifiers, and does not apply to mnemonics or SFRs.

2. Specification for Outputting Debugging Information

Option

-D

If this switch is specified, the assembler does not output the symbol information and source line information in the object file. When debugging an object file that lacks this information, source line mode cannot be used. If this switch is not specified, both types of information are output in the object file, and source line mode can be used for debugging.

3. Specification for Not Optimizing Branching Instructions

Option

-J

This switch can be specified when assembling source code that includes pseudo instructions that require optimization (JMPO, CALLO, BRO); specifying this switch suppresses the optimization operation. As a result, all pseudo instructions are interpreted as 3-byte instructions regardless of the jump destination. If this switch is not specified when assembling source code that includes pseudo instructions that require optimization, the optimization operation is performed. If there are no pseudo instructions that require optimization, the operation of the assembler is the same, whether this switch is specified or not.

4. Specification for Suppressing the Copyright Notice

Option

-N

If this switch is specified, the copyright notice, etc., is not displayed when the assembler is started up. This switch is used to keep the display screen "clean" by suppressing all unnecessary display information, aside from error messages, when starting up the assembler from a utility such as "make".

5. Reserved Word File Specification

Option

-R

The character string that starts with the first character that follows this switch and ends with the last character before the first subsequent space character that is encountered is indicated to the assembler as the name of the reserved word file. For example, assume that the following specification is made:

```
m86k -rm86krsvd.rwd source.asm,,source.lst
```

In this case, the name of the reserved word file is m86krsvd.rwd. This specification takes priority over the environment variable M86KRSVDFILE.

6. Work Buffer Size Specification

Option

-P

If a numeric value is specified after this switch, that value is adopted as the size of the assembler's internal work buffer. The work buffer is an area that is used in order to increase processing speed when the assembler is registering and expanding macros, and is allocated in main memory when the assembler is started up. The default size is 4096 bytes. This is not likely to prove to be inadequate in the case of a typical source program. However, if the buffer size is too small, the assembler displays the following error message and interrupts processing:

no more PARAMETER buffer (123) 45

(The two digits at the end of the message are internal information, and may vary.) If this type of message is displayed, use this switch to specify a larger buffer size and then repeat the assembly process. For example, if the following is specified:

m86k -p8192 source.asm

Then the work buffer size will be 8192 bytes. Only a decimal value can be specified, and must be specified directly after the switch character "P" with no intervening space. Furthermore, if only the switch is specified, with no valid number, the buffer size is unchanged and remains at its default of 4096.

7. Option List Display

Option

-?

If this switch is specified, the assembler displays the following list of options that can be used, and then terminates execution. Note that if this switch is specified, execution terminates, regardless of what other options were specified.

Usage: m86k [option] source,[object],[list],[xref]

option:

/D do not make local symbol table and source line attributes in object file

/I ignore case for user defined symbol

/J do not try to optimize

/N skip displaying copyright message

/Psize parameter buffer size in decimal

/Rfile read 'file' as reserved word file

4. Environment Variables and the Reserved Word File

1. Environment Variables

M86K references the following environment variables when necessary:

PATH	: This variable is used as the search path for the reserved word file. for details on the reserved word file and the search algorithm, refer to section 3.2 in this chapter.
CHIPNAME	: This variable defines the name of the chip that is the target of the assembly operation. Any CHIP pseudo instructions that are written in the source program are ignored. (However, if a chip name that is specified in a CHIP pseudo instruction does not match this variable, a warning message is generated.) This environment variable is referenced when assembling a source program that contains no chip pseudo instructions.
M86KRSVDFILE	: Defines names of the directory in which the reserved words file is stored and the name of that file. The file specified by this environment variable does not have a default file name extension. Be sure to specify both file name and extension, and if necessary also the drive name and path name (for the MS-DOS version) or path name (for the UNIX version).
M86KWORKFILE	: If the work memory that is dynamically allocated by M86K while the assembly operation is in progress is too large to fit in main memory, or if there is no EMS memory, or if all of the EMS memory has been used, this variable specifies the name of a work file that can be used as a type of extended memory. The name can be specified with a drive name and a path name (for the MS-DOS version), or a path name (for the UNIX version).
TMP	: If it is necessary for M86K to create a work file (refer to the item, M86KWORKFILE) and the environment variable M86KWORKFILE is not defined, a work file is created in the directory specified by this environment variable. If this environment variable is not defined, the work file is created in the current directory. In all of these cases, however, the file name is fixed to m86kwork.tmp.

1.1 Setting the Environment Variables (MS-DOS Version)

The SET command is used to set the environment variables in MS-DOS. For details on the SET command, refer to the MS-DOS manual.

Example: Setting the default chip name to LC866200

```
A> SET CHIPNAME=LC866200
```

1.2 Setting the Environment Variables (UNIX Version)

The setenv command is used to set the environment variables in UNIX. For details on the setenv command, refer to the UNIX manual.

Example: Setting the default chip name to LC866200

```
host% setenv CHIPNAME LC866200
```

2. Reserved Word File

The reserved word file is a file that is always loaded by M86K upon startup, and contains various items of information concerning the chip that is the target of the assembly operation (size of RAM/ROM, SFR mnemonics, etc.). M86K will not operate correctly without this file. When M86K is started up, it searches for the reserved word file according to the following procedure:

- 1) If the file name is explicitly specified through assembler option -R, that file is loaded. If that file does not exist, or if it is not loadable, an error results.
- 2) If the environment variable M86KRSVDFILE has been defined, the file specified by that variable is loaded. If that file does not exist, or if it is not loadable, an error results.
- 3) If there is a file named m86krsvd.rwd in the directory where M86K.EXE is located, and that file is loadable, that file is loaded.
- 4) If there is a file named m86krsvd.rwd in the current directory, and that file is loadable, that file is loaded.
- 5) M86K searches sequentially through the directory specified in the environment variable PATH, and loads the first loadable file named m86krsvd.rwd that it finds.

If searching according to the sequence described above still fails to find the reserved word file, an error is generated and M86K stops executing. Normally, the reserved word file is stored in the same directory where M86K.EXE resides. Note that the contents of the reserved word file are essential to the normal operation of M86K, and we cannot bear responsibility for any problems that arise in the operation of M86K resulting from the deletion or modification of the contents of the reserved word file. Therefore, we strongly recommended that the write-protect feature for this file be enabled.

5. Source File Input Format

A source file consists of character strings of up to 511 characters per line (including a CR or LF code at the end of each line). In addition, no distinction is made between upper- and lower-case letters, except in symbols (such as labels and macro names) that are defined in the source program. For example, both "Nop" and "nop" are recognized as the mnemonic for the NOP instruction. Furthermore, the distinction made between upper- and lower-case characters in symbols such as labels can also be disabled by specifying the assembler option "-I".

1. Statements

Statements consist of a combination of mnemonics (which define the object code that is to be produced during the assembly operation), operands and comments. One line of source code is equivalent to one mnemonic. Multi-line statements are not supported. Each statement consists of the four fields described below.

```
[label:] [operation] [operand] [;comment]
```

Field	Purpose
label	A label is applied to a statement so that other statements can use that label to access the statement. Labels must always be delimited with a colon (":").
operation	Specifies the operation of the statement.
operand	Defines the data that is the target of the operation of the statement.
comment	Describes the statement; has no effect on the assembly operation.

Note: Data that is enclosed in brackets may be omitted.

2. Label Names and Symbol Names

Label names and symbol names are character strings of any length (but of at least one character). However, only the first 32 characters of a label name or symbol name are valid. The following characters may be used in a label or symbol name:

A~Z, a~z, 0~9

\$

?

-

@

.

In addition, the first character in a label or symbol name must be a letter, "_", ".", or "@". If the assembler option "-i" has not been specified, a distinction is made between upper- and lower-case letters. Furthermore, label names must be delimited with a colon (":").

3. Comments

Comments begin with a semi-colon (";") and end with a line feed.

4. Operators

The following chart lists the operators that can be used in M86K and their priority. No distinction is made between upper- and lower-case characters in operators that consist only of letters, such as NOT. NOT and not are recognized as the same operator.

Operator	Description	Priority ranking
NOT	1's complement	1
HIGH	Upper byte	
LOW	Lower byte	
*	Multiplication	2
/	Division	
MOD	Modulo	
+	Addition	3
-	Subtraction	
SHR	Right shift	4
SHL	Left shift	
LAND	Logical product	5
LOR	Logical sum	
LXOR	Exclusive logical sum	
EQ	Equal to	6
NE	Not equal to	
LT	Less than	
LE	Less than or equal to	
GT	Greater than	
GE	Greater than or equal to	

5. Numeric constants

M86K permits description of numeric constants in four bases: binary, octal, decimal, and hexadecimal. There are two formats for writing numeric constants: one in which the base is explicitly indicated ("123H"), and one in which the pseudo instruction RADIX is used to specify the default base beforehand. Values that are written with an explicitly specified base are converted to the default base. A numeric constant for which no base is explicitly specified, such as "123," is converted to the base that was specified by the pseudo instruction RADIX. If the pseudo instruction RADIX has not been used to specify a base, the default base is "decimal."

No matter which method is used to write constants, the assembler processes the values internally in 32-bit format. In addition, when a numeric constant or the value of an expression that is ultimately returned to a numeric constant is written as immediate data for an operand, only the bits that are necessary for that operand are stored; the remaining upper bits are discarded.

Table 6.1 *Formats for numeric constants with an explicitly specified base*

Base	Format	Examples
2	Percent sign ("%"), followed by one or more digits 0 and 1	%01111011 %11111111 %0000010000000000
	One or more digits 0 and 1, followed by "B"	*01111011B 11111111B 0000010000000000B
	One or more digits 0 and 1, followed by ".B"	01111011.B 11111111.B 0000010000000000.B
8	One or more digits 0 through 7, followed by ".O"	273.O 377.O 2000.O
10	One or more digits 0 through 9, followed by ".D"	123.D 255.D 1024.D
16	Dollar sign ("\$"), followed by one or more digits 0 through 9, "a" through "f", or "A" through "F"	\$7B \$FF \$0400
	An initial digit 0 through 9, which may be followed by digits 0 through 9, "a" through "f", or "A" through "F", followed by "H"	7BH OFFH 0400H
	An initial digit 0 through 9, which may be followed by digits 0 through 9, "a" through "f", or "A" through "F", followed by ".H"	7B.H OFF.H 0400.H

*No distinction is made between upper- and lower-case characters that are used to specify the base ("B", "O", "D", and "H"). This is not affected by the assembler option "-i".

*This format is affected by the RADIX setting. For details, refer to the next table.

Table 6.2 *Interpretation of numeric constants in a format where the base is not explicitly specified*

Format	Example	Value in base specified by RADIX			
		2	8	10	16
One or more digits 0 and 1	0101	510	6510	10110	25710
One or more digits 0 through 7	123	error	8310	12310	29110
One or more digits 0 through 9	789	error	error	789110	192910
One or more digits 0 and 1, followed by "B"	"101B"	510	510	510	412310
An initial digit 0 through 9, which may be followed by digits 0 through 9, "a" through "f", or "A" through "F"	"OFF"	error	error	error	25510

6. Character Constants

Characters enclosed in single quotes (') are handled as character constants. Character constants are a type of constant, the value of which is the ASCII code of the character that is specified. In addition to being able to write all printable ASCII characters, other codes may also be written in the following format. Note that when more than one character is enclosed in quotes, it is handled as a "character string constant" (see section 4.7), not a "character constant."

Table 6.3 *Format for inputting codes within character constants and character string constants*

Format	Code (hexadecimal)	Remarks
\n	0A	Line feed
\r	0D	Return
\t	09	Horizontal tab
\b	08	Back space
\f	0C	Paper feed
\	"22	Double quotes
\	'27	Single quotes
\\	5C	Yen mark or backslash
\ooo		"ooo" represents an octal number of up to three digits
\xhh		"hh" represents a hexadecimal number of up to two digits

Example 1: ADD# 'A'

Example 2: DB 'A', '\012', 'C'

Example 3: DB " Regarded as a character string constant, and generates an error as an operand for DB.

7. Character String Constant

A character string consisting of one or more characters that is enclosed in double quotes (") or a character string consisting of two or more characters that is enclosed in single quotes (') is handled as a character string constant. A character string constant can be written as an operand for the pseudo instruction DC or .PRINTX. In addition to being able to write all printable ASCII codes in a character string, it is also possible to write other codes using the format described in section 4.6.

Example: DC "This is a sample string with special codes \007\r\n"

8. Special Symbols

When an asterisk is used as an operand, it represents the value of the described location.

Example 1: To indicate the value that was six bytes ahead of the location where the instruction is written:

BR *-

Example 2: To indicate the value that was twelve bytes after the location where the instruction is written:

Example 2: To indicate the value that was twelve bytes after the location where the instruction is written:

BR *+12

6. Errors

M86K detects three levels of errors: fatal errors, errors, and warnings. When a fatal error is detected, M86K immediately halts execution at that point. This level corresponds to problems such as "the work buffer is too small, etc." If an error is detected, M86K halts execution once the pass (pass 1 or pass 2) that was being executed at the moment when the error was detected is completed. This level corresponds to syntax errors, for example. If a warning is detected, the M86K does not halt execution, because a warning corresponds to minor problems such as an operand being out of range.

When a fatal error is detected, M86K does not generate all files that are indicated for output. If an error is detected in pass 1, M86K does not generate all files that are indicated for output; if an error is detected in pass 2, however, the list file only is generated (if the specification for generating the list file was made). The error display format is shown below.

```
filename(linenum): source line  
error message
```

Example: sample.asm(54): LD xyz

xyz: undefine symbol

The symbol "xyz" is undefined.

1. Warnings

The meanings of the warning level messages that are generated by M86K are described below. Note that "???" in these messages indicates a variable portion of the message.

???: bit number exceeds limits

In a bit manipulation instruction, the specified bit was outside of the allowable range.

absolute expression expected

An expression in which the value is finalized at the point of assembly is required.

address beyond zero

A negative value was specified for the operand of an ORG instruction.

address exceeds limits

The value that was specified for the operand of an ORG instruction exceeded the size of ROM.

address exceeds ROM size

The address of an assembled instruction exceeded the size of ROM.

chip name is different from one specified by CHIPNAME (???).

The operand of the CHIP instruction differs from that which was specified in the environment variable.

END in included file

The END pseudo instruction was found within a source file that was specified by the INCLUDE pseudo instruction.

ENDF without FUNCTION

ENDF was found even though no function was being defined.

ENDM without MACRO

ENDM was found even though no macro was being defined.

EXITM outside MACRO

EXITM was found even though no macro was being defined.

function code buffer overflow

The contents of the function definition were too large for the buffer.

illegal combination of attributes:???

The attributes (bank and segment) of both elements of a two-element operator do not agree.

illegal style expression

SET or EQU operands had an invalid format.

JMP/CALL placed at the end of memory block (FREE)

A JMP or CALL instruction appeared in which the lower 12-bits of the address were 0FFEh or 0FFFh. Because the segment placement mode is "FREE," there may be no problem, depending on the results of the link, but an error will be generated by the linker if the segment in question was placed at the start of a memory boundary.

Jump address is out of range (FREE)

The jump destination address is outside of the memory boundary. Because the segment placement mode is "FREE," there may be no problem, depending on the results of the link, but an error will be generated by the linker if the segment in question was placed at the start of a memory boundary.

LOCAL outside MACRO

LOCAL was found even though no macro was being defined.

macro name in expression

A symbol that has been registered as a macro was found in an expression.

macro name required

No macro name was found, even though a macro was being defined.

no character in string

No characters were found in a character string constant.

page width must be 72 ~ 132: ???

The operand of the WIDTH instruction must be between 72 and 132 (inclusive).

public ??? not defined

The value of a symbol declared in a PUBLIC pseudo instruction has not been defined.

SET conflicts with PUBLIC

An attempt was made to reset a value that was already set (by the SET instruction) for a symbol declared in a PUBLIC pseudo instruction.

symbol name required

No symbol was found in the operand for PUBLIC, EXTERN, or OTHER_SIDE_SYMBOL.

undefined symbol in expression

An undefined symbol was found in an expression. (This warning is detected in pass 2 only.)

value is out of range

The value is outside of the allowable range. (The "allowable range" varies for each operand.)

zero divide: ??? modulo 0

The right side of the MOD operator is "0."

zero divided: ??? / 0

The right side of the "/" operator is "0."

2. Errors

The meanings of the error level messages that are generated by M86K are described below. Note that ??? in these messages indicates a variable portion of the message.

???: 2, 8, 10 or 16 required

Only "2," "8," "10," or "16" can be specified as the operand for the pseudo instruction RADIX.

???: constant required

No numeric constant was found.

???: duplicated label

A duplicate label was found.

???: duplicated symbol

A duplicate symbol was found.

???: illegal character in numeric constant

An invalid character was found within a numeric constant.

???: no such chip in the table

The symbol that was specified by the CHIP instruction was not found in the reserved word file.

???: open error

An error was detected when a file was opened.

???: undefined symbol

An undefined symbol was referenced.

???: radix violation

Characters that are not valid for the specified base were found in a numeric constant.

???H,??? :out of internal RAM area

The data segment address allocation exceeded the allowable range.

` not seen

No single quote (') was found on the right end of a character constant.

`:` not seen

In the case of a format that explicitly specifies the segment in the EXTERN operand, no colon (:) was found that delimits the segment from the symbol.

0x??? : RAM address exceeds limits

The data segment address allocation was outside of the allowable range.

address duplicated

A duplicate address area in RAM was specified in the DS pseudo instruction.

address exceeds absolute limits

The address of an assembled instruction exceeded 65535.

bank number should be 0~15

The bank number must be within the range from 0 to 15.

Branch address beyond zero

A value smaller than address 0 (the start of the code segment in question) was specified as the branch destination address.

Branch address exceeds limits

The branch destination address exceeded the size of ROM.

CSEG conflicts with WORLD_EXTERNAL_DATA

WORLD_EXTERNAL_DATA and a pseudo instruction that specifies a segment cannot both be written within the same source file.

CSEG isn't allowed in macro

A pseudo instruction that specifies a segment cannot be written within a macro definition.

DS must be in DSEG

The DS pseudo instruction can only specify a data segment.

DSEG conflicts with WORLD_EXTERNAL_DATA

WORLD_EXTERNAL_DATA and a pseudo instruction that specifies a segment cannot both be written within the same source file.

DSEG isn't allowed in macro

A pseudo instruction that specifies a segment cannot be written within a macro definition.

ELSE without IFxxx

IFxxx corresponding to the pseudo instruction ELSE for conditional assembly was not found.

ENDF not seen

The ENDF pseudo instruction that declares the end of a function definition was not found.

ENDIF without IFxxx

IFxxx corresponding to the pseudo instruction ENDF for conditional assembly was not found.

ENDM not seen

The ENDM pseudo instruction that declares the end of a macro definition was not found.

external symbol can't be public

An external symbol was declared in a PUBLIC pseudo instruction.

Hardware configuration violation

The instruction (such as the CHANGE instruction) in question has not been implemented for the specified chip.

identifier expected

Something other than an identifier was found in a macro definition parameter list or in an EXTERN operand.

illegal character in ??? constant

illegal character in binary constant

An invalid character for the specified base was found within a numeric constant.

illegal symbol type

Declared of a symbol of an in valid type was attempted in a PUBLIC pseudo.

illegal word in external list

A syntax error was found in an EXTERN operand.

instructions can't be in DSEG

An instruction other than DS was found in a data segment.

JMP/CALL placed at the end of memory block (INBLOCK)

A JMP or CALL instruction appeared in which the lower 12-bits of the address were 0FFEh or 0FFFh. Because the segment placement mode is "INBLOCK," an error resulted.

Jump address beyond zero

A value smaller than address 0 (the start of the code segment in question) was specified as the jump destination address.

Jump address exceeds limits

The jump destination address exceeded the size of ROM.

Jump address is out of range (INBLOCK)

The jump destination address is outside of the memory boundary. Because the segment placement mode is "INBLOCK," a linker error resulted.

local symbol can't be public

A local symbol was declared in a PUBLIC pseudo instruction.

lost SET symbol

A symbol that was defined by the SET pseudo instruction was lost from in pass 2. This is possibly due to an internal error in the assembler.

macro can't be public

A macro was declared in a PUBLIC pseudo instruction.

maximum nesting of macro is 10

The maximum nesting level for macros is 10.

Multiple WORLD specified

Multiple WORLD pseudo instructions were written in the same source file.

name required for macro

No name was found in a macro definition.

no room for source line attribute object

There is insufficient memory to store source line attributes (information for debugging).

no value for EXT

Although the CHANGE instruction is being used, the register EXT was not found in the SFRs.

not the symbol defined by SET

An attempt was made to reset (with the SET command) a value for a symbol that was not the one that was defined by SET.

operand exceeds limits

The number of repetitions specified by the REPT macro pseudo instruction was not within the range from 1 to 65,535.

ORG isn't allowed in macro

The ORG pseudo instruction cannot be written within a macro.

other-side symbol isn't allowed

other-side symbol isn't allowed here

A symbol declared with OTHER_SIDE_SYMBOL cannot be specified here.

other-side symbol or absolute constant is required

A symbol declared with OTHER_SIDE_SYMBOL or a constant is required.

positive value required

A negative value cannot be used.

public ??? not defined

There was no definition of a symbol declared in a PUBLIC pseudo instruction. (This problem generates a "warning" for a symbol that is only declared in a PUBLIC pseudo instruction and has no definition of value and is not referenced, and generates an "error" when there is no definition of value but the symbol is referenced.)

string is too long

The length of a character string constant exceeded the limit (255 characters).

symbol name required

No symbol was found on the left side of SET or EQU.

symbol not defined

No symbol was found specified in the operand for PUBLIC, EXTERN, or OTHER_SIDE_SYMBOL. An internal assembler error is possible.

syntax error

A syntax error was found.

syntax error near ???

A syntax error was found in the vicinity of ???.

too complexed expression for an operand

An expression that was written for an operand was too complex and could not be interpreted.

too many CHIP pseudo operation

Multiple CHIP pseudo instructions were written in one source file.

too nested if-statements

Nesting of pseudo instructions for conditional assembly exceeded the limit (10 levels).

unbalanced conditional assembling controllers

unbalanced IF statement

The end of the source file was found while skipping due to conditional assembly.

unexpected end of file in string

The end of the source file was found within a character string constant.

unexpected end of line in string

The end of the line was found within a character string constant.

unexpected EOF in conditional assembling

The end of the source file was found while skipping due to conditional assembly.

unexpected terminator ??? in conditional assembling

The syntax analysis routine ended abnormally while skipping due to conditional assembly. An internal assembler error is possible.

unmatched ELSE in skipping

unmatched ENDIF

The end of the source file was found while skipping due to conditional assembly.

WORLD conflicts xSEG

WORLD EXTERNAL_DATA and a pseudo instruction that specifies a segment cannot both be written within the same source file.

3. Fatal Errors

The meanings of the fatal error level messages that are generated by M86K are described below. Note that ??? in these messages indicates a variable portion of the message.

???(???) : chip name not seen
???(???) : chip name not seen.
???(???) : decimal value required
???(???) : hex-value and reserved-word are required
???(???) : no chip name list

A syntax error was found in the reserved word file.

???(???) : no reserved word seen
???(???) : ROM size not seen
???(???) : too many chip names
???(???) : ??? : unknown chip name
???(???) : ??? : unknown flag

A syntax error was found in the reserved word file.

??? : illegal file name

An invalid character was found in the specified file name.

??? : no such chip in the table

The chip name that was specified by the environment variable CHIPNAME was not found in the reserved word file.

??? : no such user

The user name specified by "~user" was not found. (UNIX version only)

??? : open error

An attempt to open the specified file failed.

??? : unknown flag

An invalid assembler option was specified.

??? : unreadable

The specified file cannot be loaded.

EMM v3.2 or later is required (v???.??? found)

The EMS driver version is old and is no longer supported. Driver version 3.2 or later is required.

EMS allocation (??? pages) was failed

EMS memory allocation failed.

EMS deallocation was failed

An error was detected while opening EMS memory.

flushing error in workfile

An error was detected while flushing the work file. (There is no more free space on disk, etc.)

Getting EMM version was failed

Getting EMS status was failed

Getting free page count on EMS is failed

Getting physical page frame address was failed

During EMS memory initialization, an error was detected, such as during the EMS driver version check, etc.

making temp. name for ??? failed

An error was detected when a temporary name was given to an output file.

Neither CHIP pseudo operation nor CHIPNAME environment variable were defined. Further execution aborted.

No chip specification has been made through the CHIP pseudo instruction or the environment variable CHIPNAME; because no chip can be specified as the target chip for assembly, subsequent operation is halted.

no more MAIN memory (???) ???

Although there is an area remaining that must be allocated in main memory, there is no more space in main memory available for allocation.

no more memory (???)

There is no more memory that can be dynamically allocated (main memory, EMS memory, work files).

no more NODE buffer (???) ???

There is insufficient work area available for analyzing expressions.

no more PARAMETER buffer (???) ???

There is insufficient work area available for processing the parameter list for a macro definition or call.

no reserved word file available.

Reading of the reserved word file failed.

no room for file: ???

A file cannot be written to disk because the disk is full.

Pxxxx must be less than 65536

Specify a parameter buffer size of no more than 65535.

read error in workfile (???)

An error occurred while reading the work file.

removing ??? failed

Because an error was detected, an attempt was made to delete the output files created up to that point, but the attempt failed.

renaming ??? ==> ??? failed

An error was generated when an attempt was made to change the name of an output file that was created with a temporary name a regular name.

This error is generated when a file with the same name as the post-change name already exists, and that file is write-protected.

too many file names

Five or more file names are specified in the command line.

too many nested include files

The number of nested include files exceeded the limit (10 levels).

unlinking work file is failed

An error was detected while deleting a work file.

workfile ??? : already exist

workfile ??? : open error

An error was detected while creating a new work file.

7. Pseudo Instructions

Unlike normal instructions (instructions that indicate operations of the LC86K itself, such as ADD and MOV), pseudo instructions are used to issue instructions and make definitions to the assembler, and no machine language is generated for individual pseudo instructions (except for pseudo instructions that are used for optimization purposes, such as JMPO, and CHANGE pseudo instructions). In most cases, these pseudo instructions are used in combination with normal instructions.

Category	Pseudo instruction	Function
Link control	ORG WORLD CSEG DSEG END PUBLIC EXTERN OTHER_SIDE_SYMBOL	Specify origin Select ROM for code storage Specify code segment Specify data segment End program Specify external definition name Specify external reference name Declare CHANGE instruction jump label
Symbol definition	EQU SET	Assign value Assign temporary value
Data definition	DB DW DC DS	Define byte Define word Define character string Allocate data area (RAM)
Macro control	MACRO REPT IRP IRPC ENDM EXITM LOCAL	Define macro Repeat macro Continuous macro Character string macro End macro definition Interrupt macro expansion Define local label

Category	Pseudo instruction	Function
Conditional assembly	IFDEF IFNDEF IFB IFNB IFE IFNE IFIDN IFDIF ELSE ENDIF .PRINTX .LIST .XLIST .MACRO .XMACRO .IF .XIF	Assemble if defined Assemble if undefined Assemble if operand is empty Assemble if operand is not empty Assemble if value of expression is "0" Assemble if value of expression is not "0" Assemble if two character strings are identical Assemble if two character strings are not identical Assemble in the case of the condition that is the opposite of the above IF condition End conditional assembly Display on VDT during assembly Output list Interrupt list output Output macro expansion Interrupt macro expansion output Output conditional skip Interrupt conditional skip output
Miscellaneous	INCLUDE TITLE PAGE CHIP COMMENT WIDTH BANK CHANGE RADIX	Load file Specify list title End of page Define chip that is target of assembly Output comments to object file Specify number of columns in list file Specify RAM area bank Jump between external and internal ROM Specify default base
Optimization	JMPO BRO CALLO BZO BNZO BPO BPCO BNO DBNZO BEO BNEO	Generate optimal JMP instruction Generate optimal BR instruction Generate optimal CALL instruction Generate BZ instruction that will not generate an address error Generate BNZ instruction that will not generate an address error Generate BP instruction that will not generate an address error Generate BPC instruction that will not generate an address error Generate BN instruction that will not generate an address error Generate DBNZ instruction that will not generate an address error Generate BE instruction that will not generate an address error Generate BNE instruction that will not generate an address error

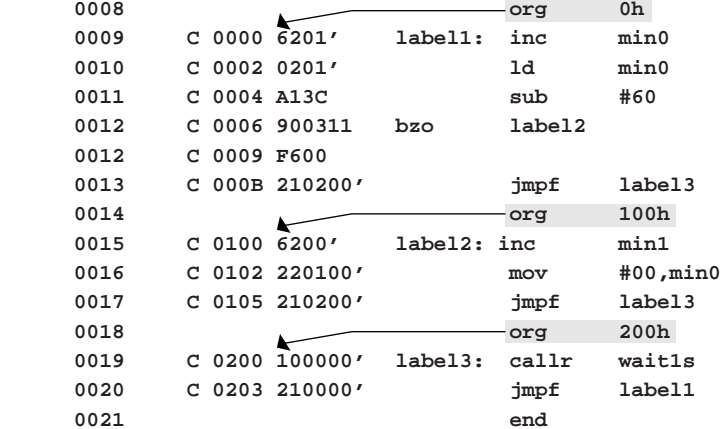
1. ORG (Specify origin)

ORG expression

The ORG pseudo instruction begins the specification of addresses in program memory (ROM) and data memory (RAM) from the value of expression. expression must be either a numeric constant or an expression that has a definite value at the time of assembly.

Example:

```
page:      1 <org.ASM>
ERR SEQ.   S LOC. OBJ.   SOURCE STATEMENTS
0001                               ;a sample program for ORG
      0002                               chip    lc866032
      0003                               extern  wait1s
      0004                               dseg
      0005      D 0000      min1:   ds        1
      0006      D 0001      min0:   ds        1
      0007                               cseg
      0008                               org      0h
      0009      C 0000 6201'   label1: inc      min0
      0010      C 0002 0201'           ld      min0
      0011      C 0004 A13C           sub      #60
      0012      C 0006 900311   bzo      label2
      0012      C 0009 F600
      0013      C 000B 210200'           jmpf    label3
      0014                               org      100h
      0015      C 0100 6200'   label2: inc      min1
      0016      C 0102 220100'          mov     #00,min0
      0017      C 0105 210200'           jmpf    label3
      0018                               org      200h
      0019      C 0200 100000'   label3: callr   wait1s
      0020      C 0203 210000'           jmpf    label1
      0021                               end
```



2. WORLD (Select ROM for code storage)

WORLD selection

This pseudo instruction specifies the ROM where the assembled code should be stored. This pseudo instruction has meaning only when the target chip is of the LC868000 Series. The following three values can be specified for selection:

INTERNAL	:Store the code in the on-chip ROM.
EXTERNAL	:Store the code in the ROM that is connected externally for code storage.
EXTERNAL_DATA	:Store the code in the ROM that is connected externally for data storage.

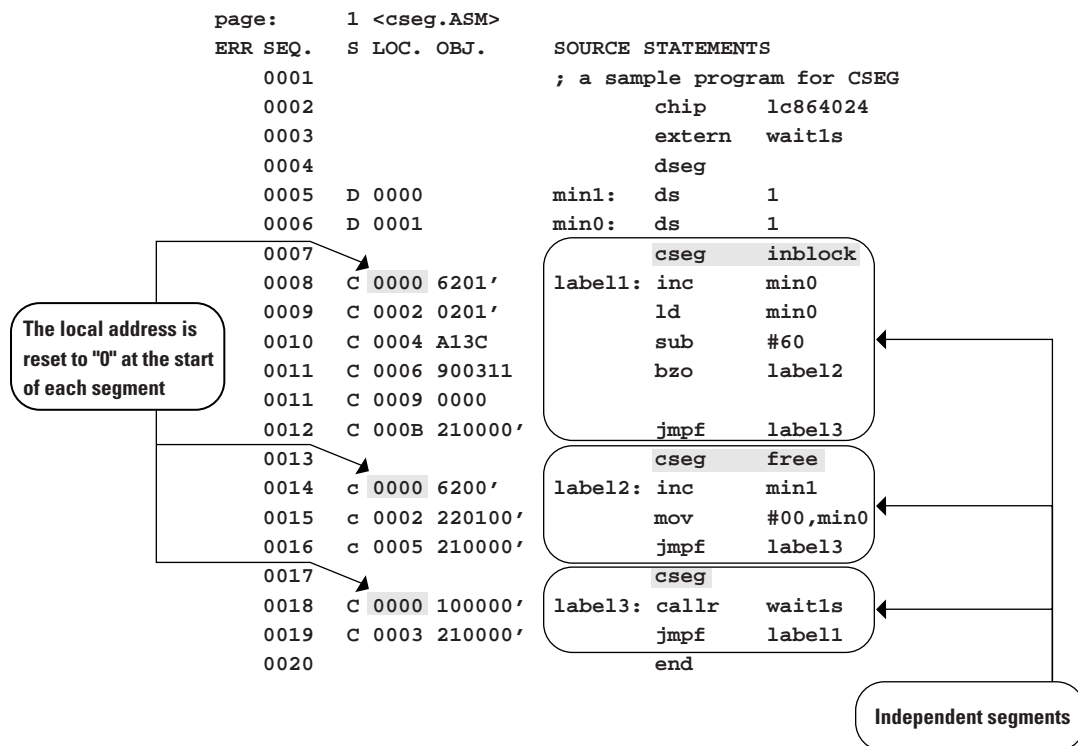
If multiple WORLD pseudo instructions are specified in one file, an error results. If a chip other than one of the LC868000 Series is specified, and a value other than INTERNAL is selected in the WORLD pseudo instruction, an error results.

3. CSEG (Declare start of code segment)

CSEG mode

This pseudo instruction declares to the assembler the start of the segment where the program code is to be stored. If mode is not specified, or if mode is specified as INBLOCK, the start of the segment is located at a 4K boundary. If mode is specified as FREE, the start of the segment has no relation to a 4K boundary.

Example:

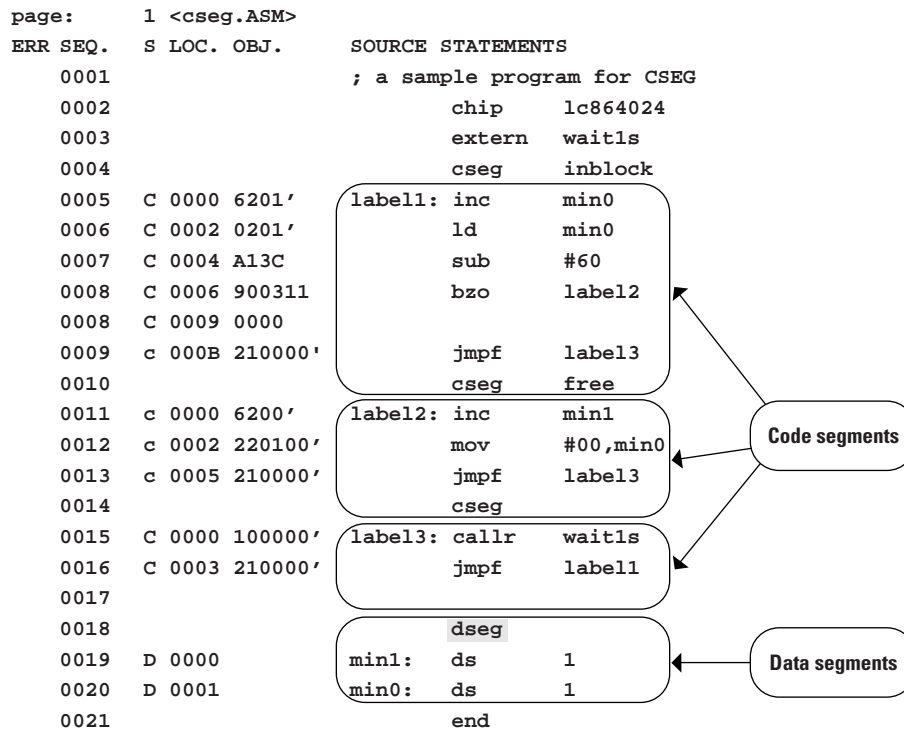


4. DSEG (Declare start of data segment)

DESG

This pseudo instruction declares to the assembler the start of the area in data memory that is to be allocated.

Example:



5. END (End program)

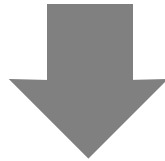
END

This pseudo instruction declares the end of the source program. Because the assembler ends the assembly operation for the pass that is being executed at the moment that this instruction is detected, any statements that follow this instruction are ignored, even if those statements are valid.

Example:

```
; a sample program for END
chip    1c866032
cseg
mov     #20h, 01h
mov     #10h, 00h
ld      00h
add     0fh
end
inc     00h
inc     01h
ld      01h
```

Statements that come after the pseudo instruction END are not assembled.



```
page: 1 <end.ASM>
ERR SEQ. S LOC. OBJ. SOURCE STATEMENTS
0001                      ; a sample program for END
0002                      chip    1c866032
0003                      cseg
0004 C 0000 220120      mov     #20h, 01h
0005 C 0003 220010      mov     #10h, 00h
0006 C 0006 0200        ld      00h
0007 C 0008 820F        add     0fh
0008                      end
```

6. PUBLIC (Specify external definition name)

PUBLIC symbol {, symbol}

The PUBLIC pseudo instruction permits symbol, which is defined in this source program, to be referenced from other source files.

Example:

```
page: 1 <extern.ASM>
ERR SEQ. S LOC. OBJ.      SOURCE STATEMENTS
0001                      ; a sample program for EXTERN
0002                      chip    1c866032
0003                      extern  label1, label2
0004
0005                      cseg     inblock
0006 C 0000 200000'        callf   label1
0007
0008 C 0003 200000'start:  callf   label2
0009 C 0006 0303          ld      c
0010 C 0008 90F9          bnz     start
0011
0012 C 000A A300          sub     a
0013
0014                      end
```

When referencing a symbol for which the value was defined in another source file, it is necessary to declare that symbol beforehand with the EXTERN pseudo instruction.

A symbol that is to be referenced from other source files must be made visible to those source files by declaring them with the PUBLIC pseudo instruction.

```
page: 1 <public.ASM>
ERR SEQ. S LOC. OBJ.      SOURCE STATEMENTS
0001                      ; a sample program for PUBLIC
0002                      chip    1c866032
0003                      public  label1, label2
0004
0005                      cseg     inblock
0006 C 0000 220000'        label1: mov    #00,  data1
0007 C 0003 23033C        mov    #60,   c
0008 C 0006 A0            ret
0009
0010 C 0007 6200'          label2: inc    data1
0011 C 0009 0200'          ld      data1
0012 C 000B 410A05        bne     #10,   label3
0013 C 000E 220000'        mov    #00,   data1
0014 C 0011 6201'          inc     data2
0015
0016 C 0013 7303          label3: dec    c
0017 C 0015 A0            ret
0018
0019                      dseg
0020 D 0000                data1:  ds     1
0021 D 0001                data2:  ds     1
0022
0023                      end
```

The PUBLIC and EXTERN pseudo instructions can be used in combination to permit referencing of symbols that are defined in other source files.

7. EXTERN (Specify external reference name)

EXTERN [segmanet:]symbol {[segment:]symbol}

The EXTERN pseudo instruction is used in order to allow the specified symbol to be referenced from other programs. The segment specification permits specification of a segment within either CSEG or DSEG. If nothing is specified, the code segment CSEG is assumed. For an example of the use of the EXTERN pseudo instruction, refer to the explanation of the PUBLIC pseudo instruction.

8. OTHER_SIDE_SYMBOL (Declare CHANGE instruction jump label)

OTHER SIDE SYMBOL label {,label}

This pseudo instruction declares an address label that is specified as an operand of the CHANGE instruction, which is used to switch between internal ROM and external ROM in the LC868000 Series. Although the label that is declared is a type of external symbol, one difference is that in a source file written with codes that are stored in internal ROM, the label is declared in external ROM (while in the case of a source file written with codes that are stored in external ROM, the label is declared in internal ROM). Note that this pseudo instruction is used only by the LC868000 Series, and generates an error in all other cases. For an example of the use of the OTHER_SIDE_SYMBOL pseudo instruction, refer to the description of the CHANGE pseudo instruction.

9. EQU (Assign value)

symbolname EQU expression

The EQU pseudo instruction assigns the value expression to symbolname. A symbol that has been defined by using the EQU pseudo instruction cannot be defined again. Using the EQU pseudo instruction effectively makes it possible to add visual meaning to constant data, which improves the efficiency with which maintenance work can be performed.

Example:

":" is not described between the symbol for which the value is being defined and "EQU".

When the defined value can be calculated,
that value is shown (in hexadecimal).

Any expression can be described.

```

page: 1 <equ.ASM>
ERR SEQ. S LOC. OBJ.

SOURCE STATEMENTS
; a sample program for EQU
chip lc866032

00000064 loop_max equ 100
00000001 mode_a equ 1
00000002 mode_b equ 2
00000003 mode_c equ 3

0009 csg inblock
0010 C 0000 220000' mov #00, loop_ctr
0011
0012 C 0003 230201 label1: mov #mode_a, b
0013 C 0006 0818' call sub1
0014 C 0008 230202 mov #mode_b, b
0015 C 000B 0818' call sub1
0016 C 000D 230303 mov #mode_c, c
0017 C 0010 6200' inc loop_ctr
0018 C 0012 0200' ld loop_ctr
0019 C 0014 4164EC bne #loop_max, label1
0020 C 0017 A0 ret
0021
0022 C 0018 0302 sub1: ld b
0023 C 001A 310107 be #mode_a, suj10
0024 C 001D 310208 be #mode_b, suj11
0025 C 0020 310309 be #mode_c, suj12
0026 C 0023 A0 suj0: ret
0027
0028 C 0024 1201' suj10: st data_a
0029 C 0026 01FB br suj0
0030 C 0028 1202' suj11: st data_b
0031 C 002A 01F7 br suj0
0032 C 002C 1203' suj12: st data_c
0033 C 002E 01F3 br suj0
0034
0035 dseg
0036 D 0000 loop_ctr: ds 1
0037 D 0001 data_a: ds 1
0038 D 0002 data_b: ds 1
0039 D 0003 data_c: ds 1
0040
0041 end

```

10. SET (Assign temporary value)

symbolname SET expression

The SET pseudo instruction assigns the value expression to symbolname. A symbol that has been defined by using the SET pseudo instruction can be defined again with the SET instruction. A symbol that has been defined by this pseudo instruction cannot be declared in a PUBLIC pseudo instruction or defined again by using the EQU instruction.

Example:

When the defined value can be calculated, that value is shown (in hexadecimal).

```

page:      1 <set.ASM>
ERR SEQ.   S LOC. OBJ.
0001
0002
0003
0004
0005
0006
0007 C 0000 220000'
0008
0009 C 0003 6300
0010 C 0005 6302
0011
0012
0013
0014 C 0007 220101'
0015
0016 C 000A 7300
0017 C 000C 7302
0018
0019
0020 D 0000      zz:
0021
0022

```

":" is not described between the symbol for which the value is being defined and "SET".

SOURCE STATEMENTS
; a sample program for SEY

```

chip      lc866032
cseg      inblock

```

```
00000000 dd set 0
```

```
mov      #dd,zz+dd
```

```
inc      a
```

```
inc      b
```

```
00000001 dd set dd+1
```

```
mov      #dd,zz+dd
```

```
dec      a
```

```
dec      b
```

```
dseg
```

```
ds      2
```

```
end

```

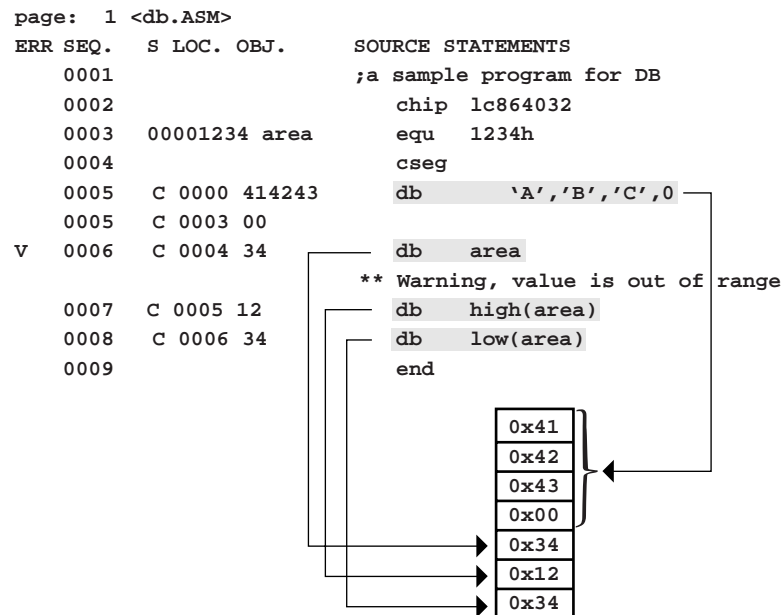
Any expression can be described, including the symbol for which the value is being defined.

11. DB (Define byte data)

labelname DB expression [,expression]

The DB pseudo instruction stores 8-bit data that corresponds to the operand expression in program memory (ROM). More than one operand can be described; delimit each operand with a comma (","). When there are two or more operands, they are evaluated in order from left to right, and are stored in sequence in ascending addresses. If no operand is described between two commas, that operand is regarded as being "0".

Example:



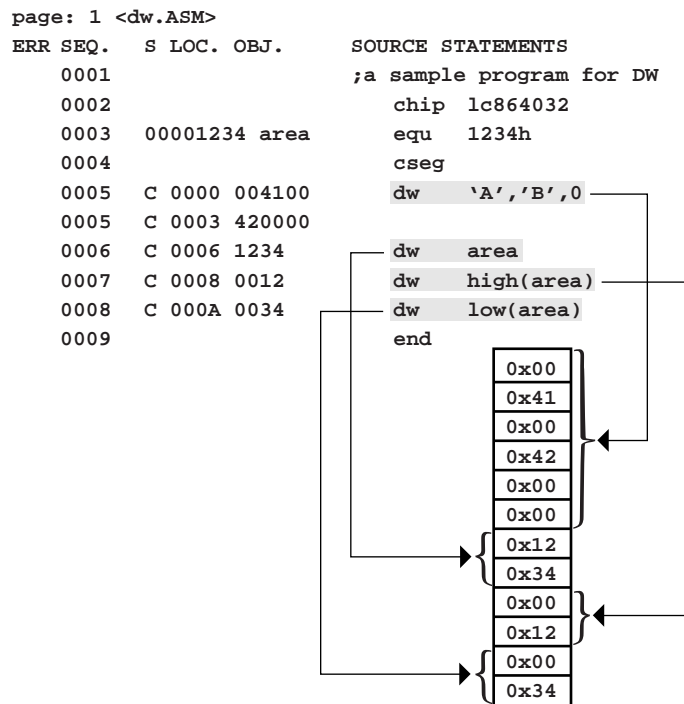
Regarding "db area" in the above example, because the value of the symbol area has a 16-bit width, a value is out of range error (warning level) will be generated during the assembly process. However, the value of the lower 8 bits will be output in the object code.

12. DW (Define word data)

labelname DW expression [,expression]

The DW pseudo instruction is used to store 16-bit data that corresponds to the operand expression in program memory (ROM). The upper byte is stored first, and the lower byte is stored in the next address (the higher address). More than one operand can be described; delimit each operand with a comma (","). When there are two or more operands, they are stored in a continuous area. If no operand is described between two commas, that operand is regarded as being "0".

Example:



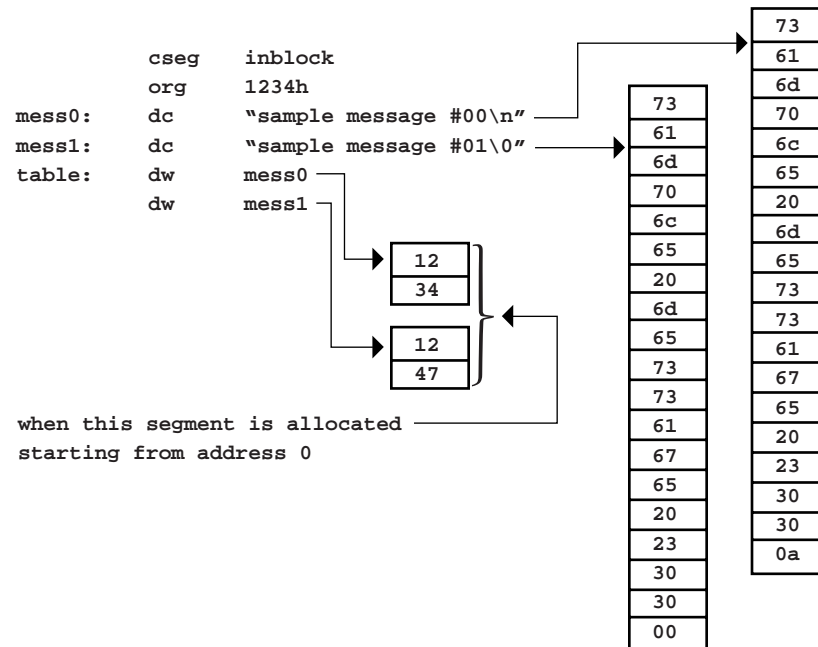
If an 8-bit value is allocated using the DW pseudo instruction, the upper 8 bits of the 16 bits are always comprised of zeroes.

13. DC (Define character string data)

labelname DC "string"

The DC pseudo instruction stores the contents of string (a character string constant) as the ASCII code values of each character in sequence in program memory (ROM). For details on character string constants, refer to section 4.7.

Example:

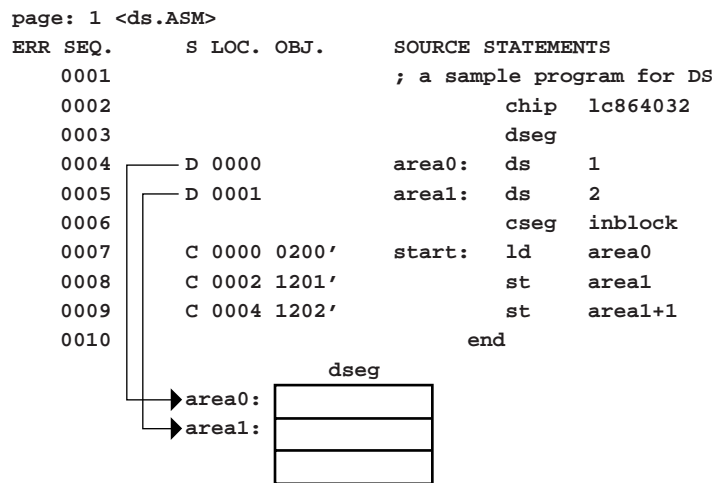


14. DS (Define byte area)

labelname DS absolute_expression

The DS pseudo instruction allocates an area consisting of the number of bytes specified by absolute_expression in data memory (RAM). The description of absolute_expression must be in absolute format (in which all of the values are determined). This pseudo instruction cannot be used unless it comes after the DSEG pseudo instruction.

Example:



In the above example, a one-byte area is allocated under the name area0, and another two-byte area under the name areal immediately after area0.

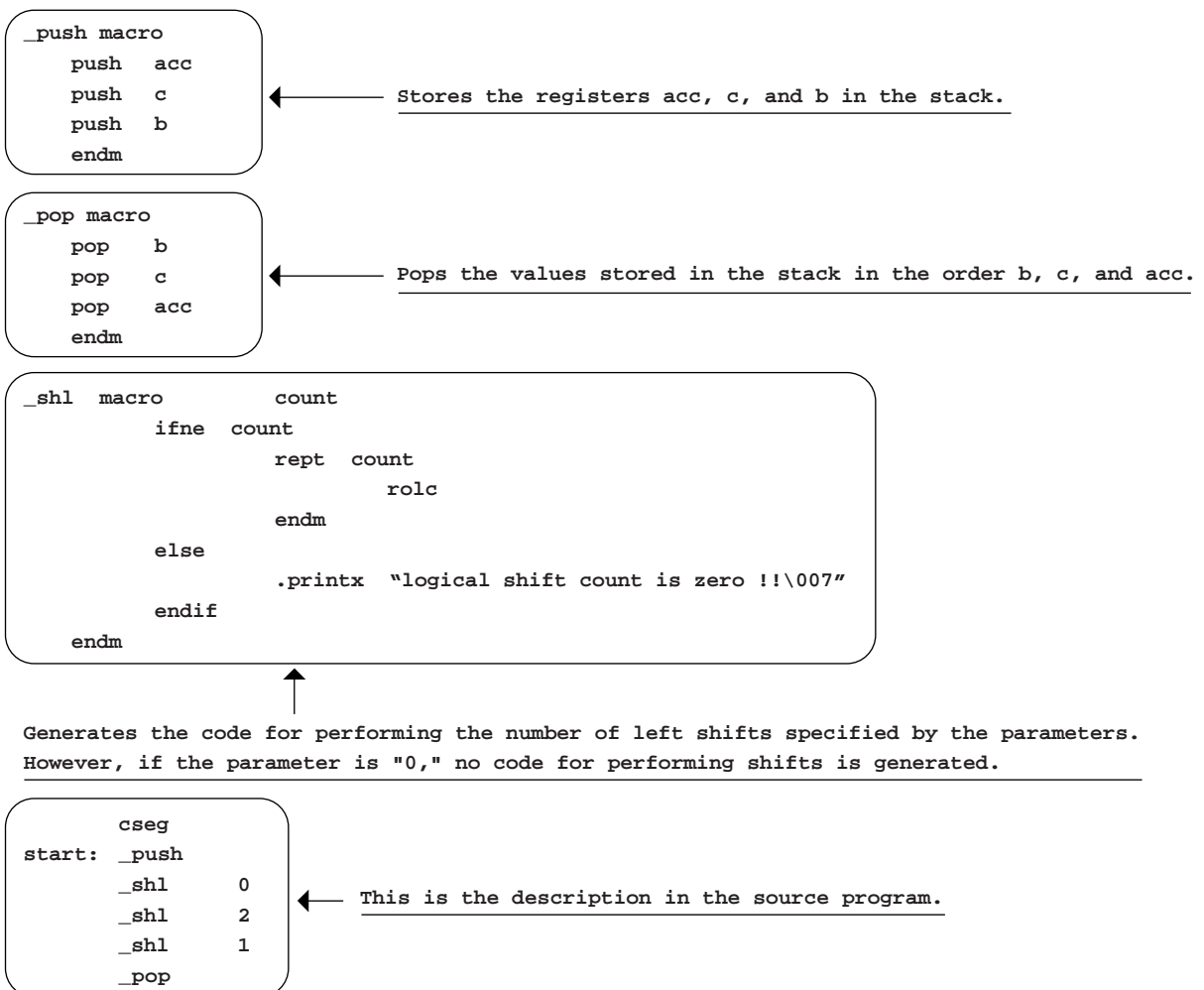
15. MACRO (Define macro)

name MACRO parameter {, parameter}

The MACRO pseudo instruction defines macros. The main body of the macro definition is formed by all of the statements that follow the MACRO pseudo instruction until the ENDM pseudo instruction is reached. name calls the macro that was defined. Because name is required so that it can be replaced with the main body of the macro definition, it must be specified. because parameters is a list of parameters, they should be specified in accordance with the contents of the macro that was defined.

Note that when calling other macros from within a macro, or when using a pseudo instruction, such as IFB, that requires "<" or ">", sufficient "<" and ">" are needed for the nesting level.

Example:



The result of assembly of the above source code is shown on the following page.

Visual Memory Unit (VMU) Environment Variables

```
0027      start:_push
0027+1    C 0000 6100    push  acc
0027+2    C 0002 6103    push  c
0027+3    C 0004 6102    push  b
0028      _shl  0
0028+1    ifne  0
0028+2    rept  0
0028+3    rolc
0028+4    endm
0028+5    else
0028+6    .printx "logical shift count is ze
0028+7    endif
0029      _shl  2
0029+1    ifne  2
0029+2    rept  2
0029+4    endm
0029+4    endm
0029+1    C 0006 F0      rolc
0029+2    C 0007 F0      rolc
0029+5    else
0029+6    .printx "logical shift count is ze
0029+7    endif
0030      _shl  1
0030+1    infe  1
0030+2    rept  1
0030+4    endm
0030+1    C 0008 F0      rolc
0030+5    else
0030+6    .print "logical shift count is ze ro !\007"
0030+7    endif
0031      _pop
0031+1    C 0009 7102    pop   b
0031+2    C 000b 7103    pop   c
```


17. IRP (Continuous macro)

IRP parameter, argument {,argument }...

The IRP pseudo instruction repeats a series of statements from the IRP pseudo instruction up to the ENDM pseudo instruction the number of times specified by argument. During the repetition, each time that parameter appears in one of the statements, an item in the argument field is substituted for parameter, in order.

Example:

```
_push macro
    irp    reg_name,acc,b,psw,c
        push    reg_name
    endm
endm
_pop macro
    irp    reg_name,c,psw,b,acc
        push    reg_name
    endm
endm
```

```
0016
0017                _push
0017+1                irp    reg_name,acc,b,psw,c
0017+3                endm
0017+1 C 0000 6100                push    acc
0017+2 C 0002 6102                push    b
0017+3 C 0004 6101                push    psw
0017+4 C 0006 6103                push    c
0018                _pop
0018+1                irp    reg_name,c,psw,b,acc
0018+3                endm
0018+1 C 0008 6103                push    c
0018+2 C 000A 6101                push    psw
0018+3 C 000C 6102                push    b
0018+4 C 000E 6100                push    acc
```

18. IRPC (Character string macro)

IRPC parameter, string

The IRPC pseudo instruction repeats a series of statements from the IRPC pseudo instruction up to the ENDM pseudo instruction a number of times equal to the number of characters in string. Unlike character string constants, string is not enclosed in quotes, etc. In addition, it is not possible to input a code that begins with the yen symbol. During the repetition, each occurrence of parameter in one of the statements is replaced with a character in string; this replacement is repeated until all of the characters in string have been used.

Example:

```

; a sample program for IRPC
chip lc866032
dseg
irpc x,01234567
ds 2
endm
end

```

parameter

string

buf&x:

Each occurrence of parameter is replaced with a character from string

Delimiter when parameter occurs as part of an identifier

Expansion
Results

```

page: 1 <irpc.ASM>
ERR SEQ.  S LOC. OBJ.      SOURCE STATEMENTS
0001      ; a sample program for IRPC
0002      chip      lc866032
0003      dseg
0004      irpc      x,01234567
0006      endm
0006+1  D 0000      buf0:  ds      2
0006+2  D 0002      buf1:  ds      2
0006+3  D 0004      buf2:  ds      2
0006+4  D 0006      buf3:  ds      2
0006+5  D 0008      buf4:  ds      2
0006+6  D 000A      buf5:  ds      2
0006+7  D 000C      buf6:  ds      2
0006+8  D 000E      buf7:  ds      2
0007      end

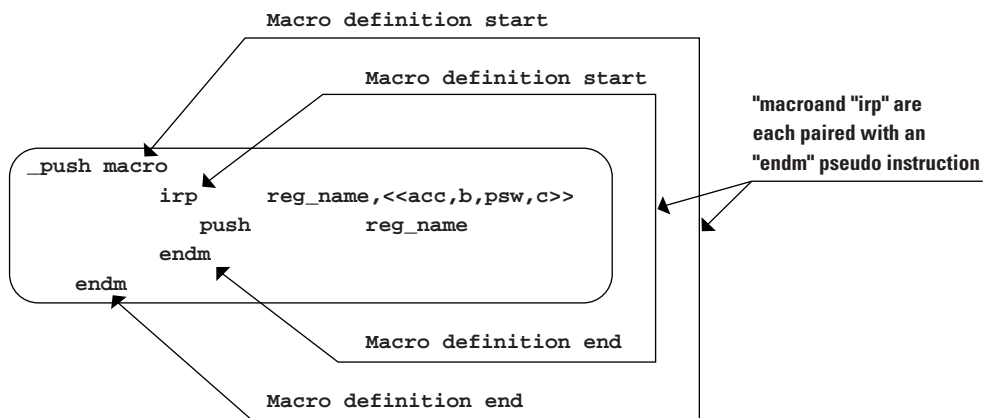
```

19. ENDM (End macro definition)

ENDM

The ENDM pseudo instruction declares the end of a macro definition statement.

Example:



20. EXITM (Interrupt macro expansion)

EXITM

The EXITM pseudo instruction interrupts macro expansion. This pseudo instruction is used in combination with conditional assembly pseudo instructions in order to obtain expansion results that differ according to the arguments that are given to identical macros.

Example:

```

page: 1 <exitm.ASM>
ERR SEQ.   S LOC. OBJ.   SOURCE STATEMENTS
0001       ; a sample program for EXITM
0002       chip LC866032
0003 rpush  macro a1,a2,a3,a4
0004       ifb    <<a1>>
0005       .printx "not enough argument"
0006       exitm
0007       endif
0008       ifnb <<a2>>
0009       push   a1
0010       push   a2
0011       push   a3
0012       push   a4
0013       endif
0014       endm
0015       cseg    inblock
0016 rpush  acc,b,psw,c
0016+1     ifb    <acc>
0016+2     .printx "not enough argument"
0016+3     exitm
0016+4     endif
0016+5     ifnb   <b>
0016+6 C 0000 6100     push   acc
0016+7 C 0002 6102     push   b
0016+8 C 0004 6101     push   psw
0016+9 C 0006 6103     push   c
0016+10    endif
0017 rpush
0017+1     ifb
0017+2     .printx "not enough argument"
0017+3     exitm
0018       end

```

Because one pair of "<>" is deleted during macro expansion, double symbols ("<>") are required

Because the first argument is given, this portion is assembled.

Because there is no second argument, this portion is expanded; when EXITM is recognized, expansion is halted.

21. LOCAL (Define local label)

LOCAL name {, name}

The LOCAL pseudo instruction is used to declare a label that can be used within a macro definition. If the name declared by the LOCAL pseudo instruction appears within a macro expansion, the macro assembler substitutes a new name for name that will not conflict with any other names.

Example:

```
; a sample program for LOCAL

                                chip                1c864008
b_ne                            macro                val,dst
                                local                skip
                                be                    val,skip
                                bro                    dst
skip:
                                endm

                                cseg
                                b_ne                #0, over

                                org                200h
over:                            b_ne                #0, under
                                nop
under:                            nop
                                end
```

In the above example, the BRO pseudo instruction is used to define the BNEO macro instruction that automatically generates an instruction word according to the branching destination. The results of assembly are shown on the next page.

```

page: 1 <local.ASM>
ERR SEQ.  S LOC. OBJ. SOURCE STATEMENTS
0001          ; a sample program for LOCAL
0002          chip    lc864008
0003          b_ne    macro  val,dst
0004          local   skip
0005          be      val,skip
0006          bro     dst
0007          skip:
0008          endm
0009
0010          cseg
0011          b_ne    #0, over
0011+1         local  _L00000000L_
0011+2 C 0000 310003  be    #0,_L00000000L_
0011+3 C 0003 11FB01  bro    over
0011+4         _L00000000L_:
0012
0013          org     200h
0014          over:   b_ne    #0, under
0014+1         local  _L00000001L_
0014+2 C 0200 310002  be    #0,_L00000001L_
0014+3 C 0203 0101   bro    under
0014+4         _L00000001L_:
0015 C 0205 00        nop
0016 C 0206 00        under:  nop
0017          end

```

The identifier declared by LOCAL is replaced by a unique name.

The name generated has the format `_L#####L_` (where ##### is a serial number, starting with 000000).

22. IFDEF (Assemble if defined)

IFDEF symbol

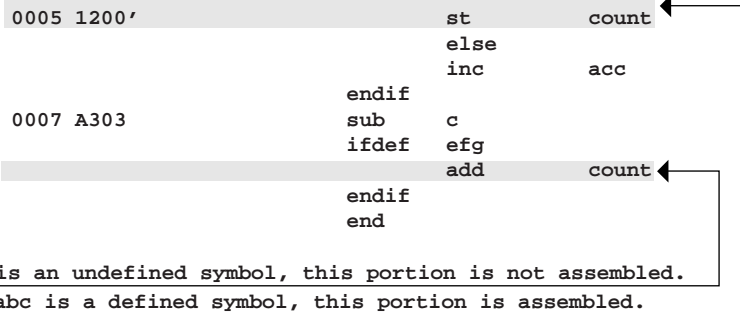
The IFDEF pseudo instruction assembles the source program until either ELSE or ENDIF appears subsequently, but only if symbol has already been defined.

Example:

```
page: 1 <ifdef.ASM>
ERR SEQ.  S LOC. OBJ.  SOURCE STATEMENTS
0001      ; a sample program for IFDEF
0002      chip 1c864024
0003 00000001 abc equ 1
0004      dseg
0005 D 0000 count: ds 1
0006
0007      cseg inblock
0008 C 0000 230010 mov #10h, acc
0009      ifdef abc
0010 C 0003 8302 add b
0011 C 0005 1200' st count
0012      else
0013      inc acc
0014      endif
0015 C 0007 A303 sub c
0016      ifdef efg
0017      add count
0018      endif
0019      end
```

Because efg is an undefined symbol, this portion is not assembled.

Because abc is a defined symbol, this portion is assembled.



23. IFNDEF (Assemble if undefined)

IFNDEF symbol

The IFNDEF pseudo instruction assembles the source program until either ELSE or ENDIF appears subsequently, but only if symbol has not been defined.

Example:

ERR	SEQ.	S	LOC.	OBJ.	SOURCE STATEMENTS
	0001				; a sample program for IFDEF
	0002				chip 1c864024
	0003	00000001			abc equ 1
	0004				dseg
	0005	D 0000			count: ds 1
	0006				
	0007				cseg inblock
	0008	C 0000	230010		mov #10h, acc
	0009				ifdef abc
	0010				add b
	0011				st count
	0012				else
	0013	C 0003	6300		inc acc
	0014				endif
	0015	C 0005	A303		sub c
	0016				ifdef efg
	0017	C 007	8200		add count
	0018				endif
	0019				end

Because efg is an undefined symbol, this portion is not assembled.

Because abc is a defined symbol, this portion is assembled.

24. IFB (Assemble if operand is empty)

IFB <argument>

The IFB pseudo instruction assembles the source program until either ELSE or ENDIF appears subsequently, but only if argument is empty. If the argument contains any space or tab characters, it is regarded as not being empty. argument must be enclosed in <>.

Example:

page:	1	<ifb.ASM>		SOURCE STATEMENTS	
ERR	SEQ.	S	LOC.	OBJ.	
	0001			; a sample program for IFB	
	0002			chip 1c864016	
	0003	tifb		macro arg	
	0004			ifb <<arg>>	← Because one pair of "<>" is deleted during macro expansion, double symbols ("<>") are required.
	0005			inc a	
	0006			else	
	0007			inc b	
	0008			endif	
	0009			endm	
	0010				
	0011			tifb xxx	
	0011+1			ifb <xxx>	
	0011+2			inc a	
	0011+3			else	
	0011+4	C 0000	6302	inc b	←
	0011+5			endif	
	0012			tifb	
	0012+1			ifb <>	
	0012+2	C 0002	6300	inc a	←
	0012+3			else	
	0012+4			inc b	
	0012+5			endif	
	0013			end	

Because the argument for IFB is empty, this portion is assembled.

Because the argument for IFB is not empty, this portion is not assembled.

25. IFNB (Assemble if operand is not empty)

IFNB <argument>

The IFNB pseudo instruction assembles the source program until either ELSE or ENDIF appears subsequently, but only if argument is not empty. If the argument contains any space or tab characters, it is regarded as not being empty. argument must be enclosed in <>.

Example:

```

page: 1 <ifnb.ASM>
ERR SEQ.   S LOC. OBJ.   SOURCE STATEMENTS
0001       ; a sample program for IFNB
0002       chip          lc864016
0003       tifb          macro   arg
0004               ifnb      <<arg>>
0005               inc       a
0006               else
0007               inc       b
0008               endif
0009               endm
0010
0011       tifb          xxx
0011+1      ifnb          <xxx>
0011+2 C 00006300      inc       a
0011+3               else
0011+4               inc       b
0011+5               endif
0012       tifb
0012+1      ifnb          <>
0012+2               inc       a
0012+3               else
0012+4 C 0002 6302      inc       b
0012+5               endif
0013       end

```

Because one pair of "<>" is deleted during macro expansion, double symbols ("<>") are required.

Because the argument for IFNB is empty, this portion is not assembled.

Because the argument for IFNB is not empty, this portion is assembled.

26. IFE (Assemble if value of expression is "0")

IFE expression

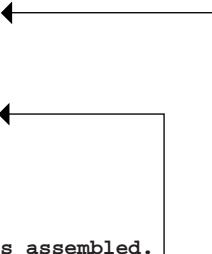
The IFE pseudo instruction assembles the source program until either ELSE or ENDIF appears subsequently, but only if the value of expression is "0."

Example:

```
page: 1 <ife.ASM>
ERR SEQ. S LOC. OBJ.      SOURCE STATEMENTS
0001                      ; a sample program for IFE
0002                      chip    lc866032
0003                      cseg
0004      00000003 aa      set     3
0005                      ife     aa-2
0006                      inc     70h
0007                      else
0008      C 0000 7270      dec     70h ←
0009                      endif
0010      00000002 aa      set     aa-1
0011                      ife     aa-2
0012      C 0002 6270      inc     70h ←
0013                      else
0014                      dec     70h
0015                      endif
0016                      end
```

Because the value of the expression is "0," this portion is assembled.

Because the value of the expression is not "0," this portion is not assembled.



27. IFNE (Assemble if value of expression is not "0")

IFNE expression]

The IFNE pseudo instruction assembles the source program until either ELSE or ENDIF appears subsequently, but only if the value of expression is not 0.

Example:

```

page: 1 <ifne.ASM>
ERR SEQ.  S LOC. OBJ.      SOURCE STATEMENTS
0001                      ; a sample program for IFNE
0002                      chip    lc866032
0003                      cseg
0004      00000003 aa      set     3
0005                      ifne    aa-2
0006      C 0000 6270      inc     70h ←
0007                      else
0008                      dec     70h
0009                      endif
0010      00000002 aa      set     aa-1
0011                      ifne    aa-2
0012                      inc     70h
0013                      else
0014      C 0002 7270      dec     70h ←
0015                      endif
0016                      end

Because the value of the expression is "0,", this portion is not assembled.
Because the value of the expression is not "0,", this portion is assembled.

```

28. IFIDN (Assemble if two character strings are identical)

IFIDN <string1>, <string2>

The IFIDN pseudo instruction assembles the source program until either ELSE or ENDIF appears subsequently, but only if string1 and string2 are identical. string1 and string2 must be enclosed in <>. The comparison includes any space or tab characters within the <>.

Example:

page: 1	<ifidn.ASM>		
ERR SEQ.	S LOC. OBJ.	SOURCE STATEMENTS	
0001		; a sample program for IFIDN	
0002		chip 1c866032	
0003		cseg	
0004		tifidn macro arg1,arg2	
0005		ifidn <<arg1>>,<<arg2>>	
0006		inc a	Because one pair of "<>" is deleted during macro expansion, double symbols ("<>") are required.
0007		else	
0008		dec a	
0009		endif	
0010		endm	
0011			
0012		tifidn same, same	
0012+1		ifidn <same>,<same>	
0012+2	C 0000 6300	inc a	Because the two character strings are different, this portion is not assembled.
0012+3		else	
0012+4		dec a	
0012+5		endif	
0013		tifidn same, not_same	
0013+1		ifidn <same>,<not_same>	
0013+2		inc a	Because the two character strings are identical, this portion is assembled.
0013+3		else	
0013+4	C 0002 7300	dec a	
0013+5		endif	
0014		end	

29. IFDIF (Assemble if two character strings are not identical)

IFDIF <string1>, <string2>

The IFDIF pseudo instruction assembles the source program until either ELSE or ENDIF appears subsequently, but only if string1 and string2 are different. string1 and string2 must be enclosed in <>. The comparison includes any space or tab characters within the <>.

Example:

```

page: 1 <ifdif.ASM>
ERR SEQ.          S LOC. OBJ. SOURCE STATEMENTS
0001              ; a sample program for IFDIF
0002              chip      lc866032
0003              cseg
0004      tifidn    macro    arg1,arg2
0005              ifdif     <<arg1>>,<<arg2>>
0006                  inc    a
0007              else
0008                  dec    a
0009              endif
0010              endm
0011
0012              tifidn    same, same
0012+1             ifdif     <same>,<same>
0012+2             inc      a
0012+3             else
0012+4 C 0000 7300    dec      a
0012+5             endif
0013              tifidn    same, not_same
0013+1             ifdif     <same>,<not_same>
0013+2 C 0002 6300    inc      a
0013+3             else
0013+4             dec      a
0013+5             endif
0014              end

```

Because one pair of "<>" is deleted during macro expansion, double symbols ("<>") are required.

Because the two character strings are different, this portion is assembled.

Because the two character strings are identical, this portion is not assembled.

30. ELSE (Assemble in case of condition opposite of the above IF condition)

ELSE

The ELSE pseudo instruction assembles the source program if the opposite of the preceding IF condition is true, until ENDIF appears. For an example, refer to the description of the IFDEF pseudo instruction, etc.

31. ENDIF (End conditional assembly)

ENDIF

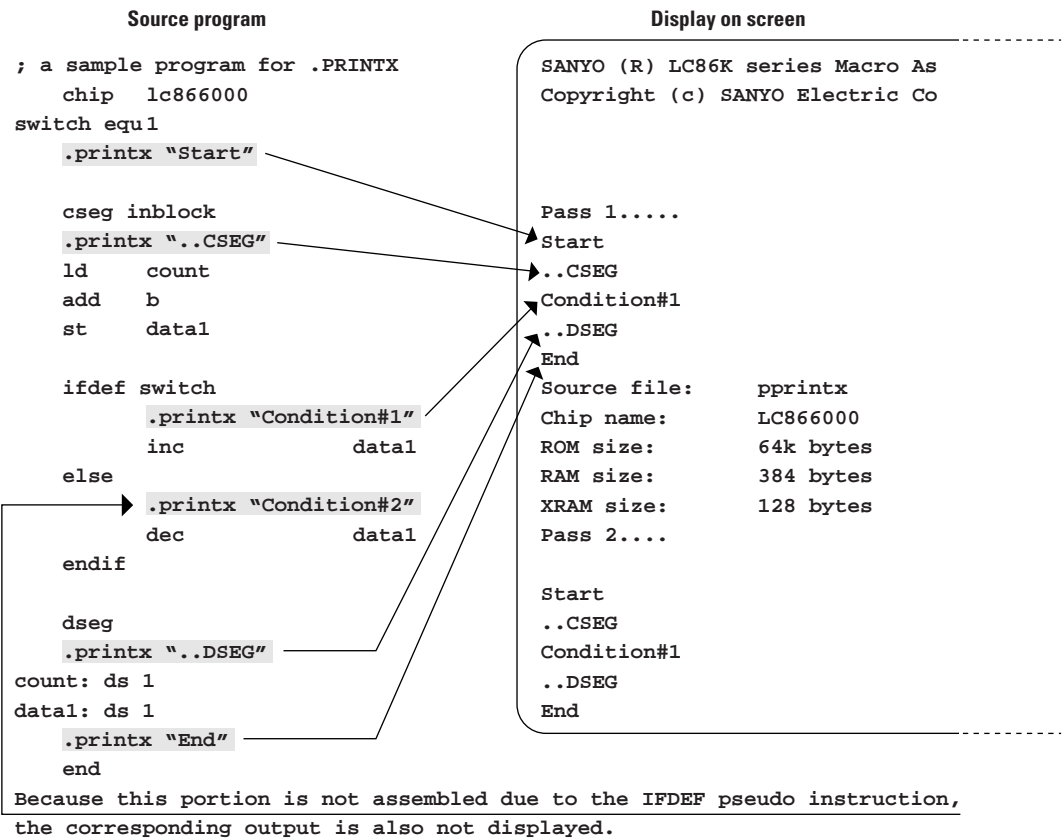
The ENDIF pseudo instruction declares the end of conditional assembly. For an example, refer to the description of the IFDEF pseudo instruction, etc.

32. PRINTX (Display on VDT during assembly)

.PRINTX "string"

The .PRINTX pseudo instruction outputs the contents of string character string constants on the VDT during the assembly process. For details on character string constants, refer to section 4.7.

Example:



33. LIST (Output list)

.LIST

The .LIST pseudo instruction releases the status in which list file output was halted by the .XLIST pseudo instruction.

Example:

```
; a sample program for LIST
chip    lc866200
cseg    inblock
mov     #00, count
ld      count
add     #10h
st      b
```

The lines that follow the line where .XLIST appears do not appear in the list file. Counting of the line continues, however, so no disagreement in line numbering occurs.

```
.xlist
abc     equ    10h
dseg
count:  ds     4
```

After the .LIST pseudo instruction, output to the list file resumes.

```
.list
cseg    inblock
ld      b
sub     #abc
st      count
end
```

page: 1 <plist.ASM>

ERR	SEQ.	S	LOC.	OBJ.	SOURCE STATEMENTS
	0001				; a sample program for LIST
	0002				chip lc866200
	0003				cseg inblock
	0004	C 0000	220000'		mov #00, count
	0005	C 0003	0200'		ld count
	0006	C 0005	8110		add #10h
	0007	C 0007	1302		st b
	0008				
	0014				.list
	0015				cseg inblock
	0016	C 0000	0302		ld b
	0017	C 0002	A110'		sub #abc
	0018	C 0004	1200'		st count
	0019				end

34. .XLIST (Interrupt list output)

.XLIST

The .XLIST pseudo instruction interrupts output to the list file. For an example, refer to the description of the .LIST pseudo instruction.

35. .MACRO (Output macro expansion)

.MACRO

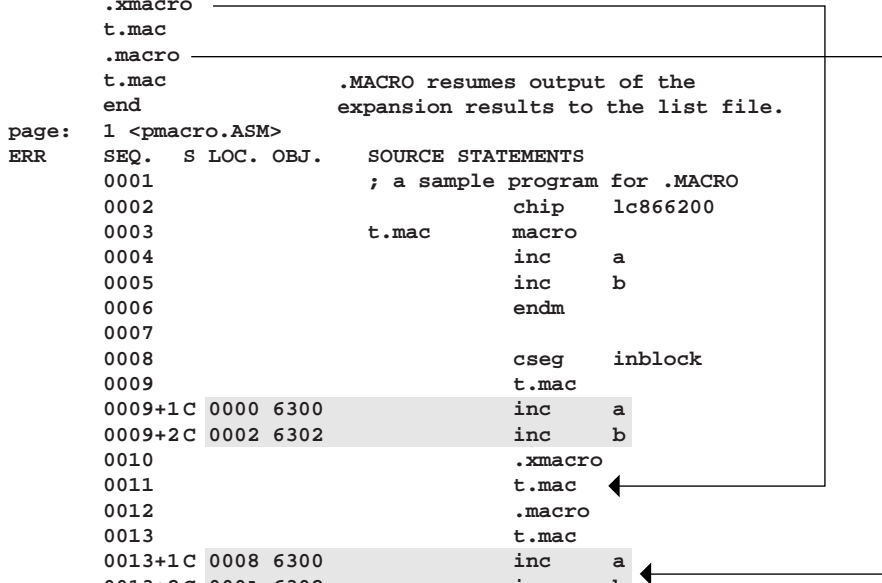
The .MACRO pseudo instruction expands the main body of a macro and outputs it to the list file when the macro is called.

Example:

```
; a sample program for .MACRO
chip 1c866200
t.mac macro
    inc a
    inc b
endm
cseg inblock
t.mac
    .xmacro
t.mac
    .macro
t.mac
    end
end
page: 1 <pmacro.ASM>
ERR  SEQ. S LOC. OBJ.  SOURCE STATEMENTS
0001      ; a sample program for .MACRO
0002      chip      1c866200
0003      t.mac      macro
0004          inc      a
0005          inc      b
0006          endm
0007
0008      cseg      inblock
0009      t.mac
0009+1C 0000 6300      inc      a
0009+2C 0002 6302      inc      b
0010      .xmacro
0011      t.mac
0012      .macro
0013      t.mac
0013+1C 0008 6300      inc      a
0013+2C 000A 6302      inc      b
0014      end
```

.XMACRO disables output of the macro expansion results to the list file. Note that the codes that are generated by the expanded statements are also not output.

.MACRO resumes output of the expansion results to the list file.



36. .XMACRO (Interrupt macro expansion output)

.XMACRO

The .XMACRO pseudo instruction temporarily interrupts the output to the list file of the results of expansion of the macro main body when a macro is called. For an example, refer to the explanation of the .MACRO item.

37. .IF (Output conditional skip)

.IF

The .IF pseudo instruction expands source program statements that were skipped during conditional instruction execution and outputs them to the list file.

Example:

```

; a sample program for .IF
chip    lc866200
t.if    macro    arg1
    ifb    <<arg1>>
    inc    a
    else
    inc    b
    endif
endm
cseg    inblock
t.if
.xif
t.if    abc
.if
t.if    def
end

```

.XIF disables the output to the list file of statements that were skipped by conditional assembly psuedo instructions. Lines wich were not skipped because the specified condition was matched are output to the list file, regardless of the .XIF psuedo instructio

.IF enables the output to the list file of statements that were skipped by conditional assembly psuedo instructions.

```

page: 1 <pif.ASM>
ERR   SEQ. S LOC. OBJ.  SOURCE STATEMENTS
0001      ; a sample program for .IF
0002      chip    lc866200
0003
0004      t.if    macro    arg1
0005          ifb    <<arg1>>
0006          inc    a
0007      else
0008          inc    b
0009      endif
0010      endm
0011      cseg    inblock
0012      t.if
0012+1          ifb    <>
0012+2 C 0000 6300          inc    a
0012+3          else
0012+4          inc    b
0012+5      endif
0013      .xif
0014      t.if    abc
0014+1          ifb    <abc>
0014+3          else
0014+4 C 0002 6302          inc    b
0014+5      endif
0015      .if
0016      t.if    def
0016+1          ifb    <def>
0016+2          inc    a
0016+3          else
0016+4 C 0004 6302          inc    b
0016+5      endif
0017      end

```

38. .XIF (Interrupt conditional skip output)

.XIF

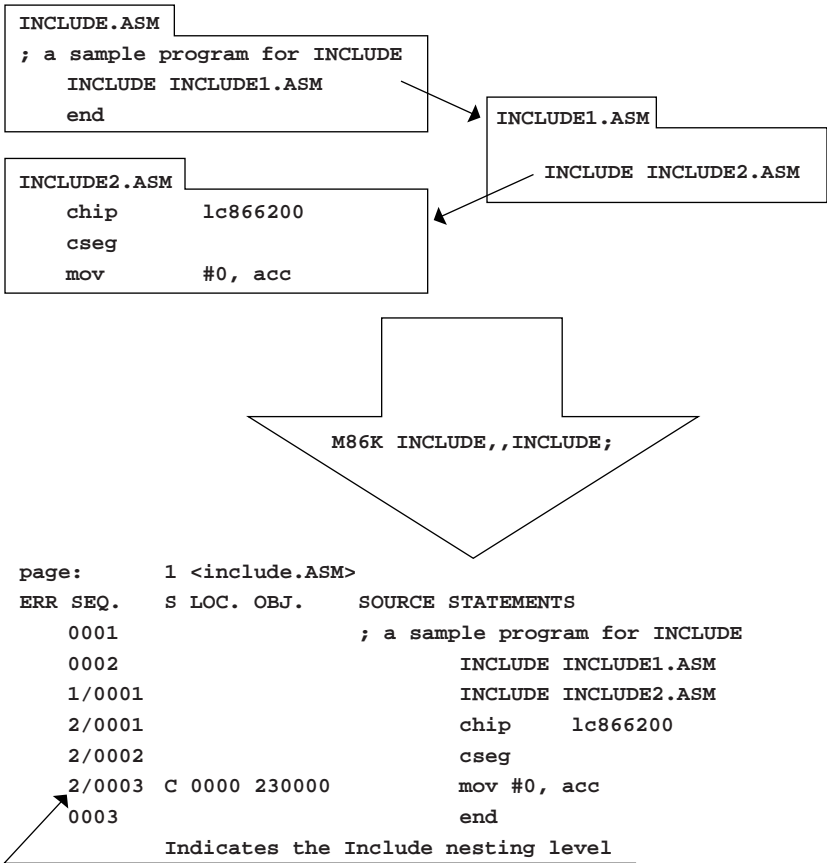
The .XIF pseudo instruction disables the expansion in the list file of source program statements that were skipped during conditional instruction execution. For an example, refer to the explanation of the .IF pseudo instruction.

39. INCLUDE (Load file)

INCLUDE filename

The INCLUDE pseudo instruction loads and assembles the source file specified by filename while assembling the source program. The filename specification must include the extension. The INCLUDE pseudo instruction can be nested up to nine levels. If the loaded file contains the END pseudo instruction, the assembly process stops there.

Example:




40. TITLE (Specify list title)

TITLE string

The parameter string of the TITLE pseudo instruction specifies a title for the list file. Unlike character string constants, string is not enclosed in quotes, etc. In addition, it is not possible to input a code that begins with the yen symbol.

Example:

string is displayed here on all pages.



```
page: 1 <title.ASM>      sample program's title for the listing
ERR SEQ.  S LOC. OBJ.    SOURCE STATEMENTS
0001                      ; a sample program for TITLE
0002                      TITLE    sample program's title for the listing
0003                      chip     1c864024
0004                      cseg
0005      C 0000 00  nop
0006                      end
```

41. PAGE (End of page)

PAGE

The PAGE pseudo instruction forcibly ends a page during output to a list file. The end-of-page character is inserted directly before this pseudo instruction.

Example:

```
Source file
; a sample program for PAGE
chip 1c866032
page
cseg
page
nop
page
end
```

List file

```
page: 1 <page.ASM>
ERR SEQ. S LOC. OBJ.      SOURCE STATEMENTS
0001                      ; a sample program for PAGE
0002                      chip      1c866032
```

```
page: 2 <page.ASM>
ERR SEQ. S LOC. OBJ.      SOURCE STATEMENTS
0003                      page
0004                      cseg
```

```
page: 3 <page.ASM>
ERR SEQ. S LOC. OBJ.      SOURCE STATEMENTS
0005                      page
0006 C 0000 00           nop
```

```
page: 4 <page.ASM>
ERR SEQ. S LOC. OBJ.      SOURCE STATEMENTS
0007                      page
0008                      end
```

42. CHIP (Define chip that is target of assembly)

CHIP chipname

The CHIP pseudo instruction notifies the assembler about which chip is the target of the assembly process. The assembler switches the reserved word list and checks the memory size on the basis of chipname. This pseudo instruction is written at the top of a source file, before any other instructions or pseudo instructions. If this pseudo instruction is not found, the assembler references the value of the environment variable CHIPNAME. If the name of the chip that is declared by this pseudo instruction does not match the name indicated by the environment variable CHIPNAME, a "warning-level" error is generated.

43. COMMENT (Output comments to object file)

COMMENT comment_string

The COMMENT pseudo instruction gives comments that are output to the assembled object file. Unlike character string constants, comment_string is not enclosed in quotes, etc. In addition, it is not possible to input a code that begins with the yen symbol. comment_string is stored in the object file starting from the 680th byte. A comment may consist of up to 255 characters.

Example:

```
Source file
; a sample program for COMMENT
chip lc866024
comment This is a comment string embedded into OBJ file
cseg
nop
end
```

Object file dump (showing only the necessary portion)

	Number of characters (one byte)
00000260 00 00 00 00 00 00 00 00-00	00 00 00 00 00 00 00
00000270 00 00 00 00 00 60 00 00-80	01 00 00 80 00 00 00
00000280 C6 92 40 2B 4D 38 36 4B-20	20 20 20 63 6F 6D 6D ***M86K comm
00000290 65 6E 74 2E 41 53 4D 20-63	6F 6D 6D 65 6E 74 20 ent.ASM comment
000002A0 4C 43 38 36 36 30 32 34-30	54 68 69 73 20 69 73 LC8660240This is
000002B0 20 61 20 63 6F 6D 6D 65-6E	74 20 73 74 72 69 6E a comment strin
000002C0 67 20 65 6D 62 65 64 64-65	64 20 69 6E 74 6F 20 g embedded into
000002D0 4F 42 4A 20 66 69 6C 65-00	00 01 01 00 01 00 05 OBJ file.....
000002E0 00 01 00 00 00 00 00 00-00	00 E0 00 00 00 00 C4
000002F0 00 00 00 00 C4 00 00 00-00	24 00 00 01 00 04 01*....\$.
00000300 00 00 00 24\$

44. WIDTH (Specify number of columns in list file)

WIDTH number

The WIDTH pseudo instruction specifies the number of columns in the list file (i.e., the number of characters per line). A number from 72 to 132 can be specified for number, but specifying the value equal to the number of columns in the source file plus at least 28 is recommended whenever possible. Furthermore, although this pseudo instruction can be described any number of times within one source file, normally it is only described once at the start of file. Note that if this pseudo instruction is not found, the default list file width is 132 columns.

Example:

Although WIDTH is evaluated in both pass 1 and pass 2, the list file is generated only in pass2. Therefore, the last evaluated result for WIDTH in pass 1 is reflected here, so this line wraps around at this position.

1	2	3	4	5	6	7	8
1234567890123456789012345678901234567890123456789012345678901234567890							
page: 1 <width.ASM>							
ERR SEQ.	S LOC. OBJ.	SOURCE STATEMENTS					
0001		; a sample program for WIDTH					
0002		chip	1c866200				
0003		cseg	; this is a long line to indicat				
0003		e WIDTH's effect					
0004		WIDTH 72					
0005	C 0000 00	nop	; this is also a long line				
0005		to indicate WIDTH's effect					
0006		WIDTH 78					
0007		end					
Because a line feed character is inserted at the 72nd character, the line wraps around at this position.							

45. BANK (Specify RAM area bank)

BANK expression

The BANK pseudo instruction gives the bank number for symbols that were defined by the DS pseudo instruction in the RAM area described subsequent to the DSEG pseudo instruction.

Example:

```

page: 1 <bank.ASM>
ERR SEQ.  S LOC. OBJ.  SOURCE STATEMENTS
0001      ; a sample program for BANK
0002      chip 1c866032
0003      cseg inblock
0004
0005 C 0000 220000'      mov  #0,data1
0006
0007 C 0003 6200'        inc   data1
0008 C 0005 0200'        ld    data1
0009 C 0007 1201'        st    data2
0010
0011 C 0009 6200'        inc   dataa
0012 C 000B 0200'        ld    dataa
0013 C 000D 1202'        st    datac
0014
0015      dseg
0016      bank 0
0017 D 0000      data1:  ds    1
0018 D 0001      data2:  ds    1
0019 D 0002      data3:  ds    1
0020
0021      bank 1
0022 D 0000      dataa:  ds    1
0023 D 0001      datab: ds    1
0024 D 0002      datac:  ds    1
0025
0026      end

```

These symbols are assigned to bank 1.

These symbols are assigned to bank 0.

46. CHANGE (Jump between external and internal ROM)

CHANGE symbol

CHANGE is a special jump instruction that is used to switch between executing code stored in external ROM and code stored in internal ROM. The operand symbol is limited to symbols that have been declared by the pseudo instruction OTHER_SIDE_SYMBOL. Note that this pseudo instruction is a special instruction for the LC868000 only, and will generate an error if executed by any other type of chip.

Example:

```
page: 1 <change.ASM>
ERR SEQ.  S LOC. OBJ.      SOURCE STATEMENTS
0001                      ; a sample program for CHANGE
0002                      chip    lc868032
0003                      other_side_symbol  far_away
0004
0005                      cseg
0006      C 0000 B80D21'    change  far_away
0006      C 0003 0000'
```

47. RADIX (Specify default base)

RADIX expression

The RADIX pseudo instruction specifies the base to which the value of a numeric constant is converted when the base of that constant is not explicitly indicated. Only certain values can be specified for expression: 2, 8, 10, and 16. Once this pseudo instruction is executed, the specified base remains valid until a different base is specified by another RADIX pseudo instruction. If this pseudo instruction is not specified, the default base is 10.

Example:

```
xxx    SET      10    → Interpreted as 1010, since the default base is 10.
        RADIX    16
xxx    SET      10    → Interpreted as 1610, since the base was set to 16.
        RADIX    2
xxx    SET      10    → Interpreted as 210, since the base was set to 2.
```

48. JMPO (Generate optimal JMP instruction)

JMPO expression

The JMPO pseudo instruction compares the current location with expression, and generates a JMP if the two locations are in the same block (i.e., the addresses are identical except for the lower 12 bits). If the two locations are not within the same block, or if the value of the destination address cannot be specifically determined because it is an external symbol, then JMP generates a JMPF.

Example:

Generates a JMP instruction when the current location and "expression" are within the same memory block

page: 1 <jmpo.ASM>

ERR	SEQ.	S	LOC.	OBJ.	SOURCE STATEMENTS
	0001				; a sample program for JMPO
	0002				chip 1c866032
	0003				cseg
	0004	C 0000	2803'		jmpo near
	0005	C 0002	00		nop
	0006	C 0003	00	near:	nop
	0007	C 0004	211000'		jmpo far
	0008				
	0009				org 1000h
	0010	C 1000	00	far:	nop
	0011				end

Generates a JMPF instruction when the current location and "expression" are in different memory blocks

49. BRO (Generate optimal BR instruction)

BRO expression

The BRO pseudo instruction compares the current location with expression, and generates a BR if the branching destination is within a range of +127 and -128. If the branching destination is outside of a range of +127 and -128, then BRO generates a BRF.

Example:

Generates a BR instruction when the branching destination is within a range of +127 and -128

page: 1 <bro.ASM>

ERR	SEQ.	S LOC.	OBJ.	SOURCE STATEMENTS
	0001			; a sample program for BRO
	0002			chip 1c866032
	0003			cseg
	0004	C 0000	0101	bro near
	0005	C 0002	00	nop
	0006	C 0003	00	near: nop
	0007	C 0004	11FA00	bro far
	0008			
	0009			org 1000h
	0010	C 0100	00	far: nop
	0011			end

Generates a BRF instruction when the branching destination is outside of a range of +127 and -128

50. CALLO (Generate optimal CAL instruction)

CALLO expression

The CALLO pseudo instruction compares the current location with expression, and generates a CALL if the two locations are in the same block (i.e., the addresses are identical except for the lower 12 bits). If the two locations are not within the same block, or if the value of the destination address cannot be specifically determined because it is an external symbol, then CALLO generates a CALLF.

Example:

Generates a CALL instruction when the current location and "expression" are within the same memory block

page: 1 <callo.ASM>

ERR	SEQ.	S	LOC.	OBJ.	SOURCE STATEMENTS
	0001				; a sample program for CALLO
	0002				chip 1c866032
	0003				cseg
	0004	C 0000	0805'		callo near
	0005	C 0002	201000'		callo far
	0006				
	0007	C 0005	00	near:	nop
	0008	C 0006	A0		ret
	0009				
	0010				org 1000h
	0011	C 1000	00	far:	nop
	0012	C 1001	A0		ret
	0013				end

Generates a CALLF instruction when the current location and expression are in different memory blocks

51. BZO (Generate BZ instruction that will not generate an address error)

BZO expression

The BZO macro generates instruction codes that are equivalent to the BZ instruction, with no restriction on the difference between the location of the instruction and the branching destination in the same segment within the same source. The BZO macro uses the BNZ instruction, which is the logical opposite of the BZ instruction, and the BRO instruction. expression describes the branching destination.

Code generation macro:

```
; ***          Branch near relative address if accumulator is zero ***
bzo           macro      r8
               local     _next_
               bnz       _next_
               bro       r8
_next_:
               endm
```

52. BNZO (Generate BNZ instruction that will not generate an address error)

BNZO expression

The BNZO macro generates instruction codes that are equivalent to the BNZ instruction, with no restriction on the difference between the location of the instruction and the branching destination in the same segment within the same source. The BNZO macro uses the BZ instruction, which is the logical opposite of the BNZ instruction, and the BRO instruction. expression describes the branching destination.

Code generation macro:

```
; ***      Branch near relative address if accumulator is not zero ***
bnzo      macro          r8
           local          _next_
           bz             _next_
           bro            r8
_next_:
           endm
```

53. BPO (Generate BP instruction that will not generate an address error)

BPO expression

The BPO macro generates instruction codes that are equivalent to the BP instruction, with no restriction on the difference between the location of the instruction and the branching destination in the same segment within the same source. The BPO macro uses the BZ instruction, the BR instruction, and the BRO instruction. expression describes the branching destination.

Code generation macro:

```
; ***      Branch near relative address if direct bit is positive ***
bpo      macro          d9,b3,r8
           local          _next_
           local          _true_
           bp             d9,b3,_true_
           br             _next_
_true_:   bro            r8
_next_:
           endm
```

54. BPCO (Generate BPC instruction that will not generate an address error)

BPCO expression

The BPCO macro generates instruction codes that are equivalent to the BPC instruction, with no restriction on the difference between the location of the instruction and the branching destination in the same segment within the same source. The BPCO macro uses the BPC instruction, the BR instruction, and the BRO instruction. expression describes the branching destination.

Code generation macro:

```
; ***      Branch near relative address if direct bit is positive,
;          and clear ***
bpc macro   d9,b3,r8
            local   _next_
            local   _true_
            bpc     d9,b3,_true_
            br      _next_
_true_:     bro     r8
_next_:
            endm
```

55. BNO (Generate BN instruction that will not generate an address error)

BNO expression

The BNO macro generates instruction codes that are equivalent to the BN instruction, with no restriction on the difference between the location of the instruction and the branching destination in the same segment within the same source. The BNO macro uses the BN instruction, the BR instruction, and the BRO instruction. expression describes the same value as in the BN instruction.

Code generation macro:

```
; ***      Branch near relative address if direct bit is negative ***
bno macro   d9,b3,r8
            local   _next_
            local   _true_
            bn      d9,b3,_true_
            br      _next_
_true_:     bro     r8
_next_:
            endm
```

56. DBNZO (Generate DBNZ instruction that will not generate an address error)

DBNZO expression

The DBNZO macro generates instruction codes that are equivalent to the DBNZ instruction, with no restriction on the difference between the location of the instruction and the branching destination in the same segment within the same source. The DBNZO macro uses the DBNZ instruction, the BR instruction, and the BRO instruction. expression describes the same value as in the DBNZ instruction.

Code generation macro:

```
; ***      Decrement direct byte and branch near relative address
;          if direct byte is not zero ***
dbnzo      macro          d9,r8
            local         _next_
            local         _true_
            dbnz          d9,_true_
            br            _next_
_true_:    bro            r8
_next_:
            endm
```

57. BEO (Generate BE instruction that will not generate an address error)

BEO expression

The BE macro generates instruction codes that are equivalent to the BE instruction, with no restriction on the difference between the location of the instruction and the branching destination in the same segment within the same source. The BE macro uses the BNE instruction and the BRO instruction. expression" describes the same value as in the BE instruction.

Code generation macro:

```
; ***      Compare immediate data or accumulator and branch
;          near relative address if equal ***
beo        macro          arg0,arg1,arg2
            local         _next_
            local         _txen_
            ifb            <<arg2>>
            bne            arg0,_next_
            bro            arg1
_next_:
            else
            bne            arg0,arg1,_txen_
            bro            arg2
_txen_:
            endif
            endm
```


58. BNEO (Generate BNE instruction that will not generate an address error)

BNEO expression

The BNEO macro generates instruction codes that are equivalent to the BNE instruction, with no restriction on the difference between the location of the instruction and the branching destination in the same segment within the same source. The BNEO macro uses the BE instruction and the BRO instruction. expression describes the same value as in the BNE instruction.

Code generation macro:

```
; ***      Compare immediate data or accumulator and branch
;          near relative address if equal ***
bneo      macro      arg0,arg1,arg2
            local      _next_
            local      _txen_
            ifb        <<arg2>>
            be         arg0,_next_
            bro        arg1

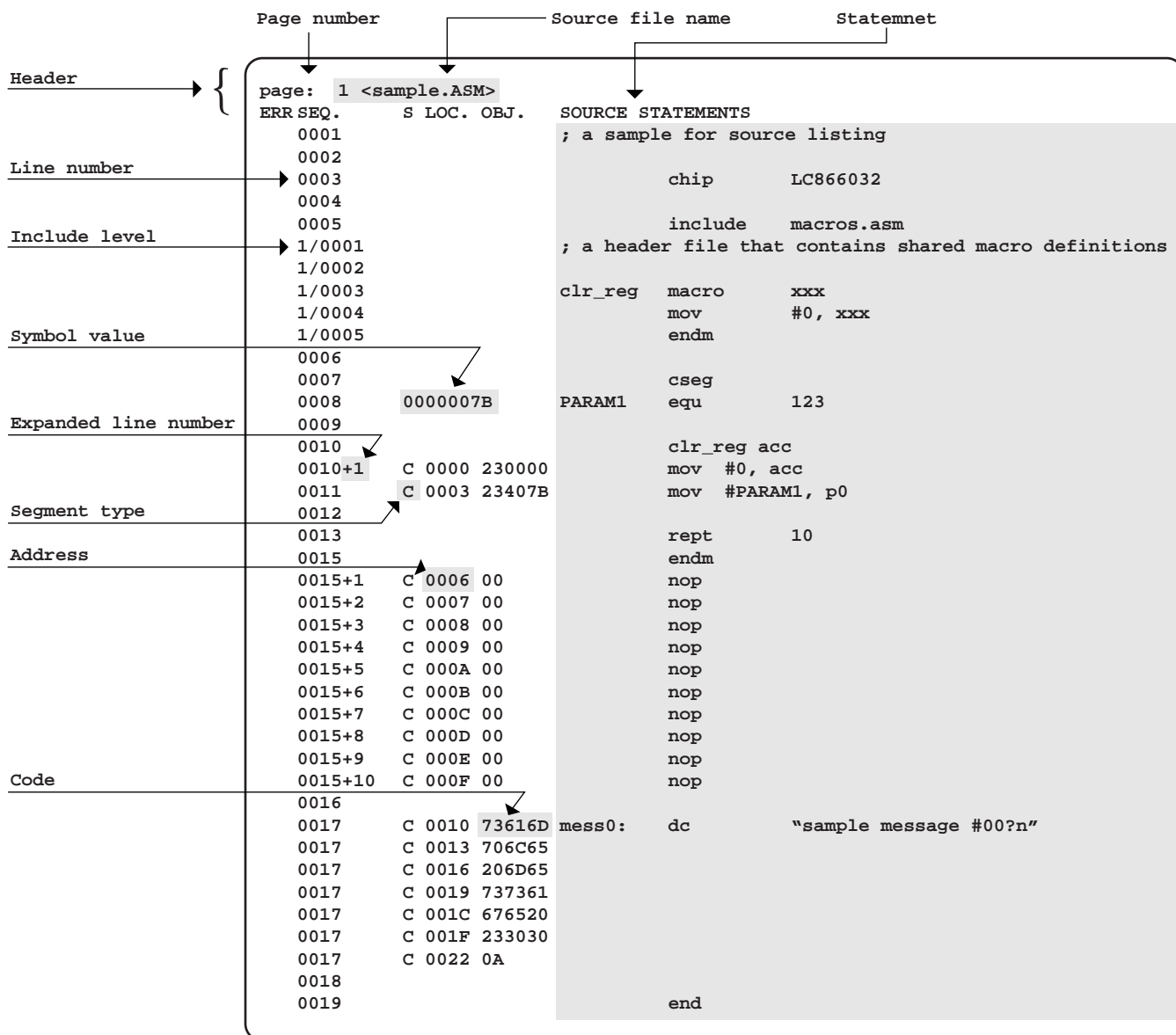
_next_:
            else
            be         arg0,arg1,_txen_
            bro        arg2

_txen_:
            endif
            endm
```


8. List File Format

The list file generated by M86K has the format shown below. Basically, this format shows the contents of the source file as is, with line numbers and machine language codes listed on the left. The placement of the lines and columns was designed with the format of the printout in mind. In short, one page consists of 60 lines (including the header), and one line consists of 132 columns. (If the source line is longer than 132 characters, it wraps around.) On the left side, the column positions are fixed so that various types of information can be displayed. Horizontal tabs that are included in the source lines are converted into spaces so that there is no visible change in positioning.

Visual Memory Unit (VMU) Environment Variables



Header:	The header appears at the start of each page, and consists of a blank line to allow space for binding, the page number, the source file name, and headings for each column position in the main body of the list.
Page number:	Pages are numbered sequentially, starting from "1."
Source file name:	This indicates the name of the source file for which assembly was specified. If the drive name and path name were also specified when the file was specified, they are shown as well.
Statement:	Indicates the contents of the source file, the results of expansion of a macro call when list output was not suppressed, and the results of expansion when include files are loaded.
Line number:	These are the line numbers in the source file (in decimal format). When a single line in the source file becomes two or more lines in the list file (for example, when the code portion spans several lines), the same line number is repeated.
Include level:	This indicates the nesting level of nested include files. This does not appear on lines that show the contents of the source file. This level is "1" for lines that show the contents of a file that is included directly from the source file. This digit becomes "2" for a file that is included from a level 1 include file. This number is separated from the line number by a slash.
Symbol value:	When a value is set in a symbol, if the value is definite at the time of assembly, that value is shown as an 8-digit hexadecimal number. If the value is not definite, nothing is shown.
Expanded line number:	This number indicates that a given line was not in the original source file, but was generated as a result of a macro call (including repeated macros). A sequential number, starting from "1," is added to the line number in the sequence of expansion. When one macro call is completed and a different one appears, the sequential numbers that are assigned start again from "1."
Segment type:	When the corresponding line generates code in CSEG or DSEG, the segment type is indicated by a single character. An upper-case "C" indicates "CSEG INBLOCK," a lower-case "c" indicates "CSEG FREE," and "D" indicates DSEG.
Address:	When the corresponding line generates code in CSEG or DSEG, the address where the first byte of the code is located is indicated with a four-digit hexadecimal number. Note that "address" as used here refers to the offset from the start of the segment in question.
Code:	When the code that is generated as a result of assembling the source line should be written in ROM, that code is depicted as two-digit hexadecimal codes, sufficient for the required number of bytes. A maximum of three bytes of codes are displayed for one line, with the two digits the farthest to the left representing the bytes corresponding to the smallest address. If the length of the code exceeds three bytes, the code is displayed on a newly generated line that has the same line number.

9. Specifying Files for Linking

There are two methods for starting up L86K and passing the necessary information to L86K.

- 1) Passing all of the information to L86K through the command line
- 2) Passing all of the information in response to the prompts that are displayed by L86K

Regardless of the method that was used to start up L86K, it can be forcibly terminated by either pressing CTRL+C (by holding down the CTRL key while pressing the C key) or pressing the STOP key.

1. File Name Specification

1.1 MS-DOS Version File Specification

Upper-case and lower-case letters can be used in any combination in a file name that is specified in the command line when starting up L86K, or in a file name that is given in response to the L86K prompts. For example, the following three file names are all equivalent:

```
sample.obj  
SAmpLE.OBJ  
SAMPLE.OBJ
```

Visual Memory Unit (VMU) Environment Variables

In addition, when a file name is specified with no extension, L86K uses the following default file name extensions.

File format	Default extension
Object file	.OBJ
Execution file	.EVA
Library file	.LIB
Option file	.OPT
Font file	.CGR
External ROM data file	.Hnn

nn is a numeric value that is specified through option -B.

1.2 UNIX Version File Specification

A distinction is made between upper-case and lower-case letters in a file name that is specified in the command line when starting up L86K, or in a file name that is given in response to the L86K command prompts. For example, the following three file names are all different:

```
sample.obj  
SAmpLE.OBJ  
SAMPLE.OBJ
```

In addition, when a file name is specified with no extension, L86K uses the following default file name extensions.

File format	Default extension
Object file	.obj
Execution file	.eva
Library file	.lib
Option file	.opt
Font file	.cgr
External ROM data file	.hnn

nn is a numeric value that is specified through option -B.

2. Specifying Parameters Through the Command line

```
L86K[option]objectfiles[, [evafilename][, [libraryfile]]][;]ø
```

1) option field

This field specifies the linker loader options that are described in section 2.1. When specifying an option, specify it in front of any field desired.

2) objectfiles field

This field specifies the names of the objects to be linked, the link start address, and the library names. At least one file name is required. When specifying multiple file names, separate the file names with a space character. If all of the file names do not fit on one line, place a plus (+) symbol at the end of the line. If the object file extension .OBJ is omitted from a file name, the .OBJ extension is automatically assumed. If “.LIB” is specified as the extension for a file name, the library is linked as well.

3) evafilename field

This field specifies the name of a file that can be executed on (downloaded to) the EVA86K. If this file name is not specified, the first file name that was specified in the objectfiles field is assumed, except that the extension is changed to .EVA.

4) Libraryfile field

This field specifies the name of the library. If no library is required, this field does not need to be specified.

Example:

```
A> L86K MAIN SUB0 SUB1,TEST,TEST.LIBØ
```

This command line links the object modules MAIN.OBJ, SUB0.OBJ, SUB1.OBJ, MAIN.OPT and MAIN.CGR. (In models that have internal EEPROM and fixed data, this line links the special option file described later as the option file.) If there are any undefined symbols when MAIN.OBJ, SUB0.OBJ and SUB1.OBJ are linked, a search for the same symbols as the undefined symbols is conducted in TEST.LIB, and the module that includes those symbols is linked.

3. Specifying Parameters in Response to Prompts

When specifying parameters for the linkage loader in response to prompts, input the following command on the command line.

```
L86K[option]Ø
```

L86K prompts the necessary input by displaying the following lines one at a time:

```
SANYO (R) LC86K series Linkage Loader Version 4.00
Copyright (c) SANYO Electric Co., Ltd. 1989-1995. All rights reserved.
Object modules[.OBJ]:
EVA filename[basefilename.EVA]:
Libraries[.LIB]:
Option filename[basefilename.OPT]:
Font filename[basefilename.CGR]:
```

L86K does not display the next line until a response has been input for the previous prompt. Section 1.1, "Specifying File Names," explains the rules for specifying file names in response to these prompts.

The responses to the prompts correspond to the fields in the L86K command line, except for Option filename and Font filename. (For details on the L86K command line, refer to section 1.2, "Specifying Parameters Through the Command Line.") The correspondence between the prompts and the command line fields is shown below:

Prompt	Command line field
Object modules	objectfiles
EVA filename	evafilename
Libraries	libraryfiles

When using the L86K prompts, after responses have been input for the above four items, input for the following two items is prompted.

Option filename :Input the name of the option file corresponding to the evafilename target chip.
Font filename :Input the font file name if the evafilename target chip belongs to the LC86100 Series, the LC864000 Series, or the LC868000 Series. (This prompt is not displayed if the target chip belongs to any series other than the LC86100 Series, the LC864000 Series, or the LC868000 Series.)

If the last character that is typed in the response line is a plus sign (+), the prompt moves to the next line, allowing you to continue inputting the response to the same prompt. In this situation, the plus sign must come at the very end of a complete file or library name, path name, or drive name.

Default Responses

To select the default response to the current prompt, simply press Return, without specifying a file name (or inputting anything else). The next prompt is then displayed.

To select the default response for the current prompt and all of the remaining prompts, type a semicolon (;) followed immediately by Return. Once the semicolon is input, it is not possible to input a response to any of the remaining prompts for that link session. Use this option in order to use the default responses or to save time. However, the semicolon cannot be input for the Object modules prompt, because that prompt has not default response.

The default responses to the L86K prompts are listed below.

Prompt	Default
EVA filename:	The first object file name that is specified for the Object modules prompt. The .OBJ extension is replaced with the .EVA extension.
Libraries:	No library search is conducted.
Option filename:	The file name specified for the EVA filename prompt. The .EVA extension is replaced with the .OPT extension. However, in models that have internal flash EEPROM and fixed data, the special option file described later becomes the default.
Font filename:	The file name specified for the Option filename prompt. The .OPT extension is replaced with the .CGR extension.

4. Files Referenced During Linking

L86K always references the following files during linking.

LC86K.LIB

Information concerning the system that is referenced by EVA86K, and the flash EEPROM access program are both stored in LC86K.LIB. L86K gets the system information concerning the link target CPU from LC86K.LIB during the linking process, and stores it in the EVA file.

In addition, if the option file name specification is omitted in the case of an object file for which the link target is a model that has internal flash EEPROM and fixed data, the following special option file is referenced:

LCnn00.OPT: nn is a two-digit integer that corresponds to the model name in question.

For details concerning models that have internal flash EEPROM and fixed data, refer to the users manual.

LC86K.LIB and LIBnn00.OPT must reside in a directory that is equivalent to the directory where L86K.EXE is stored, or in the directory that is set by the environment variable "PATH".

10. Specifying Linkage Loader Options

This section explains how to use the linkage loader options in order to specify and control the tasks performed by L86K. In the MS-DOS version, all options begin with the linkage loader option character, either "/" or "-". In the UNIX version, all options begin with the linkage loader option character "-".

1. Creating a HEX File for LC868000 Series External ROM

Option

-B=bank number

The -B option specifies the bank number of the external ROM data file in the LC868000 Series (WORLD EXTERNAL_DATA). Specify a hexadecimal value (from 1 to FF) for bank number. The bank number that is specified here becomes the data file extension.

Example:

```
A> L86K /B=1 SAMPLE;
```

The data file that is created here is SAMPLE.H01.

2. CSEG Loading Address Specification Method

Option

-C=address

The -C option, which is valid for the object module that is described immediately after this option, specifies the code segment loading address. Specify a hexadecimal value for address.

If this option is omitted, L86K loads the code segment of the object module at any suitable position.

3. DSEG Loading Address Specification Method

Option

-D=address

The -D option, which is valid for the object module that is described immediately after this option, specifies the data segment loading address. Specify a hexadecimal value for address.

If this option is omitted, L86K loads the data segment of the object module at any suitable position.

4. Enabling Duplicate Definition of DSEG Addresses

Option

-E

When the -E option is specified, no error is generated if multiple symbols are defined in the same address in DSEG.

5. No Distinction Between Upper-Case and Lower-Case

Option

-I

Under the default setting, L86K makes a distinction between upper- and lower-case, but if the -I option is specified, no distinction is made between upper- and lower-case.

6. Creating the Loading Map

Option

-P

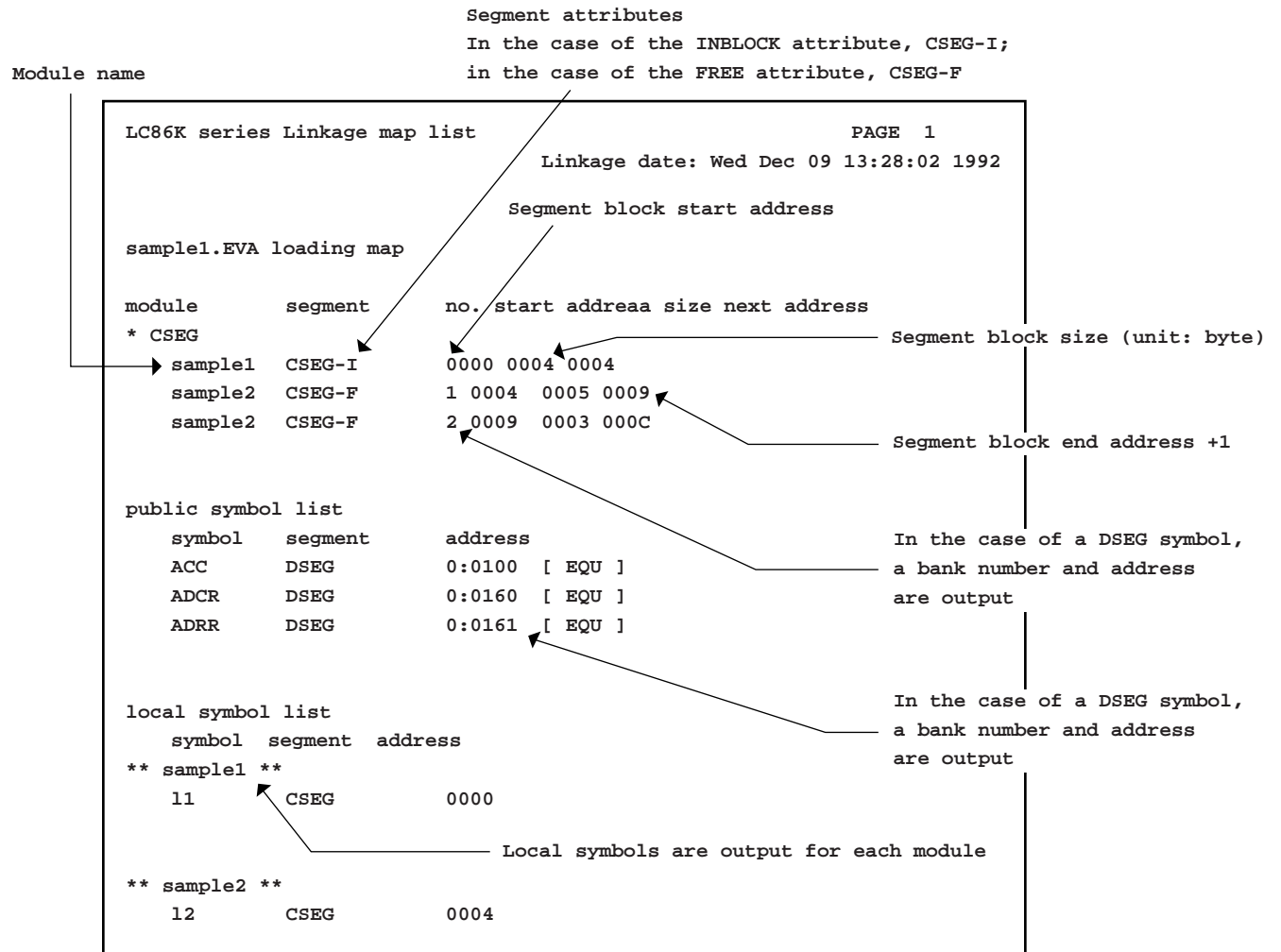
The -P option creates a file in which the link results map (a list of the link status of each segment and the placement of public symbols) is written. The file name of this map file is the name specified in the EVA file field in the command line or prompt, with the extension changed to .MAP. Note that no map file is created if a fatal error that makes it impossible for the linking process to continue occurs.

7. Creating a Local Symbol List

Option

-L

The -L option is valid only when combined with the -P option. The -L option adds a list of local symbols to each module in the map file.



8. Specifying Warning Messages Concerning Operand Data

Option

-W

The -W option displays a warning message when the value of an operand described for JMP, etc., is outside of the valid range. (In the case of JMP, the operand is 12 bits, so the valid range of values for the operand is 0 to 4095.) (The value that is stored in the instruction code consists only of valid bits; overflow bits are discarded.)

9. CSEG FREE Block Optimized Loading

L86K normally links (allocates) according to the CSEG section described in the object module and the sequence specified in the command module, but when it does so L86K aligns the executable file segment data (code segments) with 4096-byte boundaries. This results in a number of empty areas within the code segment area.

L86K has four placement functions that allocate each segment block in the optimal position in order to minimize the occurrence of such empty areas and therefore use memory efficiently.

Each of these loading methods is described below.

Option:

-A

The -A option loads all code segment blocks that are the target of the linking process in order, based on size. While doing so, if a segment block that crosses a 4096-byte boundary has the INBLOCK attribute, that block is aligned with the boundary; if the segment block has the FREE attribute, that block is placed normally without being aligned with the boundary.

Option:

-F

The -F option is valid only when used in combination with the -A option. After linking code segment blocks that have the INBLOCK attribute in the sequence specified in the command line, this option allocates code segments that have the FREE attribute in those empty areas that appear due to the reason described above. (If there is no more space to allocate code segments that have the FREE attribute, those segments are allocated after the final address in order, based on size.)

Option:

-O

The -O option is valid only when used in combination with the -A option. After linking code segment blocks that have the INBLOCK attribute in order, based on their size, this option allocates code segments that have the FREE attribute in empty areas. (If there is no more space to allocate to code segments that have the FREE attribute, those segments are allocated after the final address in order, based on size.)

Option:

-R

The -R option is valid only when used in combination with the -A option. After linking code segment blocks that have the INBLOCK attribute in order, based on their size, this option allocates code segments that have the FREE attribute in empty areas. If any two consecutive 4096-byte areas each contain empty area, the two empty areas are brought together by repositioning the second INBLOCK code segment; a code segment with the FREE attribute is then assigned to the newly formed area. (If there is no more space to allocate to code segments that have the FREE attribute, those segments are allocated after the final address in order, based on size.)

10. Specifying Symbol Sort Processing

Option:

-S

If the total number of public symbol definitions in all of the link target object files (including the relevant SFR definitions in LC86K.LIB) exceeds 8192 definitions, or if the total number of local symbol definitions in all of the link target object files exceeds 8192 definitions, symbol sort processing will become slower. A message that indicates the progress of the sort processing is displayed while processing is in progress:

Public(Local) symbol table: Sorting .. nn / nn blocks

Public(Local) symbol table: Sorting(merge) .. nn %

If the -S option is specified, however, symbol sort processing is not performed in these circumstances; instead, the following message is displayed and link processing is interrupted:

**** Link Error, Public symbol table overflow: nn symbols**

The number of symbols displayed here is the number of symbols in excess of 8192.

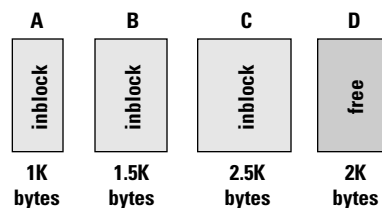
11. Object Placement

As explained in section "CSEG FREE Block Optimized Loading," the placement of objects when optimization is specified for the linking process differs from the placement that results during normal linking processing. There are four methods of optimization loading. The placement of objects when each of these optimization methods is specified is described below.

-A

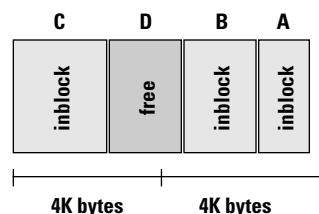
In CSEG, there are two placement specifications: the INBLOCK specification (which places objects within 4096-byte boundaries), and the FREE specification (which places objects with no regard for 4096-byte boundaries). When -A is specified, blocks are placed in the optimal position in order, based on size, regardless of the INBLOCK/FREE specification. (In this case, segments for which INBLOCK is specified are aligned with 4K-byte boundaries, while segments with the FREE specification are not aligned with the boundaries.)

For example, assume that the following group of objects exists:



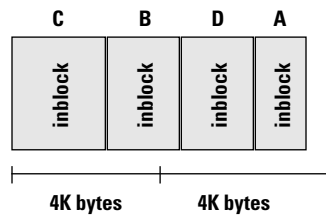
If these are linked with the -A option specified, the INBLOCK/FREE segment blocks are placed in the optimal positions in order, based on their size. The result is as follows:

L86K -A A B C D;



Visual Memory Unit (VMU) Environment Variables

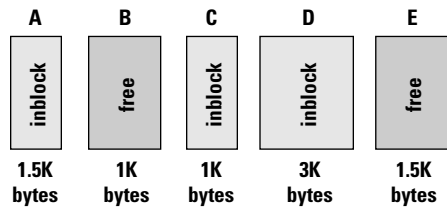
In the case of this example, because the segment (object D) that is placed on top of a 4096-byte boundary has the FREE attribute specified, object D is not aligned with the boundary. If object D had the INBLOCK attribute, the result would be as follows:



-A-F

When -A and -F are specified, the INBLOCK segment blocks are placed in the order described in the command line, and then the FREE segment blocks are placed in the optimal positions in order, based on their size.

For example, assume that the following group of objects exists:



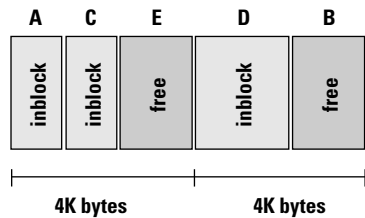
If these objects are linked with -A-F specified, A, C, and D are placed in the order indicated in the command line (D is aligned with a 4096-byte boundary), and then E and B are placed in the optimal positions in that order.

L86K -A-F A B C D E;

-A-O

When -A and -O are specified, the INBLOCK segment blocks are placed in the optimal positions in order, based on their size, and then the FREE segment blocks are placed in the optimal positions in order, based on their size.

For example, assume that the following group of objects exists:

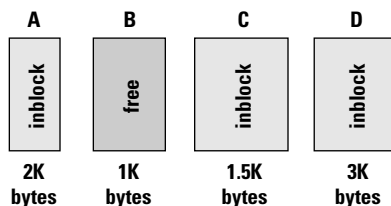


If these objects are linked with -A-O specified, D, A, and C are placed in the optimal positions in that order, and then B is placed in the optimal position.

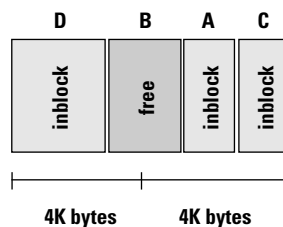
-A-R

When -A and -R are specified, the INBLOCK segment blocks are placed in the optimal positions in order, based on their size, and then, if any two consecutive 4096-byte areas each contain empty area, the two empty areas are brought together by repositioning the second segment; a segment block with the FREE attribute is then assigned to the newly formed area.

For example, assume that the following group of objects exists:

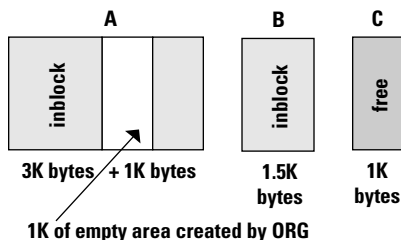


If these objects are linked with -A-R specified, D, A, and C are placed in the optimal positions in that order. A and C are then repositioned at the rear of their 4096-byte area. B is then placed in the empty area between D and A.

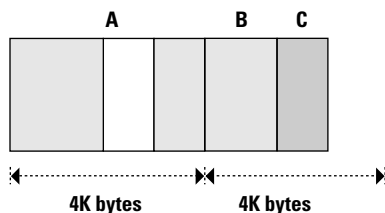


In addition, in all optimization methods, if an empty area is created within an object by the ORG pseudo instruction, that area also becomes available as a target for segment block placement.

For example, assume that the following group of objects exists:

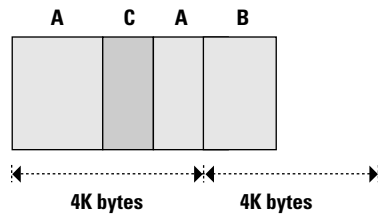


If these objects are linked normally without optimization, the result is as shown below.



If these objects are linked with optimization specified, ORG produces an empty area within segment block A, as follows:

Visual Memory Unit (VMU) Environment Variables



Furthermore, if the optimization specification and the loading address specification (-C option) are both made, and the first segment of that file has the FREE specification, the placement of that segment block (only) follows the loading address specification. (Subsequent FREE blocks are subject to optimization.)

12. Errors

1. Fatal Errors

If a fatal error is detected while the linking process is in progress, L86K displays a message on the VDT and interrupts processing. The error messages that are displayed by L86K are listed below.

Chip name unmatched

An attempt was made to link object modules for different chips.

Data file specified

A data file was specified for EVA file creation.

Data segment size exceeds

An attempt was made to link DSEG objects in excess of the RAM size.

External undefine symbol

External reference symbol not found.

Illegal bank number specified

There is an error in the external ROM bank number specification.

Illegal file format

The specified file is not intended for the LC86K Series.

Illegal option specified

Illegal option was specified.

Internal module not specified

No internal program file was specified in a link.

Loading address multiple assignment

An attempt was made to assign different objects to the same address.

No such file or directory

Specified file not found.

Program file specified

A program file was specified for the creation of an external ROM data file.

Public symbol multiple define

Duplicate definition of a public symbol.

Segment size exceeds

An attempt was made to link objects that would exceed the segment size.

WORLD attribute unmatched

A program file with the WORLD INTERNAL or WORLD EXTERNAL attribute co-exists with a data file that has the WORLD EXTERNAL_DATA attribute.

2. Non-Fatal Errors

If a non-fatal error is detected while the linking process is in progress, L86K displays a message on the VDT and continues processing. The error messages that are displayed by L86K are listed below.

Cannot access file: LC86K.LIB

The library LC86K.LIB, in which the reserved words are registered, does not exist. LC86K.LIB is a library in which reserved words for each chip are registered, and must reside in either the current directory, or in the directory that is pointed to by the environment variable PATH.

Module not in library

Reserved words for the target chip are not registered in LC86K.LIB.

Operand data overflow

The value that was described in an operand field within the prescribed range. (The range differs according to the statement.)

Operand data type mismatch

An illegal segment symbol was specified in the operand field.

13. Program Startup

There are two methods for starting up LIB86K and passing the necessary information to LIB86K.

- 1) Passing all of the information to LIB86K through the command line
- 2) Passing all of the information in response to the prompts that are displayed by LIB86K

Regardless of the method that was used to start up LIB86K, it can be forcibly terminated by either pressing CTRL+C (by holding down the CTRL key while pressing the C key) or pressing the STOP key.

1. File Name Specification

1.1 MS-DOS Version File Specification

Upper-case and lower-case letters can be used in any combination in a file name that is specified in the command line when starting up LIB86K, or in a file name that is given in response to the LIB86K prompts. For example, the following three file names are all equivalent:

```
sample.obj.  
SAmple.OBJ.  
SAMPLE.OBJ
```

In addition, when a file name is specified with no extension, LIB86K uses the following default file name extensions.

File format	Default extension
Library file	.LIB
Object file	.OBJ
List file	None

1.2 UNIX Version File Specification

A distinction is made between upper-case and lower-case letters in a file name that is given in response to the LIB86K command prompts. For example, the following three file names are all different:

```
sample.obj.  
SAmpLE.OBJ.  
SAMPLE.OBJ
```

In addition, when a file name is specified with no extension, LIB86K uses the following default file name extensions.

File format	Default extension
Library file	.lib
Object file	.obj
List file	None

2. Specifying Parameters Through the Command line

LIB86K[option] oldlibrary[commands] [, [listfile] [, [newlibrary]]] [:]ø

1) option field

The only option that can be specified in the option field is /?.

2) oldlibrary field

This field specifies the library that is the target of processing. This field cannot be omitted. If the extension of the library file is .LIB, the extension may be omitted. However, if the user's library file has an extension other than .LIB, the extension may not be omitted. Because there is no default library file, if no library file is specified, an error message is output. Furthermore, if a non-existent file name is specified, the following prompt is displayed:

```
Library file does not exist. Create? (y/n)
```

If creating a new library file, input "Y". If any character other than "Y" is input, processing is interrupted and control returns to the OS. If a semicolon is appended at the end of the name of an existing library file, a library conformity check is performed. The conformity check determines whether all of the modules in the library can be used. If an error is detected, an error message is output.

3) commands?????

The commands field is used to specify command symbols such as "+", "-", "-+", "*", and "-*" as instructions for program operation. Multiple operations can be performed by specifying one object file name or module name with one command. If a command is omitted, changes cannot be made to a library file.

Command	Meaning
+:	This is the module addition command symbol. The module in the object file that is described immediately after the command symbol is added at the end of oldlibrary. It is not possible to link libraries together.
-:	This is the module deletion command symbol. The module that is specified immediately after the command symbol is deleted from oldlibrary.
-+:	This is the module replacement command symbol. The module in the object file that is described immediately after the command symbol is substituted for a module in oldlibrary. Module replacement is accomplished by deleting the currently existing module and then adding the module with the same name at the end of the library.
*:	This is the module copy command symbol. The module that is specified immediately after the command symbol is searched for in oldlibrary, and then the contents are written to an object file with the same name. The copied module remains in oldlibrary.
-*:	This is the module move command symbol. The module that is specified immediately after the command symbol is searched for in oldlibrary, and then the contents are written to an object file with the same name. This is the same operation as the module copy operation, except that the module that was copied does not remain in oldlibrary.

4) listfile field

The Listfile field specifies the file that outputs the public symbol list, external reference symbol list, and the module name list in the library. If this field is omitted, the data is displayed on the standard output.

5) newlibrary field

The newlibrary field specifies the name of the library that is the output target. If this field is not specified, the extension of oldlibrary is changed to .BAK before beginning processing, and then the data in oldlibrary is saved as newlibrary.

Option Specification

/ ?

Specifying this option outputs a help message on the screen. In the UNIX version, this option must be enclosed in double quotation marks.

Command Line Execution Examples

Example 1: LIB86K HOME-+ROM;ø

In this example, a module named ROM is deleted from the library named HOME, and the object file ROM.OBJ is added to the end of the library.

Example 2: LIB86K HOME-ROM+ROM;ø
LIB86K HOME+ROM-ROM;ø

In the top line above, a module named ROM is deleted from the library named HOME, and then the object file ROM.OBJ is added to the library. In the bottom line, however, the object file ROM.OBJ is added to HOME and then the ROM module is deleted. Therefore, in the case of the top line, a module named ROM remains in the library, while in the case of the bottom line, there is no module named ROM remaining in the library. This is because the processing is performed in the same sequence in which the command symbols are specified.

Example 3: LIB86K HOME,LCROSS.PUBø

In this example, a conformity check is performed on HOME.LIB and then a cross-reference file named LCROSS.PUB is created.

Example 4: LIB86K FIRST -*STUFF*MORE,,SECONDø

This example copies the module STUFF from the library FIRST.LIB to a file named STUFF.OBJ and then deletes the module STUFF from the library. The module MORE is also copied from the library to MORE.OBJ, but also remains in the library. The new library is SECOND.LIB, and is the same as FIRST.LIB except that the STUFF module has been deleted.

3. Operation Using the Prompts

If only LIB86K is input on the command line, as shown below, then each field can be input one at a time in response to the prompts that are displayed on the screen.

```
LIB86K[option]Ø
```

For LIB86K, the following fields are displayed one at a time:

```
Library name:
Operations:
List file:
Output library:
```

As the prompts are displayed one at a time, LIB86K waits for input from the user. As each item is input, the next prompt is displayed and then LIB86K waits for input again.

Each of the responses to the prompts correspond to each of the fields in the command line. The correspondence between the prompts and the command line fields is shown below.

Prompt	Command line field
Library name:	This corresponds to the oldlibrary field. If a semicolon (";") is input after the file name, a library conformity check is performed.?
Operations:	This corresponds to the command field.
List file:	This corresponds to the listfile field.
Output library:	This corresponds to the newlibrary field.

Prompt Line Expansion

If an ampersand (&) is input at the end of the input at the Operations prompt, the Operations prompt is displayed again, allowing additional processing to be specified.

Default Response

Default values are set for items other than the Library name prompt. The default setting can be selected by inputting either a semicolon or just pressing the Return key. The defaults for each prompt are shown below.

Prompt	Default value
Operations:	Makes no changes to the library file.
List file:	Selects the standard output for the list file output destination. No list file is generated.
Output library:	Sets the original name as the output library file name.

14. Errors

The error messages are explained below.

cannot access file

LIB86K cannot open the specified file.

cannot create new library

Either the disk or root directory is full, or else the library file is a read-only file that is write-protected.

cannot rename old library

Because the .BAK version is read-only and is protected, LIB86K cannot rename an old library to a name with the .BAK extension.

comma or newline missing

In the command line, an expected comma or Return was not found.

error reading from library

LIB86K can not read data from the specified library file.

error writing to new library

The disk or root directory is full.

insufficient memory

Could not allocate the memory needed in order to run LIB86K.

interrupted by user

The user halted LIB86K by pressing CTRL+C.

invalid library header

The input library file has an invalid format.

module not in library; ignored

The module specified for replacement was not found in the library.

output-library specification ignored

In addition to a new library name, an output library was specified.

syntax error : illegal file specification

A command operator such as the minus sign was specified without a module name.

15. Cross-Reference List

The cross-reference list format is shown below.

```
LC86K series Library Analysis List                                PAGE  1
                                                                Tue Feb 18 13:56:12 1992
Number of Module count:      2      Library create date: Wed Oct 16 15:34:53 1991
                                                                Library update date: Tue Feb 18 10:55:23 1992
Including Modules:            1      2
```

```
-----
Module name:                1      Source name:                1.ASM
Assembler name:  SASM 1.0      Assembly date:    Tue Oct 22 15:54:43 1991
Target chip name: LC866232
Including Public symbols:
Including External symbols:
                                test          sample          label1
```

```
-----
Module name:                2      Source name:                2.ASM
Assembler name:  SASM 1.0      Assembly date:    Tue Oct 22 15:54:43 1991
Target chip name: LC866232
Including Public symbols:
Including External symbols:
                                label1          label2          label3
```


16. Program Startup

1. File Name Specification

Upper-case and lower-case letters can be used in any combination in a file name that is specified in the EVA2HEX command line. For example, the following three file names are all equivalent:

```
sample.eva
SAmpLE.EVA
SAMPLE.EVA
```

In addition, when a file name is specified with no extension, EVA2HEX uses the following default file name extensions.

File format	Default extension
EVA file	.EVA
HEX file	.HEX

2. Parameter Specification Method

```
EVA2HEX[option] EVA_filename [HEX_filename]ø
```

1) option field

This field specifies the options described in section 1.3. Specify the option field after the command name.

2) EVA_filename field

This field specifies the name of the file after debugging is completed (the file with the ".EVA" extension). (This file is known as the EVA file.)

3) HEX_filename field

This field specifies the name of the Intellec HEX format file. (This file is known as the HEX file.) If the HEX_filename is omitted, it is identical to EVA_filename. If an external ROM data file is converted, the extension is .H00.

* EVA2HEX does not support prompts for parameter input.

Visual Memory Unit (VMU) Environment Variables

Startup Example 1: A>EVA2HEX PROG012Ø
EVA file >> HEX file, external ROM HEX file
PROG012.EVAPROG012.HEX, PROG012.H00

Startup Example 2: A>EVA2HEXØ
The following message (simple help) is displayed.
SANYO LC86000 Series EVA-file to HEX-file generator V1.00A
Copyright (C) SANYO Electric Co.,Ltd. 1992
Usage: eva2hex [optiona] EVA filename [HEX filename]
Optiona: /I ... information on/off (default: on)

3. Option Specification

The option begins with "/". "-" cannot be used.

Option

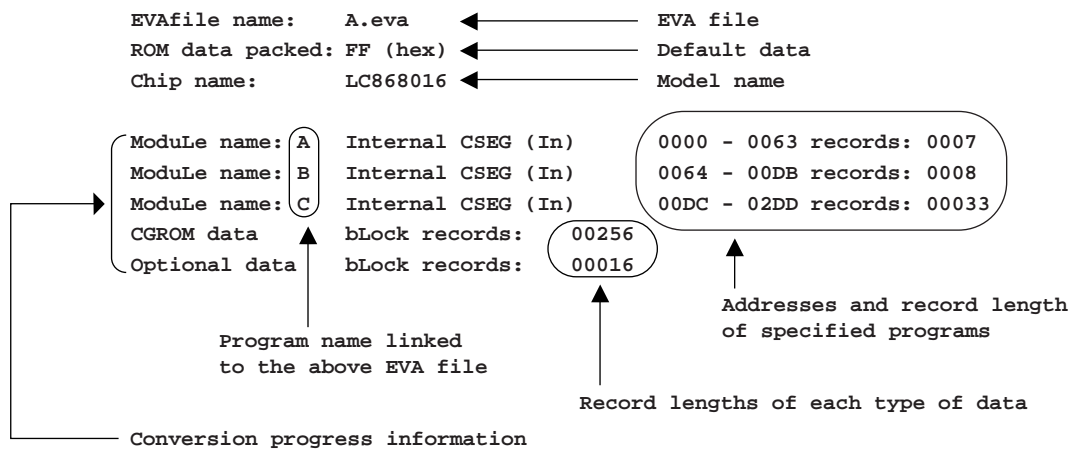
/I

The /I option disables the display (on the VDT) of information on the progress of the conversion process when converting an EVA file to a HEX file. If this option is not specified, information on the progress of the conversion process is displayed while conversion is in progress.

Monitor Display Example:

SANYO LC86000 Series EVA-file to HEX-file genetator V1.00A

Copyright (C) SANYO Electric Co.,Ltd. 1992



17. Errors

1. Fatal Errors

If a fatal error is detected while EVA2HEX is running, EVA2HEX displays a message on the VDT and interrupts processing. The error messages that are displayed by EVA2HEX are listed below.

```
Error message: Fatal error : .....messages.....  
'filename' File not close.  
    Cannot close file filename.  
'filename' File not create.  
    Cannot create file filename.  
'filename' File not open.  
    File filename not found.  
'filename' not EVA file format.  
    File filename is not an EVA-format file.  
'filename' user disk full.  
    Disk became full while writing filename.  
Chipname undefined.  
    Model name in EVA file is incorrect.  
ROM size over. (ROM size: XXXX)  
    The program size exceeded the ROM size.  
Tablename allocation error.  
    Memory allocation for tablename failed due to insufficient memory.
```


Visual Memory Unit (VMU)

VMU-BIOS Specifications

1. VMU-BIOS Specifications

1. Outline

This document describes the System BIOS functions of the backup memory system “VMU” designed for the new-generation game machine (preliminary).

2. VMU Outline

“VMU” stands for “Visual Memory Unit”. The VMU is a backup memory cartridge equipped with a liquid-crystal display and operation buttons.

When connected to a dedicated controller in the new-generation game machine (preliminary), the VMU serves as a file backup memory and it can also display game sub screens on its LCD.

When not connected to the new-generation game machine, the VMU can function as a stand alone unit that allows displaying and deleting stored files. Two VMU units can be connected to allow file transfer.

Another application of the VMU is as a highly portable miniature game machine, using simple application programs downloaded from the new-generation game machine to the VMU. (Such application programs are called “user programs”.)

2.1 System-BIOS Outline

The functions described above are implemented by several programs that are contained in an internal ROM on the VMU. These programs are called “OS programs”. OS programs consist of subroutines which can be called by user programs. Two program types (system program and header) are used to call up subroutines. The entire system consisting of OS programs, system programs, and headers is called the “System-BIOS”.

OS program subroutines are divided into subroutines that serve mainly for accessing the flash memory and subroutines for calculating time data. Application developers can use the System-BIOS to call these subroutines in user programs. This makes it easy for application developers to use VMU functions without having to deal with detailed VMU specifications.

3. Memory Space

VMU uses two types of memory space: internal memory space and external memory space.

Internal memory space consists of the internal program area and internal RAM. The external memory space is made up of flash memory.

The internal program area is 64 kilobytes and contains the OS programs and system programs. User programs can reference this area as needed. The internal program area is allocated as shown in the memory map in Fig. 3.1. (For information on OS programs and system programs, please refer to section 5.)

The flash memory space is 128 kilobytes, divided into two banks of 64 kilobytes each. Bank-0 is the program area and bank-1 the data area. User programs are stored in the program area. A memory map of the flash memory is shown in Fig. 3.2. (For information on the internal and external program area and BIOS usage, please refer to section 4 and the following sections.)

The internal RAM has a memory space of 1222 bytes, divided into the following four sections: main RAM, special register area, LCD video RAM (XRAM), work RAM (VTRBF).

The main RAM is 512 bytes, divided into two banks of 256 bytes each. Because bank-0 is reserved for the System BIOS, user programs are generally prohibited from writing to bank-0 (except for certain cases listed in appendix 1).

The special register area is allocated to VMU specific registers (timer register, LCD control register, etc.). For information on registers and corresponding addresses, please refer to the VMU user's manual.

The LCD video RAM (XRAM) consists of three banks which serve for storing LCD image data. (For information on RAM usage, please refer to the VMU user's manual.)

The work RAM is 512 bytes and serves as a buffer when VMU carries out data transfer according to the Maple Bus protocol. When the VMU is operating as a standalone unit and data transfer according to the Maple Bus protocol is therefore not being carried out, user programs can use this area.

A memory map of the internal RAM is shown in Fig. below. (The access procedure for the work RAM differs from normal RAM access. For information, please refer to the VMU user's manual.)

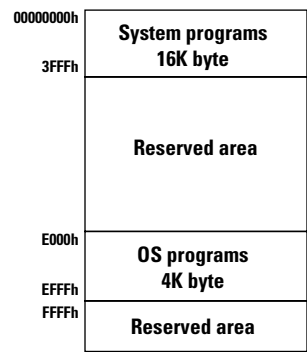
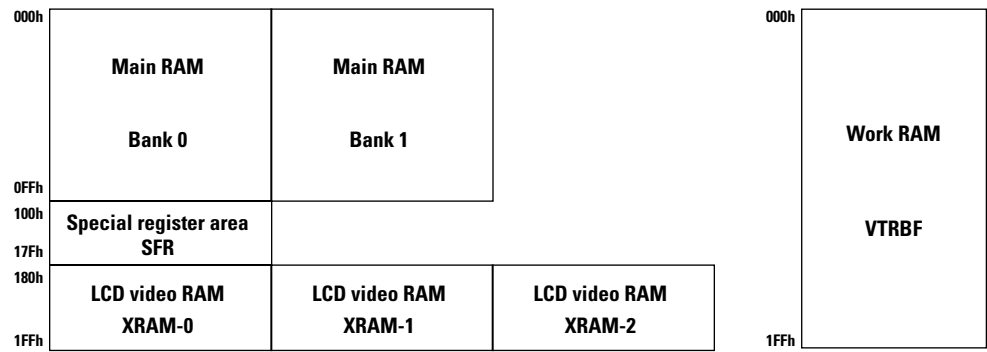


Figure 1.1 Internal program space



Figure 1.2 Flash memory space



*Bank 0 of the main RAM is reserved for system programs. Except for special cases, user programs cannot use this area.

Figure 1.3 Internal RAM space

4. System BIOS Functions

This section explains the System BIOS functions provided for VMU.

User programs running on the VMU can access System BIOS functions by calling special subroutines. However, there are certain limitations on which System BIOS functions (subroutines) can be called by user programs.

The following functions are provided by the System BIOS.

- System initialization

- VMU initialization function

- VMU execution mode selection

VMU comes with the following three execution modes:

- 1) Game data and user program management and editing
- 2) User program startup and return
- 3) Time display and adjustment

For details on execution mode selection, please refer to appendix 2.

- Subroutines

- Flash Memory Access Functions

- 1) Flash Memory Page Data Readout
- 2) Writing to Flash Memory
- 3) Flash Memory Verify

- Clock Function

- 1) Clock Countup Timer

For details on VMU initialization, please refer to section 6. For details on subroutines, please refer to section 7.

5. System BIOS Data and Memory Allocation

VMU comes with certain programs for using the System BIOS functions. These programs can be classified into the following three types:

- 1) OS programs
- 2) System programs
- 3) Header

5.1 Program Layout

The actual programs are arranged in memory as follows.

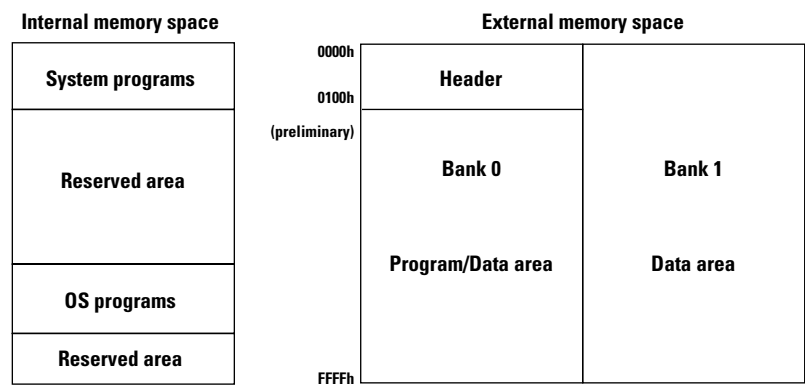


Figure 1.4 VMU memory map

Details are explained below.

System programs

System programs are required for using the VMU as a memory device. Major system programs are the file management system, clock functions, and programs for data transfer according to the Maple standard. A program for calling subroutines from the external memory space (user programs) is also located here. (A flow diagram showing the call-up process of specified subroutines is shown in section 5.2.)

The VMU initialization program is located in this area. For details on the initialization program, please refer to section 6.

OS programs

The VMU program subroutines are located here. For information on subroutines that can be called by user programs, please refer to section 7.

The method of accessing to this area is also shown in section 5.2.

Header

Contains information about internal memory space processing routines and return procedures from the internal memory space. Because this area also contains interrupt vectors for internal use by user programs, its source code is being made available to application developers. It also contains information about return from user programs to the mode selection screen. User programs must use this information to return to the file management system. (For details on mode selection screen, please refer to Appendix 2.) Within the given specifications, the area content may be modified by developers.

5.2 Subroutine Call Flow

This section explains the operation flow that occurs when a user programs calls OS program subroutines and then returns to the user program. An actual flow diagram is shown in Fig. below

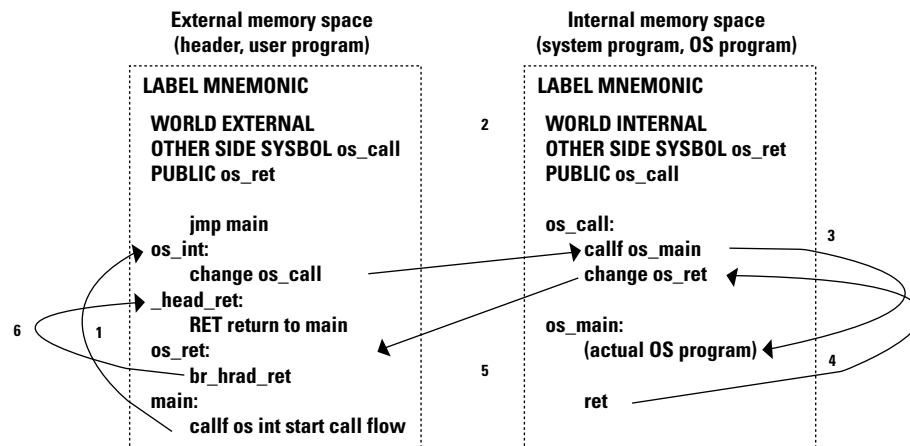


Figure 1.5 OS program call flow

Label processing description

- external memory space
 - main: Main program in user program
 - os_int: This subroutine shifts processing to internal memory space.
In the example, processing passes to the internal memory space when the subroutine is called, and the main program resumes upon return from the internal memory space. This subroutine is included in the header.
 - os_ret: Subroutine for returning to external memory space.
The “change” command serves to return to this label from the internal memory space. After returning, processing moves to the interrupt return routine in the header.
- internal memory space
 - os_call: Serves to call an OS program and return to the external memory space.
After the OS program subroutine has executed, processing returns to the external memory space.
 - os_main: Main OS program which executes the various subroutines.

The sample flow shown in Fig. 5.2 assumes that a user program is executing in the external memory space.

- 1) When wishing to use an OS program during execution of an external program, call the “os_int” subroutine. Interrupt processing routines which need to jump to an OS program contain an “os_int” subroutine.
- 2) The “change” command in the os_int subroutine jumps to the OS program call routine (os_call) placed in the internal program area.
- 3) The OS program call routine calls the actual OS program subroutine (os_main). From this point on, the OS program starts to execute.
- 4) When the OS program execution is finished, the RET command jumps to the next address of the call command in the OS program call routine. In the OS program call routine, the call command is always followed by a change command which moves processing to the external program area.
- 5) After returning from the OS program subroutine, the change command passes processing over to the external program. This program contains a subroutine (os_ret) that is called when returning from an internal program. The subroutine position is fixed. These programs are distributed to application developers as a library. Such programs are called headers. (The sample program contains the headers “os_int” and “os_ret”.)
- 6) From the above described external program return routine, processing returns to the “os_int” subroutine and then by the RET command to the main program (main).

Note: Label names in the sample program are all preliminary. Label names will be different in the actual System-BIOS.

* “change” command

The “change” command serves to move processing from the external memory space to the internal memory space, or from the internal memory space to external memory. By executing this command, a program that is currently executing in internal memory space (or external memory space) moves to external memory space (or internal memory space). The program counter is reset to the specified label (or address).

5.3 Returning From User Program to Mode Selection Screen

When a user program is executing, if the user presses the MODE button on VMU, the user program will terminate immediately and processing will return to the mode selection screen.

This section explains the operation flow from user program to the mode selection screen when the MODE button is pressed while a user program is executing.

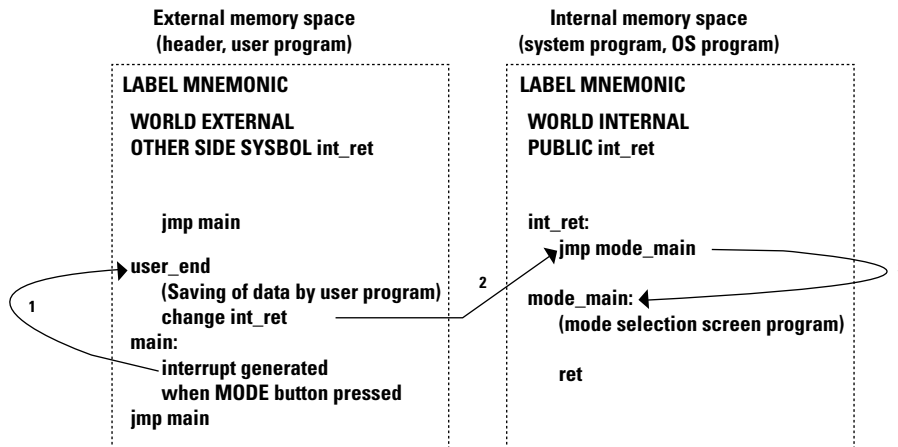


Figure 1.6 Operation flow of returning to mode selection

Label processing description

- external memory space
 - main: Main program in a user program.
A user program must contain description to allow for pressing of the MODE button to jump to the OS program return subroutine.
 - user_end: Subroutine to terminate a user program in execution and move processing to the OS program. If data in the executing user program needs to be saved, then be sure to include this information in the user program so that the subroutine will save it before returning to the OS program. (The OS program does not keep data.)
- internal memory space
 - int_ret: Return routine to serve as entry to returning to the internal memory space when a user program terminates. When processing returns to the internal memory area, the mode selection program will start.
 - mode_main: Mode selection program.
For details on mode selection specification, please refer to Appendix 2.

The sample program flow in Fig. 5.3 assumes the user program is executing in the external memory space.

- 1) While an external program is executing, pressing the MODE button will jump to the user_end subroutine. In the user_end subroutine, the “change” command will shift processing to the internal memory space. Therefore, if data in the executing user program needs to be saved, then be sure to save it before executing the “change” command.
- 2) When program control jumps from the user program to the user_end subroutine, the “change” command inside the user_end subroutine will shift processing to the mode_ret subroutine in the internal memory space.
- 3) When processing moves from the external memory space to the mode_ret subroutine, the mode selection program will start.

* “change” command

The “change” command serves to move processing from the external memory space to the internal memory space, or from the internal memory space to external memory. By executing this command, a program that is currently executing in the internal memory space (or external memory space) moves to the external memory space (or internal memory space). The program counter is reset to the specified label (or address).

5.4 VMU Initialization

This section explains the initialization that is performed at VMU startup.

The VMU is automatically initialized in the following cases.

1. VMU is connected to new-generation game machine, and power to new-generation game machine is turned ON
2. Reset switch on VMU is pressed
3. Battery is inserted in VMU

Initialization comprises the following steps.

- 1) Clear main RAM
 - Write ‘00h’ to entire main RAM area (bank 0, bank 1).

* Initialization does not change XRAM values.

All registers are initialized by a hardware reset first, and then again by software. For information on the register values after a hardware reset, please refer to the VMU user's manual.

- 2) Set system clock and cycle time
 - Switch system clock to sub-clock (crystal quartz oscillator).
 - Set cycle time to 1/6 system clock.
(The cycle time is used as reference for command execution. For details, please refer to the VMU user's manual.)
- 3) Set base timer
 - Select 14-bit base timer mode.
 - Switch base timer clock to sub-clock (crystal quartz oscillator).
 - Enable base timer 0 interrupt and start counting.

For details regarding base timer 0 operation, please refer to the VMU user's manual.

The base timer 0 is used by the clock function. For details regarding the clock function, please refer to section 7.3.

4) Set master interrupt

- Enable master interrupt.

(The master interrupt flag controls enabling/disabling of all interrupts with “High level” and “Low level” priority.)

5) Set LCD driver

- Activate LCD controller.
- Set LCD clock to 1/2 of LCD driver input clock.
- Set LCD start address to '000h' of XRAM.
- Set character register.
- Set time allocation register.
- Set LCD to ON.

6) Set port 1

- Set port 1 to all-bit input.
- Set bit 7 of port 1 to audio output pin.

* After initialization, bit 7 of port 1 is set to input mode. Therefore a user program will need to again select the output mode.

- Set bit 5 – bit 0 of port 1 (serial interface for VMU) to synchronous operation. For details regarding the synchronous serial interface, please refer to the VMU user's manual.

7) Set port 3

- Pull up all bits of port 3.
- Set port 3 to all-bit input.
- Enable interrupt triggering and HOLD mode cancel by port 3.
- Enable interrupt trigger request by port 3.

8) Initialize Maple Bus interface circuit

- Initialize Maple Bus interface circuit.

9) Set work RAM

- Enable use of work RAM.

6. Subroutine Description

This section describes the subroutines available in the System BIOS.

6.1 Flash Memory Access Functions

The following subroutines are available for flash memory access.

- 1) Flash Memory Page Data Readout
Read 128 bytes of data from the flash memory space.
- 2) Write to Flash Memory
Write 128 bytes of data to the flash memory space.
- 3) Flash Memory Verify
Verify data written to the flash memory.

* When accessing the flash memory, the main clock in use must be switched to 600 kHz. For details, please refer to the next section.

Precautions for Using Flash Memory Access Subroutines

When accessing the flash memory space, the following points must be observed.

VMU uses three types of system clock as reference for command execution (see Fig. 7.1).

When VMU is operating as a standalone unit, the quartz oscillator clock (32 kHz) will normally be used. However, for accessing the flash memory, the clock must be switched to the internal (RC) oscillator (600 kHz) before calling a flash memory access subroutine. After subroutine execution is completed, switch back to the previously used clock.

For information on the timing for clock switching, see Fig. below.

System clock source	Oscillation frequency	Command cycle time
Ceramic (CF) oscillator	6 MHz	1.0 us
Internal (RC) oscillator	600 kHz	10.0 us
Quartz (X'TAL) oscillator	32 kHz	183.0 us

Figure 1.7 *System clock table*

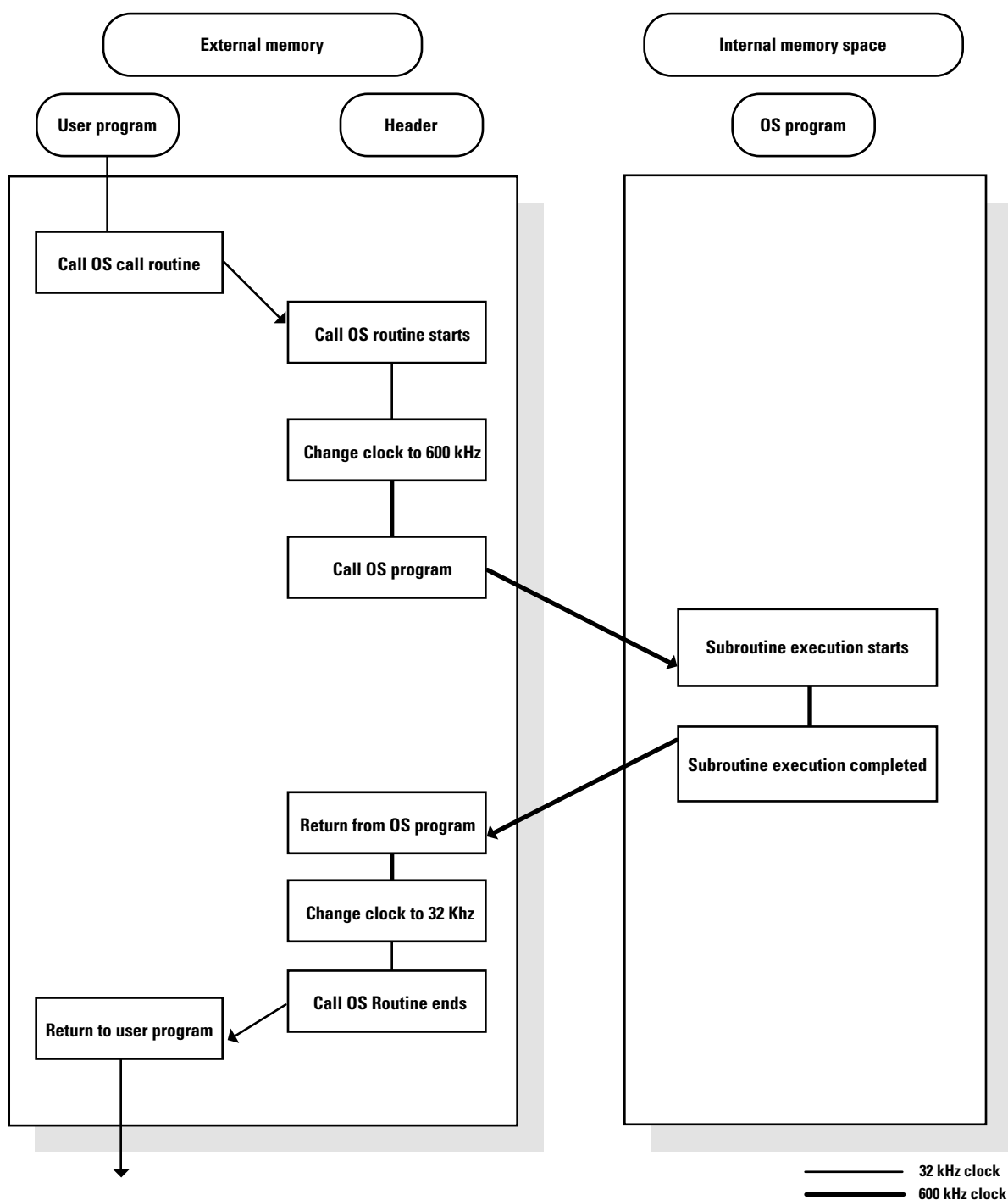


Figure 1.8 Flow diagram for clock switching during flash memory access

Flash Memory Page Data Readout

Subroutine name: fm_prd_ex (org 0120h)

Arguments: High-order start address for flash memory read: fmadd_h (RAM bank-1 07Eh)
Low-order start address for flash memory read: fmadd_l (RAM bank-1 07Fh)
Bank address for flash memory read: fmbank (RAM bank-1 07Dh)

Return value: Read data (128 bytes): 080h - 0FFh of RAM bank-1

Function: Read one continuous page of data (128 bytes) from specified address

Description: By calling this subroutine, a program can read one page of data (128 bytes) from flash memory.

Before using this subroutine, the following settings must be made.

- Select RAM bank to use
 - (1) Select RAM bank-1 (Set bit 1 of PSW to "1")
 - For information on the PSW register, please refer to the VMU user's manual.
- Set start address for flash memory read
 - (2) High-order address (8 bit): set to fmadd_h (07Eh of RAM bank-1)
 - (3) Low-order address (8 bit): set to fmadd_l (07Fh of RAM bank-1)
- Select flash memory bank to read
 - (4) Select flash memory bank-0
 - (Set 07Dh of RAM bank 1 to '00h')
 - * If another value is set, normal operation is not assured.

The read data are written to 080h - 0FFh of RAM bank-1.

When making read settings, observe the following points.

- Data extending to 2 pages cannot be read. The read start address must therefore always be set to the beginning of each page.

The start address of each page can be calculated according to the following equation:

$$\text{start address value (2 byte)} = 080\text{h} \times \text{page number (0 - 511)}$$

(Because readout is performed in single-page units, bit 0 – bit 6 of the lower-level address must always be set to "0". If an address other than the start address of a page is set, normal operation is not assured.)

- The read-out data overwrite any previous content of the RAM.

* Register values after subroutine completion

Note that the following (memory) registers will have different values before the subroutine is called and after the subroutine has completed:

- ACC (accumulator)
- TRL (table lookup register lower byte)
- TRH (table lookup register higher byte)
- r0 (RAM indirect address register)

*About pages

Beginning at the top, the flash memory space is subdivided into 128-byte units called pages. Flash memory is managed in page units. Because 1 bank of the flash memory space is 64 kilobytes, it has 512 pages.

“fm_prd_ex” execution is shown in Fig. 7.3.

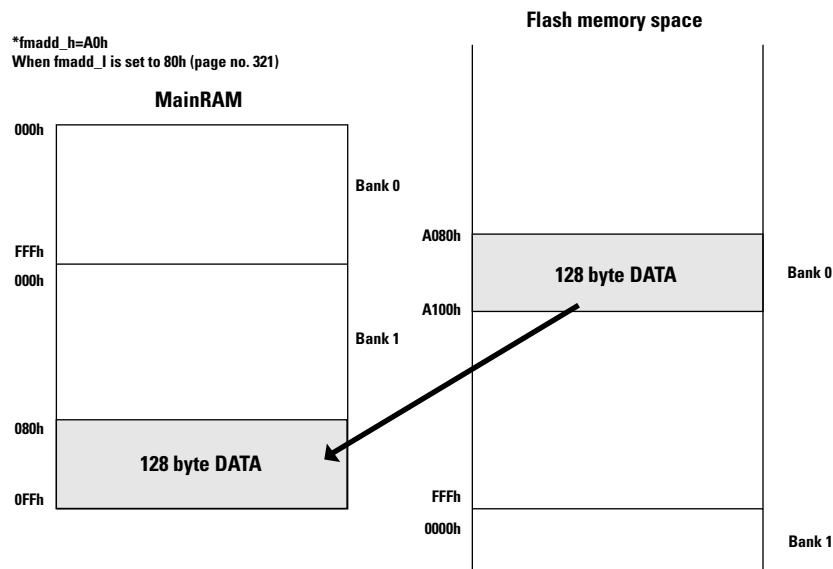


Figure 1.9 Execution of `fm_prd_ex`

Writing to Flash Memory

Subroutine name: fm_wrt_ex (org 0100h)

Arguments: High-order start address for flash memory write: fmadd_h (RAM bank-1 07Eh)
Low-order start address for flash memory write: fmadd_l (RAM bank-1 07Fh)
Bank address for flash memory write: fmbank (RAM bank-1 07Dh)
Flash memory write data (128 bytes): RAM bank-1 080h - 0FFh
Data write end detection algorithm:
Bit 0 of RAM bank-1 07Ch
(toggle bit method (0)/data polling method (1))

Return value: result of write: ACC (accumulator)
(Normal termination: 00h. Abnormal termination: FFh)

Function: Write one continuous page of data (128 bytes) to the flash memory, starting at the specified address

Description: By calling this subroutine, a program can write a page of data (128 bytes) to a continuous area in the flash memory, starting at the specified address.

Before using this subroutine, the following settings must be made.

- Select RAM bank to use
(1) Select RAM bank 1 (Set bit 1 of PSW to "1")
- Prepare data to be written to flash memory
(2) Store data to be written to flash memory in RAM bank 1, 080h - 0FFh
- Select flash memory bank to read
(3) Select flash memory bank 0
(Set 07Dh of RAM bank 1 to '00h')
* If another value is set, normal operation is not assured.
- Set address for accessing flash memory
(4) High-order address (8 bit): set to 07Eh of RAM bank-1
(5) Low-order address (8 bit): set to 07Fh of RAM bank-1
- Specify data write end detection algorithm
(6) Set data write end detection algorithm in 07Ch of RAM bank-1, as follows.
(6-1) Use toggle bit method: set 07Ch to 00h
(6-2) Use data polling method: set 07Ch to 01h
* If another value is set, normal operation is not assured.

When making write settings, observe the following points.

- fm_wrt_ex is a subroutine specifically for user programs. This subroutine can write only to the area where the user program is located. For this reason, be sure to secure an area within the user program before performing the data write.
- Data extending to 2 pages cannot be written. The write start address must therefore always be set to the beginning of each page.

The start address of each page can be calculated according to the following equation:
start address value (2 byte) = 080h x page number (0 - 511)

(Because writing is performed in single-page units, bit 0 - 6 of the lower-level address must always be set to "0". If an address other than the start address of a page is set, normal operation is not assured.)

For information on pages, please refer to section 7.1.2.

* Register values after subroutine completion

Note that the following (memory) registers will have different values before the subroutine is called and after the subroutine has completed:

- ACC (accumulator)
- B (B register)
- C (C register)
- TRL (table lookup register lower byte)
- TRH (table lookup register higher byte)
- r0 (RAM indirect access register)

fm_wrt_ex execution is shown in Fig. 7.4.

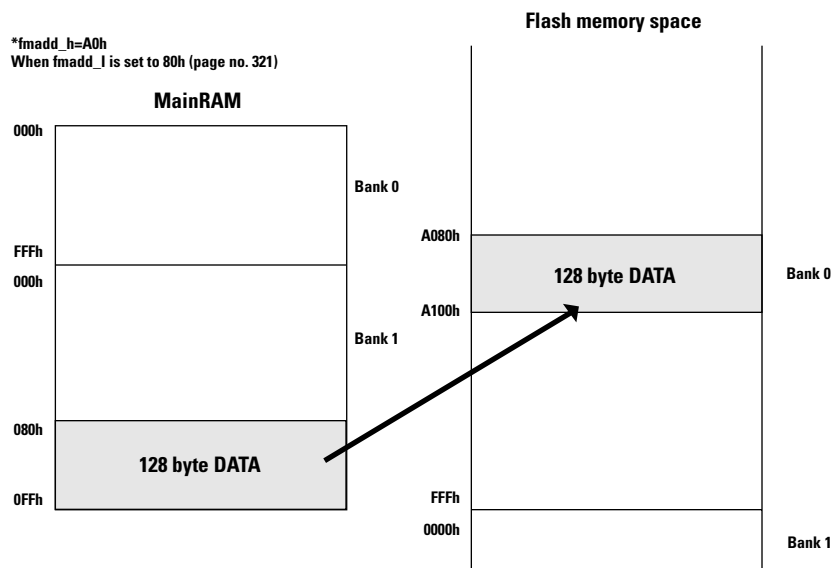


Figure 1.10 Execution of `fm_wrt_ex`

Flash Memory Verify

Subroutine name: `fm_vrf_ex` (org 0110h)

Arguments: High-order address flash memory address for verify start: `fmadd_h` (RAM bank 1 07Eh)
 Low-order address flash memory address for verify start: `fmadd_l` (RAM bank 1 07Fh)
 Flash memory bank address for verify operation: `fmbank` (RAM bank 1 07Dh)
 Data (128 bytes) for verify operation: RAM bank 1 080h - 0FFh

Return value: Verify result: accumulator (ACC) (normal end: 00h?error end: other than 00h)

Function: Serves to verify whether data were written correctly to flash memory. For use after writing data to flash memory with `fm_wrt_ex` (see section 7.1.4).

Description: This subroutine compares the 128 byte data specified when calling `fm_wrt_ex` with the data actually written to flash memory. Therefore the subroutine can only be used immediately after the `fm_wrt_ex` subroutine was called.

When calling this subroutine, the same arguments as used for the preceding `fm_wrt_ex` must be specified. If different arguments are specified, data verify will not be carried out properly.

After calling this subroutine, if all 128 bytes of data were found to match, 00h will be set in ACC, and the routine returns. If a data mismatch was detected, a value other than 00h will be set in ACC, and the routine returns.

* Register values after subroutine completion

Note that the following (memory) registers will have different values before the subroutine is called and after the subroutine has completed.

- TRL (table lookup register lower byte)
- TRH (table lookup register higher byte)
- r0 (RAM indirect access register)

`fm_vrf_ex` execution is shown in Fig. below.

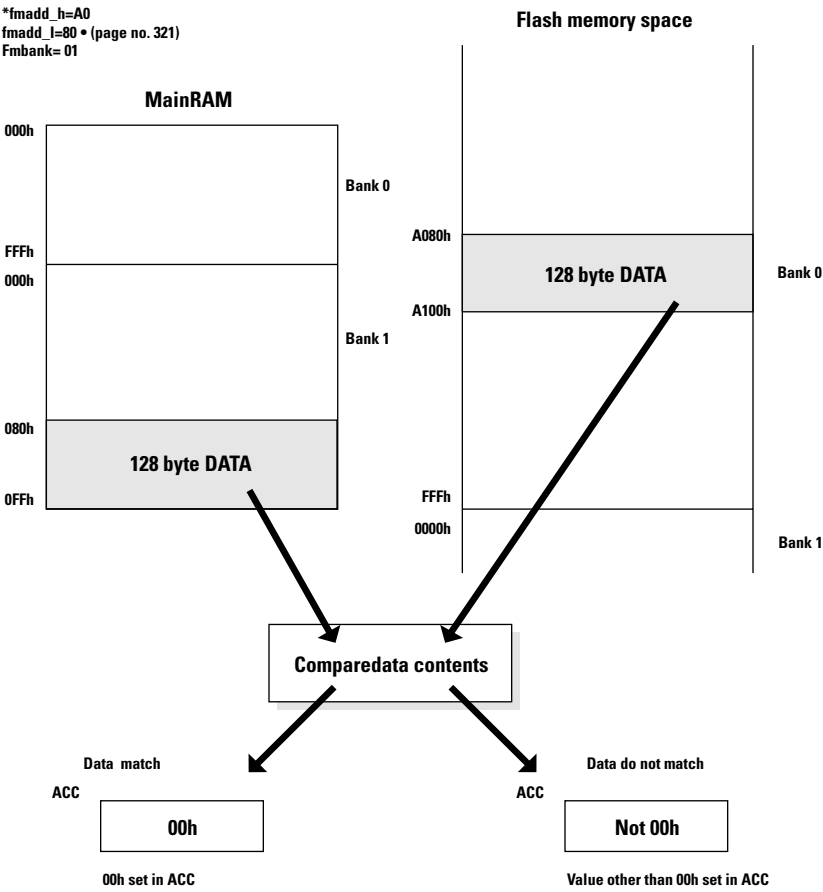


Figure 1.11 Execution of fm_vrf_ex

6.2 Clock Function

The clock functions implemented in VMU are as follows.

Time data automatic update

Clock Countup Timer

Subroutine name: timer_ex

Arguments: None

Return value: Year: year_h (RAM bank 0 017h, 18h)
Month: mon_h (RAM bank 0 019h)
Day: day_h (RAM bank 0 01Ah)
Hour: hour_h (RAM bank 0 01Bh)
Minute: min_h (RAM bank 0 01Ch)
Second: sec_h (RAM bank 0 01Dh)

* The year data are configured as 2 bytes, with the higher-level in byte in 17h and the lower level byte in 18h. Because “year_h” is assigned to RAM bank, 017h, address 018h can be accessed by specifying “year_h+1”.

Function: When the subroutine is called, it obtains the time data and places them in the specified location in RAM bank 0. (For information on the specified location, please refer to Appendix 1.)

Description: This subroutine is a time counter using the base timer interrupt. To enable use of the subroutine, the following settings for the base timer interrupt must be made.

- Base timer interrupt settings

This subroutine uses only the base timer 0 interrupt. The base timer interrupt is to be set as shown below.

- | | |
|--|--------------------|
| (1) Base timer count stop | (BTCR 6 bit = '0') |
| (2) 14 bit base timer mode selected | (BTCR 7 bit = '0') |
| (3) Sub clock used as base timer clock | (ISL 4 bit = '0') |
| (4) Base timer interrupt 0 enabled | (BTCR 0 bit = '1') |
| (5) Base timer count start | (BTCR 6 bit = '1') |

Because the base timer 0 interrupt is used by the timer_ex subroutine, user programs may not access this interrupt. Otherwise, normal operation is not assured.

This subroutine should be called after jumping to the interrupt vector of the base timer interrupt 0 source. Also, be sure to clear the base timer 0 interrupt source (BTCR 1 bit = '0').

(If this is not performed, the clock function will not operate properly.)

All time data obtained by this subroutine are in hex format. Conversion into decimal format must be performed by the user program.

7. Automatic low battery detection function

Visual Memory comes with the ability to automatically detect low battery.

The following explains how this function works.

7.1 Automatic low battery detection flag

Visual Memory can automatically check the battery's power consumption and when necessary display a low battery warning message on the screen. Gamedevelopers can use the automatic low battery detection flag to enable or disable this function.

The following describes how to use this function.

Register to use:	06Eh (Bank-0)
Register values:	00h (enable the automatic low battery detection function) FFh (disable the automatic low battery detection function) (If any value other than the above ones is used, then normal operation cannot be guaranteed.)
How it work:	The automatic low battery detection function constantly monitors the battery's voltage and if the voltage falls below a certain level it will stop the current program, wait for 3 seconds, then display the battery warning message on the screen.
Explanation:	The automatic low battery detection function consists of tasks from detecting low voltage to displaying the low battery warning message. When the automatic low battery detection flag is set to 00h, the automatic low battery detection function is enabled and when the battery is low it will display the low battery warning message, regardless of the current task of Visual Memory. If the flag is set to FFh, then the automatic low battery detection function is disabled.

When the user program is performing the following tasks, be sure to turn off the automatic low battery detection function:

1. Communicating with another Visual Memory via the serial interface
2. Writing to the flash memory space

Visual Memory Unit (VMU)
Sound Development
Specifications

1. VMU Sound Development Specifications

1. VMU Sound Output Hardware Outline

VMU can use an internal timer (timer 1) to produce sound output.

The following two output methods are possible.

- 8-bit pulse generator output
- Variable bit length pulse generator output (9 - 16bits)

Both methods use the timer 1 circuit. Normally, the 8-bit pulse generator output method is used.

2. Sound Output Principle

This section describes the VMU sound output method.

VMU sound output uses timer 1.

2.1 Timer 1 Outline

This section describes timer 1 that is used for VMU sound output.

Timer 1 incorporated in the VMU is a 16-bit timer with the following four functions.

- Mode 0: 8-bit reload timer x 2 channels
- Mode 1: 8-bit reload timer + 8-bit pulse generator
- Mode 2: 16-bit reload timer
- Mode 3: Variable bit length pulse generator (9 - 16bits)

Among these modes, VMU uses mode 1 for sound output.

For information on using the other modes, please refer to the VMU Hardware manual.

Timer 1 Block Configuration

This section describes the block configuration of timer 1.

A configuration diagram of timer 1 is shown in Fig. 2.1.

- Timer 1 lower level (T1L) 1

This is an 8-bit reload timer using the cycle clock or cycle clock 1/2 signal as clock signal.

At the overflow of T1L, the T1LR data are reloaded. When T1LRUN (T1CNT, bit6) is set to "0", the T1LR data are transferred to T1L.

- Timer 1 lower level comparator (T1LC) 2

This circuit consists of the 8-bit timer 1 lower level comparison data register (T1LC) and an 8-bit data comparator circuit. The circuit compares the T1L and T1LC data.

- Timer 1 higher level (T1H) 3

This is an 8-bit reload timer using the cycle clock or the T1L overflow as clock signal.

At the overflow of T1H, the T1HR data are reloaded. Reload also occurs when T1HRUN (T1CNT, bit7) is reset.

- Timer 1 higher level comparator (T1HC) 4

This circuit consists of the 8-bit timer 1 higher level comparison data register (T1HC) and an 8-bit data comparator circuit. The circuit compares the T1H and T1HC data.

- Timer 1 control register (T1CNT) 5

Serves for T1 mode setting and interrupt control.

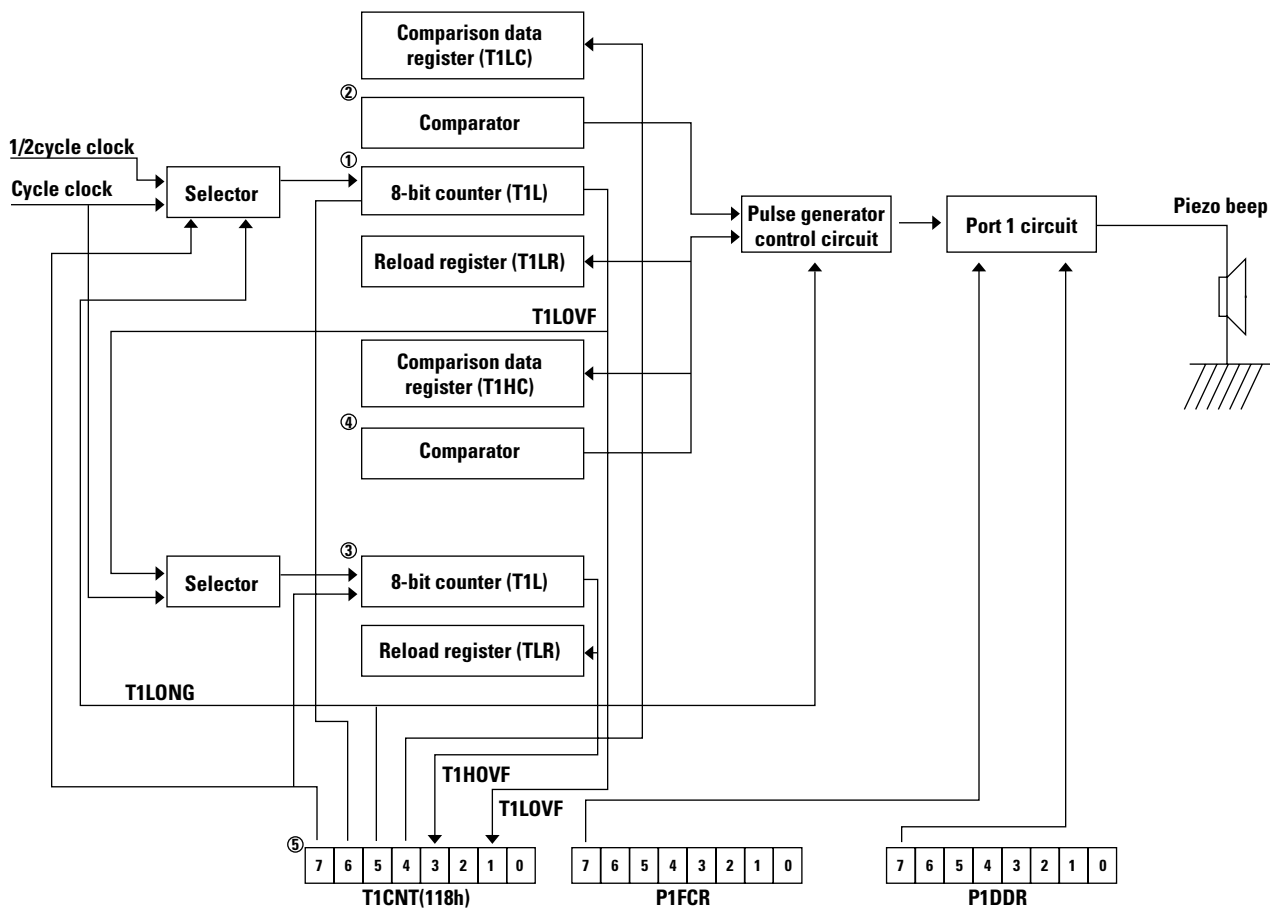


Figure 1.1 VMU Timer 1 Block Diagram

Related Registers

The following registers are required for controlling timer 1.

• T1L (11Bh)	Timer 1 lower level counter register
• T1LR (11Bh)	Timer 1 lower level reload register
• T1LC (11Ah)	Timer 1 lower level comparison data register
• T1CNT (118h)	Timer 1 control register
• P1 (114h)	Port 1 latch register
• P1DDR (145h)	Port 1 data direction register
• P1FCR (146h)	Port 1 control register
• OCR (10Eh)	Resonance control register

For details on the above timer, please refer to section 3.3 of the VMU Hardware manual.

Mode Setting

This section describes how to set timer 1 to the mode for VMU sound output (mode 1).

The following four registers are required for setting the mode.

T1CNT	(bit5: T1LONG)
P1	(bit7: P17)
P1DDR	(bit7: P17DDR)
P1FCR	(bit7: P17DDR)

The register values for the modes are listed in the table below, along with the cycle clock used for each mode.

Mode	Clock cycle	T1LONG	P17FCR	P17DDR	P17
1	Tcyc	0	1	1	0

Table 1.1 *Time 1 Mode Setting*

Tcyc in the table is the cycle clock.

To use the sound output capability of VMU, be sure to set the system clock to the sub-clock (32 KHz).

At other system clock settings, correct sound output may not be obtained.

The cycle clock is defined as follows.

System clock 32 KHz (Tcyc = 183.0 us)

For information on setting the system clock, please refer to the VMU Hardware manual.

* Problems when using other system clock settings

Besides the 32 KHz clock, the VMU can use a 600 KHz and 6 MHz system clock, but when the latter two are selected, the following problems occur.

- 600KHz When the 600 KHz clock is selected, the output frequency tolerance will be -50%, +100%, which will cause a wide fluctuation in the actual output sound.
- 6MHz When the 6 MHz clock is selected, power consumption will increase considerably, resulting in a shorter battery life.

2.2 8-Bit Counter Mode

This section describes VMU sound output when using 8-bit counter mode. For information on basic operation, please refer to the VMU Hardware manual.

Output Waveform and Parameter Settings

This section describes the signal waveform that can be output in 8-bit counter mode, and lists the parameters that determine the waveform.

The output waveform is shown in Figure below.

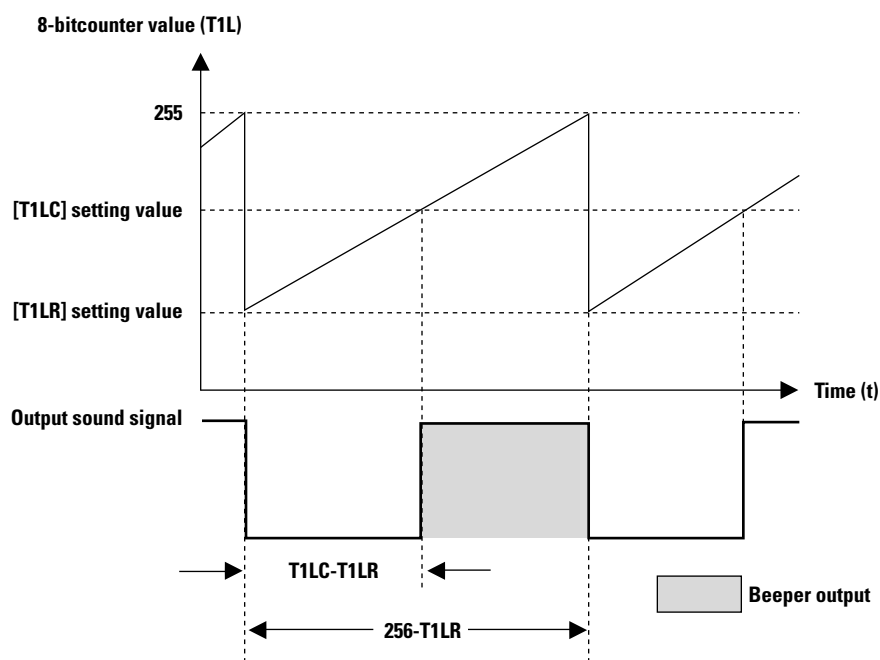


Figure 1.2 Output waveform

8-Bit Counter Mode Setting

This section describes the sound signal output procedure in 8-bit counter mode.

To output a sound signal in 8-bit counter mode, make the settings as described below.

1. Set the parameters (T1LR, T1LC) according to the desired output waveform.

Use equations (1) and (2) given below to define the waveform.

Sound output signal L level pulse width (decimal)

$$= (\text{T1LC setting value} - \text{T1LR setting value}) \times \text{Tcyc} \cdots \cdots \text{Equation (1)}$$

Sound output signal cycle (decimal)

$$= (256 - \text{T1LR setting value}) \times \text{Tcyc} \cdots \cdots \text{Equation (2)}$$

Tcyc: cycle clock

For details on output waveform and parameters, please refer to section 2.2.1.

2. Select the mode for timer 1.

The following four registers are required for setting the mode.

T1CNT (bit5: T1LONG)

P1 (bit7: P17)

P1DDR (bit7: P17DDR)

P1FCR (bit7: P17FCR)

The register values for the modes are listed in the table below, along with the cycle clock used for each mode.

Table 1.2 Time 1 Mode Setting

Mode	T1LONG	P17FCR	P17DDR	P17
1	0	1	1	0

3. Start the count for timer 1 (lower 8bits)

To start/stop the timer, make the following settings.

Waveform parameter update

Set T1CNT bit4 (ELDT1C) to "1". Note that the waveform parameters set in step 1 do not become effective until this setting is made.

If the parameters were changed while T1CNT bit4 was "1", the parameter setting value becomes effective immediately after the change.

Timer 1 count start

Set T1CNT bit6 (T1LRUN) to "1".

To stop audio output in the 8-bit counter made, make the Following setting.

4. Set the timer1(T1L) count stop flag (T1CNT bit6)to "0".

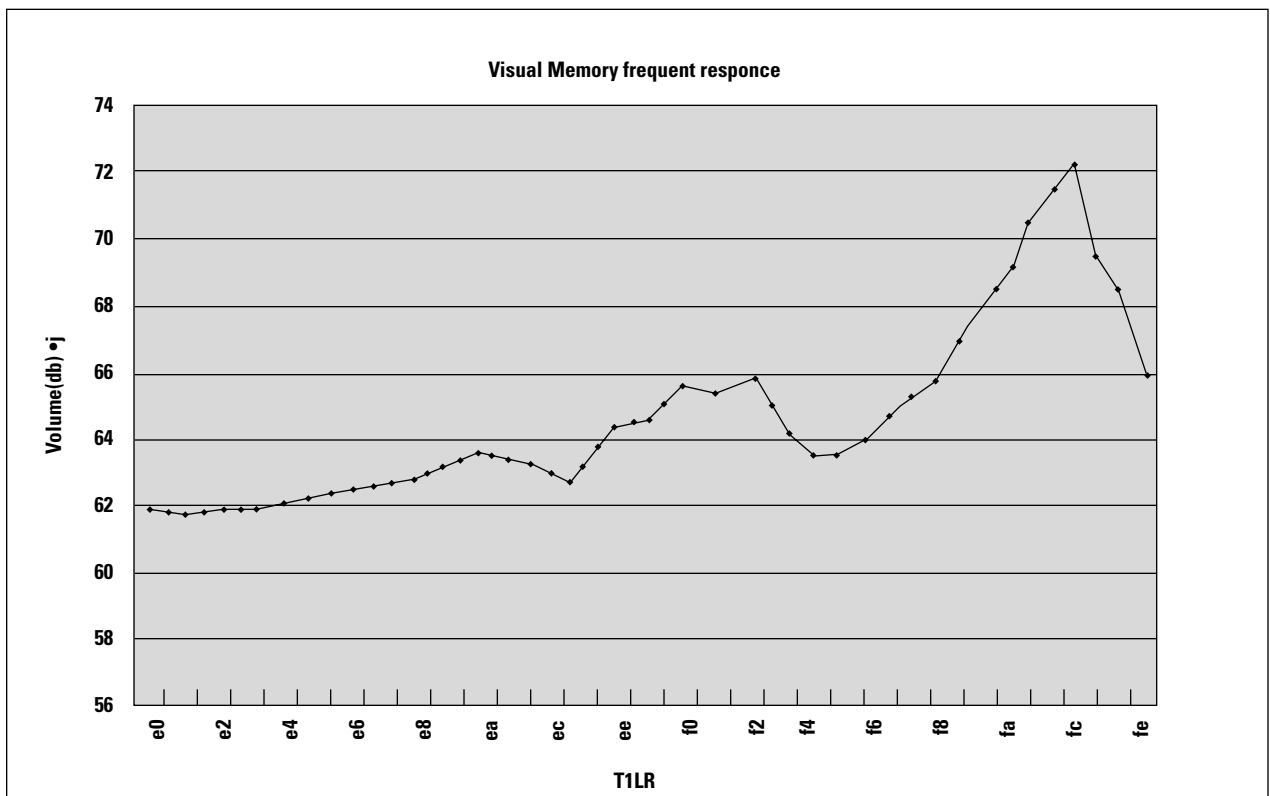
While timer 1 (lower 8bits) is operating, the waveform parameters can be changed. To output sound of a different frequency without interruption, change the waveform output parameters without stopping timer 1. (Leave T1CNT bit4 [ELDT 1C]) set to "1".)

Frequency Response Characteristics

The frequency response of the beeper in the VMU is shown below.

The T1LR value indicates the frequency range that can be output by the VMU.

For details, please refer to the explanation of the relationship between T1LR value and output frequency in section 2.3.



2.3 Table of Available Output Frequencies

The output frequencies (theoretical values) available with a system clock of 32 KHz are listed below.

Due to limitations of the beeper, not all frequencies can actually be output. You should use the recommended frequencies indicated in the table.

The L level pulse width of the output signal is set to 1/2 of the output signal cycle (duty factor = 50%).

Table 1.3 *Waveform Parameters and Output Frequencies*

T1LR(hex)	T1LC(hex)	Frequency (Hz)	T1LR(hex)	T1LC(hex)	Frequency (Hz)	T1LR(hex)	T1LC(hex)	Frequency (Hz)	T1LR(hex)	T1LC(hex)	Frequency (Hz)
00	80	21.346	40	94	28.461	80	A8	42.691	C0	E0	85.383
01	80	21.429	41	A0	28.610	81	C0	43.027	C1	E0	86.738
02	81	21.514	42	A1	28.760	82	C1	43.369	C2	E1	88.137
03	81	21.599	43	A1	28.913	83	C1	43.716	C3	E1	89.582
04	82	21.684	44	A2	29.066	84	C2	44.068	C4	E2	91.075
05	82	21.771	45	A2	29.222	85	C2	44.427	C5	E2	92.618
06	83	21.858	46	A3	29.379	86	C3	44.791	C6	E3	94.215
07	83	21.946	47	A3	29.538	87	C3	45.161	C7	E3	95.868
08	84	22.034	48	A4	29.698	88	C4	45.537	C8	E4	97.580
09	84	22.123	49	A4	29.861	89	C4	45.920	C9	E4	99.354
0A	85	22.213	4A	A5	30.025	8A	C5	46.309	CA	E5	101.194
0B	85	22.304	4B	A5	30.191	8B	C5	46.705	CB	E5	103.103
0C	86	22.395	4C	A6	30.358	8C	C6	47.108	CC	E6	105.086
0D	86	22.488	4D	A6	30.528	8D	C6	47.517	CD	E6	107.147
0E	87	22.580	4E	A7	30.699	8E	C7	47.934	CE	E7	109.290
0F	87	22.674	4F	A7	30.873	8F	C7	48.358	CF	E7	111.520
10	88	22.769	50	A8	31.048	90	C8	48.790	D0	E8	113.843
11	88	22.864	51	A8	31.226	91	C8	49.230	D1	E8	116.266
12	89	22.960	52	A9	31.405	92	C9	49.677	D2	E9	118.793
13	89	23.057	53	A9	31.587	93	C9	50.133	D3	E9	121.433
14	8A	23.155	54	AA	31.770	94	CA	50.597	D4	EA	124.193
15	8A	23.253	55	AA	31.956	95	CA	51.070	D5	EA	127.081
16	8B	23.352	56	AB	32.144	96	CB	51.552	D6	EB	130.107

1. VMU Sound Development Specifications

T1LR(hex)	T1LC(hex)	Frequency (Hz)	T1LR(hex)	T1LC(hex)	Frequency (Hz)	T1LR(hex)	T1LC(hex)	Frequency (Hz)	T1LR(hex)	T1LC(hex)	Frequency (Hz)
17	8B	23.453	57	AB	32.334	97	CB	52.043	D7	EB	133.280
18	8C	23.554	58	AC	32.527	98	CC	52.543	D8	EC	136.612
19	8C	23.656	59	AC	32.721	99	CC	53.053	D9	EC	140.115
1A	8D	23.759	5A	AD	32.919	9A	CD	53.573	DA	ED	143.802
1B	8D	23.862	5B	AD	33.118	9B	CD	54.104	DB	ED	147.689
1C	8E	23.967	5C	AE	33.320	9C	CE	54.645	DC	EE	151.791
1D	8E	24.073	5D	AE	33.524	9D	CE	55.197	DD	EE	156.128
1E	8F	24.179	5E	AF	33.731	9E	CF	55.760	DE	EF	160.720
1F	8F	24.287	5F	AF	33.941	9F	CF	56.335	DF	EF	165.590
20	90	24.395	60	B0	34.153	A0	D0	56.922	E0	F0	170.765
21	90	24.504	61	B0	34.368	A1	D0	57.521	E1	F0	176.274
22	91	24.615	62	B1	34.585	A2	D1	58.133	E2	F1	182.149
23	91	24.726	63	B1	34.806	A3	D1	58.758	E3	F1	188.430
24	92	24.839	64	B2	35.029	A4	D2	59.397	E4	F2	195.160
25	92	24.952	65	B2	35.255	A5	D2	60.049	E5	F2	202.388
26	93	25.066	66	B3	35.484	A6	D3	60.716	E6	F3	210.172
27	93	25.182	67	B3	35.716	A7	D3	61.399	E7	F3	218.579
28	94	25.299	68	B4	35.951	A8	D4	62.096	E8	F4	227.687
29	94	25.416	69	B4	36.189	A9	D4	62.810	E9	F4	237.586
2A	95	25.535	6A	B5	36.430	AA	D5	63.540	EA	F5	248.385
2B	95	25.655	6B	B5	36.674	AB	D5	64.288	EB	F5	260.213
2C	96	25.776	6C	B6	36.922	AC	D6	65.053	EC	F6	273.224
2D	96	25.898	6D	B6	37.173	AD	D6	65.837	ED	F6	287.604
2E	97	26.021	6E	B7	37.428	AE	D7	66.640	EE	F7	303.582
2F	97	26.146	6F	B7	37.686	AF	D7	67.463	EF	F7	321.440
30	98	26.272	70	B8	37.948	B0	D8	68.306	F0	F8	341.530
31	98	26.398	71	B8	38.213	B1	D8	69.171	F1	F8	364.299
32	99	26.527	72	B9	38.482	B2	D9	70.057	F2	F9	390.320
33	99	26.656	73	B9	38.755	B3	D9	70.967	F3	F9	420.345

Visual Memory Unit (VMU) Sound Development Specifications

T1LR(hex)	T1LC(hex)	Frequency (Hz)	T1LR(hex)	T1LC(hex)	Frequency (Hz)	T1LR(hex)	T1LC(hex)	Frequency (Hz)	T1LR(hex)	T1LC(hex)	Frequency (Hz)
34	9A	26.787	74	BA	39.032	B4	DA	71.901	F4	FA	455.373
35	9A	26.919	75	BA	39.313	B5	DA	72.860	F5	FA	496.771
36	9B	27.052	76	BB	39.598	B6	DB	73.844	F6	FB	546.448
37	9B	27.186	77	BB	39.887	B7	DB	74.856	F7	FB	607.165
38	9C	27.322	78	BC	40.180	B8	DC	75.896	F8	FC	683.060
39	9C	27.460	79	BC	40.478	B9	DC	76.965	F9	FC	780.640
3A	9D	27.598	7A	BD	40.780	BA	DD	78.064	FA	FD	910.747
3B	9D	27.738	7B	BD	41.086	BB	DD	79.195	FB	FD	1092.896
3C	9E	27.880	7C	BE	41.398	BC	DE	80.360	FC	FE	1366.120
3D	9E	28.023	7D	BE	41.714	BD	DE	81.559	FD	FE	1821.494
3E	9F	28.167	7E	BF	42.034	BE	DF	82.795	FE	FF	2732.240
3F	9F	28.313	7F	BF	42.360	BF	DF	84.069	FF	FF	5464.481
00	80	21.346	40	94	28.461	80	A8	42.691	C0	E0	85.383
01	80	21.429	41	A0	28.610	81	C0	43.027	C1	E0	86.738
02	81	21.514	42	A1	28.760	82	C1	43.369	C2	E1	88.137
03	81	21.599	43	A1	28.913	83	C1	43.716	C3	E1	89.582
04	82	21.684	44	A2	29.066	84	C2	44.068	C4	E2	91.075
05	82	21.771	45	A2	29.222	85	C2	44.427	C5	E2	92.618
06	83	21.858	46	A3	29.379	86	C3	44.791	C6	E3	94.215
07	83	21.946	47	A3	29.538	87	C3	45.161	C7	E3	95.868
08	84	22.034	48	A4	29.698	88	C4	45.537	C8	E4	97.580
09	84	22.123	49	A4	29.861	89	C4	45.920	C9	E4	99.354
0A	85	22.213	4A	A5	30.025	8A	C5	46.309	CA	E5	101.194
0B	85	22.304	4B	A5	30.191	8B	C5	46.705	CB	E5	103.103
0C	86	22.395	4C	A6	30.358	8C	C6	47.108	CC	E6	105.086
0D	86	22.488	4D	A6	30.528	8D	C6	47.517	CD	E6	107.147
0E	87	22.580	4E	A7	30.699	8E	C7	47.934	CE	E7	109.290
0F	87	22.674	4F	A7	30.873	8F	C7	48.358	CF	E7	111.520
10	88	22.769	50	A8	31.048	90	C8	48.790	D0	E8	113.843

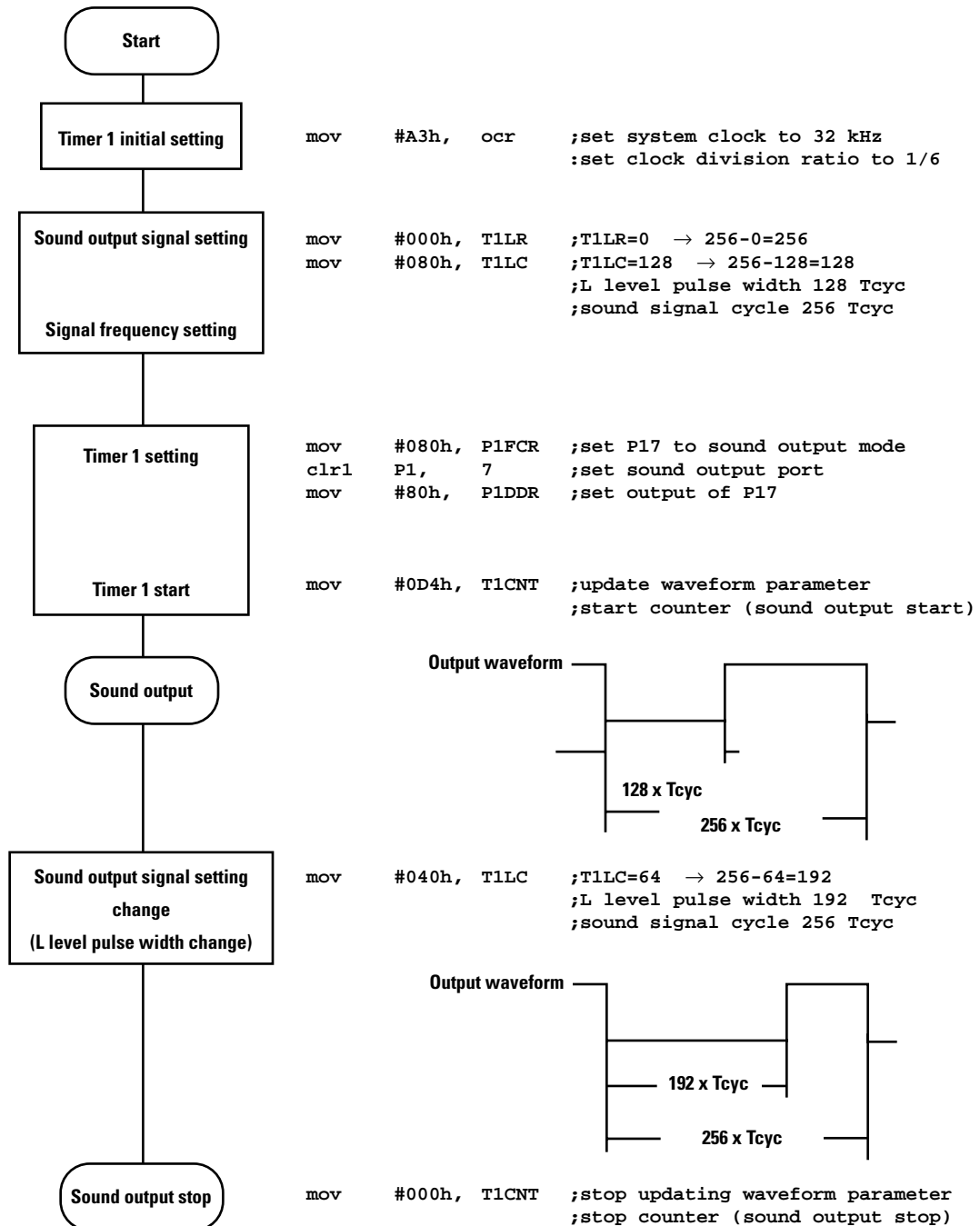
1. VMU Sound Development Specifications

T1LR(hex)	T1LC(hex)	Frequency (Hz)	T1LR(hex)	T1LC(hex)	Frequency (Hz)	T1LR(hex)	T1LC(hex)	Frequency (Hz)	T1LR(hex)	T1LC(hex)	Frequency (Hz)
11	88	22.864	51	A8	31.226	91	C8	49.230	D1	E8	116.266
12	89	22.960	52	A9	31.405	92	C9	49.677	D2	E9	118.793
13	89	23.057	53	A9	31.587	93	C9	50.133	D3	E9	121.433
14	8A	23.155	54	AA	31.770	94	CA	50.597	D4	EA	124.193
15	8A	23.253	55	AA	31.956	95	CA	51.070	D5	EA	127.081
16	8B	23.352	56	AB	32.144	96	CB	51.552	D6	EB	130.107
17	8B	23.453	57	AB	32.334	97	CB	52.043	D7	EB	133.280
18	8C	23.554	58	AC	32.527	98	CC	52.543	D8	EC	136.612
19	8C	23.656	59	AC	32.721	99	CC	53.053	D9	EC	140.115
1A	8D	23.759	5A	AD	32.919	9A	CD	53.573	DA	ED	143.802
1B	8D	23.862	5B	AD	33.118	9B	CD	54.104	DB	ED	147.689
1C	8E	23.967	5C	AE	33.320	9C	CE	54.645	DC	EE	151.791
1D	8E	24.073	5D	AE	33.524	9D	CE	55.197	DD	EE	156.128
1E	8F	24.179	5E	AF	33.731	9E	CF	55.760	DE	EF	160.720
1F	8F	24.287	5F	AF	33.941	9F	CF	56.335	DF	EF	165.590
20	90	24.395	60	B0	34.153	A0	D0	56.922	E0	F0	170.765
21	90	24.504	61	B0	34.368	A1	D0	57.521	E1	F0	176.274
22	91	24.615	62	B1	34.585	A2	D1	58.133	E2	F1	182.149
23	91	24.726	63	B1	34.806	A3	D1	58.758	E3	F1	188.430
24	92	24.839	64	B2	35.029	A4	D2	59.397	E4	F2	195.160
25	92	24.952	65	B2	35.255	A5	D2	60.049	E5	F2	202.388
26	93	25.066	66	B3	35.484	A6	D3	60.716	E6	F3	210.172
27	93	25.182	67	B3	35.716	A7	D3	61.399	E7	F3	218.579
28	94	25.299	68	B4	35.951	A8	D4	62.096	E8	F4	227.687
29	94	25.416	69	B4	36.189	A9	D4	62.810	E9	F4	237.586
2A	95	25.535	6A	B5	36.430	AA	D5	63.540	EA	F5	248.385
2B	95	25.655	6B	B5	36.674	AB	D5	64.288	EB	F5	260.213
2C	96	25.776	6C	B6	36.922	AC	D6	65.053	EC	F6	273.224
2D	96	25.898	6D	B6	37.173	AD	D6	65.837	ED	F6	287.604

Visual Memory Unit (VMU) Sound Development Specifications

T1LR(hex)	T1LC(hex)	Frequency (Hz)	T1LR(hex)	T1LC(hex)	Frequency (Hz)	T1LR(hex)	T1LC(hex)	Frequency (Hz)	T1LR(hex)	T1LC(hex)	Frequency (Hz)
2E	97	26.021	6E	B7	37.428	AE	D7	66.640	EE	F7	303.582
2F	97	26.146	6F	B7	37.686	AF	D7	67.463	EF	F7	321.440
30	98	26.272	70	B8	37.948	B0	D8	68.306	F0	F8	341.530
31	98	26.398	71	B8	38.213	B1	D8	69.171	F1	F8	364.299
32	99	26.527	72	B9	38.482	B2	D9	70.057	F2	F9	390.320
33	99	26.656	73	B9	38.755	B3	D9	70.967	F3	F9	420.345
34	9A	26.787	74	BA	39.032	B4	DA	71.901	F4	FA	455.373
35	9A	26.919	75	BA	39.313	B5	DA	72.860	F5	FA	496.771
36	9B	27.052	76	BB	39.598	B6	DB	73.844	F6	FB	546.448
37	9B	27.186	77	BB	39.887	B7	DB	74.856	F7	FB	607.165
38	9C	27.322	78	BC	40.180	B8	DC	75.896	F8	FC	683.060
39	9C	27.460	79	BC	40.478	B9	DC	76.965	F9	FC	780.640
3A	9D	27.598	7A	BD	40.780	BA	DD	78.064	FA	FD	910.747
3B	9D	27.738	7B	BD	41.086	BB	DD	79.195	FB	FD	1092.896
3C	9E	27.880	7C	BE	41.398	BC	DE	80.360	FC	FE	1366.120
3D	9E	28.023	7D	BE	41.714	BD	DE	81.559	FD	FE	1821.494
3E	9F	28.167	7E	BF	42.034	BE	DF	82.795	FE	FF	2732.240
3F	9F	28.313	7F	BF	42.360	BF	DF	84.069	FF	FF	5464.481

3. Sample Program



A. Table of Defined Variables

System BIOS requires the following variables.

Time data variables

year	010h (bank-0)	Year (BCD 4 digits)	* Not generated by timer_ex subroutine
mon	012h (bank-0)	Month (BCD 2 digits)	* Not generated by timer_ex subroutine
day	013h (bank-0)	Day (BCD 2 digits)	* Not generated by timer_ex subroutine
hour	014h (bank-0)	Hour (BCD 2 digits)	* Not generated by timer_ex subroutine
min	015h (bank-0)	Minute (BCD 2 digits)	* Not generated by timer_ex subroutine
sec	016h (bank-0)	Second (BCD 2 digits)	* Not generated by timer_ex subroutine
year_h	017h (bank-0)	Year (HEX 4 digits)	
mon_h	019h (bank-0)	Month (HEX 2 digits)	
day_h	01Ah (bank-0)	Day (HEX 2 digits)	
hour_h	01Bh (bank-0)	Hour (HEX 2 digits)	
min_h	01Ch (bank-0)	Minute (HEX 2 digits)	
sec_h	01Dh (bank-0)	Second (HEX 2 digits)	
sec_f	01Eh (bank-0)	Work RAM (used by timer, prohibited to user programs)	
leaf_f	01Fh (bank-0)	Work RAM (used by timer, prohibited to user programs)	

Low battery detection function

06Eh (bank-0)	Automatic low battery detection flag (Register name is not set.)
---------------	---

Flash memory variables

fmbank	07Dh (bank-1)	Flash memory bank designation
fmadd_h	07Eh (bank-1)	Flash memory address (upper 8 bit)
fmadd_l	07Fh (bank-1)	Flash memory address (lower 8 bit)

Note: The automatic low battery detection function cannot work if the user program performs saving data away when low battery is detected. The user program must monitor the low battery detection flag (bit 1 of PORT7) and handle it accordingly. For information on low battery detection flag, please refer to the VMU hardware manual.

VMU Mode Selection Operation Flow

The following explains the various VMU modes and their selection.

The figure shows the VMU mode selection flow diagram.

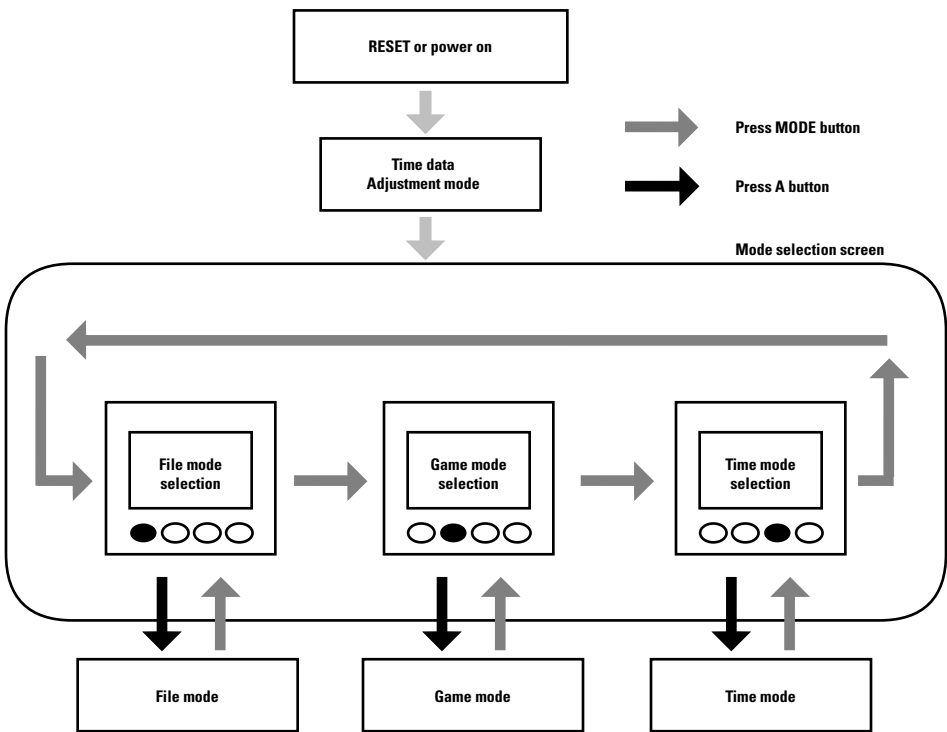


Fig. Appendix 2-1 Mode selection flow diagram

Each of the modes comes with the following functions.

- Mode selection screen This screen allows selection and execution of one of the three VMU modes. Pressing the MODE button each time selects a new mode, and pressing the A button enters the selected mode. The icon indicates the current mode. Please refer to Fig. Appendix 2-1 for the modes and their corresponding icon.
- File mode This mode handles game data and user program management and editing. Pressing the MODE button while in this mode will return to the mode selection screen.
- Game mode This mode executes user programs stored on VMU. Pressing the MODE button while in this mode will return to the mode selection screen. Because the BIOS does not support return processing from the game mode to the mode selection screen, be sure to handle the return processing in the user program. For information on the return processing to the mode selection screen, please refer to section 5.3.
- Time mode This mode handles current time display and adjustment. Pressing the MODE button while in this mode will return to the mode selection screen. Also, while in this mode, pressing the left cursor key as well as the A button at the same time will enter the time setting mode.

