**FUJITSU SEMICONDUCTOR**
CONTROLLER MANUAL

CM44-10116-2E

# F$^2$MC-16LX

## 16-BIT MICROCONTROLLER

# MB90M405 Series

# HARDWARE MANUAL

FUJITSU

# F$^2$MC-16LX

## 16-BIT MICROCONTROLLER

# MB90M405 Series

# HARDWARE MANUAL

FUJITSU LIMITED

# PREFACE

## ■ Objectives and Intended Reader

Thank you for purchasing a Fujitsu semiconductor product.

The MB90M405 series is a series of general-purpose 16-bit microcontrollers with 60 built-in high-tension-resistant output pins required for fluorescent display control. The MB90M405 series was developed for applications that require the control of a vacuum fluorescent tube panel.

This manual, intended for engineers who design products using the MB90M405 series, describes the functions and operations of MB90M405 series products.

## ■ Trademark

$F^2MC$ is the abbreviation of FUJITSU Flexible Microcontroller.

Embedded Algorithm is a trademark of Advanced Micro Devices Corporation.

Other system and product names used in this manual are trademarks of their respective companies or organizations.

The symbols $^{TM}$ and ® are sometimes omitted in the text.

## ■ License

Purchase of FUJITSU Ltd, $I^2C$ components conveys a license under the Philips $I^2C$ Patent Right to use.

These components in an $I^2C$ system, provided that the system conforms to the $I^2C$ Standard Specification as defined by Philips.

## ■ Organization of This Manual

This manual consists of the following 24 chapters and an appendix:

**CHAPTER 1 "OVERVIEW"**

This chapter summarizes the features and basic specifications of the MB90M405 series of microcontrollers.

**CHAPTER 2 "CPU"**

This chapter describes the CPU and the memory space provided by the MB90M405 series

**CHAPTER 3 "RESETS"**

This chapter describes resets for the MB90M405 series.

**CHAPTER 4 "CLOCKS"**

This chapter describes the clocks used by MB90M405 series.

**CHAPTER 5 "LOW POWER CONSUMPTION MODE"**

This chapter describes the low power consumption mode of MB90M405 series.

**CHAPTER 6 "INTERRUPTS"**

This chapter explains the interrupts and extended intelligent I/O service ($EI^2OS$) in the MB90M405 series.

**CHAPTER 7 "SETTING A MODE"**

This chapter describes the operating modes and the memory access modes of the MB90M405 series.

**CHAPTER 8 "I/O PORTS"**

This chapter describes the functions and operations of the MB90M405 series I/O ports.

**CHAPTER 9 "SERIAL I/O"**

This chapter describes the functions and operations of the serial I/O unit of the MB90M405 series.

**CHAPTER 10 "TIMEBASE TIMER"**

This chapter describes the functions and operation of the timebase timer of the MB90M405 series.

**CHAPTER 11 "WATCHDOG TIMER"**

This chapter describes the functions and operations of the watchdog timer of the MB90M405 series.

**CHAPTER 12 "16-BIT RELOAD TIMER"**

This chapter describes the functions and operations of the 16-bit reload timer of the MB90M405 series.

**CHAPTER 13 "16-BIT I/O TIMER"**

This chapter describes the functions and operations of the 16-bit I/O timer of the MB90M405 series.

**CHAPTER 14 "UART"**

This chapter describes the functions and operations of the MB90M405 series UART.

**CHAPTER 15 "DTP/EXTERNAL INTERRUPT CIRCUIT"**

This chapter describes the functions and operations of the DTP/external interrupt circuit of the MB90M405 series.

**CHAPTER 16 "I$^2$C INTERFACE"**

This chapter describes the functions and operations of the I$^2$C interface of the MB90M405 series.

**CHAPTER 17 "8/10-BIT A/D CONVERTER"**

This chapter describes the functions and operations of the MB90M405 series 8/10-bit A/D converter.

**CHAPTER 18 "FL CONTROL CIRCUIT"**

This chapter explains the functions and operation of the MB90M405 series FL control circuit.

**CHAPTER 19 "WATCH CLOCK OUTPUT"**

This chapter describes the functions and operations of MB90M405 series watch clock output.

**CHAPTER 20 "DELAYED INTERRUPT GENERATOR MODULE"**

This chapter describes the functions and operation of the MB90M405 series delayed interrupt generator module.

**CHAPTER 21 "ADDRESS MATCH DETECTION FUNCTION"**

This chapter describes the address match detection function of the MB90M405 series and its operations.

**CHAPTER 22 "ROM MIRRORING FUNCTION SELECTION MODULE"**

This chapter describes the function and operation of the MB90M405 series ROM mirroring function selection module.

**CHAPTER 23 "1M-BIT FLASH MEMORY"**

This chapter describes the functions and operations of the MB90M405 series 1M-bit flash memory.

**CHAPTER 24 "EXAMPLE OF MB90MF408 SERIAL PROGRAMMING CONNECTION"**

This chapter provides examples of connection for serial programming using the AF220 flash microcomputer programmer manufactured by YDC Corporation.

**APPENDIX**

This appendix includes I/O maps, instruction lists, and other information.

- The contents of this document are subject to change without notice. Customers are advised to consult with FUJITSU sales representatives before ordering.
- The information and circuit diagrams in this document are presented as examples of semiconductor device applications, and are not intended to be incorporated in devices for actual use. Also, FUJITSU is unable to assume responsibility for infringement of any patent rights or other rights of third parties arising from the use of this information or circuit diagrams.
- The products described in this document are designed, developed and manufactured as contemplated for general use, including without limitation, ordinary industrial use, general office use, personal use, and household use, but are not designed, developed and manufactured as contemplated (1) for use accompanying fatal risks or dangers that, unless extremely high safety is secured, could have a serious effect to the public, and could lead directly to death, personal injury, severe physical damage or other loss (i.e., nuclear reaction control in nuclear facility, aircraft flight control, air traffic control, mass transport control, medical life support system, missile launch control in weapon system), or (2) for use requiring extremely high reliability (i.e., submersible repeater and artificial satellite).
  Please note that Fujitsu will not be liable against you and/or any third party for any claims or damages arising in connection with above-mentioned uses of the products.
- Any semiconductor devices have an inherent chance of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions.
- If any products described in this document represent goods or technologies subject to certain restrictions on export under the Foreign Exchange and Foreign Trade Law of Japan, the prior authorization by Japanese government will be required for export of those products from Japan.

# CONTENTS

# CHAPTER 1 OVERVIEW

**This chapter summarizes the features and basic specifications of the MB90M405 series of microcontrollers.**

# 1.1 Features

**The MB90M405 series of general-purpose 16-bit microcontrollers was developed for applications that require control of fluorescent display tube panels. The microcontrollers in this series have 60 high dielectric output pins for fluorescent display control.**

**The instruction set inherits the AT architecture of the F$^2$MC-8L and F$^2$MC-16L, and has additional instructions that support the C language. In addition, the instruction set supports extended addressing mode, enhanced signed multiply/divide instructions, and more powerful bit manipulation instructions. The microcontrollers also have a 32-bit accumulator that enables processing of long-word data.**

## ■ MB90M405 Series Features

### ❍ Clocks

- Built-in PLL clock multiplier circuit
- Source oscillation

  Main clock that divides the source oscillation by two

  PLL clock that multiplies the source oscillation by 1 to 4 (2.1 to 16.8 MHz when the source oscillation is 4.2 MHz), which can be configured from the machine clock
- Minimum instruction execution time: 59.5 ns (when the source oscillation is 4.2 MHz, the PLL clock is multiplied by 4, and $V_{CC}$ is 3 V)
- The source oscillation can be divided by 16, 32, 64, or 128 for external clock output.

### ❍ Maximum memory address space: 16 M bytes

24-bit addressing can also be used.

### ❍ Optimum instruction set for controller applications

- Many data types (bit, byte, and long word) can be handled.
- As many as 23 addressing modes are available.
- Efficient code (compiler)
- Enhanced high-precision arithmetic operations with a 32-bit accumulator
- Enhanced signed multiply/divide instructions and RETI instruction function

### ❍ Instruction set supporting the C language and multitasking

- System stack pointer
- Instruction set symmetry and barrel shift instructions

### ❍ Program patch function (two-address pointer)

❍ **Improved execution speed**

The built-in 4-byte instruction queue prereads instructions to improve execution speed.

❍ **Interrupt function**

- Eight programmable priority levels can be set.
- An enhanced interrupt function with 32 interrupt causes is supported.

❍ **Data transfer function**

❍ **Extended intelligent I/O service function: Up to 16 channels can be set.**

❍ **Low-power mode**

- Sleep mode (in which the CPU operating clock stops)
- Timebase timer mode (in which only the source oscillation clock and Timebase timer are active)
- Stop mode (in which the source oscillation stops)
- CPU intermittent operation mode (in which the CPU operates at every specified cycle)

❍ **Package**

- QFP-100 (FPT-100P-M06: 0.65 mm pin pitch)

❍ **Process**

CMOS technology

■ **Internal Peripheral Functions (resources)**

❍ **I/O ports: Up to 26 ports (used also for internal resources)**

❍ **Timebase timer: 1 channel**

❍ **Watchdog timer: 1 channel**

❍ **16-bit reload timer: 3 channels**

❍ **16-bit free-running timer: 1 channel**

❍ **Output compare: 1 channel**

- When the counter value of the 16-bit free-running timer matches the value set in the compare register, an interrupt request can be output.

❍ **Input capture: 2 channels**

- When the effective edge of a signal that is output from an external input pin is detected, the counter value of the 16-bit free-running timer can be read into the input capture data register and an interrupt request can be output.

❍ **Serial I/O: 2 channels**

❍ **UART: 2 channels**

- With full-duplex double buffer (8-bit length)
- Capable of asynchronous or clock synchronous serial transfer (I/O extended serial)

❍ **DTP/external interrupt (4 channels)**

- The input of an external interrupt can be used to activate the extended intelligent I/O service.
- The input of an external interrupt can be used to cause an internal hardware interrupt.

❍ **Delayed interrupt generator module**

Generates an interrupt request for task switching.

❍ **8/10-bit A/D converter (16 channels)**

Selectable resolution of 8 or 10 bits

❍ **FL-control circuit**

- Enables FL driver control (automatic display control of up to 32 digit lines and up to 59 segment lines)
  - Up to 32 digit lines (can be set line by line)
  - Dimmer setting
- Permits LED driver control (automatic display control of up to 16 lines)
  - Automatic display control of up to 16 lines with a 1/2 duty factor

❍ **Clock output circuit**

- Enables the source oscillation to be divided by 32, 64, 128, or 256 for clock output.

# 1.2 Product Lineup

**Table 1.2-1 "MB90M405 Series Product Lineup" shows the MB90M405 series product lineup.**

## ■ Product Lineup

Table 1.2-1 MB90M405 Series Product Lineup

| Model | MB90MV405 | MB90MF408 (*1)<br>MB90MF408A (*2) | MB90M408 (*1)<br>MB90M408A (*2) | MB90M407 (*1)<br>MB90M407A (*2) |
|---|---|---|---|---|
| Type | Evaluation device | Built-in flash memory | Built-in mask ROM | |
| ROM size | Not installed | 128K bytes | | 96K bytes |
| RAM size | 4K bytes | 4K bytes | | 4K bytes |
| CPU function | Number of basic instructions: 351<br>Minimum instruction execution time: 59.5 ns/4.2 MHz (when PLL clock is multiplied by 4)<br>Number of addressing modes: 23<br>Program patch function: 2-address pointer<br>Maximum memory address space: 16M bytes | | | |
| Port | I/O ports (CMOS): 26 (also used for resources) | | | |
| FL control circuit | 60 FL output lines (43 FL output lines and 17 LED control lines in LED control mode)<br>Capable of FL driver control and LED driver control<br>Enables dimmer setting for both digit and segment lines in FL driver control mode | | | |
| Serial I/O (UART) | With a full-duplex double buffer<br>Capable of synchronous or asynchronous clock transfer<br>Also can be used for clock synchronous extended serial I/O<br>Dedicated built-in baud rate generator<br>Four built-in channels (two channels are also used for the UART) | | | |
| 16-bit reload timer | 16-bit reload timer operation (can be set for toggle or one-shot output)<br>Supports an event count function<br>Three built-in channels | | | |
| 16-bit free-running timer | 16-bit output compare x 1 channel (for clearing the free-running timer)<br>16-bit input capture x 2 channels | | | |
| 8/10-bit A/D converter | 16 channels (input multiplexing)<br>Capable of 8-bit or 10-bit resolution<br>Conversion time: 5.9 μs (for an the operating machine clock of 16.8 MHz) | | | |
| Timing clock output circuit | An external input clock frequency can be divided and output externally.<br>Specifiable division ratio: Programmable to 1/16, 1/32, 1/64, or 1/128 | | | |
| I$^2$C bus | One built-in I$^2$C interface channel | | | |
| DTP/external interrupt | Four independent channels (can also be used for A/D input)<br>Interrupt source: "L" --> "H" edge, "H" --> "L" edge, "L" level, or "H" level can be set. | | | |

**Table 1.2-1 MB90M405 Series Product Lineup (Continued)**

| Model | MB90MV405 | MB90MF408 (*1)<br>MB90MF408A (*2) | MB90M408 (*1)<br>MB90M408A (*2) | MB90M407 (*1)<br>MB90M407A (*2) |
|---|---|---|---|---|
| Low-power mode | Sleep mode, Timebase timer mode, stop mode, and CPU intermittent mode | | | |
| Process | CMOS | | | |
| Package | PGA256 | QFP-100 (0.65 mm pitch) | | |
| Operating voltage | 3.3V ± 0.3V (16.8 MHz: 4.2 MHz multiplied by 4) | | | |

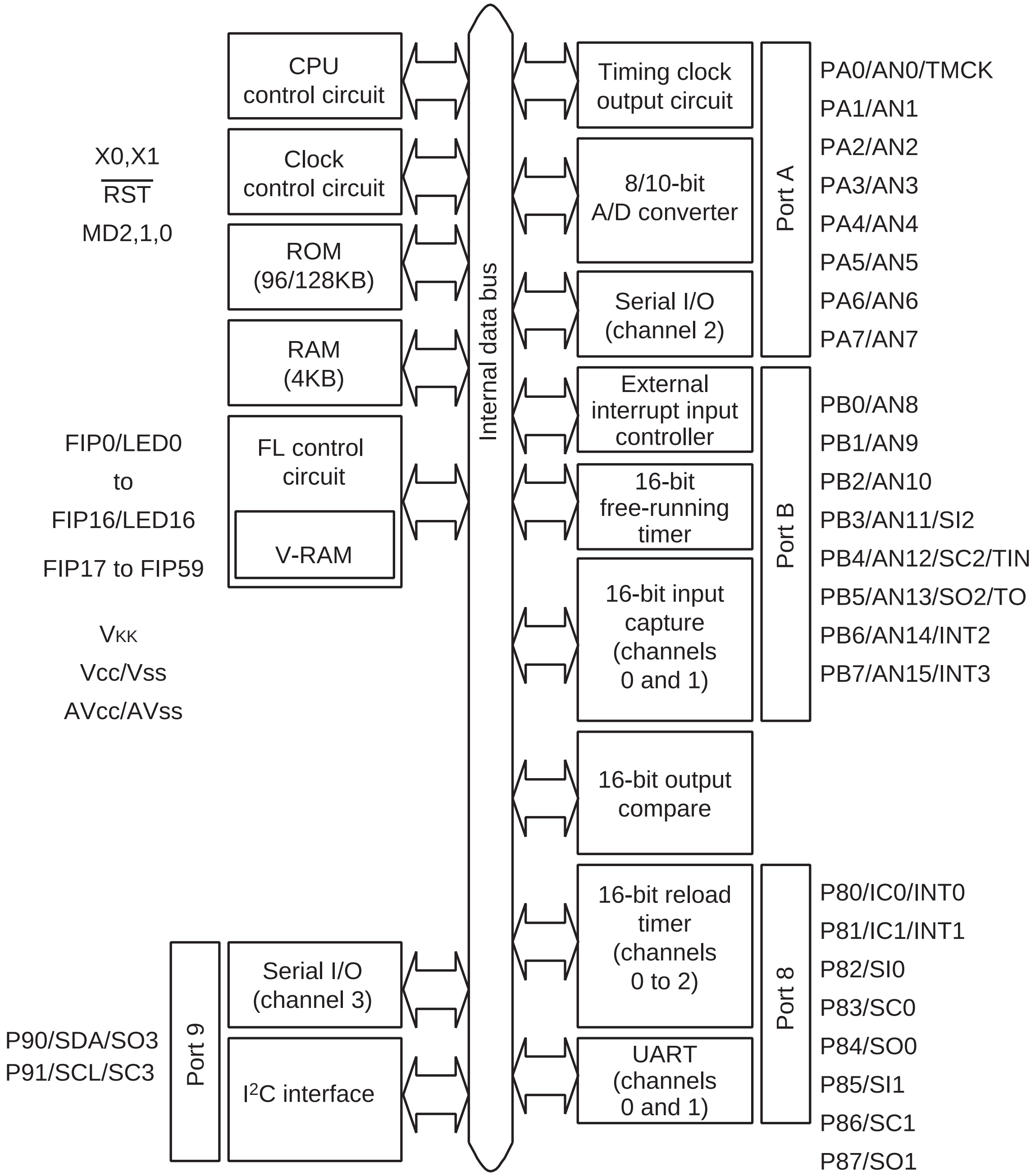
*1: The FL output pins (FIP00 to FIP59) are output with a pull-down resistor.
*2: The FL output pins (FIP00 to FIP16) are output without a pull-down resistor.
The FL output pins (FIP17 to FIP59) are output with a pull-down resistor.

# 1.3 Block Diagram

**Figure 1.3-1 "Block Diagram" shows a block diagram of the MB90M405 series of microcontrollers.**

## ■ Block Diagram

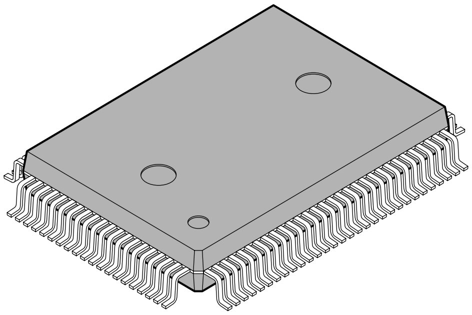
**Figure 1.3-1 Block Diagram**



## 1.4 Package Dimensions

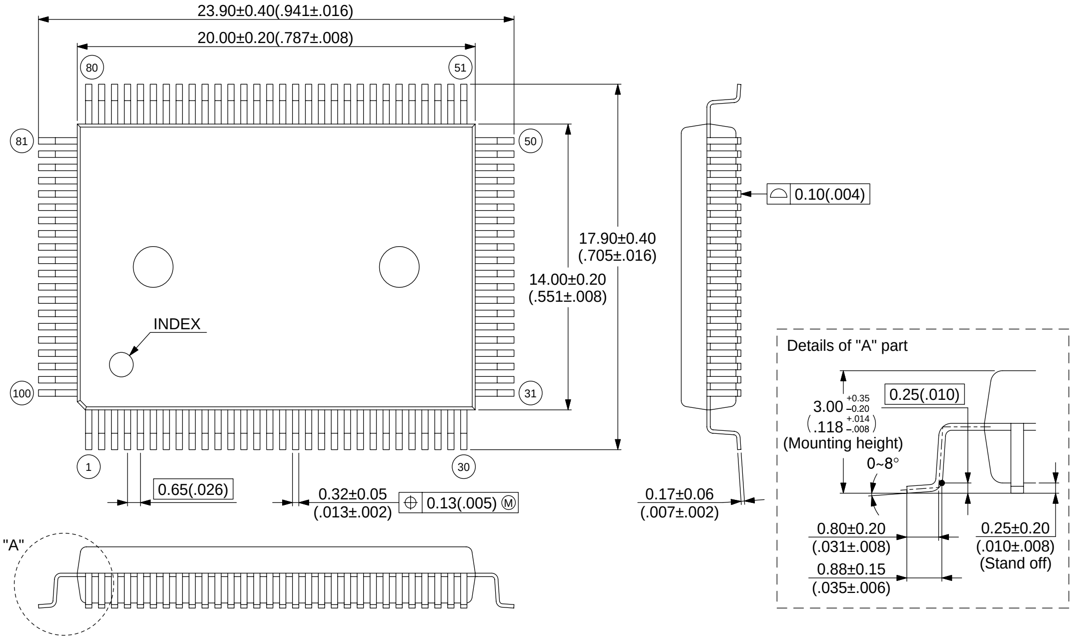
**This section provides the dimensions of the MB90M405 series package.**

### ■ FPT-100P-M06 Dimensions

| 100-pin plastic QFP<br><br>(FPT-100P-M06) | Lead pitch | 0.65 mm |
|---|---|---|
| | Package width × package length | 14.00 × 20.00 mm |
| | Lead shape | Gullwing |
| | Sealing method | Plastic mold |
| | Mounting height | 3.35 mm MAX |
| | | |
| | | |

100-pin plastic QFP
(FPT-100P-M06)

Note: Pins width and pins thickness include plating thickness.



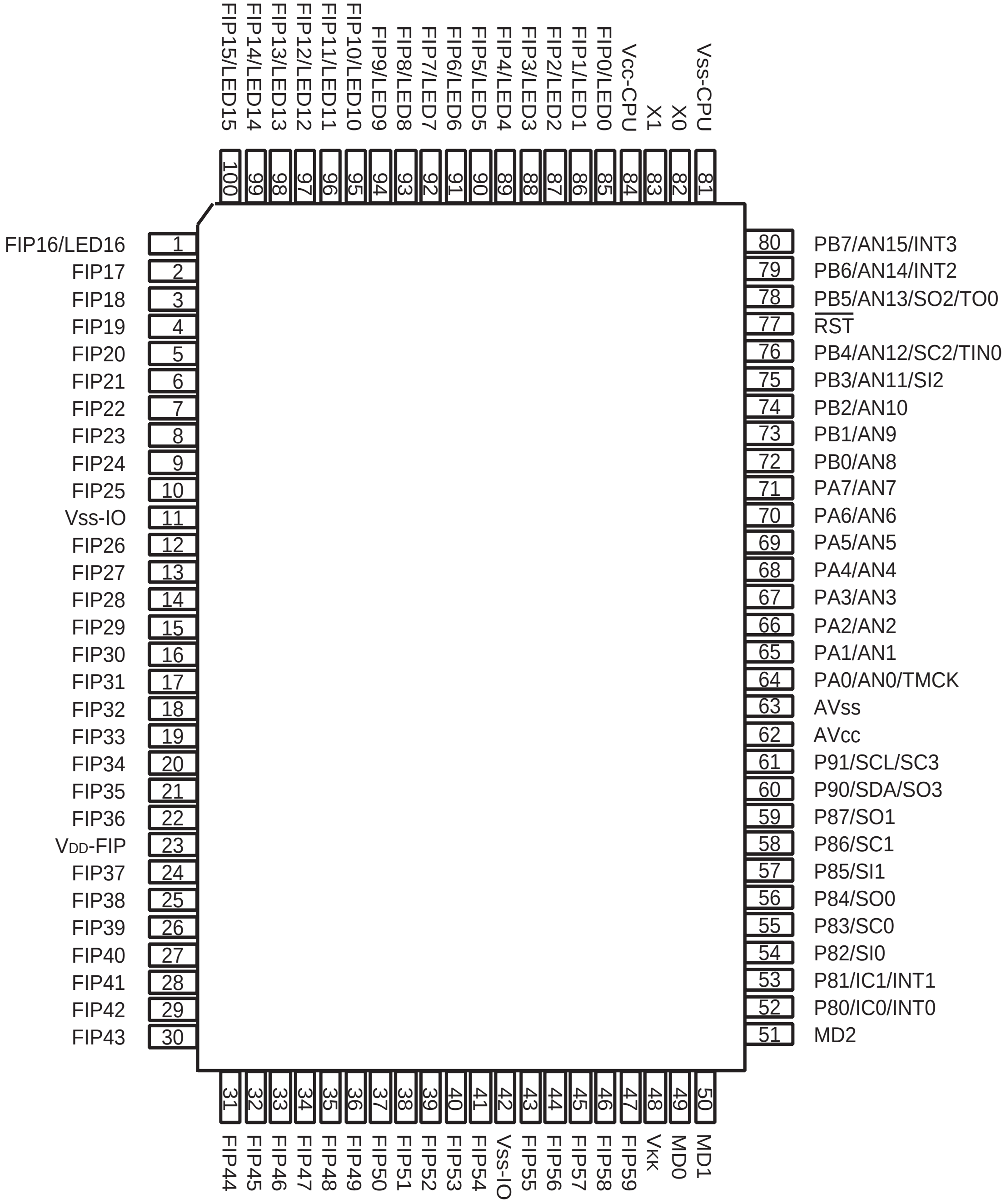
Dimensions in mm (inches).

# 1.5 Pin Assignments

**Figure 1.5-1 "Pin Assignments" shows the MB90M405 series pin assignments.**

## ■ Pin Assignments

**Figure 1.5-1 Pin Assignments**



# 1.6 Pin Functions

**Table 1.6-1 "Pin Functions" summarizes the pin names and functions, as well as the related circuit types and states at reset.**

### ■ Pin Functions

Table 1.6-1  Pin Functions

| Pin number | Pin name | Circuit type | State/function at reset | Function |
|---|---|---|---|---|
| QFP-100M06 | | | | |
| 82, 23 | X0, X1 | A | Oscillating | Oscillation input pin<br>When an external clock is connected, leave the X1 pin open. |
| 77 | $\overline{\text{RST}}$ | B | Reset input | External reset input pin |
| 85 to 100 | FIP0 to FIP15 | C | $V_{KK}$ pull-down output (when a pull-down resistor is set) | Set when the FL driver is enabled |
| | LED0 to LED15 | | | Set when the LED driver is enabled |
| 1 | FIP16 | | | Set when the FL driver is enabled |
| | LED16 | | | Set when the LED driver is enabled |
| 2 to 10<br>12 to 19 | FIP17 to FIP33 | | | Pin dedicated to FL driver output |
| 20 to 22<br>24 to 41<br>43 to 47 | FIP34 to FIP59 | D | | |
| 52 | P80 | E | Port input (Hi-z) | I/O port |
| | IC0 | | | External trigger input pin for input capture channel 0 |
| | INT0 | | | External cause input pin for external interrupt input channel 0<br>Input is enabled when the EN0 bit enables this pin. |
| 53 | P81 | | | I/O port |
| | IC1 | | | External trigger input pin for input capture channel 1 |
| | INT1 | | | External cause input pin for external interrupt input channel 1<br>Input is enabled when the EN1 bit enables this pin. |

Table 1.6-1 Pin Functions (Continued)

| Pin number | Pin name | Circuit type | State/function at reset | Function |
|---|---|---|---|---|
| QFP-100M06 | | | | |
| 54 | P82 | E | Port input (Hi-z) | I/O port |
| | SI0 | | | Serial data input pin for serial I/O channel 0 This pin is used occasionally while serial I/O channel 0 is performing an input operation. Do not use this pin for any other purpose during an input operation on serial I/O channel 0. |
| 55 | P83 | | | I/O port |
| | SC0 | | | Serial clock I/O pin for serial I/O channel 0 This function is enabled when serial I/O channel 0 is enabled for serial clock output. |
| 56 | P84 | | | I/O port |
| | SO0 | | | Serial data output pin for serial I/O channel 0 This function is enabled when serial I/O channel 0 is enabled for serial data output. |
| 57 | P85 | | | I/O port |
| | SI1 | | | Serial data input pin for serial I/O channel1 This pin is used occasionally while serial I/O channel 1 is performing an input operation. Do not use this pin for any other purpose during an input operation on serial I/O channel 1. |
| 58 | P86 | | | I/O port |
| | SC1 | | | Serial clock I/O pin for serial I/O channel 1 This function is enabled when serial I/O channel 1 is enabled for serial clock output. |
| 59 | P87 | | | I/O port |
| | SO1 | | | Serial data output pin for serial I/O channel 1 This function is enabled when serial I/O channel 1 is enabled for serial data output. |
| 60 | P90 | G | | I/O port (N-channel open drain) |
| | SDA | | | I$^2$C interface data I/O pin. This function is enabled when I$^2$C interface operation is enabled. Set the port to the input setting (DDR9 bit 8 = 0) while the I$^2$C interface is active. |
| | SO3 | | | Serial data output pin for serial I/O channel 3 This function is enabled when serial I/O channel 3 is enabled for serial data output. |

**Table 1.6-1 Pin Functions (Continued)**

| Pin number | Pin name | Circuit type | State/function at reset | Function |
|---|---|---|---|---|
| QFP-100M06 | | | | |
| 61 | P91 | G | Port input (Hi-z) | I/O port (N-channel open drain) |
| | SCL | | | $I^2C$ interface clock I/O pin. This function is effective when $I^2C$ interface operation is enabled. Set the port to the input setting (DDR9 bit 9 = 0) while the $I^2C$ interface is active. |
| | SC3 | | | Serial clock I/O pin for serial I/O channel 3<br>This function is enabled when serial I/O channel 3 is enabled for serial clock output. |
| 64 | PA0 | F | Analog input | I/O port |
| | AN0 | | | Analog input pin channel 0 for the A/D converter<br>This function is enabled when analog input is enabled (set by the ADER). |
| | TMCK | | | Timing clock output pin. This function is enabled when output is enabled.<br>The function is disabled when the ADER enables analog input. |
| 65 to 74 | PA1 to PB2 | | | I/O port |
| | AN1 to AN10 | | | Analog input pin channels 1 to 10 for the A/D converter<br>This function is enabled when analog input is enabled (set by the ADER). |
| 75 | PB3 | | | I/O port |
| | AN11 | | | Analog input pin channel 11 for the A/D converter<br>This function is enabled when analog input is enabled (set by the ADER). |
| | SI2 | | | Serial data input pin for serial I/O channel 2<br>This pin is used occasionally while serial I/O channel 2 is performing an input operation. Do not use this pin for any other purpose during an input operation on serial I/O channel 2. |
| 76 | PB4 | | | I/O port |
| | AN12 | | | Analog input pin channel 12 for the A/D converter<br>This function is enabled when analog input is enabled (set by the ADER). |
| | SC2 | | | Serial clock I/O pin for serial I/O channel 2<br>This function is enabled when serial I/O channel 2 is enabled for serial clock output. |
| | TIN0 | | | External clock input pin for reload timer channel 0<br>This function is enabled when external clock input is enabled (the ADER setting has precedence). |

Table 1.6-1 Pin Functions (Continued)

| Pin number | Pin name | Circuit type | State/function at reset | Function |
|---|---|---|---|---|
| QFP-100M06 | | | | |
| 78 | PB5 | F | Analog input | I/O port |
| | AN13 | | | Analog input pin channel 13 for the A/D converter<br>This function is enabled when analog input is enabled (set by the ADER). |
| | SO2 | | | Serial data output pin for serial I/O channel 2<br>This function is enabled when serial I/O channel 2 is enabled for serial data output. |
| | TO0 | | | External event output pin for reload timer channel 0<br>This function is enabled when external event output is enabled (the ADER setting has precedence). |
| 79, 80 | PB6 to PB7 | | | I/O port |
| | AN14 to AN15 | | | Analog input pin channels 14 and 15 for the A/D converter<br>This function is enabled when analog input is enabled (set by the ADER). |
| | INT2 to INT3 | | | External cause input pin for external interrupt input channels 2 and 3<br>Input is enabled when the EN2 and EN3 bits enable these pins. |
| 62 | $AV_{CC}$ | H | Power input | $V_{CC}$ power input pin for analog macro |
| 63 | $AV_{SS}$ | | | $V_{SS}$ power input pin for analog macro |
| 48 | $V_{KK}$ | - | | Power pin on pull-down side in high dielectric output mode |
| 49 | MD0 | B | Mode pin | Input pin for operation mode specification.<br>Connect this pin to $V_{CC}$.<br>When a flash boot program is used, be sure to change the pin to $V_{SS}$. |
| 50 | MD1 | | | Input pin for operation mode specification.<br>Connect this pin to $V_{CC}$. |
| 51 | MD2 | | | Input pin for operation mode specification.<br>Connect this pin to $V_{SS}$.<br>When a flash boot program is used, be sure to change the pin to $V_{CC}$. |
| 11, 42 | $V_{SS}$-IO | - | Power input | I/O power (0 V: GND) input pin |
| 23 | $V_{DD}$-FIP | | | FIP power (3 V: $V_{CC}$) input pin |
| 81 | $V_{SS}$-CPU | | | Control circuit power (0 V: GND) input pin |
| 84 | $V_{CC}$-CPU | | | Control circuit power (3 V: $V_{CC}$) input pin |

# 1.7 I/O Circuit Types

**Table 1.7-1 "I/O Circuit Types (continued next page)" summarizes the circuit types of individual pins.**

## ■ I/O Circuit Types

**Table 1.7-1 I/O Circuit Types (continued next page)**

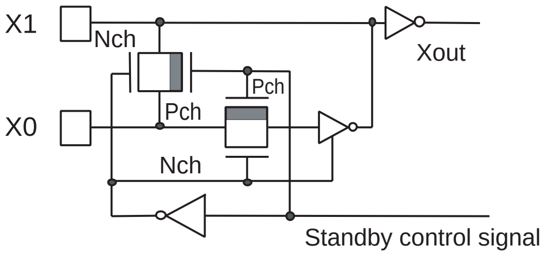
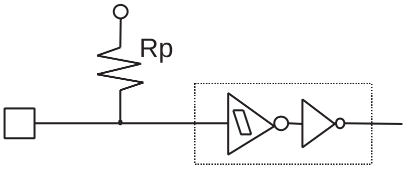
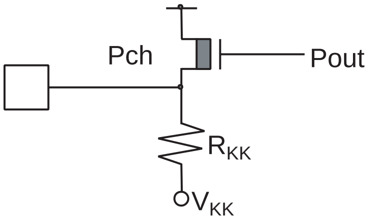
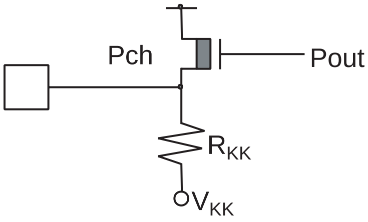
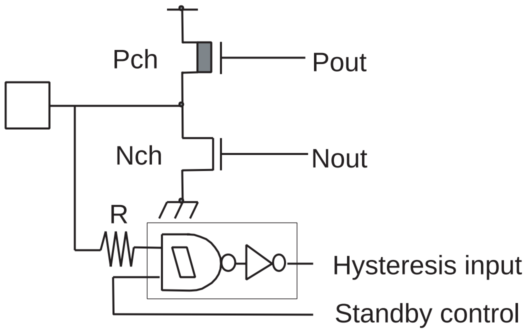
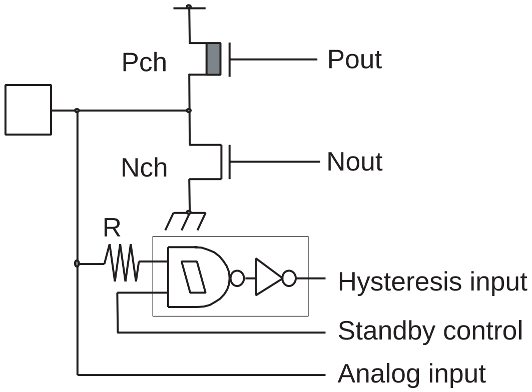
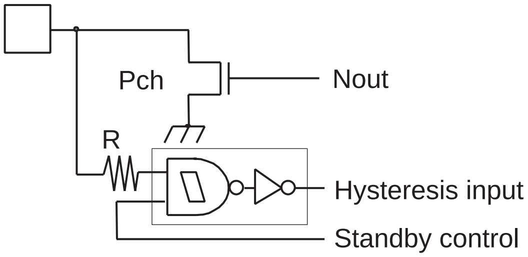
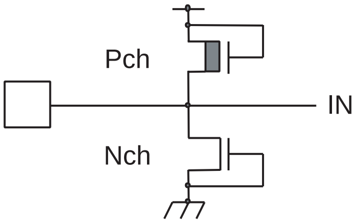
| Classification | Circuit type | Remarks |
|---|---|---|
| A |   X1<br>Nch<br>Xout<br>Pch<br>Pch<br>X0<br>Nch<br>Standby control signal | • Oscillation circuit<br>Oscillation feedback resistance = about 1 MΩ |
| B |  | • Hysteresis input pin<br>Built-in pull-up resistor (Rp) |
| C |  | • P-ch open drain output<br>- High dielectric port output<br>IOL = 25 mA<br>When this pin is used as an ordinary port, connect a diode clamp circuit to it to prevent $V_{KK}$ voltage from being applied to the pin when the "L" level is output. (See Section 1.8, "Notes on Handling Devices." |
| D |  | • P-ch open drain output<br>- High dielectric port output<br>IOL = 12 mA<br>When this pin is used as a normal port, connect a diode clamp circuit to it to prevent $V_{KK}$ voltage from being applied to the pin when the "L" level is output. (See Section 1.8 "Notes on Handling Devices." |

Table 1.7-1 I/O Circuit Types (continued next page)

| Classification | Circuit type | Remarks |
|---|---|---|
| E |  Pch<br>Pout<br>Nch<br>Nout<br>R<br>Hysteresis input<br>Standby control | • CMOS hysteresis I/O pin<br>- CMOS output<br>- CMOS hysteresis input (with standby control for input rejection)<br>IOL = 4 mA |
| F |  Pch<br>Pout<br>Nch<br>Nout<br>R<br>Hysteresis input<br>Standby control<br>Analog input | • Analog/CMOS hysteresis I/O pin<br>- CMOS output<br>- CMOS hysteresis input (with standby control for input rejection)<br>- Analog input (Analog input is enabled when the corresponding ADER bit is "1".)<br>IOL = 4 mA |
| G |  | • N-ch open drain output<br>- CMOS hysteresis input (with standby control for input rejection)<br>Unlike the CMOS I/O pin, this pin has no Pch transistor. Therefore, even when voltage is applied externally to this pin while the device power is off, no current flows into the device power supply ($V_{CC}$-IO/$V_{CC}$-CPU). |
| H |  | • Analog power input protection circuit |

## 1.8 Notes on Handling Devices

**When handling devices, note the following:**

- **Strict observation of maximum rated voltage (latchup prevention)**
- **Stabilization of supply voltage**
- **Note on power-on**
- **Treatment of unused input pins**
- **Note on external clocks**
- **Power supply pins**
- **Crystal oscillation circuit**
- **Power-on sequence for A/D converter power supply analog input**
- **Treating pins when the A/D converter is not used**
- **Output of high dielectric output pin (circuit type C or D)**
- **Note on during operation of PLL clock mode**

### ■ Strict Observation of Maximum Rated Voltage (Latchup Prevention)

- Do not apply a voltage higher than $V_{CC}$ or lower than $V_{SS}$ to CMOS IC input and output pins that are not medium or high dielectric pins. Also, do not apply a voltage higher than the rating between $V_{CC}$ and $V_{SS}$. Disregard of these precautions may result in latchup.
- A latchup rapidly increases supply current and may result in thermal damage to elements. Be careful not to apply voltage exceeding the maximum rating.
- When turning power to analog circuits on or off, ensure that the analog power supply ($AV_{CC}$) and analog input voltage do not exceed the digital supply voltage ($V_{CC}$).

### ■ Stabilization of Supply Voltage

Even within the operation guarantee range of the $V_{CC}$ supply voltage, a malfunction can be caused if the supply voltage changes abruptly. To prevent problems from occurring, stabilize the $V_{CC}$ supply voltage.

As guidelines for voltage stabilization, the $V_{CC}$ ripple fluctuations (peak-to-peak value) at commercial frequencies (50 to 60 Hz) should be suppressed to 10% or less of the reference $V_{CC}$ value. During a momentary change such as when a supply voltage is switched, voltage functions should also be suppressed so that the transient fluctuation rate does not exceed 0.1 V/ms.

### ■ Note on Power-on

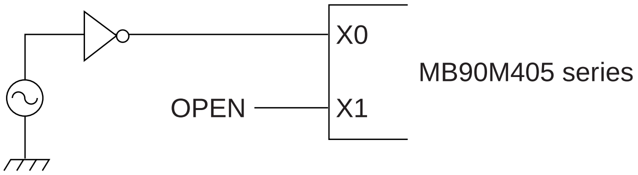
To prevent a malfunction in the built-in voltage drop circuit during power-on, ensure 50 µs (between 0.2 V and 2.7 V) or more for the supply voltage ($V_{CC}$) rise time.

### ■ Treatment of Unused Input Pins

An unused input pin, if left open, may cause a malfunction or a permanent damage due to a latchup. Every unused input pin must be pulled up or down using resistance of 2 kΩ or more. An unused I/O pin must be either opened by setting it to output mode or handled in the same way as an input pin after it has been set to input mode.

## ■ Note on External Clocks

When an external clock is used, connect only the X0 pin; leave the X1 pin open. A sample application of the external clock is shown below:



## ■ Power Supply Pins

- When a device has two or more $V_{CC}$ or $V_{SS}$ pins, the pins that should have equal potential are connected within the device in order to prevent a latchup or other malfunction. To reduce extraneous emissions, to prevent a malfunction of the strobe signal due to an increase in the ground level, and to maintain the total output current rating, connect the $V_{CC}$ or $V_{SS}$ pins to the power supply or to ground.
- Connect the current supply source to the $V_{CC}$ and $V_{SS}$ pins of the MB90M405 series device with minimum impedance.
- As a measure against power supply noise in an MB90M405 series device, connect a bypass capacitor of about 0.1 µF between $V_{CC}$ and $V_{SS}$ near the $V_{CC}$ and $V_{SS}$ pins.

## ■ Crystal Oscillation Circuit

Noise at the X0 and X1 pins can cause malfunctioning of MB90M405 series devices. Design the printed wiring board so that bypass capacitors to the X0 and X1 pins, crystal oscillator (or ceramic oscillator), and ground are provided near the X0 and X1 pins and that the X0 and X1 pin wirings do not cross other wirings.

Printed wiring board artwork that encloses the X0 and X1 pins with ground should result in stable operation.

## ■ Power-on Sequence for A/D Converter Power Supply Analog Input

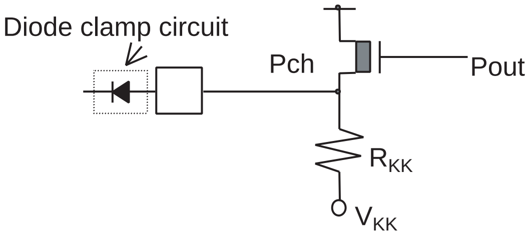
- Before turning on power to the A/D converter power supply pin ($AV_{CC}$) and analog input pins (AN0 to AN15), be sure power to the digital power supply pin ($V_{CC}$) has already been turned on.
- When turning off the power, turn off the power to the digital power supply pin ($V_{CC}$) after turning off the power to the A/D converter and analog inputs.
- When a pin that is used for analog input is also used as an input port, prevent the analog input voltage from exceeding $AV_{CC}$. (The analog power and digital power can be simultaneously turned on or off with no problem.)

## ■ Treating Pins When the A/D Converter Is Not Used

When the A/D converter is not used, connect $AV_{CC}$ and $V_{CC}$, and $AV_{SS}$ and $V_{SS}$.

## ■ Output of High Dielectric Output Pin (Circuit Type C or D)

When a high dielectric output pin (circuit type C or D) is used as an ordinary output port, the value obtained by pulling down the $V_{KK}$ pin voltage is output in "L" level output mode. In this case, the $V_{KK}$ pin level voltage is applied to the external circuit. Add a diode clamp circuit as shown in the figure below.



## ■ Note on during Operation of PLL Clock Mode

In the case of the PLL clock is selected as the machine clock, if an oscillator is off or if the clock input is stopped, the microcontroller may be continued to operation by the free running frequency of the internal PLL self oscillator circuit.

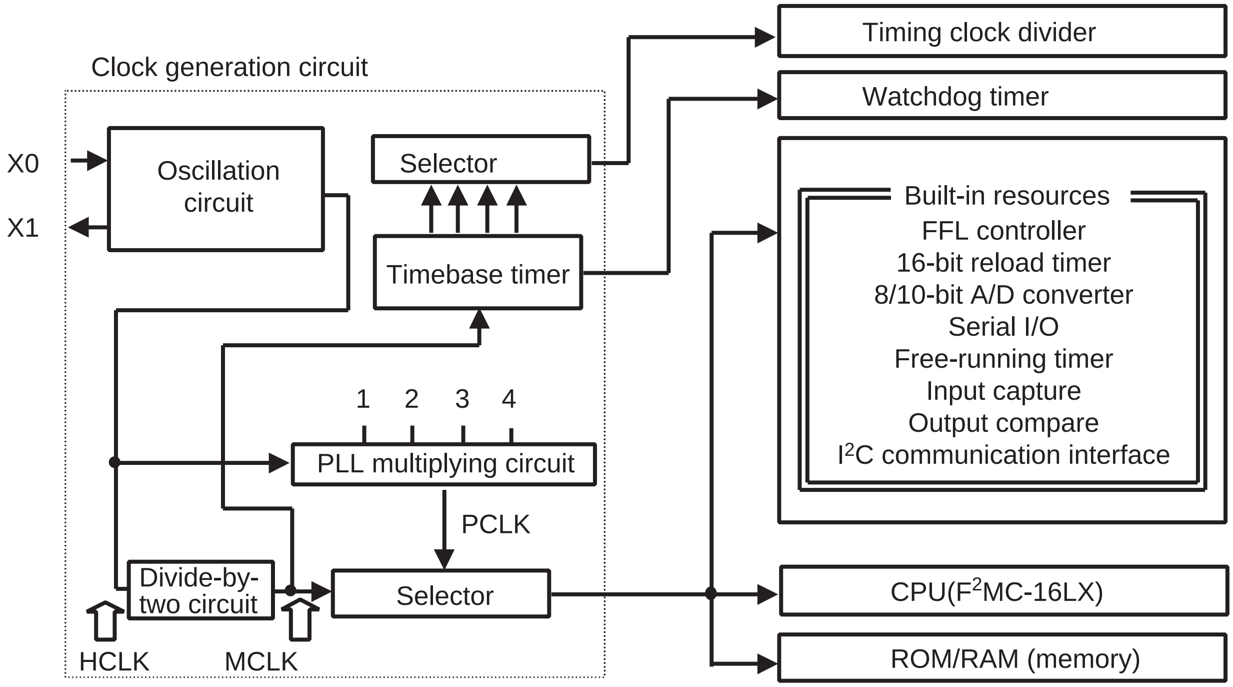
This operation is warranted.

# 1.9 Clock Supply Map

**Figure 1.9-1 "Clock Supply Map" shows a clock supply map of the MB90M405 series.**

## ■ Clock Supply Map

**Figure 1.9-1 Clock Supply Map**



HCLK : Oscillation clock frequency
MCLK : Main clock frequency
PCLK : PLL clock frequency

# 1.10 Low Power Consumption Mode

**This section provides an overview of low-power mode. The MB90M405 series uses the operation modes listed in the table below, and functions and clocks stop differently depending on the mode. For more information, see CHAPTER 4 "CLOCKS."**

## ■ Relationships between operation Modes and Power

**Table 1.10-1 Relationships between Operation Modes and Power**

| Operation mode | Main clock | PLL clock | CPU | Built-in resources | Timing clock |
|---|---|---|---|---|---|
| PLL Run | Operating | Operating | Operating | Operating | Operating |
| Main Run | Operating | Stopped | Operating | Operating | Operating |
| PLL Sleep | Operating | Operating | Stopped | Operating (*1) | Operating |
| Main Sleep | Operating | Stopped | Stopped | Operating (*1) | Operating |
| Pseudo Clock | Operating | Stopped | Stopped | Stopped | Operating |
| Stop | Stopped | Stopped | Stopped | Stopped | Stopped |

In the above table, the operation modes are arranged in descending order of power consumption
In PLL Run mode, operation is performed based on PCLK obtained by multiplying the source oscillation by 1 to 4. In Main Run mode, operation is performed based on MCLK obtained by dividing the source clock by 2.
*1: In Sleep mode, since the CPU has stopped, access to the built-in resources from the CPU is disabled.

# CHAPTER 2 CPU

**This chapter describes the CPU and the memory space provided by the MB90M405 series.**

## 2.1 CPU

**The F$^2$MC-16LX CPU core is a 16-bit CPU designed for use in applications, such as welfare and mobile equipment, which require high-speed real-time processing. The instruction set of the F$^2$MC-16LX was designed for controllers so that it can perform various types of control at high speeds and efficiencies.**

**The F$^2$MC-16LX CPU core processes 32-bit data using a built-in 32-bit accumulator. Memory space, which can be extended to up to 16M bytes, can be accessed in either linear or bank access mode. The instruction set inherits the AT architecture of the F$^2$MC-8L and has additional instructions for supporting the C language. In addition, it has an extended addressing mode, enhanced multiply/divide instructions, and improved bit manipulation instructions. The features of the F$^2$MC-16XL CPU are shown below:**

### ■ CPU

❍ **Minimum instruction execution time: 59.5 ns (source oscillation at 4.2 MHz and clock multiplication by 4)**

❍ **Maximum memory address space: 16M bytes. Can be accessed in linear or bank mode.**

❍ **Instruction set optimum for controller applications**

- Many data types (bit, byte, word, and long word)
- As many as 23 addressing modes
- Enhanced high-precision arithmetic operation by a 32-bit accumulator
- Enhanced signed multiply/divide instructions and RETI instruction function

❍ **Interrupt function**

Eight programmable priority levels

❍ **Automatic transfer function independent to CPU**

Extended intelligent I/O service using up to 16 channels

❍ **Instruction set supporting high-level language (C) and multi-tasking**

System stack pointer, instruction set symmetry, and barrel shift instructions

❍ **Increased execution speed: 4-byte instruction queue**

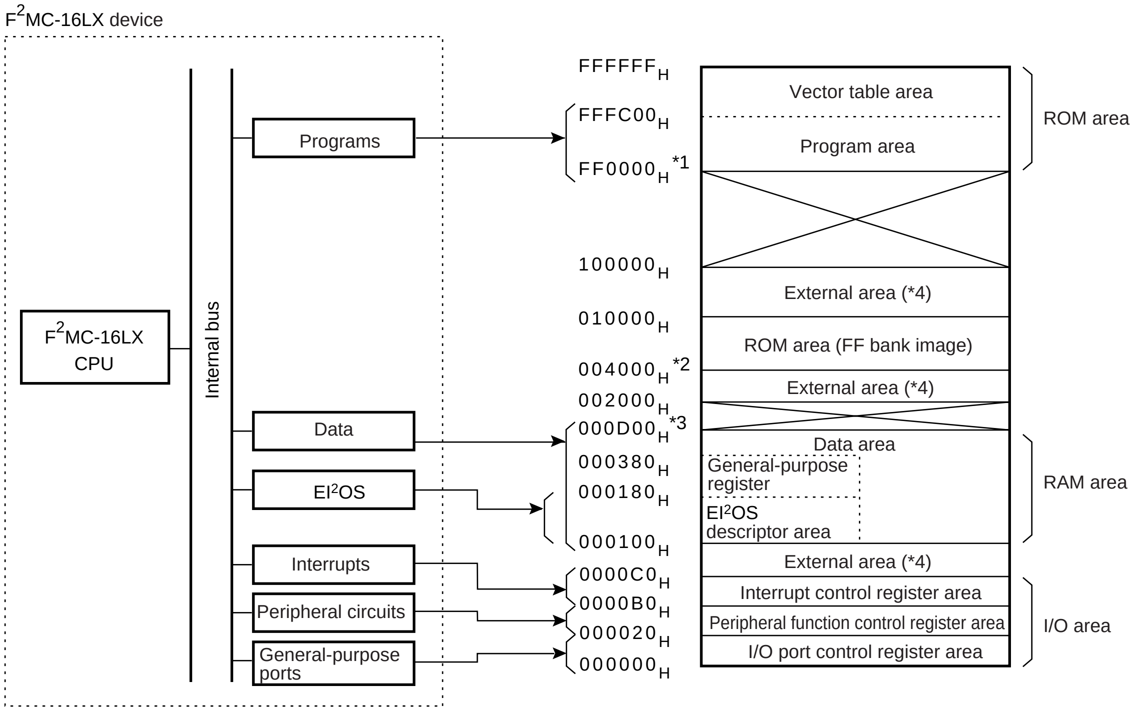
# 2.2 Memory Space

**All I/O, programs, and data are located in the 16M-byte memory space of the $F^2$MC-16LX. The RAM space is used for extended intelligent I/O service ($EI^2OS$) descriptors, general-purpose registers, and vector tables.**

## ■ Memory Space

All I/O, programs, and data are located in the 16M-byte memory space of the $F^2$MC-16LX CPU. The CPU is able to access each built-in peripheral function (resource) through a memory space address indicated by the 24-bit address bus.

**Figure 2.2-1 Sample Relationship between the $F^2$MC-16LX System**



*1 The size of internal ROM differs for each model.

*2 The area accessible by the image differs for each model.

*3 The size of internal RAM differs for each model.

*4 There is no access in single-chip mode.

## ■ ROM Area

### ❍ Vector table area (address: "FFFC00$_H$ to FFFFFF$_H$")

- This area is used as a vector table for vector call instructions, interrupt vectors, and reset vectors.
- This area is allocated at the highest addresses of the ROM area. The start address of the corresponding processing routine is stored in each vector table address.

### ❍ Program area (address: Up to "FFFBFF$_H$")

- ROM is built in as an internal program area.
- The size of the internal ROM differs for each model.

## ■ RAM Area

### ❍ Data area (address: From "000100$_H$")

- The static RAM is built in as an internal data area.
- The size of internal RAM differs for each model.

### ❍ General-purpose register area (address: "000180$_H$ to 00037F$_H$")

- Auxiliary registers used for 8-bit, 16-bit, and 32-bit arithmetic operations and transfer are allocated in this area.
- When this area is not used as a general purpose register, it can be used as ordinary RAM.
- When this area is used as a general-purpose register, general-purpose register addressing enables high-speed access within a few instruction cycles.

### ❍ Extended intelligent I/O service (EI$^2$OS) descriptor area (address: "000100$_H$ to 00017F$_H$")

- This area retains the transfer modes, I/O addresses, transfer count, and buffer addresses of the Extended Intelligent I/O Service (EI$^2$OS).
- If the Extended Intelligent I/O Service (EI$^2$OS) is not used, this area can be used as ordinary RAM.

## ■ I/O Area

### ❍ Interrupt control register area (address: "0000B0$_H$ to 0000BF$_H$")

The interrupt control registers (ICR00 to ICR15) correspond to the built-in peripheral functions that have an interrupt function. These registers set interrupt levels and control the extended intelligent I/O service (EI$^2$OS).

### ❍ Peripheral function control register area (address: "000020$_H$ to 0000AF$_H$")

This register controls the built-in peripheral functions (resources).

### ❍ I/O port control register area (address: "000000$_H$ to 00001F$_H$")

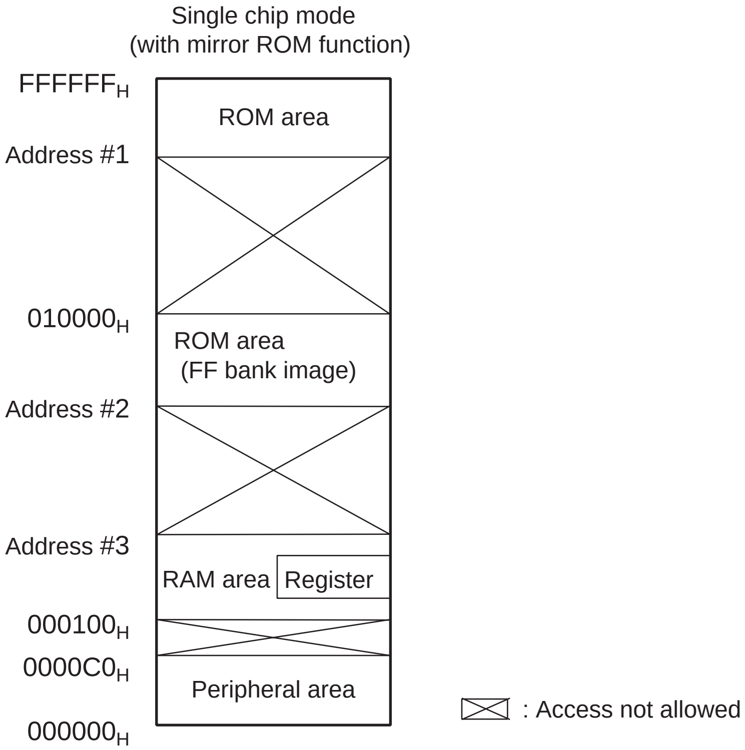
This register controls the I/O port.

# 2.3 Memory Maps

**This section shows the memory map for each model of MB90M405 series.**

## ■ Memory Maps

**Figure 2.3-1 Memory Maps**



| Model | Address #1 | Address #2 | Address #3 |
|---|---|---|---|
| MB90M407/M407A | FE8000$_H$ | 004000$_H$ | 001100$_H$ |
| MB90M408/M408A | FE0000$_H$ | 004000$_H$ | 001100$_H$ |
| MB90MF408/MF408A | FE0000$_H$ | 004000$_H$ | 001100$_H$ |
| MB90MV405 | F80000$_H$ (*1) | 004000$_H$ | 001100$_H$ |

*1: V products have no built-in ROM. This area should be understood as the ROM decode area of the tool.

**Reference:**

The ROM mirroring function allows the small model of the C compiler to be used.

The low-order 16-bit address of FF bank becomes the same as the low-order 16-bit address of 00 bank. However, not all the data in the ROM area can be seen as a mirror image in 00 bank because the ROM area of FF bank exceeds 48K bytes.

To use the small model of the C compiler, store the data table in "$FF4000_H$ to $FFFFFF_H$" to show the data table as a mirror image in "$004000_H$ to $00FFFF_H$". Thus, the data table in the ROM area can be referenced without the "far" specification declared in the pointer.

**Note:**

- If the ROM mirroring function register is set, the data in the upper side of FF bank ("$FF4000_H$ to $FFFFFF_H$") can be seen as a mirror image in the upper side of 00 bank ("$004000_H$ to $00FFFF_H$").
- For information on setting the ROM mirroring function, see Chapter 22 "ROM MIRRORING FUNCTION SELECTION MODULE".

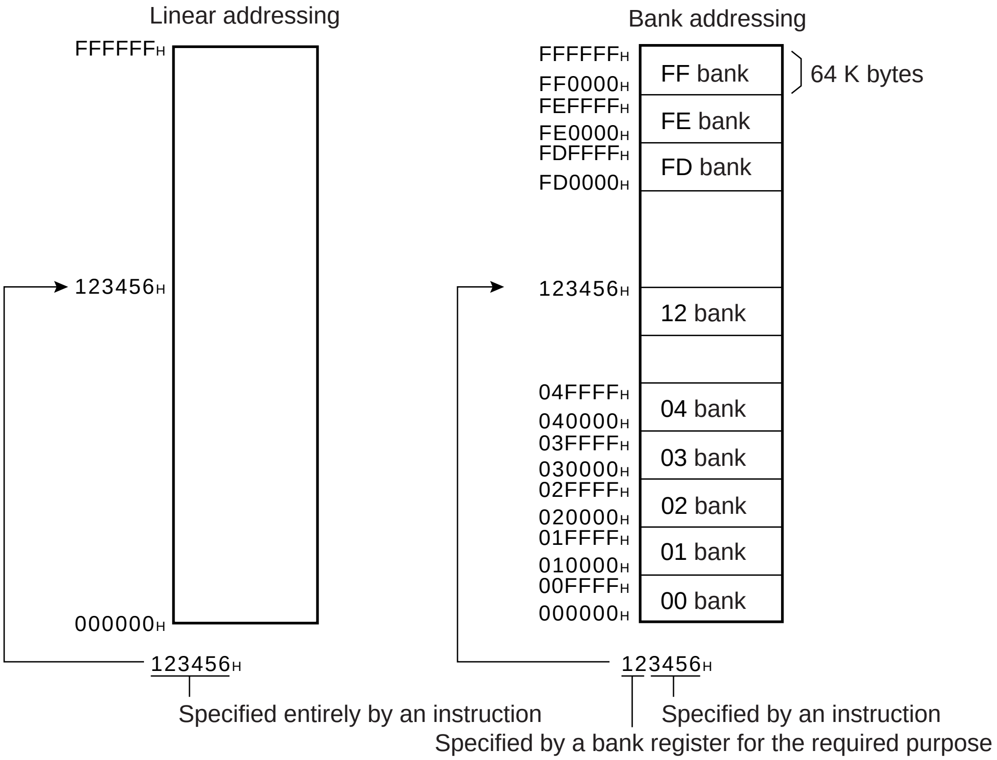
## 2.4 Addressing

**Addresses can be generated using linear addressing and bank addressing.**
**In linear addressing, the 16M-byte space is directly specified using a continuous 24-bit address.**
**In bank addressing, the 16M-byte space is divided into 256 64K-byte banks. The upper 8 bits of the address are specified by a bank register and the lower 16 bits of the address are directly specified by an instruction.**

**The F$^2$MC-16LX series basically uses bank addressing.**

### ■ Linear Addressing and Bank Addressing

**Figure 2.4-1 Linear Addressing and Bank Addressing Memory Management**



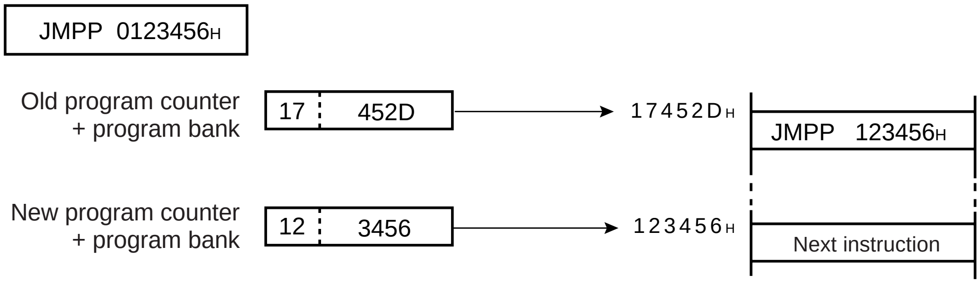
## 2.4.1 Address Specification by Linear Addressing

**The linear addressing has two types of address specified:**

- **Specify 24-bit address directly in the instruction as operand**
- **Specify 24-bit address in a 32-bit general-purpose registers, using the lower 24 bits**

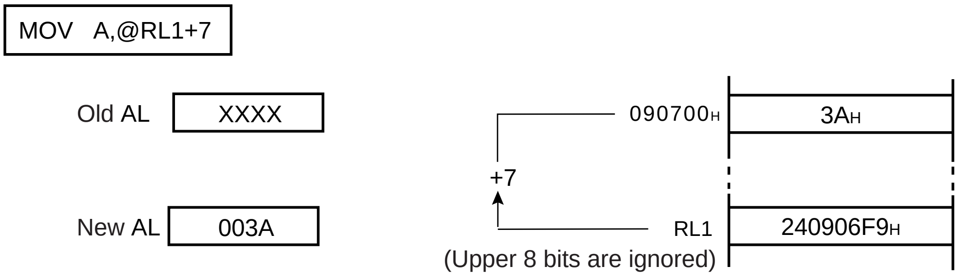
### ■ Linear Addressing by 24-Bit Operand Specification

**Figure 2.4-2 Example of Direct Specification of a 24-bit Physical Address in Linear Addressing**



### ■ Addressing by Indirect Specification with a 32-Bit Register

**Figure 2.4-3 Example of Indirect Specification with a 32-bit General-Purpose Register in Linear Addressing**



RL1: 32-bit (long-word) general-purpose register

## 2.4.2 Address Specification by Bank Addressing

**In address specification by bank addressing, the 16M-byte space is divided into 256 64K-byte banks. The upper 8 bits of the address are specified by a bank register and the lower 16 bits of the address are directly specified by an instruction.**

**The five types of bank registers classified by function are as follows:**

- **Program bank register (PCB)**
- **Data bank register (DTB)**
- **User stack bank register (USB)**
- **System stack bank register (SSB)**
- **Additional bank register (ADB)**

### ■ Bank Registers and Access Space

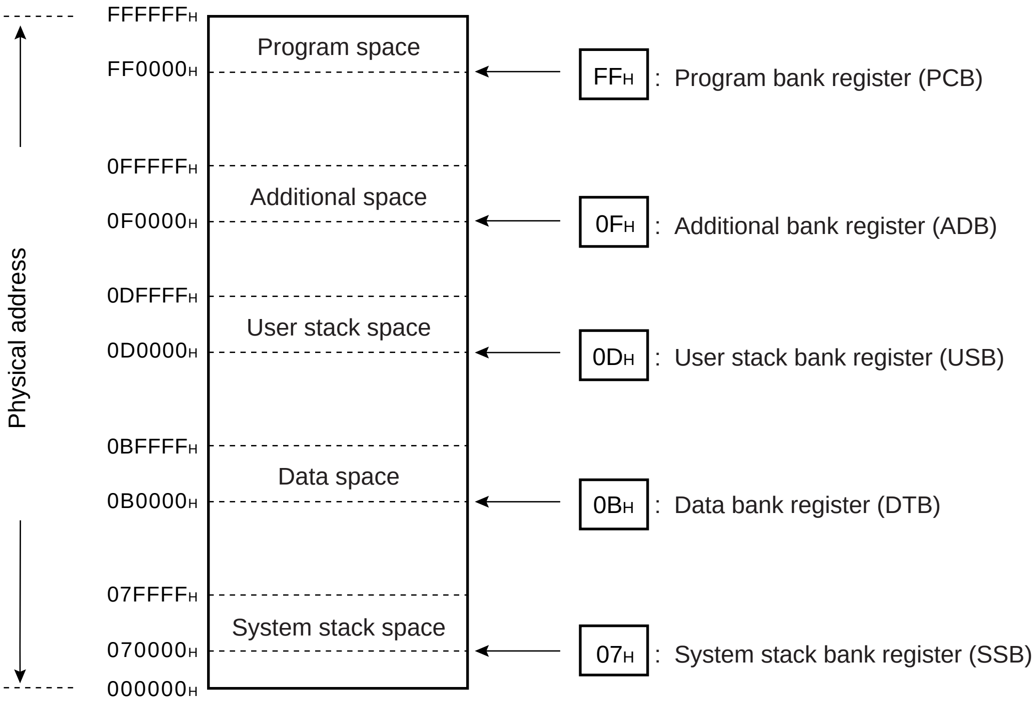
**Table 2.4-1 Access Space and Main Function of Each Bank Register**

| Bank register name | Access space | Main function | Initial value after a reset |
|---|---|---|---|
| Program bank register (PCB) | Program (PC) space | Instruction codes, vector tables, and immediate-value data are stored. | $FF_H$ |
| Data bank register (DTB) | Data (DT) space | Read/write data is stored. Internal or external peripheral control registers and data registers are accessed. | $00_H$ |
| User stack bank register (USB) | Stack (SP) space | This area is used for stack accesses such as when PUSH/POP instructions and interrupt registers are saved.<br>The SSB is used when the stack flag in the condition register (CCR:S) is 1. The USB is used when the stack flag in the condition register (CCR:S) is 0. (*1) | $00_H$ |
| System stack bank register (SSB) (*1) | | | $00_H$ |
| Additional bank register (ADB) | Additional (AD) space | Data that overflows from the data (DT) space is stored. | $00_H$ |

*1: The SSB is always used as an interrupt stack.

See Section 2.7.9 "Bank Registers (PCB, DTB, USB, SSB, ADB)" for details.

Figure 2.4-4 Sample Bank Addressing


FFFFFFH
FF0000H
0FFFFFH
0F0000H
0DFFFFH
0D0000H
0BFFFFH
0B0000H
07FFFFH
070000H
000000H
Physical address
Program space
Additional space
User stack space
Data space
System stack space
FFH : Program bank register (PCB)
0FH : Additional bank register (ADB)
0DH : User stack bank register (USB)
0BH : Data bank register (DTB)
07H : System stack bank register (SSB)

## ■ Bank Addressing and Default Space

To improve instruction coding efficiency, each instruction has a defined default space for each addressing method, as shown in Table 2.4-2 "Addressing and Default Spaces". To use a space other than the default space, specify a prefix code for a bank before the instruction. This enables the bank space that corresponds to the specified prefix code to be accessed.See Section 2.9 "Prefix Codes" for details about prefix codes.

Table 2.4-2 Addressing and Default Spaces

| Default space | Addressing |
|---|---|
| Program space | PC indirect, program access, branching |
| Data space | Addressing using @RW0, @RW1, @RW4, and @RW5, @A, addr16, dir |
| Stack space | Addressing using PUSHW, POPW, @RW3, and @RW7 |
| Additional space | Addressing using @RW2 and @RW6 |

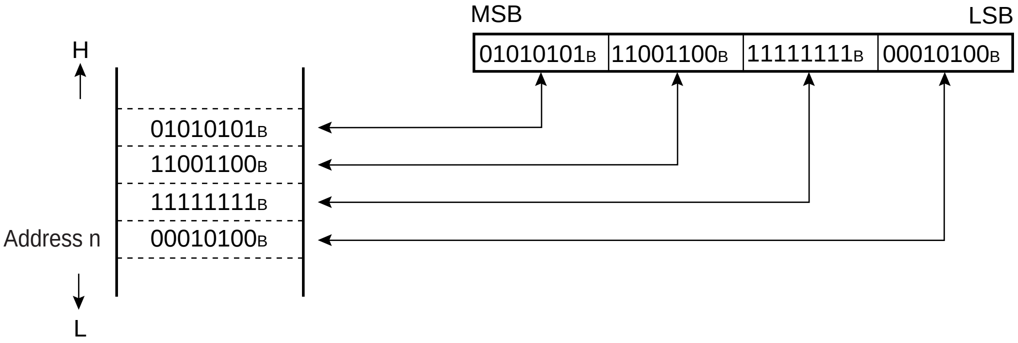
# 2.5 Memory Location of Multibyte Data

**Multibyte data is written to memory sequentially from the lower address. If multibyte data is 32-bit data, the lower 16 bits are transferred followed by the upper 16 bits.**
**If a reset signal is input immediately after the low-order data is written, the high-order data may not be written.**

## ■ Storage of Multibyte Data in RAM

The lower 8 bits of the data is located at address n, and subsequent data is located at address n + 1, address n + 2, address n + 3, and so on, in this sequence.

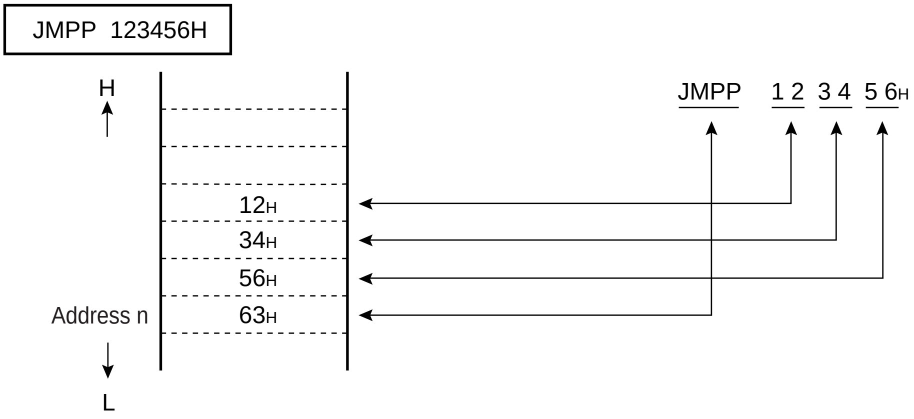
**Figure 2.5-1 Storage of Multibyte Data in RAM**



MSB : Most Significant Bit
LSB : Least Significant Bit

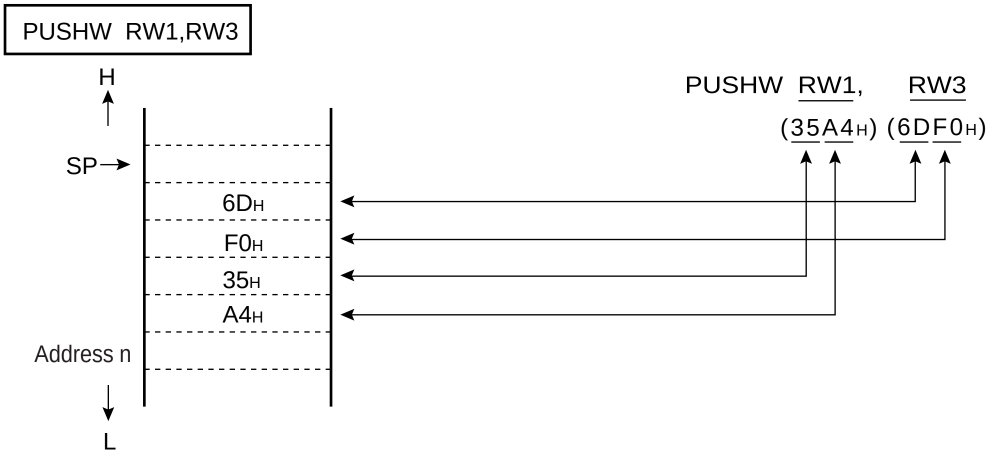
## ■ Storage of Multibyte Operand

**Figure 2.5-2 Storage of a Multibyte Operand in RAM**



## ■ Storage of Multibyte Data in a Stack

**Figure 2.5-3 Storage of Multibyte Data in a Stack**

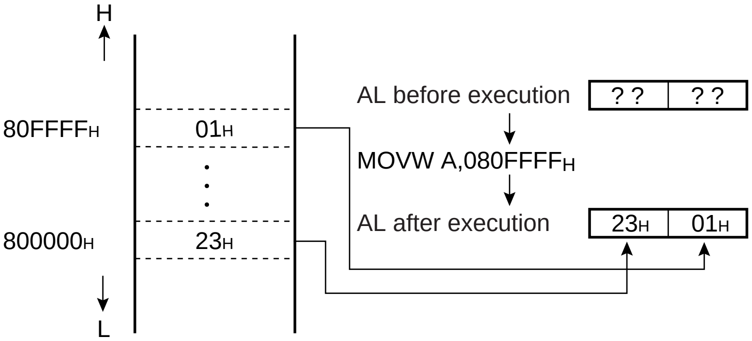


RW1: 35A4H
RW3: 6DF0H
*1 Stack status after execution of the PUSHW instruction

## ■ Multibyte Data Access

Multibyte data is generally accessed within a bank. For an instruction that accesses multibyte data, the address "FFFF$_H$" is followed by "0000$_H$" in the same bank.

**Figure 2.5-4 Multibyte Data Access on a Bank Boundary**



# 2.6 Registers

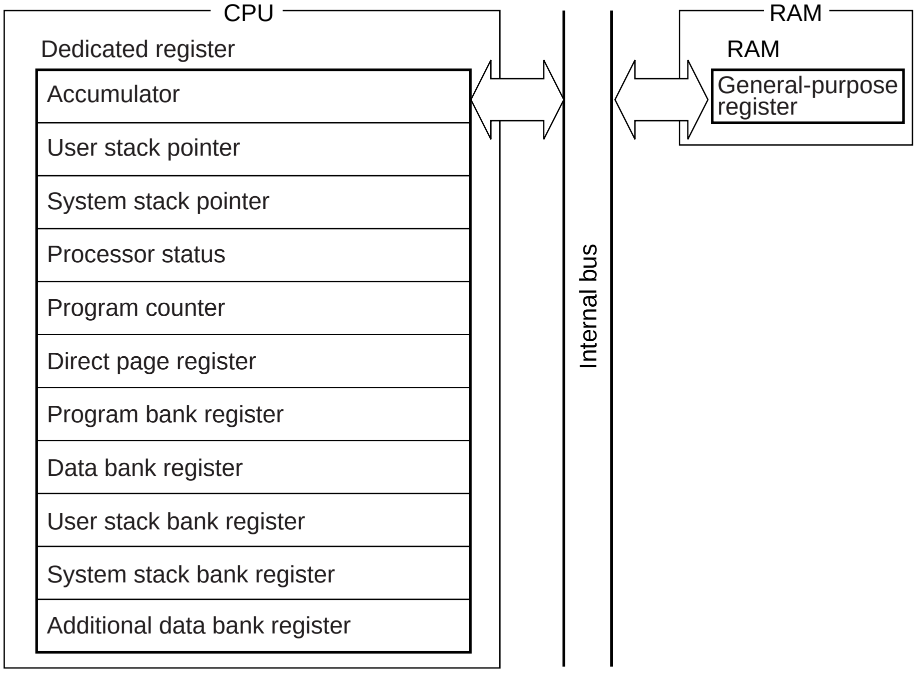
**F²MC-16LX registers are classified into internal dedicated CPU registers and built-in RAM general-purpose registers.**

## ■ Dedicated Registers and General-Purpose Registers

he dedicated registers are built-in hardware components of the CPU. Consequently, their use is limited by the CPU architecture.

General-purpose registers are allocated together with RAM in the CPU address space. General-purpose registers are like dedicated registers in that they can be accessed without address specifications, but are like regular memory in that their use can be defined by the user.

**Figure 2.6-1 Dedicated Registers and General-Purpose Registers**



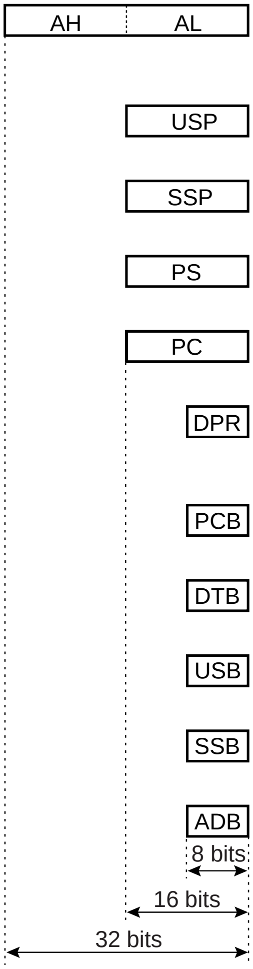
# 2.7 Dedicated Registers

**The following 11 registers are dedicated registers in the CPU.**

- **Accumulator (A)**
- **User stack pointer (USP)**
- **System stack pointer (SSP)**
- **Processor status (PS)**
- **Program counter (PC)**
- **Direct page register (DPR)**
- **Program bank register (PCB)**
- **Data bank register (DTB)**
- **User stack bank register (USB)**
- **System stack bank register (SSB)**
- **Additional data bank register (ADB)**

## ■ Configuration of Dedicated Registers

**Figure 2.7-1 Configuration of Dedicated Registers**



: Accumulator (A)
Two 16-bit registers used for the storage of arithmetic operation results.
The two registers can be combined as a contiguous 32-bit register.

: User stack pointer (USP)
16-bit stack pointer that indicates the user stack address

: System stack pointer (SSP)
16-bit stack pointer that indicates the system stack address

: Processor status (PS)
16-bit status register that indicates the system status

: Program counter (PC)
16-bit count register that indicates the current instruction storage location

: Direct page register (DPR)
8-bit page register that specifies bits 8 to 15 of the operand address
when the short direct addressing is executed

: Program bank register (PCB)
8-bit bank register that indicates the program space

: Data bank register (DTB)
8-bit bank register that indicates the data space

: User stack bank register (USB)
8-bit bank register that indicates the user stack space

: System stack bank register (SSB)
8-bit bank register that indicates the system stack space

: Additional data bank register (ADB)
8-bit bank register that indicates the additional space

**Table 2.7-1 Initial Values of the Dedicated Registers**

| Dedicated register | Initial value |
|---|---|
| Accumulator (A) | Undefined |
| User stack pointer (USP) | Undefined |
| System stack pointer (SSP) | Undefined |
| Processor status (PS) | bit15 to bit13 bit12 to bit8 bit7 to bit0<br>PS ILM RP CCR<br>0 0 0 0 0 0 0 0 - 0 1 x x x x x |
| Program counter (PC) | Value in reset vector (contents of FFFFDC$_H$, FFFFDD$_H$) |
| Direct page register (DPR) | 01$_H$ |
| Program bank register (PCB) | Value in reset vector (contents of FFFFDE$_H$) |
| Data bank register (DTB) | 00$_H$ |
| User stack bank register (USB) | 00$_H$ |
| System stack bank register (SSB) | 00$_H$ |
| Additional data bank register (ADB) | 00$_H$ |

**Note:**

Shown above are the initial values for use in a device, not for use in an ICE (such as an emulator).

## 2.7.1 Accumulator (A)

**The accumulator (A) consists of two 16-bit arithmetic operation registers (AH and AL). The accumulator is used to temporarily store the results of an arithmetic operation and data.**

**The accumulator (A) can be used as a 32-bit, 16-bit, or 8-bit register. Arithmetic operations can be performed between memory and other registers or between the higher 16-bit arithmetic operation register (AH) and the lower 6-bit arithmetic operation register (AL). The A register has a data retention function: When data not longer than a word is transferred to the AL register, data stored in the AL register before the transfer is transferred to the AH register. (Data is not retained for some instructions.)**

### ■ Accumulator (A)

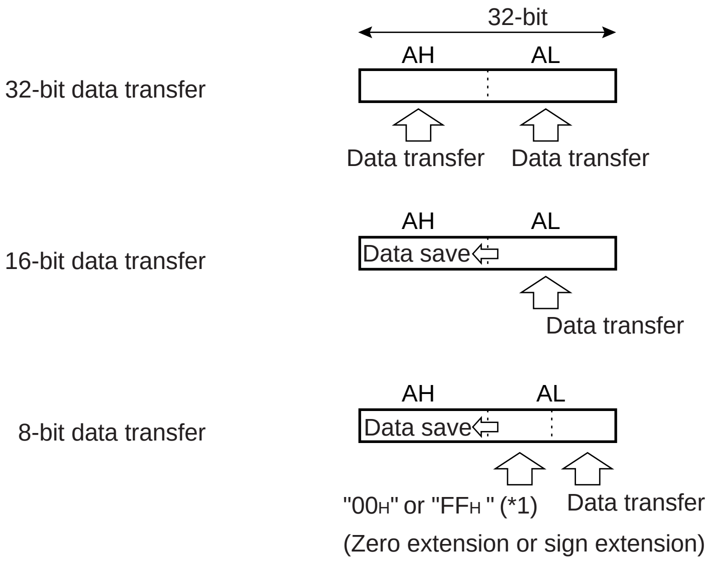
**❍ Data transfer to the accumulator**

The accumulator (A) can handle 32-bit (long word), 16-bit (word), and 8-bit (byte) data. The four-bit data transfer instruction (MOVN) is exceptionally provided but the data is processed in the same way as that for 8-bit data.

- For 32-bit data processing, the AH and AL registers are used in combination.
- For 16-bit and 8-bit data, the AL register is used while the AH register retains data in the AL register.
- Data not longer than a byte, when transferred to the AL register, becomes 16 bits long through sign or zero extension and is stored in the AL register. Data stored in the AL register can be handled as 16-bit or 8-bit data.

Figure 2.7-3 "Example of AL-AH Transfer in the Accumulator (A) (8-bit Immediate Value, Zero Extension)" to Figure 2.7-6 "Example of AL-AH Transfer in the Accumulator (A) (16 bits, Register Indirect)" show specific examples of transfer.

**Figure 2.7-2 Data Transfer to the Accumulator**



*1 Becomes "000H" or "FFFH" for a 4-bit transfer instruction.

### ❍ Accumulator byte-processing arithmetic operation

When a byte-processing arithmetic operation instruction is executed for the AL register, the upper 8 bits of the AL register before the arithmetic operation is executed are ignored. The upper 8 bits of the arithmetic operation results are all zeros.

### ❍ Initial value of the accumulator

The initial value after a reset is undefined.

**Figure 2.7-3 Example of AL-AH Transfer in the Accumulator (A) (8-bit Immediate Value, Zero Extension)**

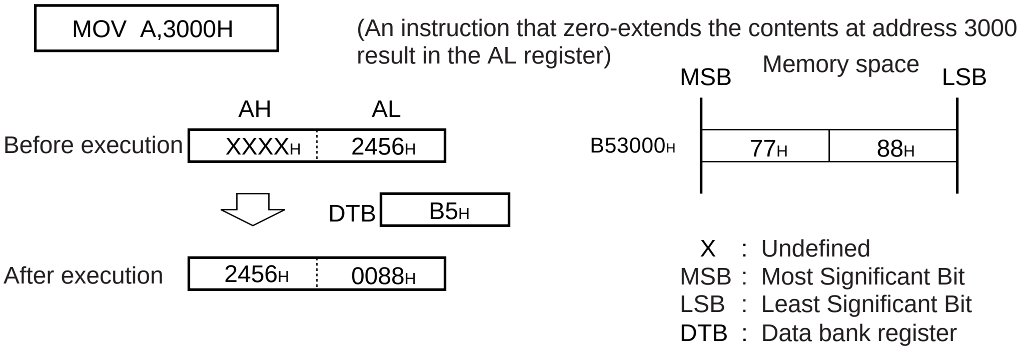

MOV A,3000H
(An instruction that zero-extends the contents at address 3000 result in the AL register)
MSB
Memory space
LSB
AH
AL
Before execution
XXXXH
2456H
B53000H
77H
88H
DTB
B5H
After execution
2456H
0088H
X : Undefined
MSB : Most Significant Bit
LSB : Least Significant Bit
DTB : Data bank register

Figure 2.7-4 Example of AL-AH Transfer in the Accumulator (A) (8-bit Immediate Value, Sign Extension)

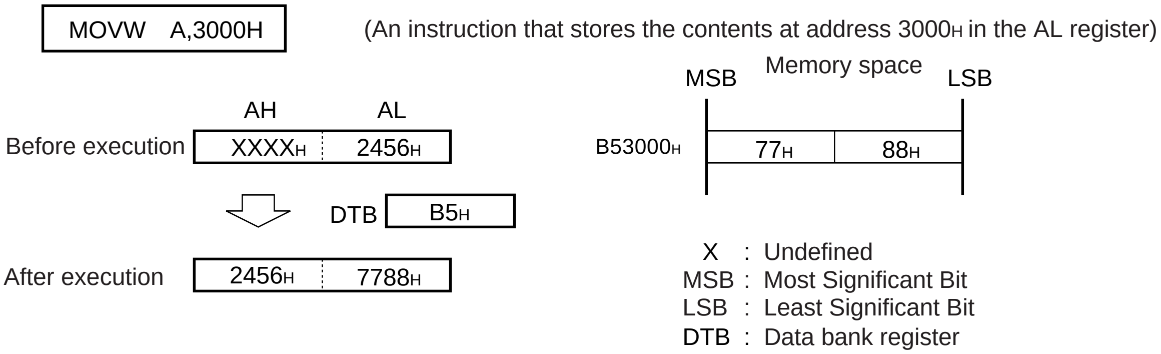


Figure 2.7-5 Example of 32-bit Data Transfer to the Accumulator (A) (Register Indirect)

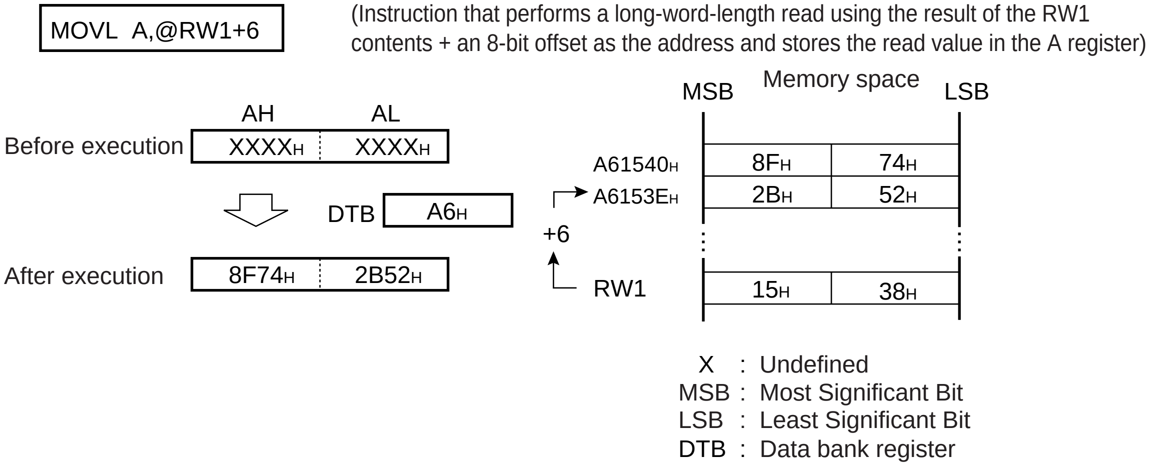
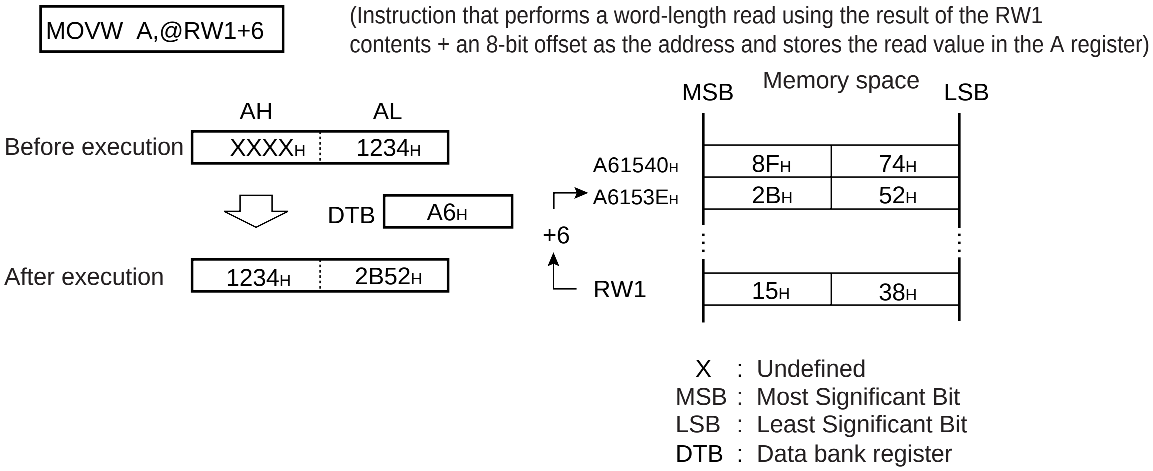

MOVL A,@RW1+6
(Instruction that performs a long-word-length read using the result of the RW1 contents + an 8-bit offset as the address and stores the read value in the A register)
Memory space
MSB
LSB
AH
AL
Before execution
XXXXH
XXXXH
A61540H
8FH
74H
A6153EH
2BH
52H
DTB
A6H
+6
After execution
8F74H
2B52H
RW1
15H
38H
X : Undefined
MSB : Most Significant Bit
LSB : Least Significant Bit
DTB : Data bank register

Figure 2.7-6 Example of AL-AH Transfer in the Accumulator (A) (16 bits, Register Indirect)


MOVW A,@RW1+6
(Instruction that performs a word-length read using the result of the RW1 contents + an 8-bit offset as the address and stores the read value in the A register)
Memory space
MSB
LSB
AH
AL
Before execution
XXXXH
1234H
A61540H
8FH
74H
A6153EH
2BH
52H
DTB
A6H
+6
After execution
1234H
2B52H
RW1
15H
38H
X : Undefined
MSB : Most Significant Bit
LSB : Least Significant Bit
DTB : Data bank register

## 2.7.2 Stack Pointers (USP, SSP)

**There are two types of stack pointers: a user stack pointer (USP) and a system stack pointer (SSP). Each stack pointer is a register that indicates the memory address of the location of the destination for saved data or a return address when PUSH instructions, POP instructions, and subroutines are executed. The upper 8 bits of the stack address are specified by the user stack bank register (USB) or system stack bank register (SSB).**

**When the S flag of the condition code register (CCR) is 0, the USP and USB registers are valid. When the S flag is 1, the SSP and SSB registers are valid.**

### ■ Stack Selection

The F$^2$MC-16LX uses two types of stack: a system stack and a user stack.

The stack address is determined, as shown in Table 2.7-2 "Stack Address Specification", by the S flag in the processor status (PS:CCR).

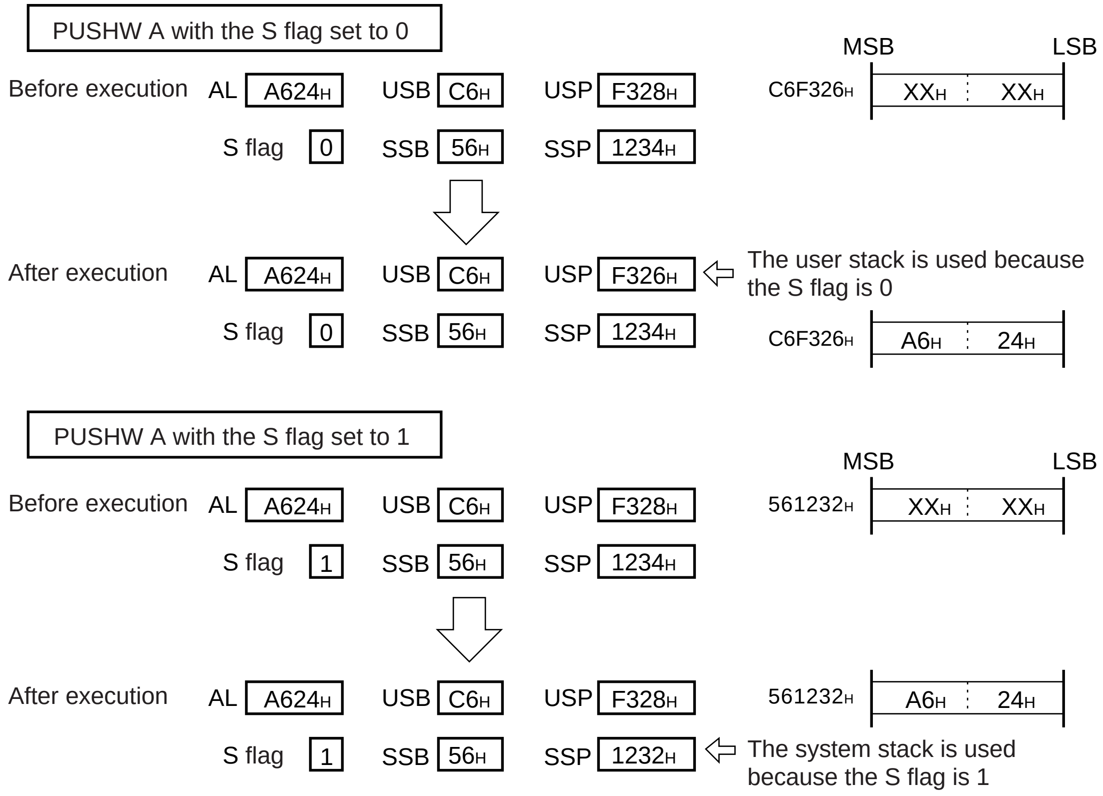
**Table 2.7-2 Stack Address Specification**

| S flag | Stack address | |
|---|---|---|
| | Upper 8 bits | Lower 16 bits |
| 0 | User stack bank register (USB) | User stack pointer (USP) |
| 1 (*1) | System stack bank register (SSB) | System stack pointer (SSP) |

*1: Initial value

Since a reset initializes the S flag to "1", the system stack is used by default. When an interrupt is received, the stack flag (CCR:S) is set to "1" and the system stack pointer will be used. The user stack is used for all types of stack operations except those for interrupt routines. Unless the stack space is divided , the system stack should be used.

Figure 2.7-7 Stack Operation Instruction and Stack Pointer



X : Undefined
MSB : Most Significant Bit
LSB : Least Significant Bit

**Note:**

- To set a stack address in the stack pointer, use an even-numbered address. If an odd-numbered address is used, a word is accessed in two separate operations, reducing access efficiency.
- The initial values for the USP and SSP registers are undefined.
- Allocate the system stack area, user stack area, and data area so that they do not overlap.

## ■ System Stack Pointer (SSP)

To use the system stack pointer (SSP), set the S flag in the condition code register (CCR) to "1". If the S flag is set to "1", the upper 8 bits of the address to be used for the stack operation are indicated by the system stack bank register (SSB).

For more information on the condition code register (CCR), see Section 2.7.4 "Condition Code Register (PS: CCR)". For more information on the system stack bank register (SSB), see Section 2.7.9 "Bank Registers (PCB, DTB, USB, SSB, ADB)".

## ■ User Stack Pointer (USP)

To use the user stack pointer (USP), set the S flag in the condition code register (CCR) to "0". If the S flag is set to "0", the upper 8 bits of the address to be used for the stack operation are indicated by the user stack bank register (USB).

For more information on the condition code register (CCR), see Section 2.7.4 "Condition Code Register (PS: CCR)". For more information on the system stack bank register (SSB), see Section 2.7.9 "Bank Registers (PCB, DTB, USB, SSB, ADB)".

## 2.7.3 Processor Status (PS)

**The processor status register (PS) contains CPU control bits and bits that indicate the CPU status. The PS register consists of the following three registers:**

- **Condition code register (CCR)**
- **Register bank pointer (RP)**
- **Interrupt level mask register (ILM)**

### ■ Processor Status (PS) Configuration

The processor status register (PS) contains CPU control bits and bits that indicate the CPU status.

**Figure 2.7-8 Processor Status (PS) Configuration**

| | ILM | | | RP | | | | | CCR | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PS | ILM2 | ILM1 | ILM0 | B4 | B3 | B2 | B1 | B0 | – | I | S | T | N | Z | V | C |

#### ❍ Condition Code Register (CCR)

This register consists of flags that are set to "1" or reset to "0" in accordance with instruction execution results or interrupts.

For more information on the flags, see Section 2.7.4 "Condition Code Register (PS: CCR)".

#### ❍ Register Bank Pointer (RP)

This pointer points to the first address of the memory block (register bank) used as the general-purpose register in the RAM area.

There are 32 banks for general-purpose registers. Set values "$00_H$ to $1F_H$" in the RP to specify a bank.

For the setting method and more information on this pointer, see Section 2.7.5 "Register Bank Pointer (PS: RP)".

#### ❍ Interrupt Level Mask Register (ILM)

This register indicates the level of an interrupt currently accepted by the CPU. The value is compared with that of the interrupt level setting bits (ICR: IL0 to IL2) in the interrupt control register (ICR00 to ICR15) set so that an interrupt request corresponds to each peripheral function (resource).

For the setting method and more information on this register, see Section 2.7.6 "Interrupt Level Mask Register (PS: ILM)".

## 2.7.4 Condition Code Register (PS: CCR)

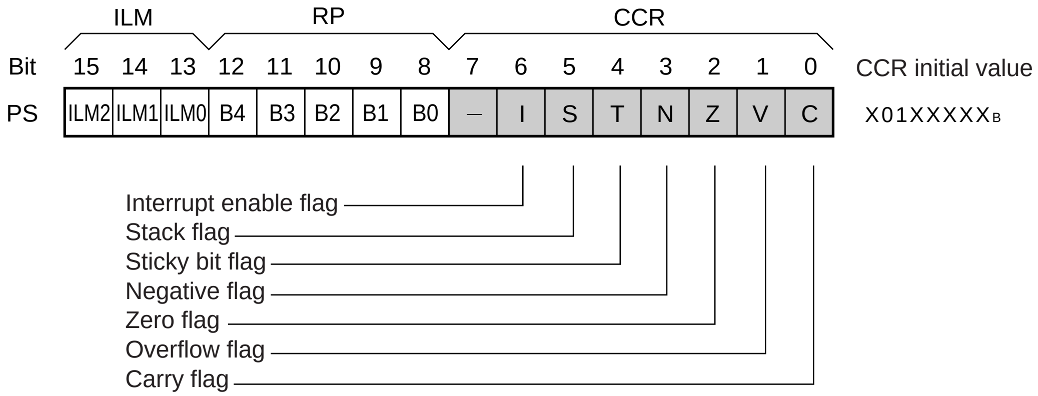
**The condition code register (CCR) is an 8-bit register that consists of the following bits:**

- **Bits that indicate the result of an arithmetic operation and the contents of transfer data**
- **Bits that control the acceptance of an interrupt request**

### ■ Condition Code Register (CCR) Configuration

Refer to the programming manual for details about the status of the condition code register (CCR) during instruction execution.

**Figure 2.7-9 Condition Code Register (CCR) Configuration**



X : Not used
- : Undefined

❍ **Interrupt enable flag (I)**

Interrupts are enabled when the I flag is set to "1", or disabled when the I flag is reset to "0", in response to any interrupt request other than software interrupts. The I flag is reset to "0" by an external or software reset.

❍ **Stack flag (S)**

This flag indicates the pointer used for a stack operation. The user stack pointer (USP) is valid if the S flag is reset to "0". The system stack pointer (SSP) is valid if the S flag is set to "1". The S flag is set to "1" when an interrupt is accepted or when an external or software reset is asserted.

For more information on the stack pointers, see Section 2.7.2 "Stack Pointers (USP, SSP)"

❍ **Sticky bit flag (T)**

The T flag is set to "1" if the data shifted out of by the carry contains "1" during execution of a logical or arithmetic right shift instruction. Otherwise, the T flag is reset to "0". The T flag is also reset to "0" if the shift amount is zero.

- **Negative flag (N)**

  The N flag is set to "1" if the most significant bit (MSB) of the general-purpose registers (RL0 to RL3) that store the operation result is "1".  Otherwise, the N flag is reset to "0".

  For more information on general-purpose registers, see Section 2.8 "General-Purpose Registers".

- **Zero flag (Z)**

  The Z flag is set to "1" if the general-purpose registers (RL0 to RL3) that store the operation result are "$0000_H$".  Otherwise, the Z flag is reset to "0".

- **Overflow flag (V)**

  The V flag is set to "1" if an overflow occurs in a signed numeric value.  Otherwise, the V flag is reset to "0".

- **Carry flag (C)**

  The C flag is set to "1" if a carry from the most significant bit or a borrow to the least significant bit occurs during an arithmetic operation.  Otherwise, the C flag is reset to "0".

## 2.7.5 Register Bank Pointer (PS: RP)

**The register bank pointer (RP) is a five-bit register that points to the first address of the general-purpose register bank currently used.**

### ■ Register Bank Pointer (RP)

**Figure 2.7-10 Configuration of the Register Bank Pointer (RP)**

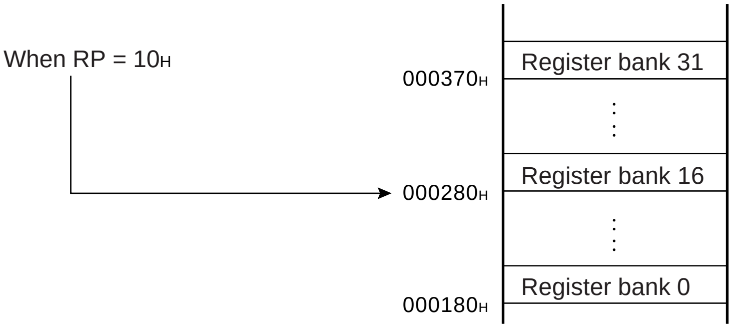
| | ILM | | | RP | | | | | CCR | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | RP initial value |
| PS | ILM2 | ILM1 | ILM0 | B4 | B3 | B2 | B1 | B0 | — | I | S | T | N | Z | V | C | $00000_B$ |

### ■ General-Purpose Register Area and Register Bank Pointer

The register bank pointer (RP) points to the relationship between the general-purpose register of the $F^2$MC-16LX and the address in internal RAM. The relationship between the contents of the RP register and the address follows the conversion rules shown in Figure 2.7-11 "Conversion Rules for Physical Address of General-Purpose Register Area".

**Figure 2.7-11 Conversion Rules for Physical Address of General-Purpose Register Area**

Conversion formula [$000180_H$ + (RP) × $10_H$]



- The register bank pointer (RP) can assume values from $00_H$ to $1F_H$. The first address of a register bank can be set to a value from $000180_H$ to $00037F_H$.
- Although an assembler instruction can use an 8-bit immediate value transfer instruction for transfer to the register bank pointer (RP). However, only the lower 5 bits of the data are valid.
- A reset initializes the register bank pointer (RP) to $00000_B$.

## 2.7.6 Interrupt Level Mask Register (PS: ILM)

**The interrupt level mask register (ILM) is a 3-bit register that indicates the level of the interrupt currently accepted by the CPU.**

### ■ Interrupt Level Mask Register (ILM)

See CHAPTER 6 "INTERRUPTS" for details about interrupts.

**Figure 2.7-12 Configuration of the Interrupt Level Mask Register (ILM)**

| | ILM | | | RP | | | | | CCR | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ILM initial value |
| PS | ILM2 | ILM1 | ILM0 | B4 | B3 | B2 | B1 | B0 | — | I | S | T | N | Z | V | C | $000_B$ |

The interrupt level mask register (ILM) indicates the level of the interrupt currently accepted by the CPU.

A level of the interrupt currently accepted by the CPU can be set in the ILM register. The CPU does not accept any interrupt with an interrupt level lower than that set in the ILM register.

- A reset sets the highest interrupt level in the ILM register, causing no interrupt to be accepted.
- Although an assembler instruction can use an 8-bit immediate value transfer instruction for transfer to the interrupt level mask register (ILM). However, only the lower 3 bits of the data are valid.

**Table 2.7-3 Interrupt Level Mask Register (ILM) and Interrupt Level Priority**

| ILM2 | ILM1 | ILM0 | Interrupt level | Interrupt level priority |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Highest (interrupts disabled)<br>↕<br>Lowest |
| 0 | 0 | 1 | 1 | |
| 0 | 1 | 0 | 2 | |
| 0 | 1 | 1 | 3 | |
| 1 | 0 | 0 | 4 | |
| 1 | 0 | 1 | 5 | |
| 1 | 1 | 0 | 6 | |
| 1 | 1 | 1 | 7 | |

## 2.7.7 Program Counter (PC)

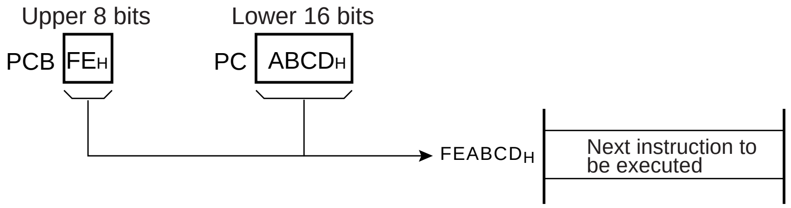
**The program counter (PC) is a 16-bit counter that indicates the lower 16 bits of the address of the next instruction to be executed by the CPU.**

### ■ Program Counter (PC)

The program bank register (PCB) specifies the upper 8 bits, and the program counter (PC) specifies the lower 16 bits, of the address of the next instruction to be executed by the CPU. The address of the next instruction to be executed is as shown in Figure 2.7-13 "Program Counter (PC)". The contents of the PC are updated by conditional branch instructions, subroutine call instructions, interrupts, and resets. The PC can also be used as a base pointer for reading operands.

For more information on the PCB, see Section 2.7.9 "Bank Registers (PCB, DTB, USB, SSB, ADB)".

**Figure 2.7-13 Program Counter (PC)**



**Note:**

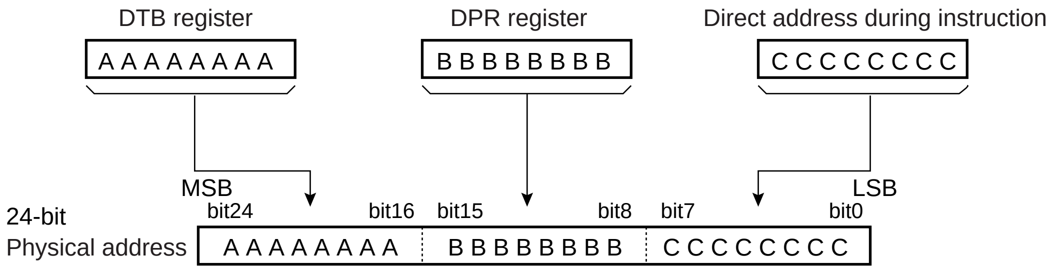
The PC and PCB cannot be rewritten directly by a program (instructions such as MOVPC and #0FF$_H$).

## 2.7.8 Direct Page Register (DPR)

**The direct page register (DPR) is an 8-bit register that specifies bits 8 to 15 (addr8 to addr15) of the operand address when a short direct addressing instruction is executed. A reset initializes the DPR to "01$_H$".**

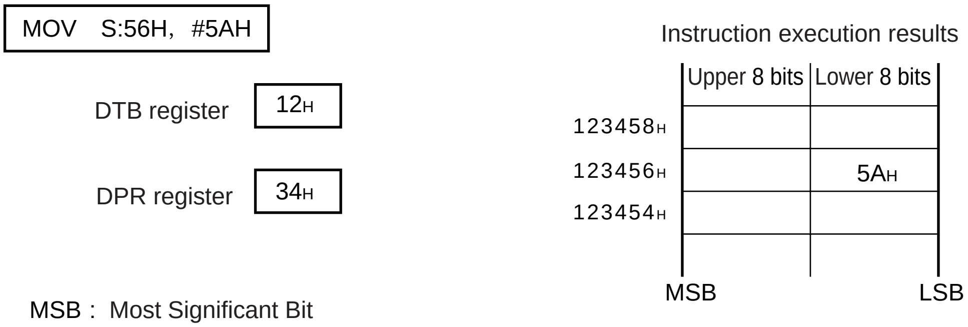
### ■ Direct Page Register (DPR)

**Figure 2.7-14 Physical Address Generation by the Direct Page Register (DPR)**



MSB : Most Significant Bit
LSB : Least Significant Bit

**Figure 2.7-15 Example of Direct Page Register (DPR) Setting and Data Access**



MSB : Most Significant Bit
LSB : Least Significant Bit

## 2.7.9 Bank Registers (PCB, DTB, USB, SSB, ADB)

**Bank registers specify the highest 8-bit address by bank addressing. The five bank registers are as follows:**

- **Program bank register (PCB)**
- **Data bank register (DTB)**
- **User stack bank register (USB)**
- **System stack bank register (SSB)**
- **Additional bank register (ADB)**

**The PCB, DTB, USB, SSB, and ADB registers indicate the individual memory banks where the program space, data space, user stack space, system stack space, and additional space are located.**

### ■ Bank Registers (PCB, DTB, USB, SSB, ADB)

#### ❍ Program bank register (PCB)

The PCB is a bank register that specifies the program (PC) space. The PCB is rewritten whenever a JMPP, CALLP, RETP, or RETI instruction causes a branch anywhere within the 16M-byte space, when an software interrupt instruction is executed, when a hardware interrupt occurs, or when an exception occurs.

#### ❍ Data bank register (DTB)

The DTB is a bank register that specifies the data (DT) space.

#### ❍ User stack bank register (USB), system stack bank register (SSB)

The USB and SSB are bank registers that specify the stack (SP) space. Either the USB or SSB is used depending on the value of the S flag in the processor status register (PS:CCR). For more information, see Section 2.7.2 "Stack Pointer (USP, SSP)".

#### ❍ Additional bank register (ADB)

The ADB is a bank register that specifies the additional (AD) space.

#### ❍ Bank settings and data access

All bank registers are 8 bits in length. A reset initializes the PCB to "$FF_H$" and the DTB, USB, SSB, and ADB to "$00_H$". The PCB can be read but cannot be written to. Bank registers other than the PCB can be read and written to.

**Note:**

The MB90M405 series supports up to the memory space contained in the device.

See Section 2.4.2 "Address Specification by Bank Addressing" for the operation of each register.

## 2.8 General-Purpose Registers

**The general-purpose registers are a memory block allocated in RAM at 000180$_H$ to 00037F$_H$ as register banks, each of which consists of eight 16-bit segments.**

**The general-purpose registers can be used as general-purpose 8-bit registers (byte registers R0 to R7), 16-bit registers (word registers RW0 to RW7), or 32-bit registers (long-word registers RL0 to RL7).**

**General-purpose registers can access RAM with a short instruction at high speed. Since general-purpose registers are blocked into register banks, protection of register contents and division into function units can readily be performed. When a general-purpose register is used as a long-word register, it can be used as a linear pointer that directly accesses the entire space.**

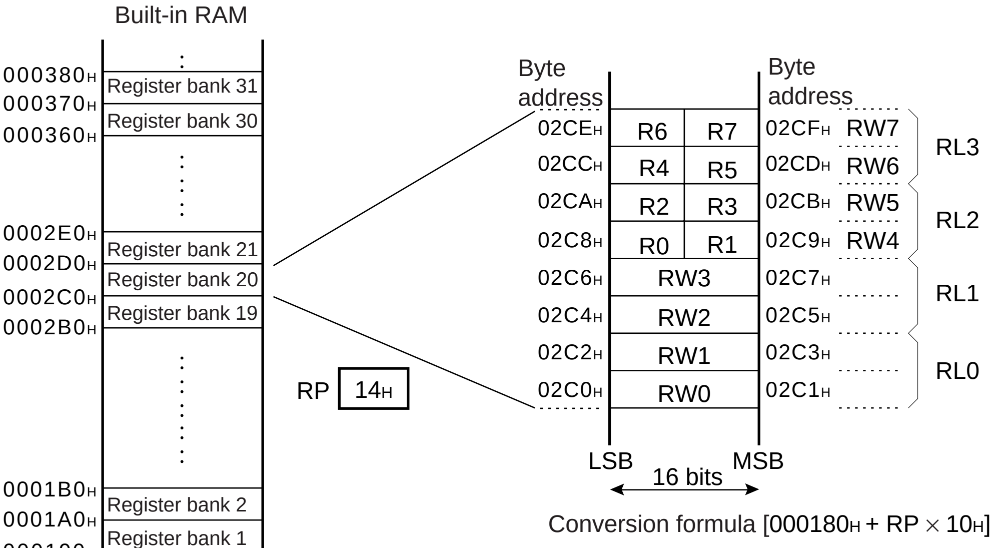
### ■ Configuration of a General-Purpose Register

General-purpose registers exist in RAM at "000180$_H$" to "00037F$_H$" and are configured as 32 banks. The register bank pointer (RP) specifies the bank. The RP determines the first address of each bank as shown in the following equation. 16 bits multiplied by 8 are defined as one register bank.

First address of general-purpose register = 000180$_H$ + RP x 10$_H$

For more information on the PR, see Section 2.7.5 "Register Bank Pointer (PS: RP)".

**Figure 2.8-1 Location and Configuration of the General-Purpose Register Banks in the Memory Space**



| | | |
|---|---|---|
| R0 to R7 | : | Byte registers |
| RW0 to RW7 | : | Word registers |
| RL0 to L3 | : | Long-word registers |
| MSB | : | Most Significant Bit |
| LSB | : | Least Significant Bit |

**Note:**

The register bank pointer (RP) is initialized to $00_H$ after a reset.

## ■ Register Bank

Register banks can be used as general-purpose registers (byte registers R0 to R7, word registers RW0 to RW7, and long word registers RL0 to RL3) for various arithmetic operations and pointers. Long word registers can be used also as linear pointers that directly access all the memory space.

A reset does not initialize the contents of the register bank as with RAM but the status before reset is retained. A power-on reset, however, makes the contents undefined.

**Table 2.8-1 Typical Functions of General-Purpose Registers**

| Register name | Function |
|---|---|
| R0 to R7 | Used as an operand in various instructions<br>**Note:**<br>R0 is also used as a barrel shift counter and an instruction normalization counter |
| RW0 to RW7 | Used as a pointer<br>Used as an operand in various instructions<br>**Note:**<br>RW0 is used also as a string instruction counter |
| RL0 to RL3 | Used as a long pointer<br>Used as an operand in various instructions |

# 2.9 Prefix Codes

**Prefix codes are placed before an instruction to partially change the operation of the instruction. The three types of prefix codes are as follows:**

- **Bank select prefix (PCB, DTB, ADB, SPB)**
- **Common register bank prefix (CMR)**
- **Flag change suppression prefix (NCC)**

## ■ Prefix Codes

### ❍ Bank select prefix (PCB, DTB, ADB, SPB)

A bank select prefix is placed before an instruction to select the memory space to be accessed by the instruction regardless of the addressing method.

For more information, see Section 2.9.1 "Bank Select Prefix (PCB, DTB, ADB, SPB)".

### ❍ Common register bank prefix (CMR)

The common register bank prefix is placed before an instruction that accesses a register bank to change the register accessed by the instruction to the common bank (register bank selected when RP = 0) at $000180_H$ to $00018F_H$ regardless of the current register bank pointer (RP) value.

For more information, see Section 2.9.2 "Common Register Bank Prefix (CMR)".

### ❍ Flag change suppression prefix (NCC)

The flag change suppression prefix code is placed before an instruction to suppress a flag change accompanying the execution of the instruction.

For more information, see Section 2.9.3 "Flag Change Suppression Prefix (NCC)".

## 2.9.1 Bank Select Prefix (PCB, DTB, ADB, SPB)

**Memory space used for data access is determined for each addressing method. However, placing a bank select prefix before an instruction selects the memory space to be accessed by the instruction regardless of the addressing method.**

### ■ Bank Select Prefixes (PCB, DTB, ADB, SPB)

**Table 2.9-1 Bank Select Prefix Codes**

| Bank select prefix | Selected space |
|---|---|
| PCB | Program space |
| DTB | Data space |
| ADB | Additional space |
| SPB | When the value of the S flag in the condition code register (CCR) is 0 and the user stack space is 1, the system stack space is used. |

If a bank select prefix is used, some instructions perform an unexpected operation.

**Table 2.9-2 Instructions Not Affected by Bank Select Prefix Codes**

| Instruction type | Instruction | | | | Effect of bank select prefix |
|---|---|---|---|---|---|
| String instruction | MOVS<br>SCEQ<br>FILS | | MOVSW<br>SCWEQ<br>FILSW | | The bank register specified by the operand is used irrespective of whether a prefix is used. |
| Stack operation instruction | PUSHW | | POPW | | When the S flag is 0, the user stack bank (USB) is used whether or not there is a prefix. When the S flag is 1, the system stack bank (SSB) is used regardless of whether a prefix is used. |
| I/O access instruction | MOV<br>MOVW<br>MOV<br>MOV<br>MOVB<br>SETB<br>BBC<br>WBTC | A,io<br>A,io<br>io, A<br>io,#imm8<br>A,io:bp<br>io:bp<br>io:bp, rel<br>io,bp | MOVX<br><br>MOVW<br>MOVW<br>MOVB<br>CLRB<br>BBS<br>WBTS | A,io<br><br>io, A<br>io,#imm16<br>io:bp,A<br>io:bp<br>io:bp, rel<br>io:bp | The I/O space ($000000_H$ to $0000FF_H$) is accessed whether or not there is a prefix. |
| Interrupt return instruction | RETI | | | | The system stack bank (SSB) is used whether or not a prefix is used. |

**Table 2.9-3 Instructions Whose Use Requires Caution When Bank Select Prefix**

| Instruction type | Instruction | Explanation |
|---|---|---|
| Flag change instruction | ANDCCR, #imm8<br>ORCCR, #imm8 | The effect of the prefix extends to the next instruction. |
| ILM setting instruction | MOVILM, #imm8 | The effect of the prefix extends to the next instruction. |
| PS return instruction | POPWPS | Do not place a bank select prefix before the PS return instruction. |

## 2.9.2 Common Register Bank Prefix (CMR)

**Placing the common register bank prefix (CMR) before an instruction that accesses a register bank changes the register accessed by it to the common bank at "$000180_H$" to "$00018F_H$" (register bank selected when RP = "$00_H$") regardless of the current register bank pointer (RP) value.**

### ■ Common Register Bank Prefix (CMR)

To facilitate data exchange between multiple tasks, the $F^2$MC-16LX provides a common bank that can be commonly used by these tasks. The common bank is located at addresses "$000180_H$" to "$00018F_H$".

However, be careful when you use this prefix with the instructions listed in Table 2.9-4 "Instructions Whose Use Requires Caution When the Common Register Bank Prefix (CMR) Is Used ".

**Table 2.9-4 Instructions Whose Use Requires Caution When the Common Register Bank Prefix (CMR) Is Used**

| Instruction type | Instruction | Explanation |
|---|---|---|
| String instruction | MOVSMOVSW<br>SCEQ SCWEQ<br>FILS FILSW | Do not place the CMR prefix before the string instruction. |
| Flag change instruction | AND CCR,#imm8 OR CCR,#imm8 | The effect of the prefix extends to the next instruction. |
| PS return instruction | POPW PS | The effect of the prefix extends to the next instruction. |
| ILM setting instruction | MOV ILM,#imm8 | The effect of the prefix extends to the next instruction. |

## 2.9.3 Flag Change Suppression Prefix (NCC)

**The flag change suppression prefix (NCC) code is set before an instruction to suppress a flag change accompanying the execution of the instruction.**

### ■ Flag Change Suppression Prefix (NCC)

Use the flag change suppression prefix (NCC) to suppress unnecessary flag changes. Changes in the T, N, Z, V, and C flags can be suppressed.

Be careful when you use this prefix with the instructions listed in Table 2.9-5 "Instructions Whose Use Requires Caution When the Flag Change Suppression Prefix (NCC) Is Used ".

For more information on the T, N, Z, V, and C flags, see Section 2.7.4 "Condition Code Register (PS: CCR)".

**Table 2.9-5 Instructions Whose Use Requires Caution When the Flag Change Suppression Prefix (NCC) Is Used**

| Instruction type | Instruction | | Explanation |
|---|---|---|---|
| String instruction | MOVS<br>SCEQ<br>FILS | MOVSW<br>SCWEQ<br>FILSW | Do not place the NCC prefix before the string instruction. |
| Flag change instruction | AND CCR, #imm8<br>OR CCR, #imm8 | | The condition code register (CCR) changes as defined in the instruction specification whether or not a prefix is used.<br>The effect of prefix extends to the next instruction. |
| PS return instruction | POPW PS | | The condition code register (CCR) changes as defined in the instruction specification whether or not a prefix is used.<br>The effect of prefix extends to the next instruction. |
| ILM setting instruction | MOV ILM, #imm8 | | The effect of prefix extends to the next instruction. |
| Interrupt instruction<br>Interrupt return instruction | INT #vct8<br>INT adder16<br>RETI | INT9<br>INTP addr24 | The condition code register (CCR) changes as defined in the instruction specification whether or not a prefix is used. |
| Context switch instruction | JCTX @A | | The condition code register (CCR) changes as defined in the instruction specification whether or not a prefix is used. |

## 2.9.4 Restrictions on Prefix Codes

**The following restrictions are imposed on the use of prefix codes:**

- **Interrupt requests are not accepted during the execution of prefix codes and interrupt suppression instructions.**
- **If a prefix code is placed before an interrupt instruction, the effect of the prefix code is delayed.**
- **If consecutively placed prefix codes conflict, the last prefix code is valid.**

### ■ Prefix Codes and Interrupt Suppression Instructions

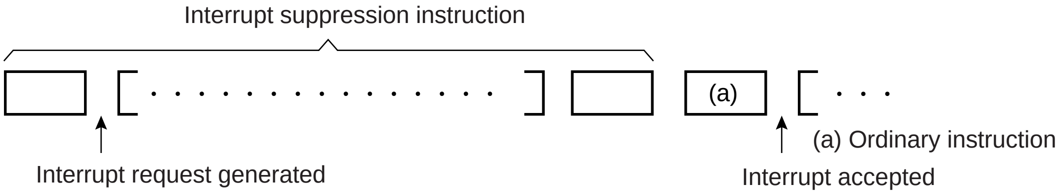
**Table 2.9-6 Prefix Codes and Interrupt Suppression Instructions**

| | Prefix codes | Interrupt suppression instructions (instructions that delay the effect of prefix codes) |
|---|---|---|
| Instructions that do not accept interrupt requests | PCB<br>DTB<br>ADB<br>SPB<br>CMR<br>NCC | MOV ILM, #imm8<br>OR CCR, #imm8<br>AND CCR, #imm8<br>POPW PS |

#### ❍ Interrupt Suppression

As shown in Figure 2.9-1 "Interrupt Suppression", an interrupt request generated during the execution of prefix codes and interrupt instructions is not accepted. The interrupt is not processed until the first instruction that is not governed by a prefix code or that is not an interrupt suppression instruction is executed.

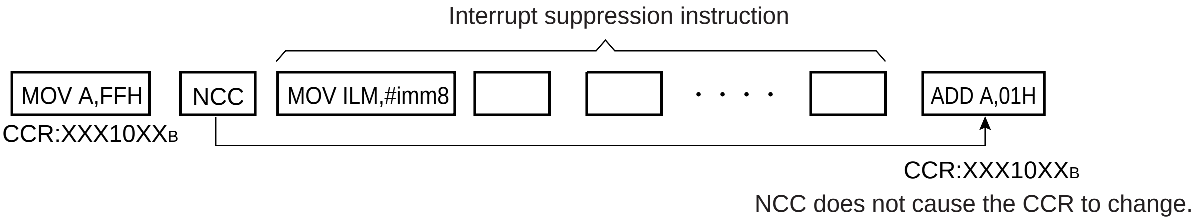
**Figure 2.9-1 Interrupt Suppression**


Interrupt suppression instruction
(a)
(a) Ordinary instruction
Interrupt request generated
Interrupt accepted

### ❍ Delay of the effect of prefix codes

If a prefix code is placed before an interrupt/hold suppression instruction as shown in Figure 2.9-2 "Interrupt Suppression Instructions and Prefix Codes", the prefix code takes effect on the first instruction executed after the interrupt/hold suppression instruction.

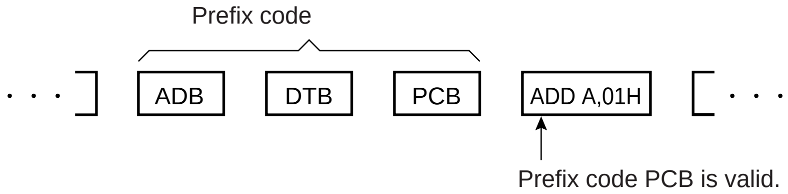
**Figure 2.9-2 Interrupt Suppression Instructions and Prefix Codes**


Interrupt suppression instruction
MOV A,FFH
NCC
MOV ILM,#imm8
· · · ·
ADD A,01H
CCR:XXX10XXB
CCR:XXX10XXB
NCC does not cause the CCR to change.

## ■ Consecutive Prefix Codes

When consecutive conflicting prefix codes (PCB, ADB, DTB, and SPB) are specified, the last prefix code is valid.

**Figure 2.9-3 Consecutive Prefix Codes**



# CHAPTER 3 RESETS

**This chapter describes resets for the MB90M405 series.**

# 3.1 Resets

**If a reset cause is generated, the CPU stops the current execution process and waits for the reset to be cleared. When the reset is cleared, the CPU begins processing at the address indicated by the reset vector.**
**There are four causes of a reset:**

- **Power-on reset (at power-on)**
- **Watchdog timer overflow (during the use of a watchdog timer)**
- **External reset input via the $\overline{\text{RST}}$ pin**
- **Setting "0" in the internal reset signal generation bit (RST) of the low power consumption mode control register (software reset)**

## ■ Reset Causes

**Table 3.1-1 Reset Causes**

| Type of reset | Cause | Machine clock | Watchdog timer | Oscillation stabilization wait |
|---|---|---|---|---|
| External pin | "L" level input to $\overline{\text{RST}}$ pin | Main clock frequency (MCLK) | Stop | No |
| Software | "0" written to the internal reset signal generation bit (RST) of the low power consumption mode control register (LPMCR) | Main clock frequency (MCLK) | Stop | No |
| Watchdog timer | Watchdog timer overflow | Main clock frequency (MCLK) | Stop | No |
| Power-on | Power-on | Main clock frequency (MCLK) | Stop | Yes |

MCLK: Main clock frequency (oscillation clock frequency divided by 2: 2/HCLK)

### ❍ External reset

An external reset is generated if the external reset terminal ($\overline{\text{RST}}$ terminal) is set to the "L" level. The "L" level must be input at least for 16 machine cycles (16/ϕ). While the machine clock is used, no oscillation stabilization wait time is placed even if a reset occurs due to the "L" level input to the external reset pin.

**Reference:**

If the external reset pin is set to the "L" level while an instruction is being executed (while a transfer instruction such as MOV is being executed), the external reset input becomes valid after the completion of processing by the instruction being executed.

For a string-processing instruction (such as MOVS), however, the reset input may become valid before the transfer resulting from the specified counter value is completed.

If the external reset pin is set to the "L" level, the port pin enters the reset status regardless of the instruction execution cycle (if set to the "L" level, the operation is asynchronous).

### ❍ Software reset

A software reset is a reset for three machine cycles (3/$\phi$) generated by writing "0" to the internal reset signal generation bit (RST) of the low power consumption mode control register (LPMCR). The oscillation stabilization wait time is not required for software resets.

### ❍ Watchdog timer reset

A watchdog timer reset is generated unless "0" is written to the watchdog timer control bit (WTE) of the watchdog timer control register (WDTC) within the time specified in the interval time setting bits (WT1, WT0) of the WDTC after the watchdog timer is activated.

### ❍ Power-on reset

A power-on reset is generated when the power is turned on.

The oscillation stabilization wait time is fixed at $2^{17}$/HCLK (about 31.25 ms if the source oscillation is 4.194 MHz). A reset occurs after the oscillation stabilization wait time has elapsed.

**Information: Definition of clocks**

HCLK: Oscillation clock frequency (Clock supplied from the oscillation pin)

MCLK: Main clock frequency (Clock obtained by dividing the source oscillation by two)

$\phi$ : Machine clock (CPU operating clock)

1/$\phi$: Machine cycle (CPU operating clock cycle)

See Section 4.1 "Clocks" for details about clocks.

## 3.2 Reset Causes and Oscillation Stabilization Wait Time

**The F$^2$MC-16LX has four reset causes. The oscillation stabilization wait time for a reset depends on the reset cause.**

### ■ Reset Causes and Oscillation Stabilization Wait Time

**Table 3.2-1 Reset Causes and Oscillation Stabilization Wait Time**

| Reset cause | Oscillation stabilization wait time<br>The corresponding time interval for an oscillation clock frequency of 4 MHz is given in parentheses. |
|---|---|
| Power-on reset | $2^{17}$/HCLK (about 31.25 ms) |
| Watchdog timer | None. (The WS1 and WS0 bits are initialized to "$11_B$".) |
| External reset from $\overline{\text{RST}}$ pin | None. (The WS1 and WS0 bits are initialized to "$11_B$".) |
| Software reset | None. (The WS1 and WS0 bits are initialized to "$11_B$".) |

HCLK: Oscillation clock frequency (MHz)

**Table 3.2-2 Oscillation Stabilization Wait Time Depending on Settings of Clock Selection Register (CKSCR)**

| WS1 | WS0 | Oscillation stabilization wait time<br>The corresponding time interval for an oscillation clock frequency of 4.194 MHz is given in parentheses. |
|---|---|---|
| 0 | 0 | $2^{10}$/HCLK (about 244 μs) |
| 0 | 1 | $2^{13}$/HCLK (about 1.95 ms) |
| 1 | 0 | $2^{15}$/HCLK (about 7.81 ms) |
| 1 | 1 | $2^{17}$/HCLK (about 31.25 ms) |

HCLK: Oscillation clock frequency (MHz)

**Note:**

Oscillation clock oscillators generally require an oscillation stabilization wait time from the start of oscillation until they stabilize at their natural frequency. Be sure to set a proper oscillation stabilization wait time for the oscillator to be used.

### ■ Oscillation Stabilization Wait Reset Status

For a power-on reset or an external reset in stop mode, the reset operation occurs after the oscillation stabilization wait time generated by the timebase timer elapses. Unless the external reset input is released, the reset operation occurs after the external reset is released.

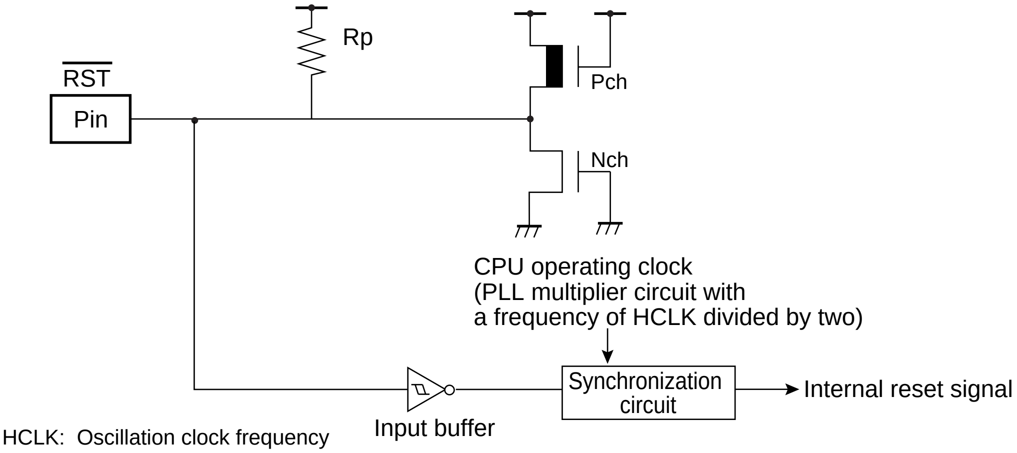
# 3.3 External Reset Pin

**An internal reset occurs if an "L" level signal is input to the external reset pin ($\overline{\text{RST}}$ pin). In the MB90M405 series, a reset occurs in synchronization with the CPU operating clock. However, resets of external pins (I/O ports) occur asynchronously.**

## ■ Block Diagrams of the External Reset Pin

### ❍ Block diagram of components related to internal resets

**Figure 3.3-1 Block Diagram of Components Related to Internal Resets**

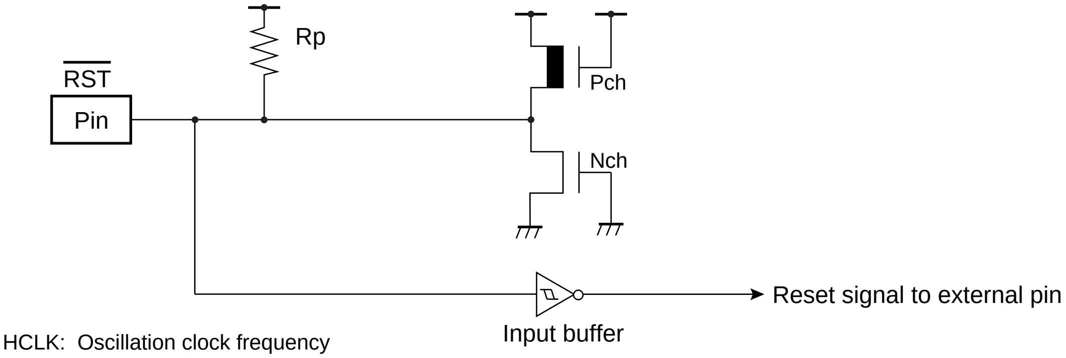


**Note:**

The machine clock is required to initialize the internal circuit. When a reset signal is input, the clock must be supplied from the oscillation pin.

### ❍ Block diagram of components related to internal resets for external pins (I/O ports)

**Figure 3.3-2 Block Diagram of Components Related to Internal Resets for External Pins**

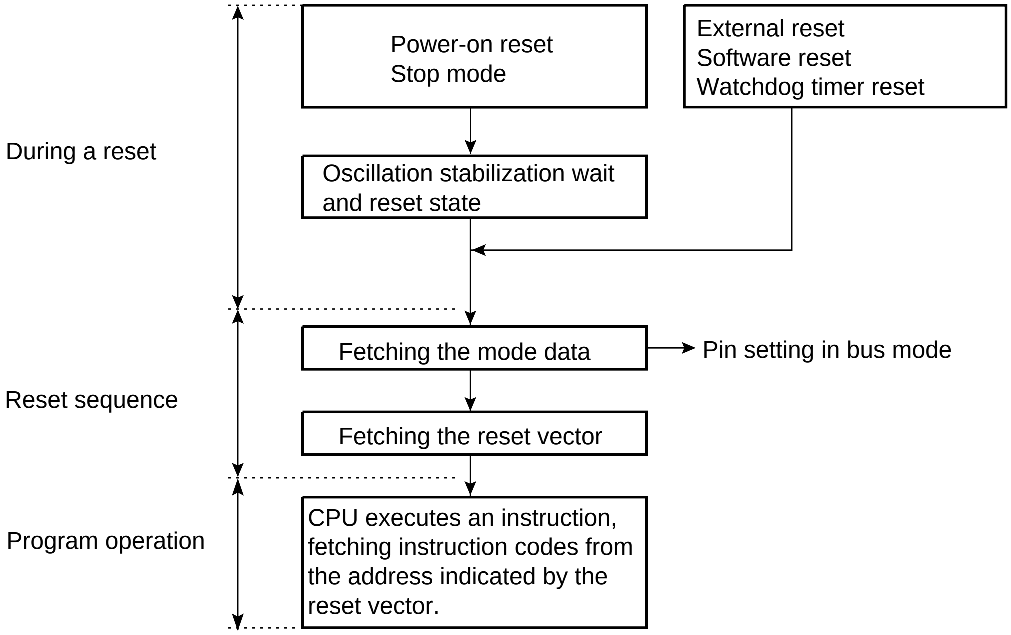

Rp
$\overline{\text{RST}}$
Pin
Pch
Nch
Reset signal to external pin
Input buffer
HCLK: Oscillation clock frequency

## 3.4 Reset Operation

**When a reset is cleared, the mode data and the reset vector stored in the internal or external memory are fetched. The mode data register determines the CPU operating mode. The reset vector determines the execution start address used after a reset sequence ends.**

### ■ Overview of Reset Operation

**Figure 3.4-1 Reset Operation Flow**


Power-on reset
Stop mode
External reset
Software reset
Watchdog timer reset
During a reset
Oscillation stabilization wait and reset state
Fetching the mode data
Pin setting in bus mode
Reset sequence
Fetching the reset vector
Program operation
CPU executes an instruction, fetching instruction codes from the address indicated by the reset vector.

### ■ Mode Pins

Setting the mode pins (MD0 to MD2) specifies how to fetch the mode data and the reset vector. Fetching the mode data and the reset vector is performed in the reset sequence. See Section 7.2 "Mode Pins (MD2 to MD0)" for details about mode pins.

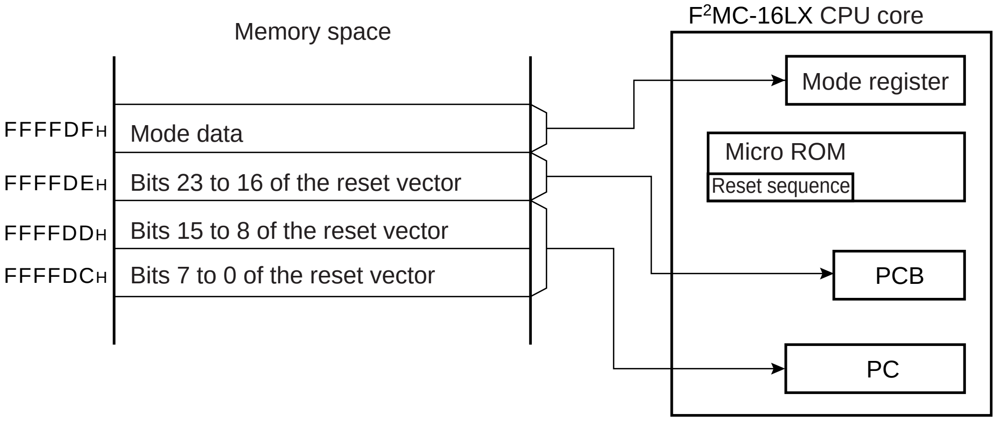
## ■ Mode Data Fetch

When a reset is cleared, the CPU transfers mode data to the mode data register. After the mode data is transferred, the reset vector is transferred to the program counter (PC) and the program counter bank register (PCB).

The mode data register can determine the bus mode and the bus width. The reset vector can determine the program start address.

For more information, see CHAPTER 7 "SETTING A MODE".

**Figure 3.4-2 Transfer of Reset Vector and Mode Data**



### ❍ Mode data register (address: FFFFDF$_H$)

The mode data register setting can be changed while a reset sequence is executed. The mode data register setting is valid after a reset vector is fetched. No new contents can be written to the mode data register even if an instruction is used to specify mode data at "FFFFDF$_H$".

For more information, see Section 7.3 "Mode Data Register".

### ❍ Reset vector (address: "FFFFDC$_H$" to "FFFFDE$_H$")

The reset vector determines the program start address used after a reset is cleared. A program is executed from the address specified in the reset vector.

# 3.5 Reset Cause Bits

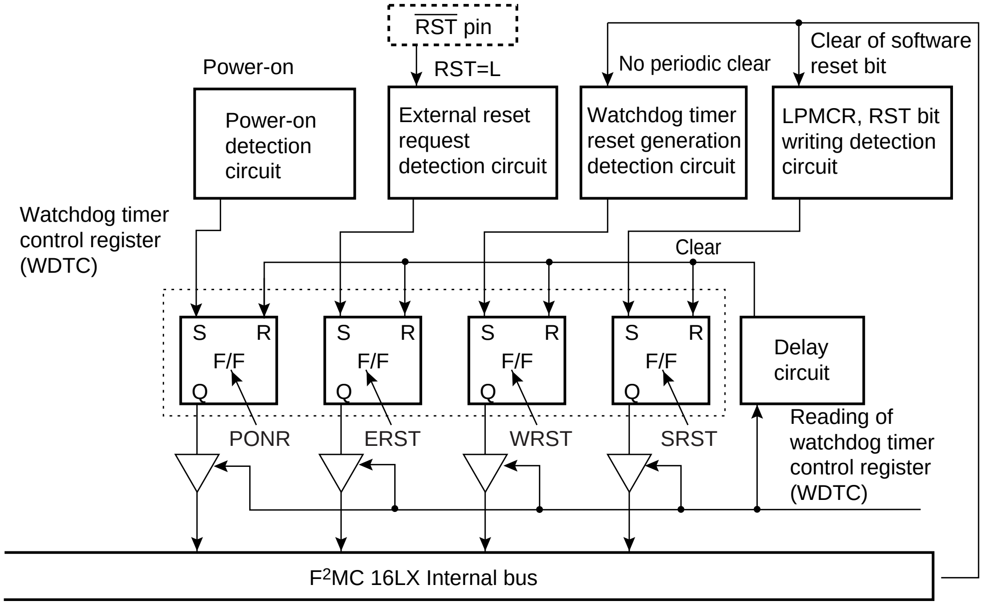
**Read the watchdog timer control register (WDTC) to identify a reset cause.**

## ■ Reset Cause Bits

Read the reset cause flag bits PONR, WRST, ERST, and SRST of the watchdog timer control register (WDTC) to identify a reset cause. If a reset cause needs to be identified after a reset is cleared, read the reset cause flag bits PONR, WRST, ERST, and SRST of the watchdog timer control register (WDTC).

The PONR, WRST, ERST, and SRST bits are cleared to "0" if the watchdog timer control register (WDTC) is read.

**Figure 3.5-1 Block Diagram of Reset Cause Bits**

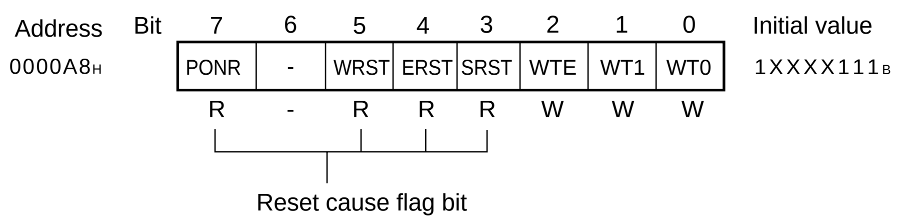

RST pin
RST=L
Power-on
No periodic clear
Clear of software reset bit
Power-on detection circuit
External reset request detection circuit
Watchdog timer reset generation detection circuit
LPMCR, RST bit writing detection circuit
Watchdog timer control register (WDTC)
Clear
S
R
F/F
Q
Delay circuit
PONR
ERST
WRST
SRST
Reading of watchdog timer control register (WDTC)
F2MC 16LX Internal bus

S : Set
R : Reset
Q : Output
F/F : Flip Flop

## ■ Correspondence between Reset Cause FLAG Bits and Reset Causes

### Figure 3.5-2 Configuration of Reset Cause Bits (Watchdog Timer Control Register)

Watchdog timer control register (WDTC)



R : Read only
W : Write only
X : Undefined
- : Undefined bit

### Table 3.5-1 Correspondence between Reset Cause Bits and Reset Causes

| Reset cause | PONR | WRST | ERST | SRST |
|---|---|---|---|---|
| Power-on reset | 1 | X | X | X |
| Watchdog timer overflow | * | 1 | * | * |
| External reset request via $\overline{\text{RST}}$ pin | * | * | 1 | * |
| Software reset request (LPMCR:RST) | * | * | * | 1 |

*: Previous state retained
X: Undefined

## ■ Notes about Reset Cause Bits

### ❍ Multiple reset causes generated at the same time

When multiple reset causes are detected, the corresponding reset cause bits of the watchdog timer control register (WDTC) are set to "1". For example, if an external reset and a watchdog timer reset occur at the same time, the reset cause flag bits (ERST, WRST) of the watchdog timer control register (WDTC) are set to "1".

### ❍ Power-on reset

If a power-on reset occurs, the PONR bit of the watchdog timer control register (WDTC) is set to "1" and the WRST, ERST, and SRST bits are undefined.

If the PONR bit is set to "1", the contents of the WRST, ERST, and SRST bits should be ignored.

### ❍ Clearing the reset cause bits

The PONR, WRST, ERST, and SRST bits are cleared to "0" if the watchdog timer control register (WDTC) is read. In other words, these reset cause flag bits are not cleared to "0" unless the watchdog timer control register (WDTC) is read even if a reset occurs.

**Note:**

> The value of the WDTC register cannot be assured if the power is turned on under conditions where a power-on reset does not occur.

# 3.6 Status of Pins in a Reset

**This section describes the status of pins when a reset occurs.**

## ■ Status of Pins during a Reset

The status of pins during a reset is determined by the settings of the mode pins (MD2 to MD0 = $011_B$).

### ❍ When internal vector mode is specified

All the I/O pins are set to high impedance output, and the mode data is read from the internal ROM.

## ■ Status of Pins after Mode Data is Read

The status of pins after mode data is read is determined by the mode data (M1 and M0 = $00_B$).

### ❍ When single-chip mode is specified (M1 and M0 = $00_B$)

All the I/O pins become high impedance output, and the mode data is read from the internal ROM.

**Note:**

Specify an external pin level that disables external circuits.

# CHAPTER 4 CLOCKS

**This chapter describes the clocks used by MB90M405 series.**

# 4.1 Clocks

**The clock generation block controls the operating clock of the CPU and peripheral functions (resources). The following four clocks are available:**

- **Oscillation clock**
- **Main clock**
- **PLL clock**
- **Machine clock**

## ■ Clocks

The clock generation block contains the oscillation circuit and the PLL clock multiplier circuit. The clock generation block controls the oscillation stabilization wait time and PLL clock multiplication as well as controls the operation of switching the clock with a clock selector.

### ❍ Oscillation clock frequency (HCLK)

The oscillation clock is generated either from an oscillator connected to the X0 and X1 pins or by input of an external clock.

### ❍ Main clock (MCLK)

The main clock, which is the oscillation clock divided by 2, supplies the clock input to the timebase timer and the clock selector.

### ❍ PLL clock (PCLK)

The PLL clock is obtained by multiplying the oscillation clock in the internal PLL clock multiplier circuit. Four different clocks (multiplied by 1 through 4) can be generated.

### ❍ Machine clock(ϕ)

The machine clock is the operating clock of the CPU and peripheral functions (resources). One machine clock cycle is called a machine cycle. Either the main clock or a PLL clock can be selected.

**Reference:**

PLL oscillation, which may be from 3 to 16.8 MHz, varies depending on the operating voltage and the frequency multiplier.

For more information, see the "data sheet".

**Note:**

The maximum operating frequency of the CPU and the peripheral function circuits is 16.8 MHz. If a frequency multiplier is specified that results in a frequency higher than the maximum operating frequency, devices will not operate normally.

For example, if an oscillation clock of 16.8 MHz is generated, a multiplier of 1 or division by 2 can be specified.

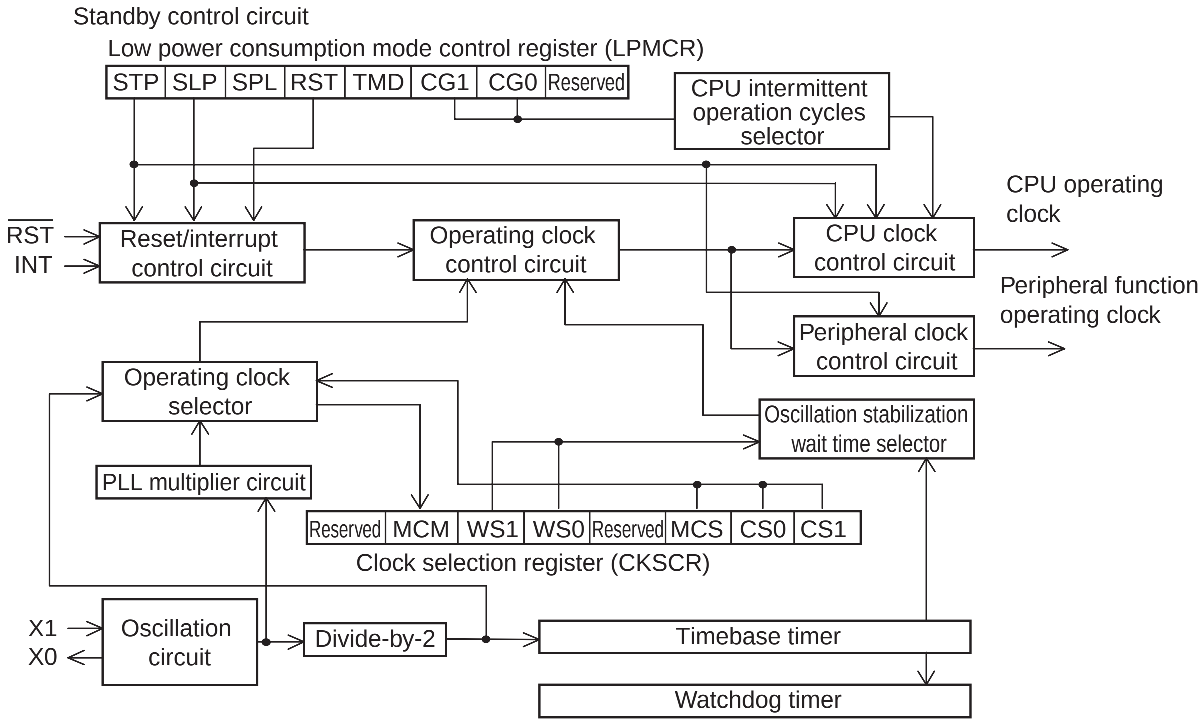
# 4.2 Block Diagram of the Clock Generation Block

**The clock generation block consists of the following five blocks:**

- **System clock generation circuit**
- **PLL multiplier circuitS**
- **Clock selector**
- **Clock selection register (CKSCR)**
- **oscillation stabilization wait time selector**

## ■ Block Diagram of the Clock Generation Block

**Figure 4.2-1 Block Diagram of the Clock Generation Block**


Standby control circuit
Low power consumption mode control register (LPMCR)
STP
SLP
SPL
RST
TMD
CG1
CG0
Reserved
CPU intermittent operation cycles selector
CPU operating clock
RST
INT
Reset/interrupt control circuit
Operating clock control circuit
CPU clock control circuit
Peripheral function operating clock
Peripheral clock control circuit
Operating clock selector
Oscillation stabilization wait time selector
PLL multiplier circuit
Reserved
MCM
WS1
WS0
Reserved
MCS
CS0
CS1
Clock selection register (CKSCR)
X1
X0
Oscillation circuit
Divide-by-2
Timebase timer
Watchdog timer

**Reference:**

Figure 4.2-1 "Block Diagram of the Clock Generation Block" includes the standby control circuit and the timebase timer circuit.

### ❍ System clock generation circuit

The system clock generation circuit generates an oscillation clock from an oscillator connected to the X0 and X1 pins or by input of an external clock.

**Reference:**

Figure 4.2-1 "Block Diagram of the Clock Generation Block" includes the standby control circuit and the timebase timer circuit.

❍ **PLL multiplier circuit**

The PLL multiplier circuit multiplies the oscillation clock and supplies the resultant clock to the clock selector.

❍ **Clock selector**

The clock selector selects the clock to be supplied to the CPU and peripheral clock control circuits from among the main clock and the PLL clock.

❍ **Clock selection register (CKSCR)**

The clock selection register selects the machine clock and determines the oscillation stabilization wait time and the PLL clock multiplier, etc.

❍ **oscillation stabilization wait time selector**

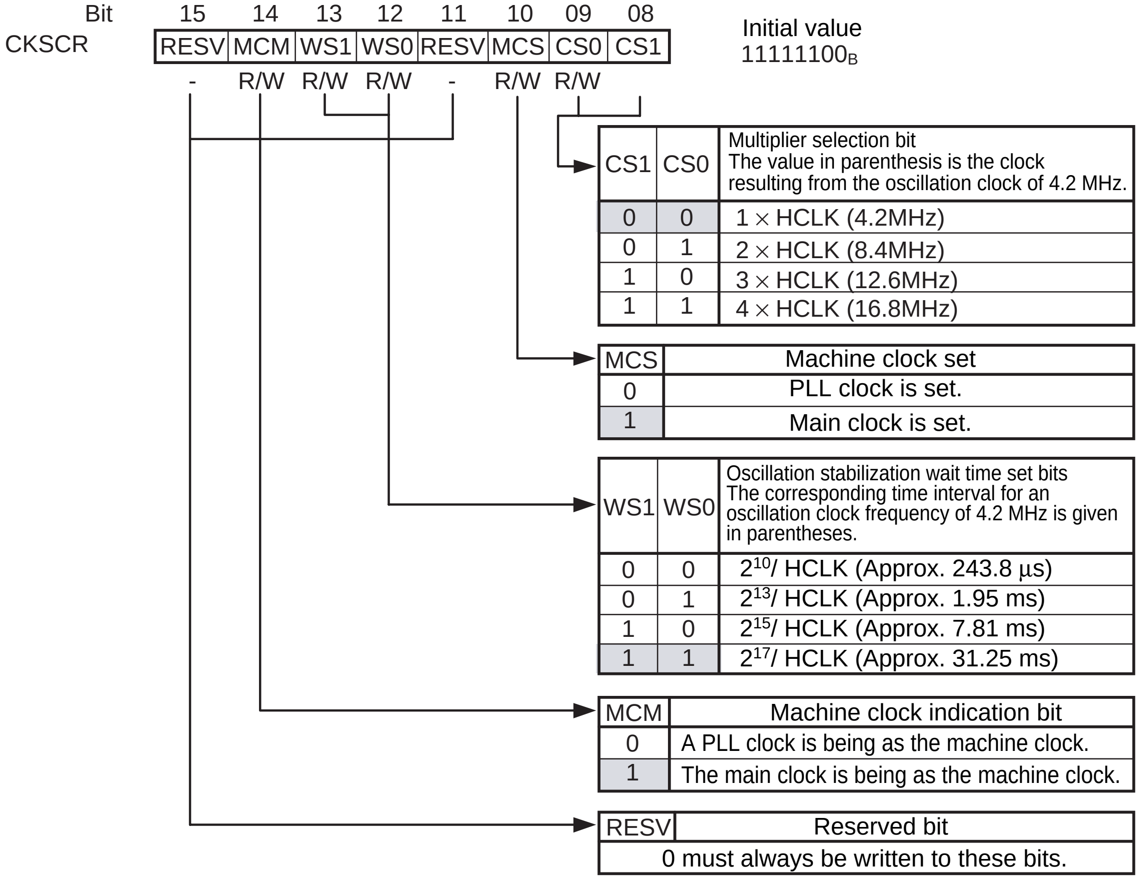
The oscillation stabilization wait time selector selects the oscillation stabilization wait time of the oscillation clock when the stop mode is cleared. One of the four time-base timer outputs is selected to determine the oscillation stabilization wait time.

# 4.3 Clock Selection Register (CKSCR)

**The clock selection register (CKSCR) switches the machine clock and sets the oscillation stabilization wait time and the PLL clock multiplier, etc.**

## ■ Configuration of the Clock Selection Register (CKSCR)

Figure 4.3-1 Configuration of the Clock Selection Register (CKSCR)



| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 |
|---|---|---|---|---|---|---|---|---|
| CKSCR | RESV | MCM | WS1 | WS0 | RESV | MCS | CS0 | CS1 |
| | - | R/W | R/W | R/W | - | R/W | R/W | |

Initial value
11111100B

| CS1 | CS0 | Multiplier selection bit<br>The value in parenthesis is the clock resulting from the oscillation clock of 4.2 MHz. |
|---|---|---|
| 0 | 0 | 1 × HCLK (4.2MHz) |
| 0 | 1 | 2 × HCLK (8.4MHz) |
| 1 | 0 | 3 × HCLK (12.6MHz) |
| 1 | 1 | 4 × HCLK (16.8MHz) |

| MCS | Machine clock set |
|---|---|
| 0 | PLL clock is set. |
| 1 | Main clock is set. |

| WS1 | WS0 | Oscillation stabilization wait time set bits<br>The corresponding time interval for an oscillation clock frequency of 4.2 MHz is given in parentheses. |
|---|---|---|
| 0 | 0 | $2^{10}$/ HCLK (Approx. 243.8 µs) |
| 0 | 1 | $2^{13}$/ HCLK (Approx. 1.95 ms) |
| 1 | 0 | $2^{15}$/ HCLK (Approx. 7.81 ms) |
| 1 | 1 | $2^{17}$/ HCLK (Approx. 31.25 ms) |

| MCM | Machine clock indication bit |
|---|---|
| 0 | A PLL clock is being as the machine clock. |
| 1 | The main clock is being as the machine clock. |

| RESV | Reserved bit |
|---|---|
| 0 must always be written to these bits. | |

| | | |
|---|---|---|
| HCLK | : | Oscillation clock frequency |
| R/W | : | Read/write |
| R | : | Read only |
| - | : | Undefined bit |
| ▨ | : | Initial value |

**Reference:**

A reset initializes the machine clock selection bit to the main clock setting.

**Table 4.3-1 Function Description of Each Bit of the Clock Selection Register (CKSCR)**

| Bit name | | Function |
|---|---|---|
| bit 15<br>bit 11 | RESV:<br>Reserved bit | **Note:**<br>Always set "0". |
| bit 14 | MCM:<br>Machine clock indication bit | • This bit indicates whether the main clock or a PLL clock has been selected as the machine clock.<br>• When this bit is set to "0", a PLL clock has been selected.<br>• When this bit is set to "1", the main clock has been selected.<br>• If the machine clock selection bit (MCS) is set to "0" and MCM is set to "1", the PLL clock oscillation stabilization wait time is in effect. |
| bit 13<br>bit 12 | WS1, WS0:<br>oscillation stabilization wait time selection bits | • These bits select the oscillation stabilization wait time for the oscillation clock after the stop mode has been cleared due to an external interrupt.<br>• A reset cause initializes these bits to "$11_B$".<br>• Specify an oscillation stabilization wait time appropriate for the oscillator used. |
| bit 10 | MCS:<br>Machine clock selection bit | • This bit specifies whether the main clock or a PLL clock is selected as the machine clock.<br>• When this bit is set to "0", a PLL clock is selected.<br>• When this bit is set to "1", the main clock is selected.<br>• If this bit has been set to "1" and is reset to "0", the oscillation stabilization wait time for the PLL clock starts. As a result, the time-base timer counter and the interrupt request flag bit (TBOF) of the time-base timer counter control register (TBTC) are cleared to "0".<br>• For PLL clocks, the oscillation stabilization wait time is fixed to $2^{14}$/HCLK. The oscillation stabilization wait time is about 3.9 ms if the oscillation clock frequency is 4.194 MHz.)<br>• When the main clock has been selected, the oscillation clock divided by 2 is used as the machine clock. The machine clock frequency is 2 MHz if the oscillation clock frequency is 4 MHz.<br>• A reset initializes this bit to 1.<br>**Note:**<br>The MCS bit set to "1" can be reset to "0" while the interrupt request enable bit (TBIE) of the time-base timer counter control register (TBTC) or the interrupt level mask register (ILM) are set to disable timer-base timer interrupt requests. |
| bit 9<br>bit 8 | CS1, CS0:<br>Multiplier selection bits | • These bits select a PLL clock multiplier.<br>• One of the four multipliers can be selected.<br>• A reset initializes these bits to "$00_B$".<br>**Note:**<br>These bits cannot be set while the machine clock selection bit (MCS) or the machine clock indication bit (MCM) is set to "0". Set these bits only after setting the MCS bit to "1". |

HCLK: Oscillation clock frequency

# 4.4 Clock Mode

**Two clock modes are provided: main clock mode and PLL clock mode.**

## ■ Main Clock Mode and PLL Clock Mode

### ❍ Main clock mode

In main clock mode, the main clock is used as the operating clock of the CPU and peripheral resources while the PLL clocks are disabled.

### ❍ PLL clock mode

In PLL clock mode, a PLL clock is used as the machine clock of the CPU and peripheral functions (resources). Specify a PLL clock multiplier in the multiplier selection bits (CS1 and CS0) of the clock selection register (CKSCR).

## ■ Clock Mode Transition

Setting the machine clock selection bit (MCS) of the clock selection register (CKSCR) causes switching between main clock mode and PLL clock mode.

### ❍ Switching from main clock mode to PLL clock mode

When the MCS bit of the CKSCR that is set to "1" is reset to "0", switching from the main clock to a PLL clock occurs after the PLL clock oscillation stabilization wait time ($2^{14}$/HCLK) has elapsed.

### ❍ Switching from PLL clock mode to main clock mode

When the MCS bit of the CKSCR that is set to "0" is reset to "1", switching from a PLL clock to the main clock occurs when the edges of the PLL clock and the main clock coincide (after 1 to 8 PLL clocks).

**Note:**

Before setting the peripheral functions (resources) after the machine clock switching, make sure that the machine clock has been switched by referring to the MCM bit of the CKSCR.

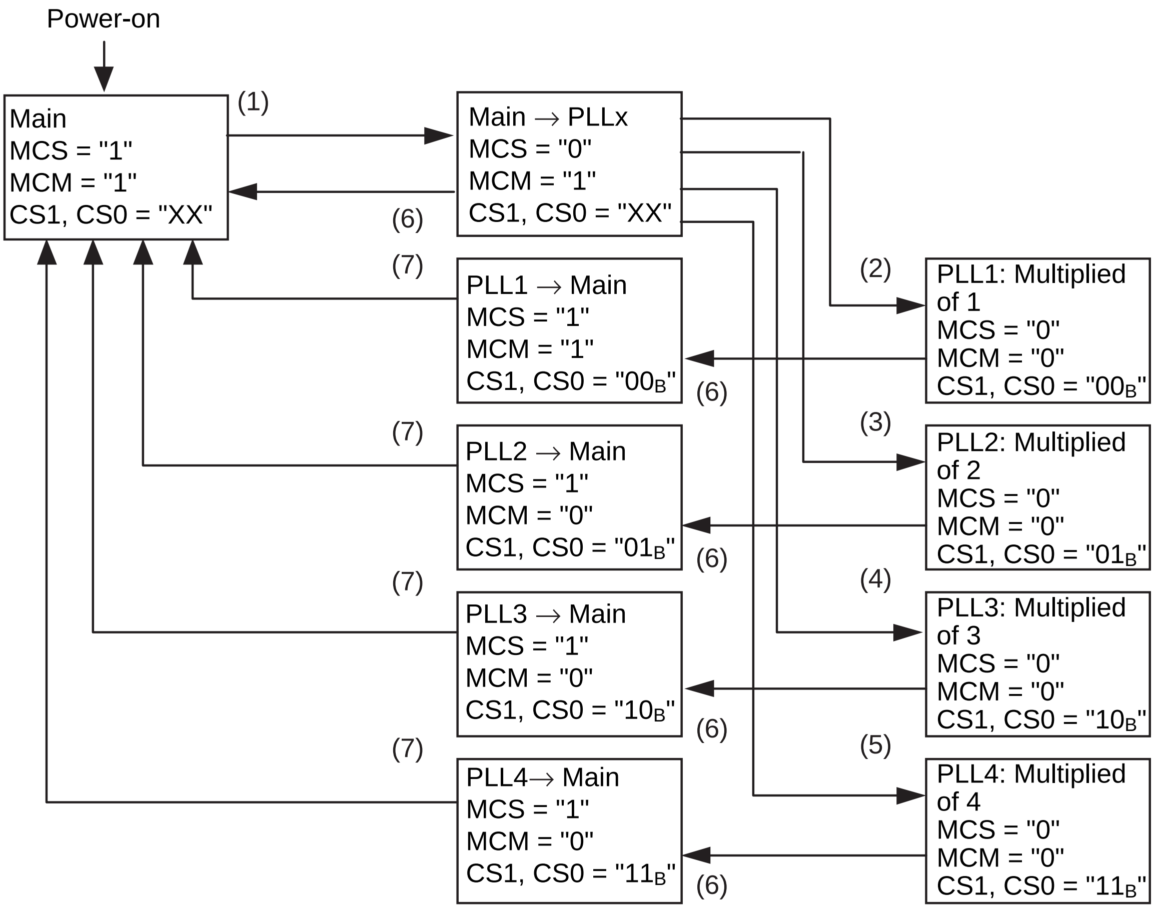
## ■ Selection of a PLL Clock Multiplier

Set the multiplier selection bits (CS1 and CS0) of the CKSCR to "$00_B$" and "$11_B$" to set one of the four PLL clock multipliers (1 through 4).

## ■ Machine Clock

Either the main clock or a PLL clock is used as the machine clock. The machine clock is an operating clock of the CPU and peripheral functions (resources). Set either the main clock or a PLL clock in the MCS bit of the CKSCR.

**Figure 4.4-1 Status Change Diagram for Machine Clock Selection**



(1) The MCS bit is cleared.
(2) The PLL clock oscillation stabilization wait ends with CS1 and CS0 = $00_B$.
(3) The PLL clock oscillation stabilization wait ends with CS1 and CS0 = $01_B$.
(4) The PLL clock oscillation stabilization wait ends with CS1 and CS0 = $10_B$.
(5) The PLL clock oscillation stabilization wait ends with CS1 and CS0 = $11_B$.
(6) The MCS bit is set (including hardware standby and watchdog timer resets).
(7) PLL clock and main clock frequency synchronization timing

| | | |
|---|---|---|
| MCS | : | Machine clock set bit of CKSCR |
| MCM | : | Machine clock indication bit of CKSCR |
| CS1, CS0 | : | Multiplier set bits of CKSCR |

**Note:**

The initial value for the machine clock setting is main clock (CKSCR:MCS = 1).

## 4.5 Oscillation Stabilization Wait Time

**Whenever the power is turned on, or whenever stop mode is cleared, oscillation begins following a state in which there was no oscillation. Accordingly, an oscillation stabilization wait time is required. Also, whenever the switching from the main clock to a PLL clock occurs, an oscillation stabilization wait time is required after the oscillation of the PLL clock starts.**

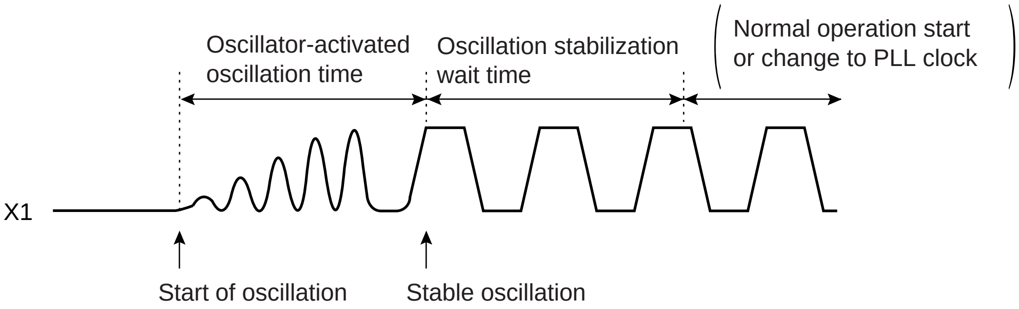
### ■ Oscillation Stabilization Wait Time

Specify an oscillation stabilization wait time appropriate for the oscillator used because the oscillation stabilizes in different lengths of time depending on the oscillator type. Specify an appropriate oscillation stabilization wait time in the oscillation stabilization wait time selection bits (WS1 and WS0) of the clock selection register (CKSCR).

When switching from the main clock to a PLL clock occurs, the CPU operates on the main clock during an oscillation stabilization wait time and starts to operate on a PLL clock.

The timebase timer counts the specified oscillation stabilization wait time.

**Figure 4.5-1 Operation When Oscillation Starts**


Oscillator-activated oscillation time
Oscillation stabilization wait time
Normal operation start or change to PLL clock
X1
Start of oscillation
Stable oscillation

# 4.6 Connection of an Oscillator or an External Clock to the Microcontroller

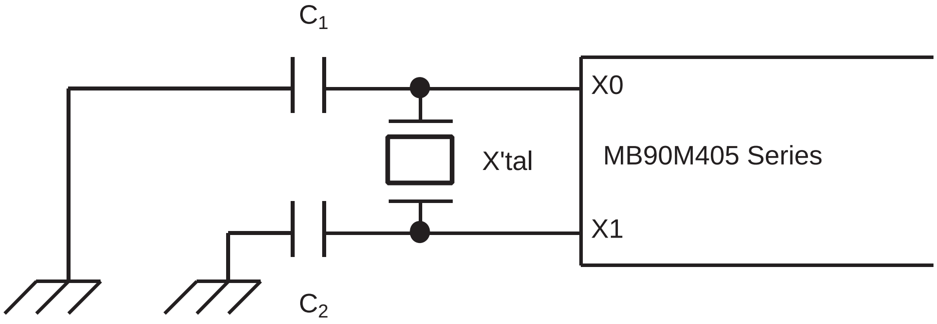
**The MB90M405 series contains a system clock generation circuit. An oscillator can be connected to the X0 and X1 pins.**
**Alternatively, pulses from an external clock may be input.**

## ■ Connection of an Oscillator or an External Clock to the Microcontroller

### ❍ Example of connecting a crystal or ceramic oscillator to the microcontroller

Connect a crystal or ceramic oscillator as shown in the example in Figure 4.6-1 "Example of Connecting a Crystal or Ceramic Oscillator to the Microcontroller".

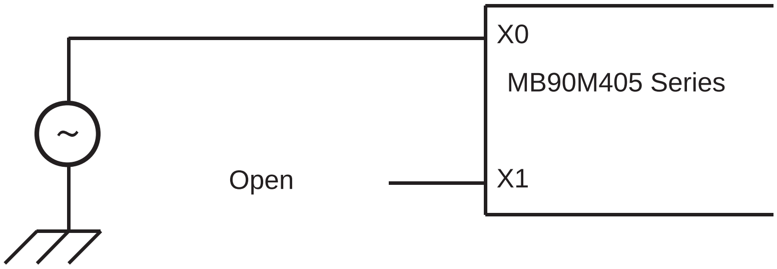
**Figure 4.6-1 Example of Connecting a Crystal or Ceramic Oscillator to the Microcontroller**



### ❍ Example of connecting an external clock to the microcontroller

As shown in Figure 4.6-2 "Example of Connecting an External Clock to the Microcontroller", connect an external clock to pin X0. Pin X1 must be open.

**Figure 4.6-2 Example of Connecting an External Clock to the Microcontroller**



# CHAPTER 5 LOW POWER CONSUMPTION MODE

**This chapter describes the low power consumption mode of MB90M405 series.**

# 5.1 Low Power Consumption Mode

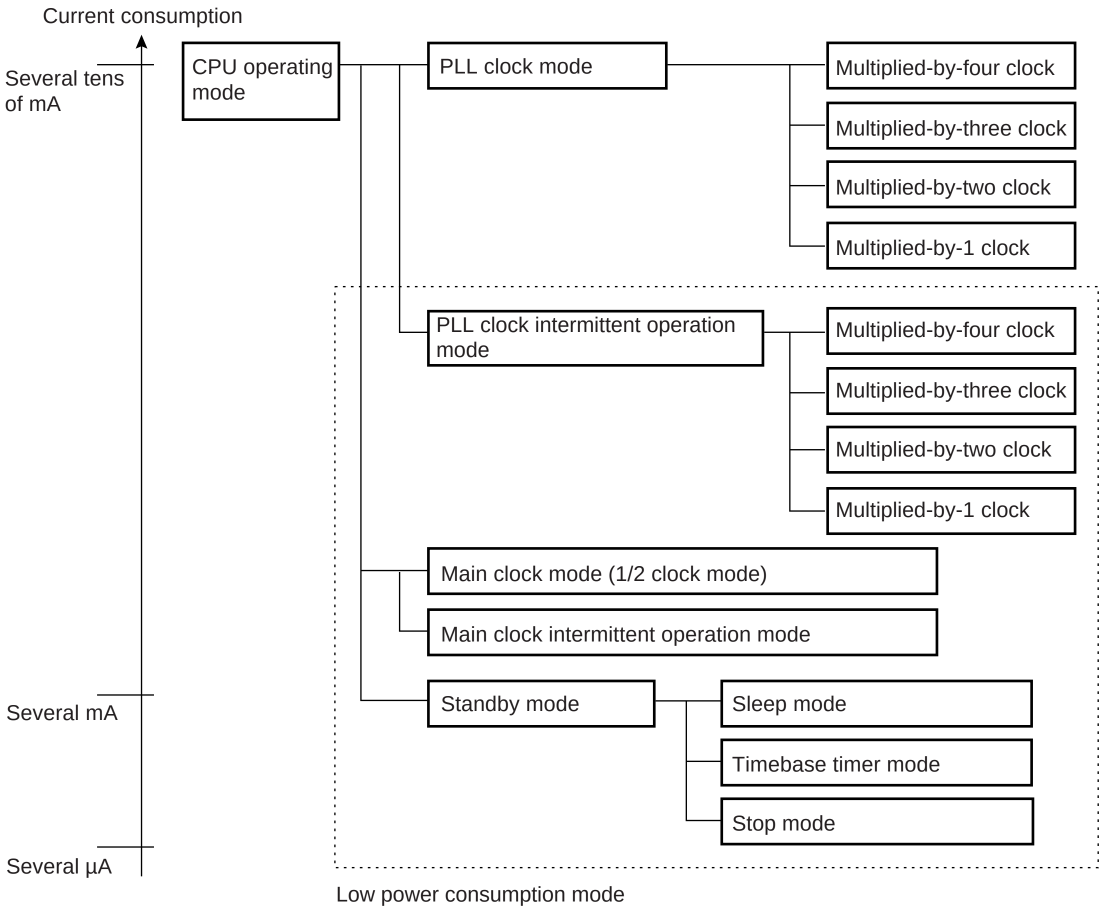
**The MB90M405 series has the following low power consumption modes, one of which can be selected depending on the operating clock setting and the clock operation control.**

- **CPU intermittent operation mode (PLL clock intermittent operation mode and main clock intermittent operation mode)**
- **Standby mode (sleep mode, timebase timer mode)**

**All modes other than PLL clock mode are low power consumption modes.**

## ■ CPU Operating Modes and Current Consumption

**Figure 5.1-1 CPU Operating Modes and Current Consumption**



Note:
This figure is only an indication of the degree of power consumption for each mode.
Actual current consumption values may not agree with those in the figure.

## ■ Clock Mode

### ❍ PLL clock mode

The CPU and peripheral functions (resources) operate on a PLL clock.

### ❍ Main clock mode

The CPU and peripheral functions (resources) operate on the main clock. In the main clock mode, the PLL multiplier circuit is disabled.

See Section 4.4 "Clock Mode", for details about clock mode.

## ■ CPU Intermittent Operation Mode

The CPU operates intermittently while the machine clock is supplied to the peripheral functions (resources).

## ■ Standby Mode

### ❍ PLL sleep mode

The CPU operating clock is stopped. Other components continue to operate on a PLL clock.

### ❍ Main sleep mode

The CPU operating clock is stopped. Other components continue to operate on the main clock.

### ❍ Timebase timer mode

All the components but the oscillation clock and the timebase timer are stopped.

### ❍ Stop mode

The oscillation clock is stopped. All the functions are stopped.

**Note:**

> In stop mode, data can be retained at the minimum power consumption because the oscillation clock has stopped.

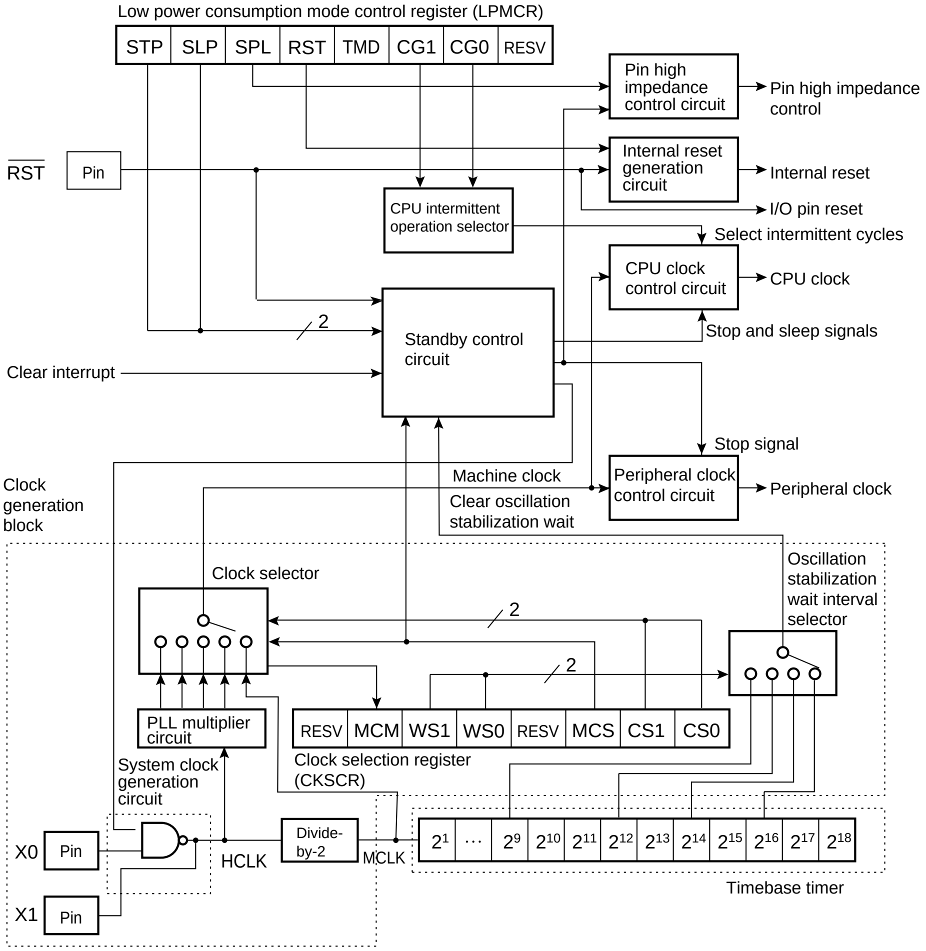
# 5.2 Block Diagram of the Low Power Consumption Control Circuit

**The low power consumption control circuit consists of the following circuits and register:**

- **CPU intermittent operation selector**
- **Standby clock control circuit**
- **CPU clock control circuit**
- **Peripheral clock control circuit**
- **Pin high-impedance control circuit**
- **Internal reset generation circuit**
- **Low power consumption mode control register (LPMCR)**

## ■ Block Diagram of the Low Power Consumption Control Circuit

**Figure 5.2-1 Block Diagram of the Low Power Consumption Control Circuit**


Low power consumption mode control register (LPMCR)
STP
SLP
SPL
RST
TMD
CG1
CG0
RESV
Pin high impedance control circuit
Pin high impedance control
RST
Pin
Internal reset generation circuit
Internal reset
I/O pin reset
CPU intermittent operation selector
Select intermittent cycles
CPU clock control circuit
CPU clock
2
Standby control circuit
Stop and sleep signals
Clear interrupt
Stop signal
Machine clock
Peripheral clock control circuit
Peripheral clock
Clear oscillation stabilization wait
Clock generation block
Clock selector
2
Oscillation stabilization wait interval selector
2
PLL multiplier circuit
RESV
MCM
WS1
WS0
RESV
MCS
CS1
CS0
Clock selection register (CKSCR)
System clock generation circuit
X0
Pin
HCLK
Divide-by-2
MCLK
2^1
…
2^9
2^10
2^11
2^12
2^13
2^14
2^15
2^16
2^17
2^18
Timebase timer
X1
Pin

❍ **CPU intermittent operation selector**

This selector selects the number of clock pulses during which the CPU is halted in CPU intermittent operation mode.

❍ **Standby control circuit**

This circuit controls the CPU clock control circuit, the peripheral clock control circuit, and the pin high-impedance control circuit to switch to, or release low power consumption mode.

❍ **CPU clock control circuit**

This circuit control the clocks supplied to the CPU.

❍ **Peripheral clock control circuit**

This circuit control the clocks supplied to the peripheral functions (resources).

❍ **Pin high-impedance control circuit**

A setting of this circuit to timebase timer mode or stop mode causes the I/O pins to enter a high impedance state. For those I/O pins configured to accept the connection of a pull-up resistor, this circuit disconnects the pull-up resistor in stop mode.

❍ **Internal reset generation circuit**

If the RST bit of the low power consumption mode control register (LPMCR:RST) is set to "0" by software, this circuit generates a three-cycle reset signal to the internal circuit.

❍ **Low power consumption mode control register (LPMCR)**

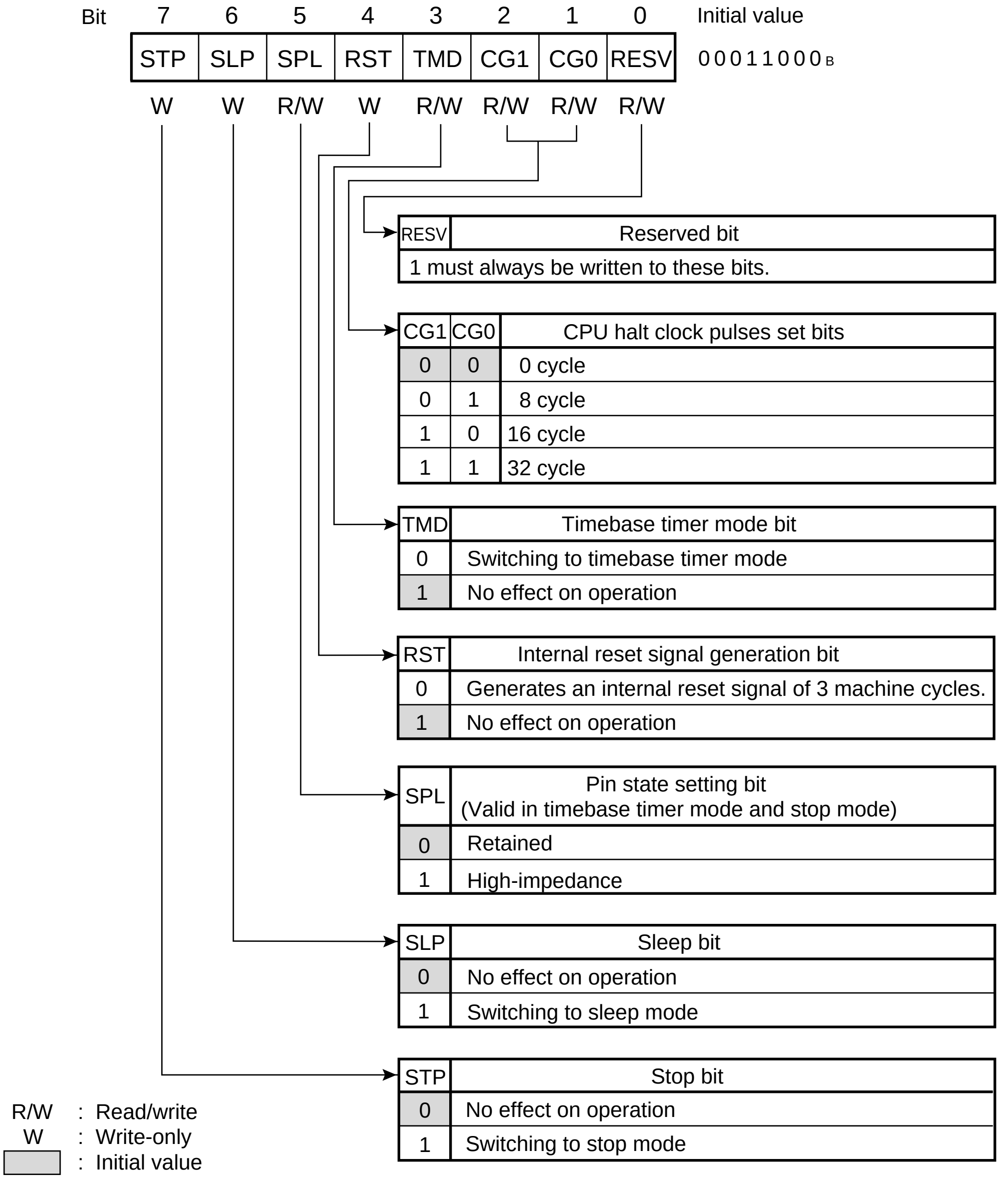
This register selects either switching to or release of low power consumption mode and the number of clock pulses during which the CPU is halted in CPU intermittent operation mode.

# 5.3 Low Power Consumption Mode Control Register (LPMCR)

**The low power consumption mode control register (LPMCR) selects the switching to, or release of low power consumption mode and the number of clock pulses during which the CPU is halted in CPU intermittent operation mode.**

## ■ Low Power Consumption Mode Control Register (LPMCR)

**Figure 5.3-1 Configuration of the Low Power Consumption Mode Control Register (LPMCR)**



| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | STP | SLP | SPL | RST | TMD | CG1 | CG0 | RESV | 00011000B |
| | W | W | R/W | W | R/W | R/W | R/W | R/W | |

| RESV | Reserved bit |
|---|---|
| 1 must always be written to these bits. | |

| CG1 | CG0 | CPU halt clock pulses set bits |
|---|---|---|
| 0 | 0 | 0 cycle |
| 0 | 1 | 8 cycle |
| 1 | 0 | 16 cycle |
| 1 | 1 | 32 cycle |

| TMD | Timebase timer mode bit |
|---|---|
| 0 | Switching to timebase timer mode |
| 1 | No effect on operation |

| RST | Internal reset signal generation bit |
|---|---|
| 0 | Generates an internal reset signal of 3 machine cycles. |
| 1 | No effect on operation |

| SPL | Pin state setting bit<br>(Valid in timebase timer mode and stop mode) |
|---|---|
| 0 | Retained |
| 1 | High-impedance |

| SLP | Sleep bit |
|---|---|
| 0 | No effect on operation |
| 1 | Switching to sleep mode |

| STP | Stop bit |
|---|---|
| 0 | No effect on operation |
| 1 | Switching to stop mode |

R/W : Read/write
W : Write-only
▨ : Initial value

**Table 5.3-1 Function Description of Each Bit of the Low Power Consumption Mode Control Register (LPMCR)**

| Bit name | | Function |
|---|---|---|
| bit 7 | STP:<br>Stop bit | • This bit selects the stop mode.<br>• When this bit is set to "1", the microcontroller enters stop mode.<br>• When this bit is set to "0", there is no effect on operation.<br>• An external reset or the output of a hardware interrupt resets this bit to "0".<br>• The read value of this bit is "0". |
| bit 6 | SLP:<br>Sleep bit | • This bit selects sleep mode.<br>• When this bit is set to "1", the microcontroller enters sleep mode.<br>• When this bit is set to "0", there is no effect on operation.<br>• An external reset or the output of a hardware interrupt resets this bit to "0".<br>• The read value of this bit is "0". |
| bit 5 | SPL:<br>Pin state setting bit valid in timebase timer mode or stop mode) | • This bit selects a pin state in timebase timer mode or stop mode.<br>• When this bit is set to "0", an I/O pin has a retained level.<br>• When this bit is set to "1", an I/O pin has high impedance.<br>• An external reset resets this bit to "0". |
| bit 4 | RST:<br>Internal reset signal generation bit | • This bit selects an internal reset.<br>• When this bit is set to "0", an internal reset signal of three machine cycles is generated.<br>• When this bit is set to "1", there is no effect on operation.<br>• The read value of this bit is "1". |
| bit 3 | TMD:<br>Timebase timer mode bit | • This bit selects the switching to timebase timer mode.<br>• When this bit is set to "0", the microcontroller enters timebase timer mode.<br>• When this bit is set to "1", there is no effect on operation.<br>• An external reset or a hardware interrupt return sets this bit to "1".<br>• The read value of this bit is "1". |
| bit 2<br>bit 1 | CG1, CG0:<br>CPU halt clock pulses selection bits | • These bits set the number of CPU halt clock pulses in CPU intermittent operation mode.<br>• The clock supplied to the CPU is stopped for the specified number of clock cycles every time after an instruction is executed.<br>• These bits select one of the four clock pulses.<br>• A reset initializes these bits to "$00_B$". |
| bit 0 | RESV:<br>Reserved bit | • This bit must be set to "0". |

## ■ Access to the Low Power Consumption Mode Control Register

Use one of the instructions listed in Table 5.3-2 "Instructions to Be Used for Switching to Low Power Consumption Mode" to set low power consumption mode control register. The operation is not assured if any instruction other than that listed in Table 5.3-2 "Instructions to Be Used for Switching to Low Power Consumption Mode" is used to enter low power consumption mode. Any instruction may be used provided it is not an instruction that controls switching to low power consumption mode via the low power consumption mode control register.

When using word access to set the low power consumption mode control register, use an even-numbered address. A malfunction may occur if an odd-numbered address is used to enter low power consumption mode.

**Table 5.3-2 Instructions to Be Used for Switching to Low Power Consumption Mode**

| | | | |
|---|---|---|---|
| MOV io, #imm8 | MOV dir, #imm8 | MOV eam, #imm8 | MOV eam, Ri |
| MOV io, A | MOV dir, A | MOV addr, A | MOV eam, A |
| MOV @RLi+disp8, A | MOVP addr24, A | | |
| MOVW io, #imm16 | MOVW dir, #imm16 | MOVW eam, #imm16 | MOVW eam, RWi |
| MOVW io, A | MOVW dir, A | MOVW addr16, | MOVW eam, A |
| MOVW @RLi+disp8, A | MOVPW addr24, A | | |

## ■ Priority of STP, SLP and TMD bits

If stop mode (LPMCR: STP), sleep mode (LPMCR: SLP), and timebase timer mode (LPMCR: TMD) are simultaneously specified, the microcontroller enters the following order of priority:

stop mode, timebase timer mode, or sleep mode

# 5.4 CPU Intermittent Operation Mode

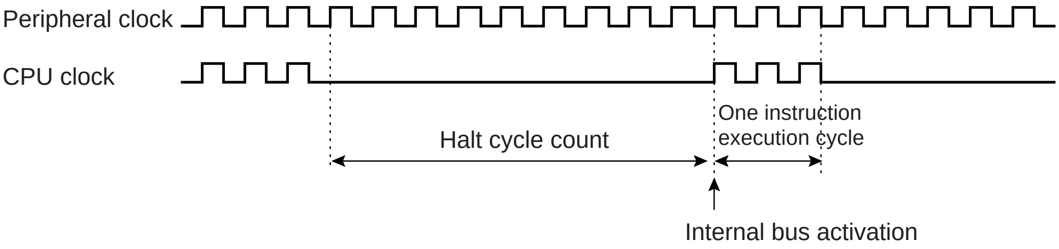
**In CPU intermittent operation mode, the CPU operates intermittently while the peripheral functions (resources) operate on the machine clock.**

## ■ CPU Intermittent Operation Mode

In CPU intermittent operation mode, the machine clock to be supplied to the CPU is halted for a certain period every time after an instruction is executed, so that the activation of an internal bus cycle is delayed. Reducing the CPU speed while supplying a fast peripheral clock to the peripheral function circuits allows processing with low power consumption.

- Use the CG1 and CG0 bits of the low power consumption mode control register (LPMCR) to specify the number of cycles during which the clock supplied to the CPU is halted.
- For external bus operation, use the same clock as that for the peripheral functions.
- You can calculate the instruction execution time required when CPU intermittent operation mode is used as follows: Multiply the instruction execution count required to access registers, built-in memory, built-in peripheral functions (resources), and external buses by the halt cycle count, and add this correction value to the regular execution time.

**Figure 5.4-1 Clock Pulses during CPU Intermittent Operation**



## 5.5 Standby Mode

**Standby mode includes the sleep mode (PLL sleep mode and main sleep mode), timebase timer mode, and stop mode.**

### ■ Operating Status during Standby Mode

**Table 5.5-1 Operation Statuses in Standby Mode**

| Standby mode | | Condition for switch | Oscil-lation | Clock | CPU | Timebase timer Watchdog timer | Peripheral | Pin | Release event |
|---|---|---|---|---|---|---|---|---|---|
| Sleep mode | PLL sleep mode | MCS = 0<br>SLP = 1 | Active | Active | In-active | Active | Active | Active | External reset or hardware interrupt |
| | Main sleep mode | MCS = 1<br>SLP = 1 | | | | | | | |
| Time-base timer mode | Timebase timer mode (SPL = 0) | TMD = 0 | | In-active | In-active | | Inactive | Hold | |
| | Timebase timer mode (SPL = 1) | TMD = 0 | | | | | | Hi-z | |
| Stop mode | Stop mode (SPL = 0) | STP = 1 | In-active | | | Inactive | Inactive | Hold | |
| | Stop mode (SPL = 1) | STP = 1 | | | | | | Hi-z | |

SPL: Pin state setting bit of low power consumption mode control register (LPMCR)
SLP: Sleep bit of low power consumption mode control register (LPMCR)
STP: Timebase timer or stop bit of low power consumption mode control register (LPMCR)
TMD: Timebase timer mode bit of low power consumption mode control register (LPMCR)
MCS: Machine clock selection bit of clock set register (CKSCR)
Hi-z: High-impedance

## 5.5.1 Sleep Mode

**In sleep mode, the CPU operating clock is halted while components other than the CPU continue to operate. When switching to sleep mode is specified, the microcontroller enters PLL sleep mode if PLL clock mode has been specified or the main sleep mode if the main clock mode has been specified.**

### ■ Switching to Sleep Mode

Set the sleep mode bit (SLP) of the low power consumption mode control register (LPMCR) to "1", the timebase timer mode bit (TMD) to "1", and the stop mode bit (STP) to "0" to enter sleep mode. When switching to sleep mode is specified, the microcontroller enters PLL sleep mode if the machine clock setting bit (MCS) of the clock selection register (CKSCR) is set to "0". Alternatively, the microcontroller enters main sleep mode if the MCS is set to "1".

**Note:**

If you make the stop mode bit (STP), sleep mode bit (SLP), and timebase timer mode bit (TMD) effective at the same time, the stop mode bit (STP) has precedence. If you make the sleep mode bit (SLP) and timebase timer mode bit (TMD) effective at the same time, the timebase timer mode bit (TMD) has precedence. The order of priority is as follows:

Stop mode > Timebase timer mode > Sleep mode

#### ❍ Data retention function

In sleep mode, the contents of both the dedicated registers and internal RAM are retained.

For more information on dedicated registers, see Section 2.7 "Dedicated Registers".

#### ❍ Operation during output of an interrupt request

While an interrupt request is output, the microcontroller does not enter sleep mode but executes the next instruction even if the sleep mode bit (SLP) of the low power consumption mode control register (LPMCR) is set to "1".

#### ❍ Pin state

In sleep mode, the pin state existing just before sleep mode is entered is retained.

### ■ Release of Sleep Mode

An external reset or the output of a hardware interrupt releases sleep mode.

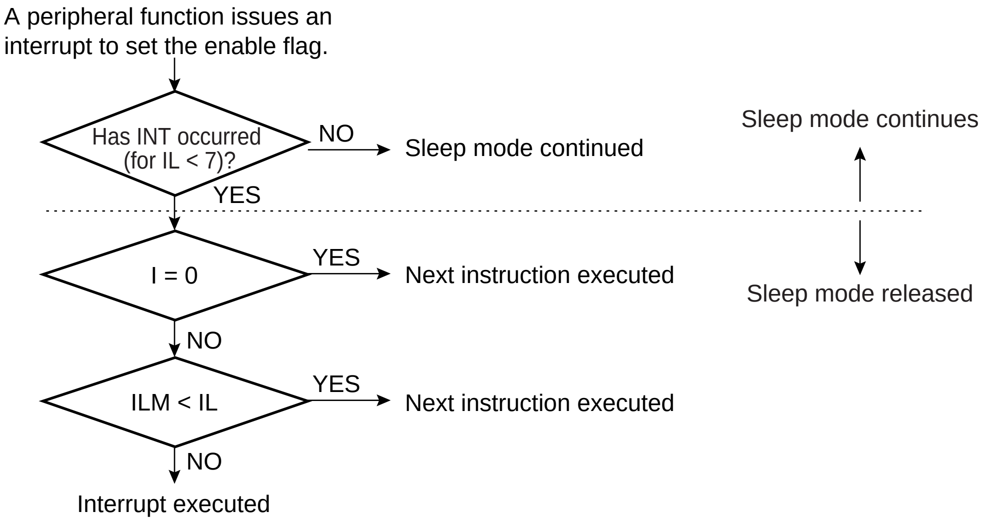
#### ❍ Release by a reset

For more information, see Section 3.4 "Reset Operation".

#### ❍ Release by a hardware interrupt

An interrupt request with an interrupt level higher than 7 (Interrupt control register ICR: IL2, IL1, IL0 = "$000_B$" to "$110_B$") is required to cause a hardware interrupt to release sleep mode.

**Figure 5.5-1 Release of Sleep Mode upon Occurrence of a Hardware Interrupt**


A peripheral function issues an interrupt to set the enable flag.
Has INT occurred (for IL < 7)?
NO
Sleep mode continued
YES
Sleep mode continues
I = 0
YES
Next instruction executed
Sleep mode released
NO
ILM < IL
YES
Next instruction executed
NO
Interrupt executed

**Note:**

To handle an interrupt, the microcontroller normally starts with interrupt processing after executing the instruction following the instruction that specifies sleep mode. However, the microcontroller may start interrupt processing before executing the next instruction if a sleep mode request and an external bus hold request are accepted at the same time.

## 5.5.2 Timebase Timer Mode

**In timebase timer mode, all of the operations other than the source oscillation and the timebase timer are stopped.**

### ■ Switching to Timebase Timer Mode

If the timebase timer mode bit (TMD) of the low power consumption mode control register (LPMCR) is set to "0", the microcontroller enters timebase timer mode.

#### ❍ Data retention function

In timebase timer mode, the contents of both the dedicated registers and internal RAM are retained.

For more information on dedicated registers, see Section 2.7 "Dedicated Registers".

#### ❍ Operation during output of an interrupt request

While an interrupt request is output, the microcontroller does not enter timebase timer mode even if the timebase timer mode bit (TMD) of the low power consumption mode control register (LPMCR) is set to "0".

#### ❍ Status of pins

The I/O pins in timebase timer mode can be set to retain a previous level or have high impedance in the pin status setting bit (SPL) of the low power consumption mode control register (LPMCR).

### ■ Release of Timebase Timer Mode

Timebase timer mode is released by an external reset, timebase timer interrupt, or hardware interrupt resulting from input of an external interrupt.

#### ❍ Release by a reset

For more information, see Section 3.4 "Reset Operation".

#### ❍ Release by a hardware interrupt

An interrupt request with an interrupt level higher than 7 (IL2, IL1 and IL0 of the interrupt control register (ICR) are $000_B$ to $110_B$) is required to release timebase timer mode with a hardware interrupt.

**Note:**

When interrupt processing is executed, the microcontroller normally enters interrupt processing after executing the instruction after the instruction that specifies timebase timer mode.

## 5.5.3 Stop Mode

**In stop mode, the source oscillation is stopped. Since all the functions are stopped, data can be retained while minimum power is consumed.**

### ■ Switching to Stop Mode

If the stop mode bit (STP) of the low power consumption mode control register (LPMCR) is set to "1", the microcontroller enters stop mode.

#### ❍ Data retention function

In stop mode, the contents of both the dedicated registers and RAM are retained.

For more information on dedicated registers, see Section 2.7 "Dedicated Registers".

#### ❍ Operation during acceptance or execution of an interrupt request

While an interrupt request is being accepted or executed, the microcontroller does not enter stop mode even though the stop mode bit (STP) of the low power consumption mode control register (LPMCR) is set to "1".

#### ❍ Status of pins

The I/O pins in stop mode can be set to retain a previous level or have high impedance in the pin status setting bit (SPL) of the low power consumption mode control register (LPMCR).

### ■ Release of Stop Mode

Stop mode is released by an external reset or hardware interrupt resulting from input of an external interrupt.

#### ❍ Release by a reset

For more information, see Section 3.4 "Reset Operation".

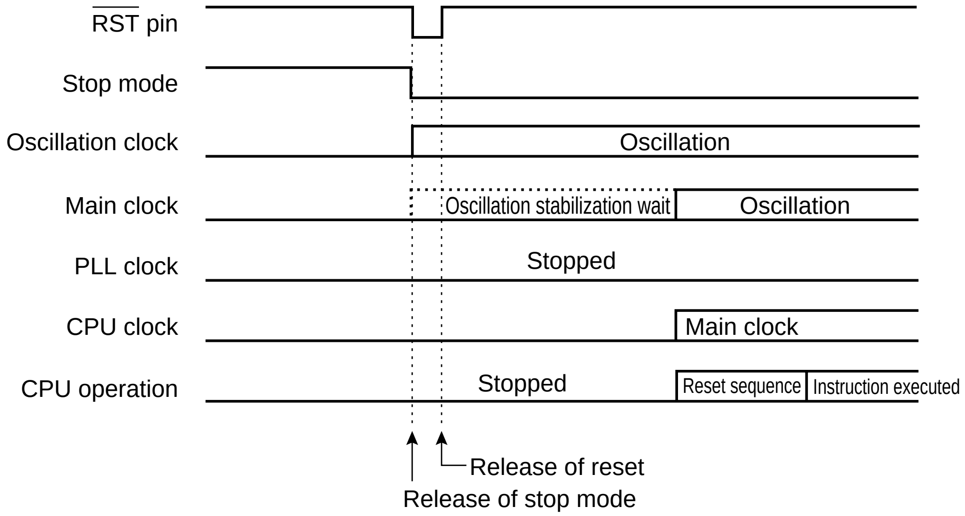
#### ❍ Release by a hardware interrupt

An interrupt request with an interrupt level higher than 7 (IL2, IL1 and IL0 of the interrupt control register (ICR) are $000_B$ to $110_B$) is required to release stop mode with an external interrupt.

**Note:**

> To handle an interrupt, the microcontroller enters interrupt processing in ordinary cases after executing the instruction following the instruction that specifies stop mode. However, the microcontroller may enter interrupt processing before executing the next instruction if a request to enter stop mode and an external bus hold request are received at the same time.

**Figure 5.5-2 Release of Stop Mode (External Reset)**

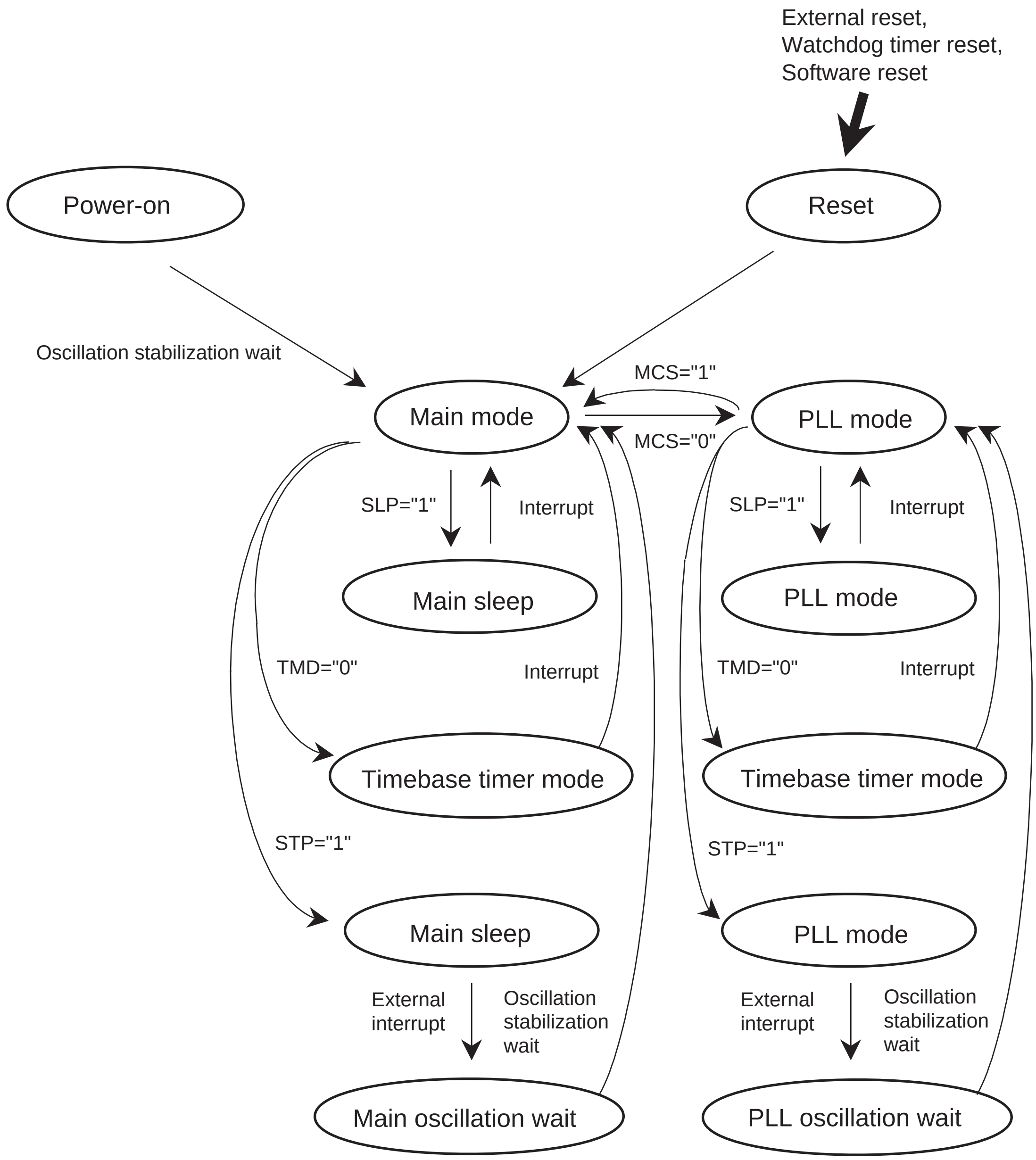


# 5.6 Status Change Diagram

**Figure 5.6-1 "Status Change Diagram" shows the CPU operation modes of the MB90M405 series and a state transition diagram.**

## ■ Status Change Diagram

**Figure 5.6-1 Status Change Diagram**



## ■ Operating States in Low Power Consumption Mode

**Table 5.6-1 Operating States in Low Power Consumption Mode**

| Low power consumption mode | Entry condition | Oscillation | Machine clock | CPU | Peripheral | Pin | Release method |
|---|---|---|---|---|---|---|---|
| Main sleep | MCS="1"<br>SLP="1" | Running | Running | Stopped | Running | Running | Reset<br>interrupt |
| PLL sleep | MCS="0"<br>SLP="1" | Running | Running | Stopped | Running | Running | Reset<br>interrupt |
| Timebase timer (SPL="0") | TMD="0" | Running | Stopped | Stopped | Stopped | Retained | Reset<br>interrupt |
| Timebase timer (SPL="1") | TMD="0" | Running | Stopped | Stopped | Stopped | Hi-z | Reset<br>interrupt |
| Stop (SPL="0") | MCS="1"<br>STP="1" | Stopped | Stopped | Stopped | Stopped | Retained | Reset<br>interrupt |
| Stop (SPL="1") | MCS="1"<br>STP="1" | Stopped | Stopped | Stopped | Stopped | Hi-z | Reset<br>interrupt |

## 5.7 Pin Status in Standby Mode and during Reset

**Table 5.7-1 "State of Pins in Single-Chip Mode" shows the status of pins in standby mode and during a reset.**

### ■ Software Pull-Up Resistor

For I/O pins configured in software to accept the connection of a pull-up resistor, set the port to an output setting that disconnects the pull-up resistor.

### ■ Status of Pins in Single-Chip Mode

**Table 5.7-1 State of Pins in Single-Chip Mode**

| Pin name | Standby mode | | | Reset |
|---|---|---|---|---|
| | Sleep mode | Stop mode | | |
| | | SPL=0 | SPL=1 | |
| P82 to P87<br>P90 to P91<br>PA0 to PA7<br>PB0 to PB5 | The preceding status is retained. (*2) | The preceding status is retained. (*2) | Input shut off (*3) /output Hi-z | Output Hi-z (*4) |
| P80, P81<br>PB6, PB7 | | Input enabled (*1) | | |

*1: "Input enabled" means that the input function is enabled. However, the input function is enabled only if an external interrupt is enabled. These pins, when used as output ports, conform to the setting of the pin status setting bit (SPL) of the low power consumption mode control register (LPMCR).

*2: "The preceding status is retained" means that the output status of the pin immediately before switching to standby mode is retained (*5). However, note that the input is disabled (*6) if the pin was in the input status.

*3: "Input shut off" means that the input to the pin is inhibited.

*4: "Output Hi-Z" means that the pin-driving transistor is disabled and that the pin is made to have high impedance.

*5: "the output status is retained as is" means that the output value of a peripheral function (resource) or a port is retained.

*6: "the input is disabled" means that a value input to the pin cannot be accepted internally because an internal circuit is not running.

# 5.8 Usage Notes on Low Power Consumption Mode

**Note the following items when using low power consumption mode:**

- **Switching to standby mode and interrupts**
- **Release of standby mode by an interrupt**
- **Release of stop mode by an external interrupt**
- **Oscillation stabilization wait time**

## ■ Switching to Standby Mode and Interrupts

While an interrupt request is output, the microcontroller does not enter standby mode even if the stop mode bit (STP) of the low power consumption mode control register (LPMCR) is set to "1", the sleep mode bit (SLP) is set to "1", or the timebase timer mode bit (TMD) is set to "0".

## ■ Release of Standby Mode by an Interrupt

Standby mode is released if an interrupt request with an interrupt level higher than 7 (Interrupt control register ICR: IL2, IL1, IL0 = "$000_B$" to "$110_B$") is output in sleep mode, timebase timer mode, or stop mode.

If the interrupt level setting bit (ICR: IL2, IL1, IL0) corresponding to an interrupt request has a priority higher than the interrupt level mask register (ILM) and the interrupt enable flag of the condition code register is enabled (CCR:I = 1), the interrupt is accepted and the interrupt processing routine is executed. Unless the interrupt is accepted, the processing starts again from the next instruction to the one that set standby mode.

**Note:**

Interrupt disable setting is required before the setting of standby mode unless an interrupt processing routine is executed immediately after standby mode is released.

## ■ Release of Stop Mode by an External interrupt

To release stop mode by an external interrupt, set the DTP/interrupt enable register (ENIR) and the request level setting register (ELVR) before the microcontroller enters stop mode.

Select one of the "H" level, "L" level, rising edge, and falling edge as an input cause.

## ■ Oscillation Stabilization Wait Time

### ❍ Source clock oscillation stabilization wait time

An oscillation stabilization wait time is required after stop mode is released because the source oscillation has been halted in stop mode. The oscillation stabilization wait time can be set in the oscillation stabilization wait time setting bits (WS1, WS0) of the clock selection register (CKSCR). On a return due to a reset, the registers are set to the initial value. The clock cycle is therefore fixed to $2^{17}$/HCLK.

### ❍ PLL clock oscillation stabilization wait time

A PLL clock oscillation stabilization wait time is required after the operating clock is changed from the main clock to the PLL clock because the PLL clock is halted while the CPU operates on the main clock. While waiting for PLL clock oscillation stabilization, the CPU operates on the main clock.

The PLL clock oscillation stabilization wait time is fixed at $2^{14}$/HCLK (HCLK: clock oscillation frequency).

# CHAPTER 6 INTERRUPTS

**This chapter explains the interrupts and extended intelligent I/O service (EI$^2$OS) in the MB90M405 series.**

# 6.1 Interrupts

**The MB90M405 series has the following interrupt functions and exception processing function:**

- **Hardware interrupts**
- **Software interrupts**
- **Interrupts from extended intelligent I/O service (EI$^2$OS)**
- **Exception processing**

## ■ Interrupt Types and Functions

### ❍ Hardware interrupt

A hardware interrupt transfers control to an interrupt processing program in response to an interrupt request from a peripheral function (resource). For more information, see Section 6.4 "Hardware Interrupts".

### ❍ Software interrupt

A software interrupt transfers control to an interrupt processing program if a software interrupt instruction (INT instruction) is executed on a program. For more information, see Section 6.5 "Software Interrupts".

### ❍ Interrupt from extended intelligent I/O service (EI$^2$OS)

The extended intelligent I/O service (EI$^2$OS) can transfer data between a register contained in a peripheral function (resource) and internal memory if settings are made in the interrupt control registers (ICR00 to ICR15) and the extended intelligent I/O service descriptor (ISD).

When the data transfers have been terminated, the interrupt processing program is executed. For more information, see Section 6.6 "Interrupt of Extended Intelligent I/O Service (EI$^2$OS)".

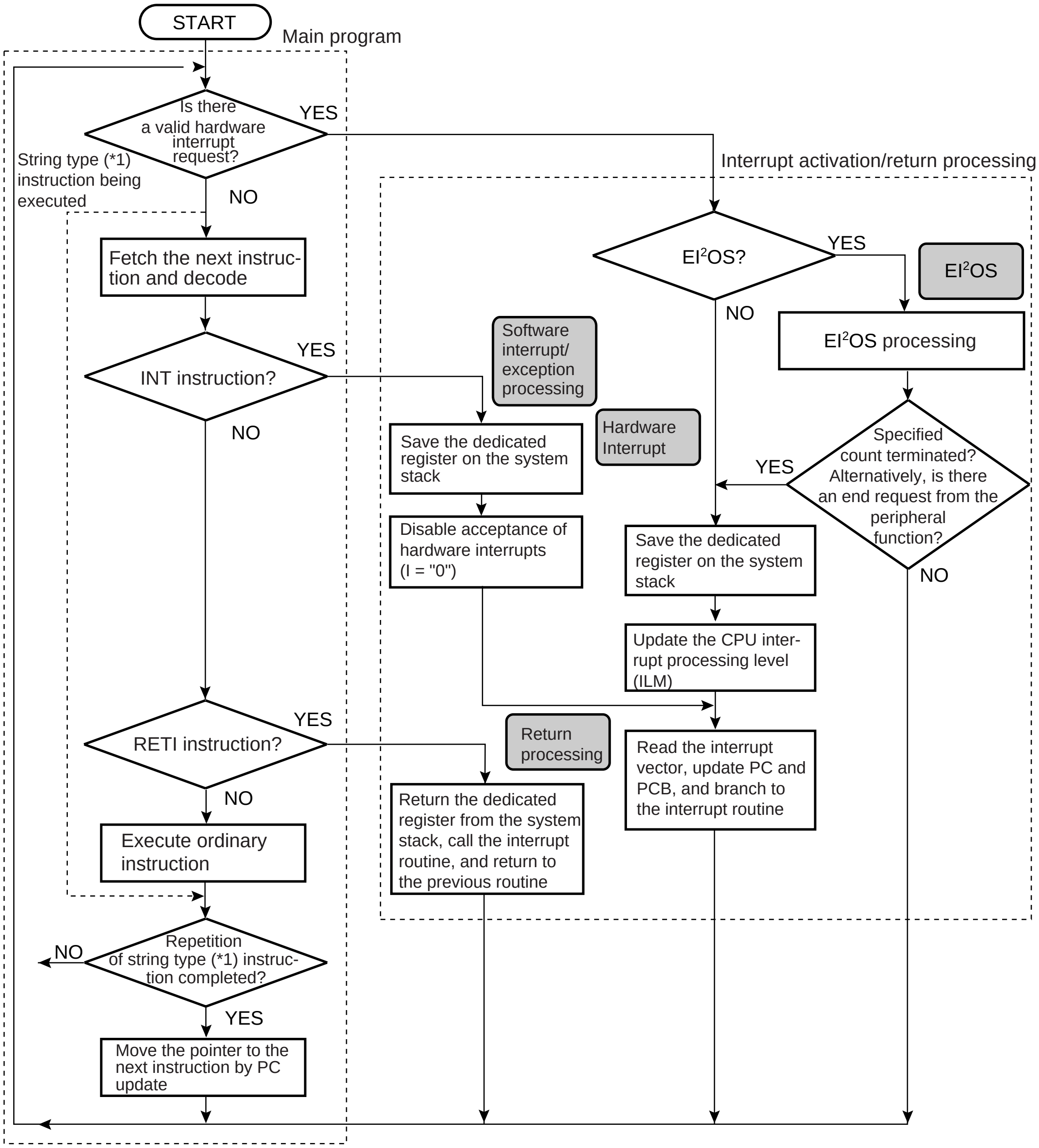
### ❍ Exception processing

Exception processing is performed if an undefined instruction code is executed.

If exception processing is performed, the register value currently processed is saved to the system stack and the processing branches to the exception processing routine. For more information, see Section 6.7 "Exception Processing Interrupt".

## ■ Interrupt Operation

**Figure 6.1-1 Overall Flow of Interrupt Operation**


START
Main program
Is there a valid hardware interrupt request?
YES
NO
String type (*1) instruction being executed
Interrupt activation/return processing
Fetch the next instruction and decode
INT instruction?
YES
NO
RETI instruction?
YES
NO
Execute ordinary instruction
Repetition of string type (*1) instruction completed?
NO
YES
Move the pointer to the next instruction by PC update
$EI^2OS$?
YES
NO
$EI^2OS$
$EI^2OS$ processing
Specified count terminated? Alternatively, is there an end request from the peripheral function?
YES
NO
Software interrupt/ exception processing
Hardware Interrupt
Save the dedicated register on the system stack
Disable acceptance of hardware interrupts (I = "0")
Save the dedicated register on the system stack
Update the CPU interrupt processing level (ILM)
Return processing
Read the interrupt vector, update PC and PCB, and branch to the interrupt routine
Return the dedicated register from the system stack, call the interrupt routine, and return to the previous routine

*1 When a string type instruction is being executed, the interrupt is evaluated in each step.

## 6.2 Interrupt Causes and Interrupt Vectors

**The MB90M405 series have functions for handling 256 types of interrupt cause. The 256 interrupt vector tables are allocated to the memory at the highest addresses. Software interrupts can use 256 interrupt instructions (INT0 to INT255). Note that INT8 is shared with a reset vector interrupt and that INT10 is shared with exception processing. INT11 to INT42 are shared with an interrupt from a peripheral function (resource).**

### ■ Interrupt Vectors

Interrupt vector tables referenced during interrupt processing are allocated to the highest addresses in the memory area (FFFC00$_H$ to FFFFFE$_H$). Interrupt vectors share the same area with EI$^2$OS, exception processing, hardware, and software interrupts.

Table 6.2-1 "Interrupt Vectors" shows the assignment of software interrupt instructions, interrupt numbers, and interrupt vectors.

**Table 6.2-1 Interrupt Vectors**

| Software interrupt instruction | Vector address L | Vector address M | Vector address H | Mode data | Interrupt No. | Hardware interrupt |
|---|---|---|---|---|---|---|
| INT0 | FFFFFC$_H$ | FFFFFD$_H$ | FFFFFE$_H$ | Not used | #0 | None |
| : | : | : | : | : | : | : |
| INT7 | FFFFE0$_H$ | FFFFE1$_H$ | FFFFE2$_H$ | Not used | #7 | None |
| INT8 | FFFFDC$_H$ | FFFFDD$_H$ | FFFFDE$_H$ | FFFFDF$_H$ | #8 | (RESET Vector) |
| INT9 | FFFFD8$_H$ | FFFFD9$_H$ | FFFFDA$_H$ | Not used | #9 | None |
| INT10 | FFFFD4$_H$ | FFFFD5$_H$ | FFFFD6$_H$ | Not used | #10 | <Exception processing> |
| INT11 | FFFFD0$_H$ | FFFFD1$_H$ | FFFFD2$_H$ | Not used | #11 | Hardware interrupt #0 |
| INT12 | FFFFCC$_H$ | FFFFCD$_H$ | FFFFCE$_H$ | Not used | #12 | Hardware interrupt #1 |
| INT13 | FFFFC8$_H$ | FFFFC9$_H$ | FFFFCA$_H$ | Not used | #13 | Hardware interrupt #2 |
| INT14 | FFFFC4$_H$ | FFFFC5$_H$ | FFFFC6$_H$ | Not used | #14 | Hardware interrupt #3 |
| : | : | : | : | : | : | : |
| INT254 | FFFC04$_H$ | FFFC05$_H$ | FFFC06$_H$ | Not used | #254 | None |
| INT255 | FFFC00$_H$ | FFFC01$_H$ | FFFC02$_H$ | Not used | #255 | None |

**Note:**

Interrupt vectors not defined during software design should be set at the exception processing address.

## ■ Interrupt Causes and Interrupt Vectors/Interrupt Control Registers

**Table 6.2-2 Interrupt Causes, Interrupt Vectors, and Interrupt Control Registers**

| Interrupt cause | EI$^2$OS support | Interrupt vector | | | Interrupt control register | | Priority |
|---|---|---|---|---|---|---|---|
| | | Number *1 | | Address | ICR | Address | |
| Reset | X | #08 | 08$_H$ | FFFFDC$_H$ | - | - | |
| INT9 instruction | X | #09 | 09$_H$ | FFFFD8$_H$ | - | - | High |
| Exception processing | X | #10 | 0A$_H$ | FFFFD4$_H$ | - | - | ↑ |
| DTP/external interrupt channel 0 | O | #11 | 0B$_H$ | FFFFD0$_H$ | ICR00 | 0000B0$_H$ | |
| DTP/external interrupt channel 1 | O | #13 | 0D$_H$ | FFFFC8$_H$ | ICR01 | 0000B1$_H$ | |
| Serial I/O channel 2 | △ | #15 | 0F$_H$ | FFFFC0$_H$ | ICR02 | 0000B2$_H$ | |
| DTP/external interrupt channels 2/3 | O | #16 | 10$_H$ | FFFFBC$_H$ | | | |
| Serial I/O channel 3 | △ | #17 | 11$_H$ | FFFFB8$_H$ | ICR03 | 0000B3$_H$ | |
| 16-bit free-running timer | △ | #18 | 12$_H$ | FFFFB4$_H$ | | | |
| Reserved | - | #20 | - | FFFFAC$_H$ | ICR04 | 0000B4$_H$ | |
| 16-bit reload timer channel 2 | △ | #21 | 15$_H$ | FFFFA8$_H$ | ICR05 | 0000B5$_H$ | |
| 16-bit reload timer channel 0 | △ | #23 | 17$_H$ | FFFFA0$_H$ | ICR06 | 0000B6$_H$ | |
| 16-bit reload timer channel 1 | △ | #24 | 18$_H$ | FFFF9C$_H$ | | | |
| Input capture channel 0 | △ | #25 | 19$_H$ | FFFF98$_H$ | ICR07 | 0000B7$_H$ | |
| Input capture channel 1 | △ | #26 | 1A$_H$ | FFFF94$_H$ | | | |
| Reserved | - | #27 | - | FFFF90$_H$ | ICR08 | 0000B8$_H$ | |
| Output compare match | X | #29 | 1D$_H$ | FFFF88$_H$ | ICR09 | 0000B9$_H$ | |
| Reserved | - | #31 | - | FFFF80$_H$ | ICR10 | 0000BA$_H$ | |
| Timebase timer | X | #33 | 21$_H$ | FFFF78$_H$ | ICR11 | 0000BB$_H$ | |
| Reserved | - | #34 | - | FFFF74$_H$ | | | |
| UART0 receive end | ◎ | #35 | 23$_H$ | FFFF70$_H$ | ICR12 | 0000BC$_H$ | |
| UART0 send end | △ | #36 | 24$_H$ | FFFF6C$_H$ | | | |
| End of A/D conversion | O | #37 | 25$_H$ | FFFF68$_H$ | ICR13 | 0000BD$_H$ | |
| I$^2$C interface | △ | #38 | 26$_H$ | FFFF64$_H$ | | | |
| UART1 receive end | ◎ | #39 | 27$_H$ | FFFF60$_H$ | ICR14 | 0000BE$_H$ | |
| UART1 send end | △ | #40 | 28$_H$ | FFFF5C$_H$ | | | ↓ |
| Flash memory status | X | #41 | 29$_H$ | FFFF58$_H$ | ICR15 | 0000BF$_H$ | Low |
| Delayed interrupt generator module | X | #42 | 2A$_H$ | FFFF54$_H$ | | | |

O: Can be used.

X: Cannot be used.

◎ : Usable if used with the EI$^2$OS stop function.

△ : Usable when an interrupt cause that shares the ICR is not used.

*1: If multiple interrupts of the same level are output simultaneously, an interrupt cause with a smaller interrupt vector number takes precedence.

## 6.3 Interrupt Control Registers and Peripheral Functions

**The interrupt control registers ICR00 to ICR15 correspond to all peripheral functions that have the interrupt function. These registers control interrupts and the extended intelligent I/O service (EI$^2$OS).**

### ■ Interrupt Control Registers

**Table 6.3-1 Interrupt Control Registers**

| Address | Register | Abbreviation | Corresponding peripheral function (Resource) |
|---|---|---|---|
| $0000B0_H$ | Interrupt control register 00 | ICR00 | DTP/external interrupt channel 0 |
| $0000B1_H$ | Interrupt control register 01 | ICR01 | DTP/external interrupt channel 1 |
| $0000B2_H$ | Interrupt control register 02 | ICR02 | Serial I/O channel 2<br>DTP/external interrupt channels 2/3 |
| $0000B3_H$ | Interrupt control register 03 | ICR03 | Serial I/O channel 3<br>16-bit free-running timer |
| $0000B4_H$ | Interrupt control register 04 | ICR04 | Reserved |
| $0000B5_H$ | Interrupt control register 05 | ICR05 | 16-bit reload timer channel 2 |
| $0000B6_H$ | Interrupt control register 06 | ICR06 | 16-bit reload timer channels 0/1 |
| $0000B7_H$ | Interrupt control register 07 | ICR07 | Input capture channels 0/1 |
| $0000B8_H$ | Interrupt control register 08 | ICR08 | Reserved |
| $0000B9_H$ | Interrupt control register 09 | ICR09 | Output compare |
| $0000BA_H$ | Interrupt control register 10 | ICR10 | Reserved |
| $0000BB_H$ | Interrupt control register 11 | ICR11 | Timebase timer |
| $0000BC_H$ | Interrupt control register 12 | ICR12 | UART0 receive end<br>UART0 send end |
| $0000BD_H$ | Interrupt control register 13 | ICR13 | A/D converter<br>I$^2$C bus interface |
| $0000BE_H$ | Interrupt control register 14 | ICR14 | UART1 receive end<br>UART1 send end |
| $0000BF_H$ | Interrupt control register 15 | ICR15 | Flash memory status<br>Delayed interrupt generator module |

The following four settings can be made in an interrupt control register (ICR).

- Set the interrupt level of the corresponding peripheral function (resource).
- Set an interrupt of the peripheral function (resource) either as an interrupt or as the extended intelligent I/O service.
- Set the descriptor address of the extended intelligent I/O service (EI$^2$OS).

- Display the status of the extended intelligent I/O service (EI$^2$OS)

Interrupt control registers (ICRs) have different functions during the writing and reading of data.

**Note:**

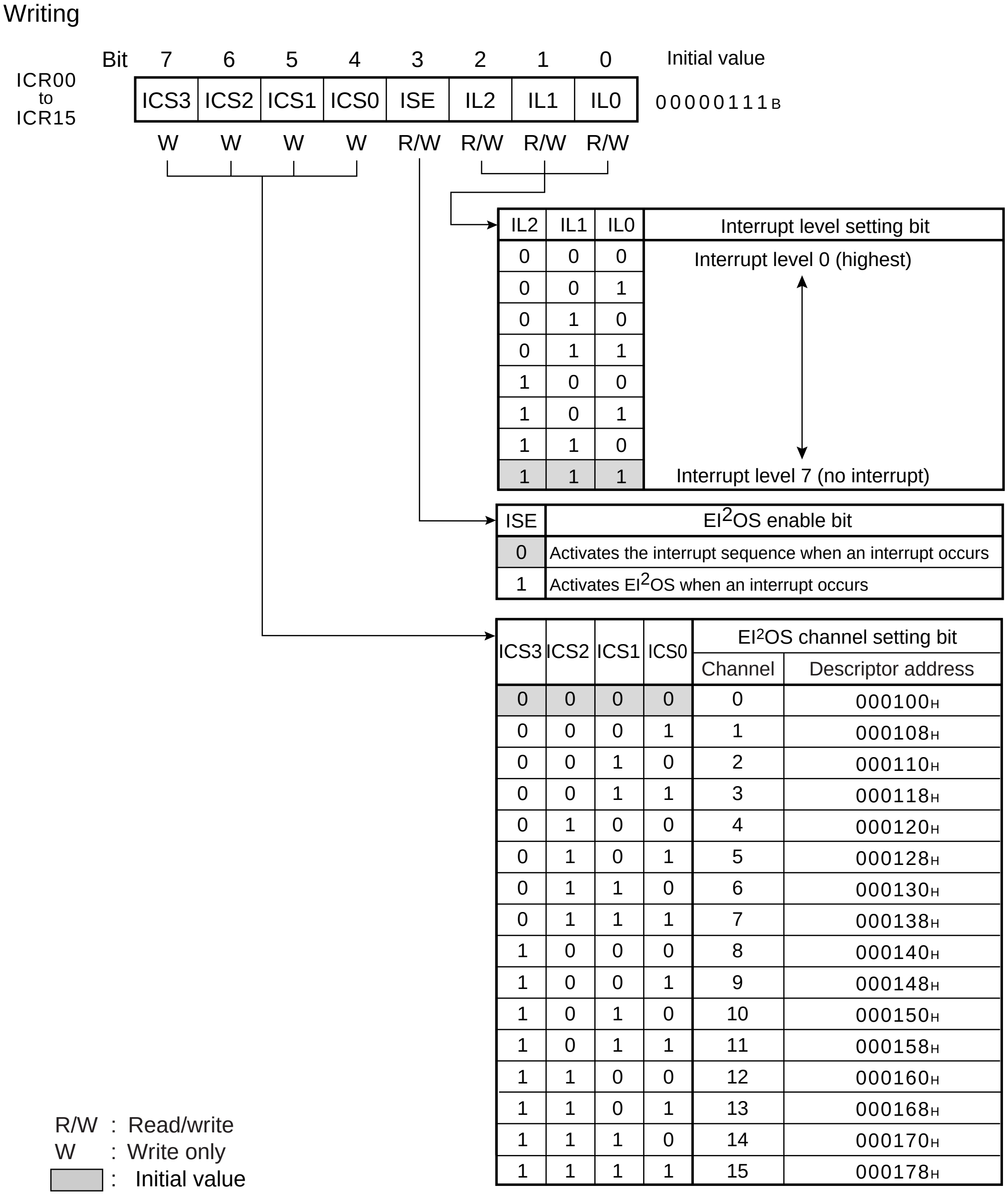
When interrupt control registers (ICRs) are set, a read-modify-write instruction of SETB and CLRB cannot be used to access it.

## 6.3.1 Interrupt Control Registers (ICR00 to ICR15)

**Interrupt control registers can determine the interrupt processing or the extended intelligent I/O service processing when an interrupt request is output. Interrupt control registers (ICRs) have different bit functions during the writing and reading of data.**

### ■ Interrupt Control Registers (ICR00 to ICR15)

Figure 6.3-1 Interrupt Control Registers (ICR00 to ICR15) during Writing



| IL2 | IL1 | IL0 | Interrupt level setting bit |
|---|---|---|---|
| 0 | 0 | 0 | Interrupt level 0 (highest) |
| 0 | 0 | 1 | |
| 0 | 1 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 0 | |
| 1 | 0 | 1 | |
| 1 | 1 | 0 | |
| 1 | 1 | 1 | Interrupt level 7 (no interrupt) |

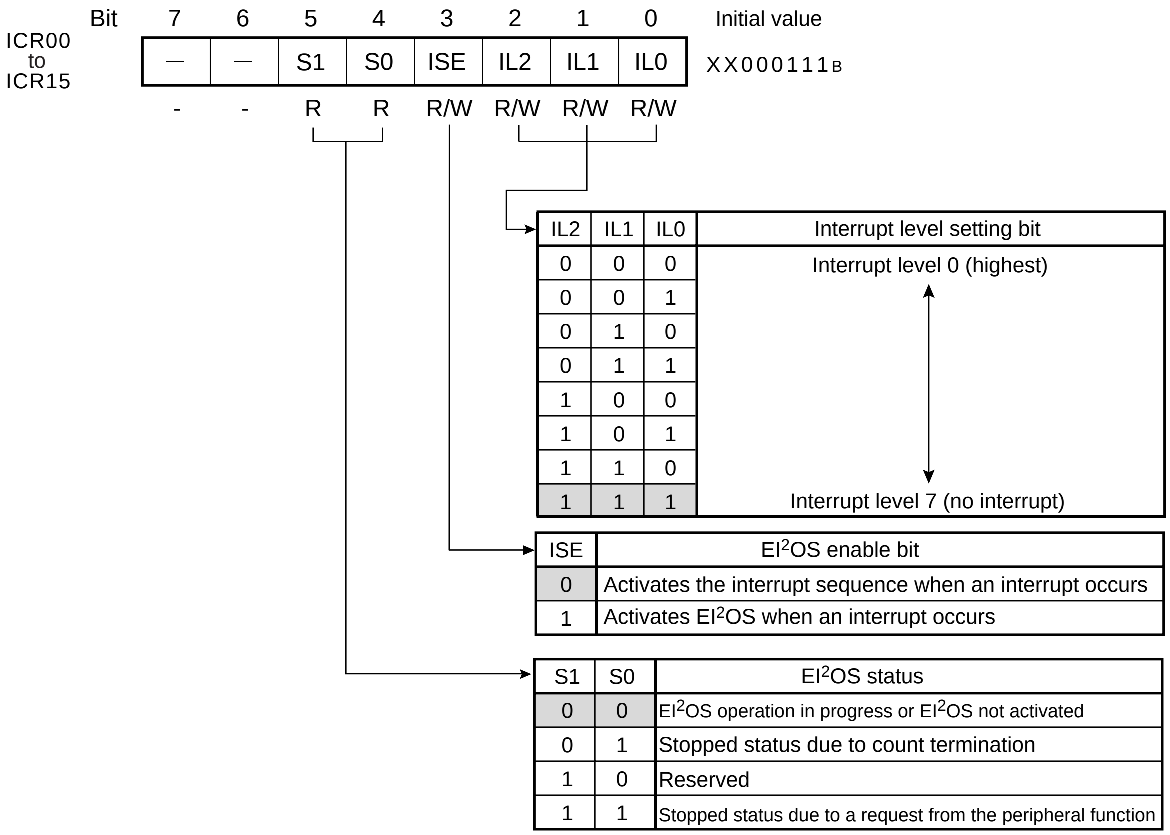
| ISE | EI²OS enable bit |
|---|---|
| 0 | Activates the interrupt sequence when an interrupt occurs |
| 1 | Activates EI²OS when an interrupt occurs |

| ICS3 | ICS2 | ICS1 | ICS0 | EI²OS channel setting bit | |
|---|---|---|---|---|---|
| | | | | Channel | Descriptor address |
| 0 | 0 | 0 | 0 | 0 | 000100H |
| 0 | 0 | 0 | 1 | 1 | 000108H |
| 0 | 0 | 1 | 0 | 2 | 000110H |
| 0 | 0 | 1 | 1 | 3 | 000118H |
| 0 | 1 | 0 | 0 | 4 | 000120H |
| 0 | 1 | 0 | 1 | 5 | 000128H |
| 0 | 1 | 1 | 0 | 6 | 000130H |
| 0 | 1 | 1 | 1 | 7 | 000138H |
| 1 | 0 | 0 | 0 | 8 | 000140H |
| 1 | 0 | 0 | 1 | 9 | 000148H |
| 1 | 0 | 1 | 0 | 10 | 000150H |
| 1 | 0 | 1 | 1 | 11 | 000158H |
| 1 | 1 | 0 | 0 | 12 | 000160H |
| 1 | 1 | 0 | 1 | 13 | 000168H |
| 1 | 1 | 1 | 0 | 14 | 000170H |
| 1 | 1 | 1 | 1 | 15 | 000178H |

R/W : Read/write
W : Write only
▨ : Initial value

## Figure 6.3-2 Interrupt Control Registers (ICR00 to ICR15) during Reading

Reading



| | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|---|
| ICR00<br>to<br>ICR15 | | — | — | S1 | S0 | ISE | IL2 | IL1 | IL0 | XX000111B |
| | | - | - | R | R | R/W | R/W | R/W | R/W | |

| IL2 | IL1 | IL0 | Interrupt level setting bit |
|---|---|---|---|
| 0 | 0 | 0 | Interrupt level 0 (highest) |
| 0 | 0 | 1 | |
| 0 | 1 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 0 | |
| 1 | 0 | 1 | |
| 1 | 1 | 0 | |
| 1 | 1 | 1 | Interrupt level 7 (no interrupt) |

| ISE | EI$^2$OS enable bit |
|---|---|
| 0 | Activates the interrupt sequence when an interrupt occurs |
| 1 | Activates EI$^2$OS when an interrupt occurs |

| S1 | S0 | EI$^2$OS status |
|---|---|---|
| 0 | 0 | EI$^2$OS operation in progress or EI$^2$OS not activated |
| 0 | 1 | Stopped status due to count termination |
| 1 | 0 | Reserved |
| 1 | 1 | Stopped status due to a request from the peripheral function |

| | | |
|---|---|---|
| R/W | : | Read/write |
| R | : | Read only |
| - | : | Undefined bit |
| X | : | Undefined |
| ▨ | : | Initial value |

## 6.3.2 Interrupt Control Register Functions

**The interrupt control registers (ICR00 to ICR15) can specify the following settings:**

- **Interrupt level setting**
- **Extended intelligent I/O service (EI$^2$OS) enable setting**
- **Extended intelligent I/O service (EI$^2$OS) descriptor address setting**
- **Extended intelligent I/O service (EI$^2$OS) operation status display**

### ■ Configuration of Interrupt Control Registers (ICR)

**Figure 6.3-3 Configuration of Interrupt Control Registers (ICR)**

Writing to interrupt control register (ICR)

|  | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|---|
| ICR00<br>to<br>ICR15 |  | ICS3 | ICS2 | ICS1 | ICS0 | ISE | IL2 | IL1 | IL0 | 00000111B |

Reading of interrupt control register (ICR)

|  | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|---|
| ICR00<br>to<br>ICR15 |  | — | — | S1 | S0 | ISE | IL2 | IL1 | IL0 | XX000111B |

- : Undefined bit

**Reference:**

Set the EI$^2$OS descriptor address setting bits (ICS3 to ICS0) to start the extended intelligent I/O service (EI$^2$OS). Set the EI$^2$OS enable bit (ISE) to "1" to activate it. Alternatively, set the ISE to "0" to refrain from activating it. If EI$^2$OS is not activated, the ICS3 to ICS0 bits need not be set.

## ■ Interrupt Control Register Functions

### ❍ Interrupt level setting bits (IL2 to IL0)

These bits can set the interrupt level of the corresponding peripheral function (resource). A reset initializes these bits to level 7 (no interrupts). (No interrupts can be generated at level 7.)

**Table 6.3-2 Correspondence between the Interrupt Level Setting Bits and Interrupt Levels**

| IL2 | IL1 | IL0 | Interrupt level |
|---|---|---|---|
| 0 | 0 | 0 | 0 (highest priority)<br>↕<br>6 (lowest priority) |
| 0 | 0 | 1 | |
| 0 | 1 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 0 | |
| 1 | 0 | 1 | |
| 1 | 1 | 0 | |
| 1 | 1 | 1 | 7 (no interrupts) |

### ❍ Extended intelligent I/O service (EI$^2$OS) enable bit (ISE)

When an interrupt request is output, EI$^2$OS is started if the EI$^2$OS enable bit (ISE) is set to "1". Alternatively, an interrupt sequence is started if the EI$^2$OS enable bit (ISE) is set to "0". When EI$^2$OS processing is completed, the ISE bit is reset to "0". If a peripheral function (resource) has no EI$^2$OS function, set the ISE bit to "0" using software. A reset initializes the ISE bit to "0".

### ❍ Extended intelligent I/O service (EI$^2$OS) channel setting bits (ICS3 to ICS1)

The EI$^2$OS descriptor address setting bits (ICS3 to ICS1) are valid when a descriptor is set. Set the EI$^2$OS descriptor address in these bits. Set values in the EI$^2$OS descriptor address setting bits to set the EI$^2$OS descriptor address. A reset initializes the ICS3 to ICS0 bits to 0000$_B$.

**Table 6.3-3 Correspondence between EI$^2$OS Channel Setting Bits and Descriptor Addresses**

| ICS3 | ICS2 | ICS1 | ICS0 | Selected channel | Descriptor address |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | $000100_H$ |
| 0 | 0 | 0 | 1 | 1 | $000108_H$ |
| 0 | 0 | 1 | 0 | 2 | $000110_H$ |
| 0 | 0 | 1 | 1 | 3 | $000118_H$ |
| 0 | 1 | 0 | 0 | 4 | $000120_H$ |
| 0 | 1 | 0 | 1 | 5 | $000128_H$ |
| 0 | 1 | 1 | 0 | 6 | $000130_H$ |
| 0 | 1 | 1 | 1 | 7 | $000138_H$ |
| 1 | 0 | 0 | 0 | 8 | $000140_H$ |
| 1 | 0 | 0 | 1 | 9 | $000148_H$ |
| 1 | 0 | 1 | 0 | 10 | $000150_H$ |
| 1 | 0 | 1 | 1 | 11 | $000158_H$ |
| 1 | 1 | 0 | 0 | 12 | $000160_H$ |
| 1 | 1 | 0 | 1 | 13 | $000168_H$ |
| 1 | 1 | 1 | 0 | 14 | $000170_H$ |
| 1 | 1 | 1 | 1 | 15 | $000178_H$ |

### ❍ Extended intelligent I/O service (EI$^2$OS) status bits (S1 and S0)

The EI$^2$OS status bits (S1 and S0) are valid during reading. Read the S1 and S0 bits while EI$^2$OS is activated to determine whether EI$^2$OS is running or terminated. A reset initializes the S1 and S0 bits to "$00_B$".

**Table 6.3-4 Relationship between EI$^2$OS Status Bits and the EI$^2$OS Status**

| S1 | S0 | EI$^2$OS status |
|---|---|---|
| 0 | 0 | EI$^2$OS operation in progress or EI$^2$OS not activated |
| 0 | 1 | Stopped status due to count termination |
| 1 | 0 | Reserved |
| 1 | 1 | Stopped status due to a request from the peripheral function (resource) |

# 6.4 Hardware Interrupts

**A hardware interrupt operates as follows: an interrupt request that is output by a peripheral function (resource) temporarily interrupts a program being executed by the CPU and transfers control to a user-defined interrupt processing program. The extended intelligent I/O service (EI$^2$OS) is also handled as a hardware interrupt.**

## ■ Hardware Interrupts

### ❍ Hardware interrupt function

The hardware interrupt function determines whether an interrupt can be accepted. To do so, it compares the interrupt level of an interrupt request that is output by a peripheral function (resource) with the interrupt level mask register (PS: ILM) while referring to the contents of the I flag (PS: I).

If a hardware interrupt is accepted, the contents of the direct page register (DPR), accumulator (A), program counter (PC), processor status register (PS), and bank registers (ADB, DTB, and PCB) are saved to the system stack. An interrupt level stored in the ICR register is then stored in the interrupt level mask register (ILM). Finally, the processing branches to the interrupt vector and the interrupt processing program is executed.

### ❍ Multiple interrupts

A hardware interrupt can be activated while the interrupt processing program is being executed.

### ❍ Extended intelligent I/O service (EI$^2$OS)

EI$^2$OS is a data transfer function between memory and I/O registers. When the transfer of data to the extended intelligent I/O service descriptor is completed, a hardware interrupt is activated. EI$^2$OS cannot be activated in duplicate. While EI$^2$OS is being processed, no interrupt request or EI$^2$OS request is accepted. When the processing of EI$^2$OS is completed, interrupt request or EI$^2$OS request is accepted.

### ❍ External interrupt

An external interrupt is accepted as a hardware interrupt if a circuit that can output an interrupt request from an external terminal (DTP/external interrupt circuit) detects an interrupt request.

### ❍ Interrupt vector

Interrupt vector tables referenced during interrupt processing are allocated to memory at FFFC00$_H$ to FFFFFF$_H$.

**Reference:**

See Section 6.2 "Interrupt causes and Interrupt Vectors", for more information about the allocation of interrupt numbers and interrupt vectors.

## ■ Hardware Interrupt Structure

The mechanisms shown in Table 6.4-1 "Mechanisms Used for Hardware Interrupts" are used for hardware interrupts. These mechanisms must be configured in a user program before hardware interrupts can be used.

**Table 6.4-1 Mechanisms Used for Hardware Interrupts**

| | Mechanism | Function |
|---|---|---|
| Peripheral function (resource) | Interrupt enable bit, interrupt request bit | Controls interrupt requests from a peripheral function (resource) |
| Interrupt controller | Interrupt control register (ICR) | Sets the interrupt level and controls $EI^2OS$ |
| CPU | Interrupt enable flag (I) | Identifies the interrupt enable status |
| | Interrupt level mask register (ILM) | Compares the request interrupt level and current interrupt level |
| | Microcode | Executes the interrupt processing routine |
| $FFFC00_H$ to $FFFFFF_H$ in memory | Interrupt vector table | Stores the branch destination address for interrupt processing |

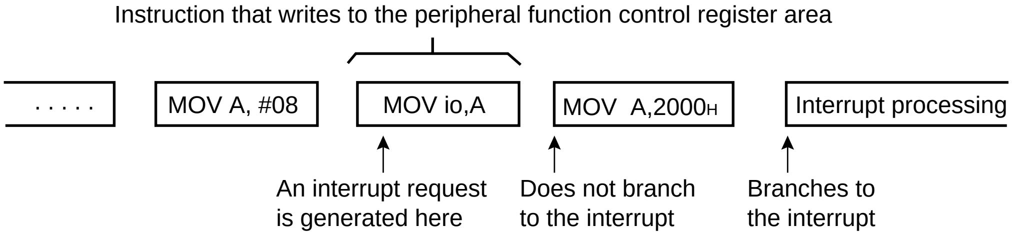
## ■ Hardware Interrupt Disable

Acceptance of a hardware interrupt request is disabled under the following conditions:

### ❍ Hardware interrupt acceptance disable during writing to the peripheral function (resource) control register

No hardware interrupt request is accepted while data is written to the peripheral function (resource) control register.

**Figure 6.4-1 Hardware Interrupt Request While Writing to the Peripheral Function Control Register Area**


Instruction that writes to the peripheral function control register area
. . . . .
MOV A, #08
MOV io,A
MOV A,2000H
Interrupt processing
An interrupt request is generated here
Does not branch to the interrupt
Branches to the interrupt

### ❍ Hardware interrupt acceptance disable by interrupt suppression instructions

The hardware interrupt suppression instructions listed in Table 6.4-2 "Hardware Interrupt Suppression Instruction" ignore interrupt requests without detecting whether a hardware interrupt request exists.

If a hardware interrupt request is output while a hardware interrupt suppression instruction is being executed, the interrupt is accepted and processed after completion of the hardware interrupt suppression instruction and the subsequent execution of an instruction other than the hardware interrupt suppression instruction.

**Table 6.4-2 Hardware Interrupt Suppression Instruction**

| | Prefix code | Interrupts/hold suppression instructions (instructions that delay the effect of the prefix code) |
|---|---|---|
| Instructions that do not accept interrupts and hold requests | PCB<br>DTB<br>ADB<br>SPB<br>CMR<br>NCC | MOV ILM, #imm8<br>OR CCR, #imm8<br>AND CCR, #imm8<br>POPW PS |

### ❍ Hardware interrupt acceptance disable during execution of a software interrupt

When a software interrupt is activated, the I flag is cleared to 0. In this state, other interrupt requests cannot be accepted.

## 6.4.1 Operation of Hardware Interrupts

**This section explains hardware interrupt operation from generation of a interrupt request to the completion of interrupt processing.**

### ■ Hardware Interrupt Activation

❍ **Peripheral function (resource) operation (generation of an interrupt request)**

A peripheral function (resource) that has a hardware interrupt request function has an "interrupt request flag bit" and an "interrupt enable flag" in the corresponding peripheral function (resource) control registers. The interrupt request flag bit indicates the presence of an interrupt request. The interrupt enable flag determines whether a CPU interrupt request is enabled or disabled. When an interrupt cause defined in a peripheral function is detected, an interrupt request is output to an interrupt controller as long as the interrupt request flag bit is set to "1" and the interrupt enable bit is set to enable an interrupt request to the CPU.

❍ **Interrupt controller operation (interrupt request control)**

The interrupt controller compares the interrupt levels (ILs) to accept the request having the highest level. If multiple interrupts of the same level are output simultaneously, an interrupt request with the smallest number takes precedence (see Table 6.2-1 "Interrupt Vectors").

❍ **CPU operation (interrupt request acceptance and interrupt processing)**

The CPU compares the received interrupt level (ICR: IL2 to IL0) and the interrupt level mask register (ILM). If IL2 to IL0 are greater than ILM and interrupts are enabled (PS: CCR: I = "1"), the CPU terminates the instruction being executed and performs the interrupt processing. If the EI$^2$OS enable bit (ISE) of the interrupt control register (ICR) is set to "0", the CPU performs the interrupt processing. If the ISE is set to "1", the CPU starts EI$^2$OS and then performs the interrupt processing.

Interrupt processing saves the contents of the dedicated registers (12 bytes including A, DPR, ADB, DTB, PCB, PC, and PS) on the system stack (the system stack space indicated by the SSB and SSP).

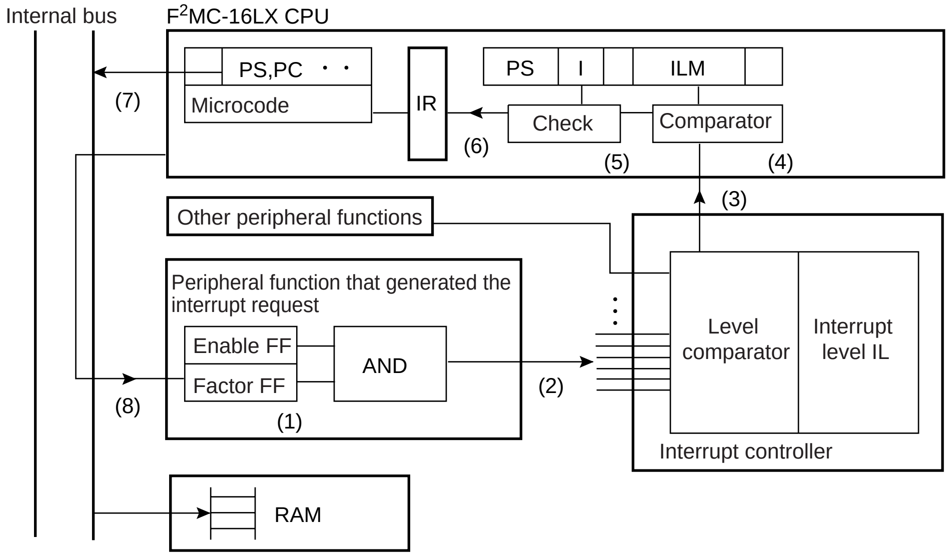
The CPU then loads data into the interrupt vector program counters (PCB and PC), updates the ILM, and sets the stack flag (S) (sets CCR: S = 1 and activates the system stack).

### ■ Returning from a Hardware Interrupt

If an interrupt processing program writes "0" to the interrupt request flag bit of a peripheral function (resource) that output an interrupt cause and the RETI instruction is executed, data saved on the system stack is restored to the dedicated registers and the program processing that was executed before branching due to an interrupt is resumed.

## ■ Hardware Interrupt Operation

**Figure 6.4-2 Hardware Interrupt Operation**



IL : Interrupt level setting bit in the interrupt control register (ICR)
PS : Processor status
I : Interrupt enable flag
ILM : Interrupt level mask register
IR : Instruction register

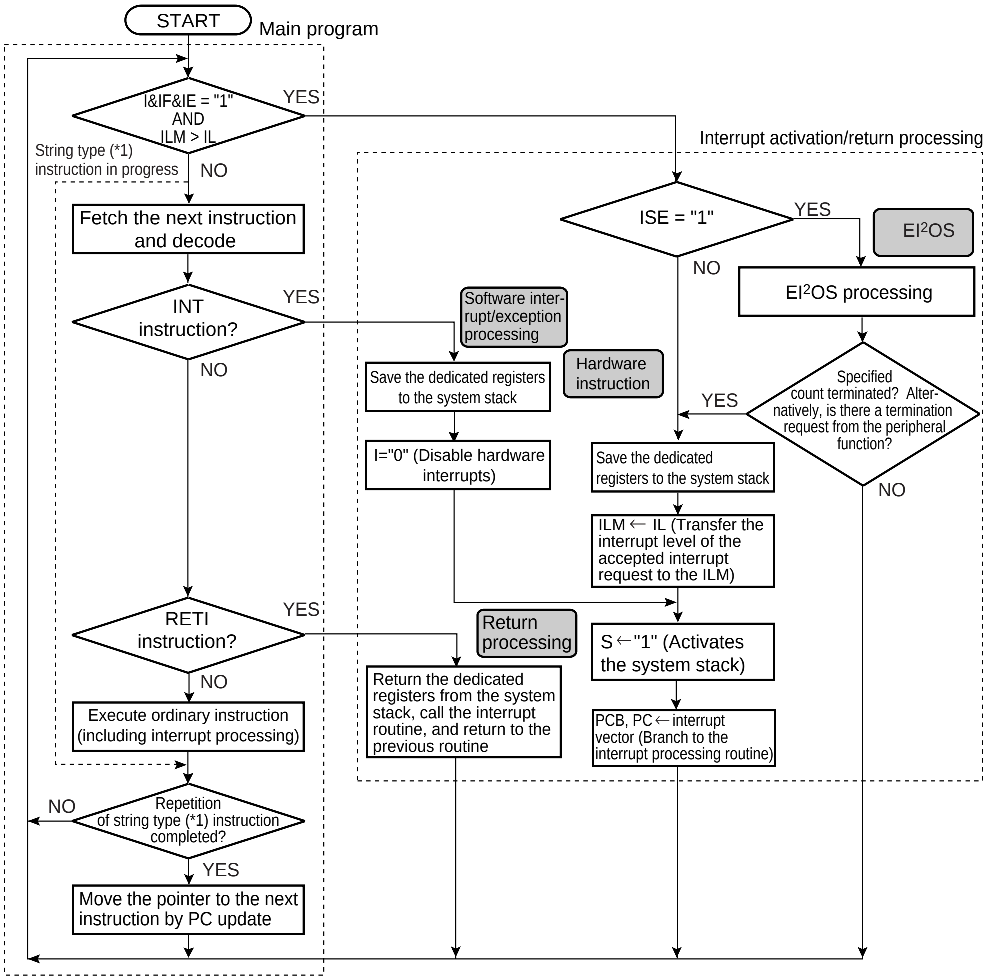
1. An interrupt cause is output within the peripheral functions (resources).
2. If the interrupt enable bit in the peripheral functions (resources) is set to enable interrupts, interrupt requests are output from the peripheral functions (resources) to the interrupt controller.
3. The interrupt controller that receives interrupt requests from the peripheral functions (resources) checks the priorities of interrupt requests simultaneously received and transfers the interrupt level (IL) of an interrupt request with the highest priority to the CPU.
4. The CPU compares the interrupt level (IL) requested by the interrupt controller with the interrupt level mask register (ILM).
5. If the comparison indicates a higher priority than the current interrupt processing level, the CPU checks the contents of the I flag in the condition code register (CCR).
6. If, in the check, the I flag in the CCR is found to be set to Enabled (CCR: I = "1"), the CPU waits until the execution of an instruction being executed is terminated. When it is terminated, the CPU sets the requested level (IL2 to IL0) in the ILM.
7. The values in the dedicated registers are saved to the system stack. The processing branches to the interrupt processing routine.
8. If a program in the interrupt processing routine sets the interrupt request flag bit of a peripheral function (resource) to "0" and the RETI instruction is executed, data saved on the system stack is restored to the dedicated registers and the interrupt processing is terminated.

## 6.4.2 Processing for Interrupt Operation

**When a peripheral function (resource) outputs an interrupt request and the CPU accepts it, the interrupt processing is performed after the instruction currently being executed is terminated. If the EI$^2$OS enable bit (ISE) of the interrupt control register (ICR) is set to "0", the CPU performs the interrupt processing. If the ISE is set to "1", the CPU activates the extended intelligent I/O service (EI$^2$OS). If a software interrupt is output by the INT instruction, the instruction currently being executed is suspended, the interrupt processing routine is performed, and hardware interrupts are disabled.**

### ■ Processing for Interrupt Operation

**Figure 6.4-3 Flow of Interrupt Processing**


START
Main program
I&IF&IE = "1" AND ILM > IL
YES
NO
String type (*1) instruction in progress
Interrupt activation/return processing
Fetch the next instruction and decode
ISE = "1"
YES
NO
EI2OS
EI2OS processing
INT instruction?
YES
NO
Software inter-rupt/exception processing
Hardware instruction
Save the dedicated registers to the system stack
I="0" (Disable hardware interrupts)
YES
Specified count terminated? Alter-natively, is there a termination request from the peripheral function?
NO
Save the dedicated registers to the system stack
ILM ← IL (Transfer the interrupt level of the accepted interrupt request to the ILM)
RETI instruction?
YES
NO
Return processing
S ← "1" (Activates the system stack)
Return the dedicated registers from the system stack, call the interrupt routine, and return to the previous routine
PCB, PC ← interrupt vector (Branch to the interrupt processing routine)
Execute ordinary instruction (including interrupt processing)
NO
Repetition of string type (*1) instruction completed?
YES
Move the pointer to the next instruction by PC update

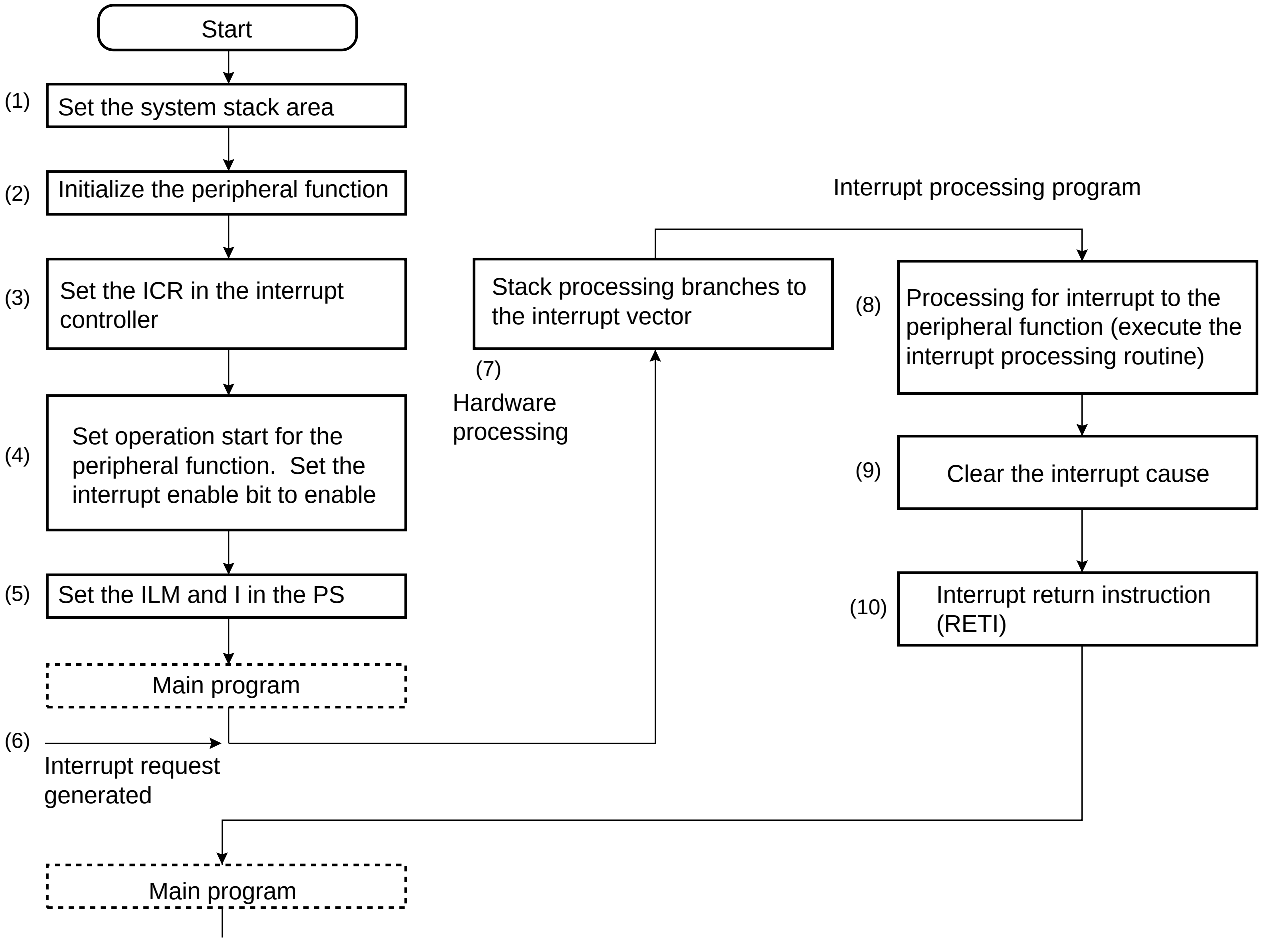
*1 When a string type instruction is being executed, the interrupt is evaluated in each step.
I : Interrupt enable flag of the condition code register (CCR)
IF : Interrupt request flag of the peripheral function
IE : Interrupt enable flag of the peripheral function
ILM : Interrupt level mask register (in the PS)
ISE : EI$^2$OS enable flag of the interrupt control register (ICR)
IL : Interrupt level setting bit of the interrupt control register (ICR)
S : Stack flag of the condition code register (CCR)
PCB : Program bank register
PC : Program counter:

# 6.4.3 Procedure for Using Hardware Interrupts

**Before hardware interrupts can be used, the system stack area, peripheral function (resource), and interrupt control register (ICR) must be set.**

## ■ Procedure for Using Hardware Interrupts

**Figure 6.4-4 Procedure for Using Hardware Interrupts**


Start
(1)
Set the system stack area
(2)
Initialize the peripheral function
(3)
Set the ICR in the interrupt controller
(4)
Set operation start for the peripheral function. Set the interrupt enable bit to enable
(5)
Set the ILM and I in the PS
Main program
(6)
Interrupt request generated
Main program
Stack processing branches to the interrupt vector
(7)
Hardware processing
Interrupt processing program
(8)
Processing for interrupt to the peripheral function (execute the interrupt processing routine)
(9)
Clear the interrupt cause
(10)
Interrupt return instruction (RETI)

1. Set the system stack area.
2. Set the operation of a peripheral function (resource).
3. Set the interrupt control register (ICR).
4. Set the interrupt enable bit of the peripheral function (resource) to enable the output of interrupt requests.
5. Set the interrupt level mask register (ILM) and interrupt enable flag (I) to interrupt acceptable.
6. If an interrupt request of the peripheral function (resource) is detected, a hardware interrupt request is output.
7. The interrupt processing hardware saves the dedicated register values to the system stack. The processing then branches to the interrupt processing program.

8. The interrupt processing program processes the peripheral function (resource) in response to the generated interrupt.
9. Clear the peripheral function (resource) interrupt request.
10. Execute the interrupt return instruction (RETI instruction), and return to the program before branching.

## 6.4.4 Multiple Interrupts

**Multiple hardware interrupts can be implemented in response to multiple interrupt requests from peripheral functions (resources). However, multiple extended intelligent I/O services cannot be activated.**

### ■ Multiple Interrupts

**❍ Operation of multiple interrupts**

If an interrupt request with an interrupt level that is higher than the one being executed is output, the current interrupt processing is suspended and the higher-priority interrupt request is executed. When the processing of the higher-priority interrupt ends, the previous interrupt processing resumes.

If, during execution of interrupt processing, an interrupt request with a level equal to or lower than the current interrupt processing is output, the new interrupt request is suspended until the current interrupt processing ends, unless the I flag of the condition code register (ICR) or the interrupt level mask register (ILM) is changed. When the current interrupt processing ends, the suspended interrupt request is executed.

Other multiple interrupts to be activated during an interrupt can be temporarily disabled by setting the I flag in the condition code register (CCR) in the interrupt processing routine to interrupts not allowed (CCR: I = 0) or the interrupt level mask register (ILM) to interrupts not allowed (ILM = $000_B$).

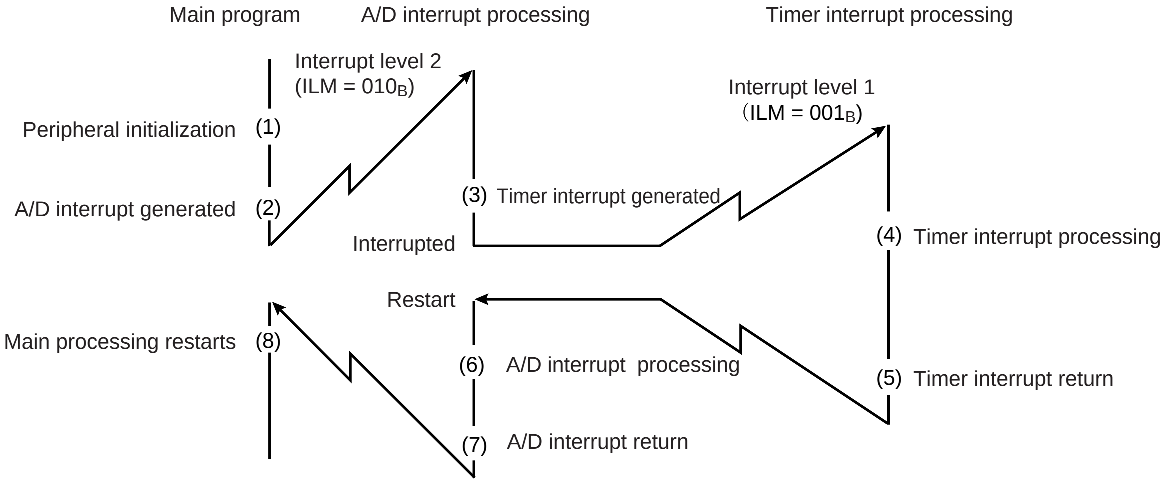
**Note:**

- 0 to 7 can be set as the interrupt level. If level 7 is set, the CPU does not accept interrupt requests.
- The extended intelligent I/O service ($EI^2OS$) cannot be used for the activation of multiple interrupts. During processing of the extended intelligent I/O service ($EI^2OS$), all other interrupt requests and extended intelligent I/O service requests are held.

### ❍ Example of multiple interrupts

This example of multiple interrupt processing assumes that a timer interrupt is given a priority that is higher than an A/D converter interrupt. In this example, the A/D converter interrupt level is set to 2, and the timer interrupt level is set to 1. If a timer interrupt is generated during processing of the A/D converter interrupt, the processing shown in Figure 6.4-5 "Example of Multiple Interrupt" is performed.

**Figure 6.4-5 Example of Multiple Interrupt**


Main program
A/D interrupt processing
Timer interrupt processing
Interrupt level 2
(ILM = $010_B$)
Interrupt level 1
(ILM = $001_B$)
Peripheral initialization (1)
A/D interrupt generated (2)
(3) Timer interrupt generated
Interrupted
(4) Timer interrupt processing
Restart
Main processing restarts (8)
(6) A/D interrupt processing
(5) Timer interrupt return
(7) A/D interrupt return

- At the start of A/D converter interrupt processing, the value in the interrupt level mask register (ILM) indicates the A/D converter interrupt level (ICR:IL2 to IL0) (2 in this example). If an interrupt request with level 1 or 0 is generated, the interrupt processing for level 1 or 0 takes precedence.
- When the interrupt processing ends and the return instruction (RETI) is executed, the values of the dedicated registers (A, DPR, ADB, DTB, PCB, PC, and PS) saved to the stack are restored and the interrupt mask register (ILM) is set back to the value that it had before the interruption.

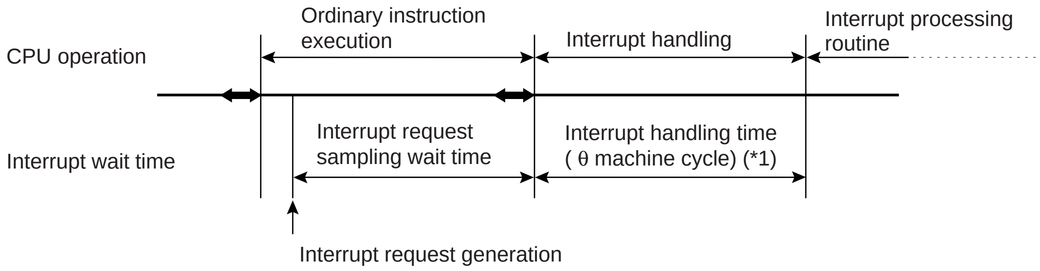
# 6.4.5 Hardware Interrupt Processing Time

**The time for completion of the currently executed instruction and the interrupt handling time apply from the output of a hardware interrupt request until the interrupt processing routine is executed.**

## ■ Hardware Interrupt Processing Time

The interrupt request sampling wait time and the interrupt handling time (time required to prepare for interrupt processing) are required from the output of a hardware interrupt request until the interrupt processing routine is executed.

**Figure 6.4-6 Interrupt Processing Time**


Ordinary instruction execution
Interrupt handling
Interrupt processing routine
CPU operation
Interrupt request sampling wait time
Interrupt handling time ( θ machine cycle) (*1)
Interrupt wait time
Interrupt request generation

⟷: The final instruction cycle samples the interrupt request here.
* : One machine cycle corresponds to one machine clock ( ϕ ).

### ❍ Interrupt request sampling wait time

The interrupt request sampling wait time refers to the time required from the output of an interrupt request by a peripheral function (resource) until the instruction being executed terminates. The CPU checks through sampling whether an interrupt request is output in the final cycle of an instruction being executed. The interrupt request sampling wait time is required because no interrupt request can be recognized while an instruction is being executed.

**Reference:**

The maximum interrupt request sampling wait time applies when an interrupt request is output immediately after execution of the POPW RW0, ... RW7 instruction (45 machine cycles) with the longest execution cycle is started.

### ❍ Interrupt handling time ($\phi$ machine cycle)

After accepting an interrupt request, the CPU saves the values of dedicated registers to the system stack and fetches the interrupt vector. The interrupt handling time is therefore required. Obtain the interrupt handling time by using the following equations:

When an interrupt starts to be processed: $\phi$ =24 + 6 x Z machine cycles
When control is returned from an interrupt: $\phi$ =11 + 6 x Z machine cycles (RETI instruction)

The interrupt handling time varies depending on the address of the stack pointer.

**Table 6.4-3 Interpolation Values (Z) for the Interrupt Handling Time**

| Address pointed to by the stack pointer | Interpolation value (Z) |
|---|---|
| External 8-bit | +4 |
| External even-numbered address | +1 |
| External odd-numbered address | +4 |
| Internal even-numbered address | 0 |
| Internal odd-numbered address | +2 |

**Reference:**

One machine cycle equals one clock cycle of the machine clock ($\phi$).

## 6.5 Software Interrupts

**When the software interrupt instruction is executed, control is transferred from the main program to the interrupt processing program. Hardware interrupts are disabled while the software interrupt instruction is being executed.**

### ■ Software Interrupt Activation

#### ❍ Software interrupt activation

The INT instruction is used to activate a software interrupt. A software interrupt does not have an interrupt request flag bit or enable flag like that of a hardware interrupt. Thus, an interrupt request is output whenever the INT instruction is executed.

#### ❍ Hardware interrupt suppression

Since the INT instruction does not have interrupt levels, the interrupt level mask register (ILM) is not updated. During the execution of the INT instruction, the I flag of the condition code register (CCR) is set to 0, and hardware interrupts are masked.

To enable hardware interrupts during software interrupt processing, set the I flag of the condition code register (CCR) to "1" in the software interrupt processing routine.

#### ❍ Software interrupt operation

When the CPU fetches the INT instruction, the software interrupt processing microcode is activated. This microcode saves the internal CPU registers on the system stack, masks hardware interrupts (CCR: I = 0), and branches to the corresponding interrupt vector.

**Reference:**

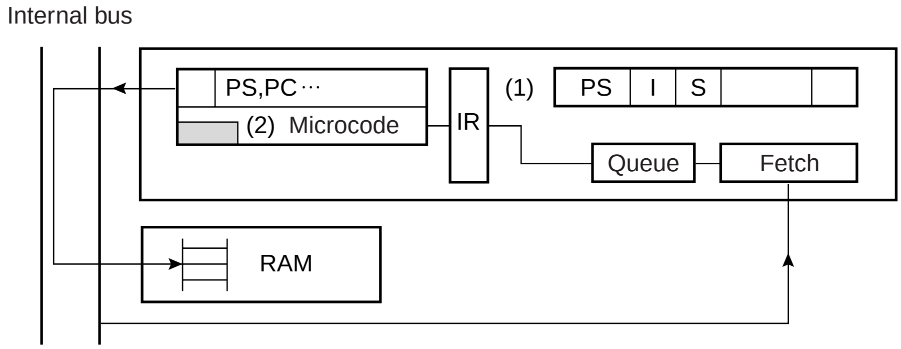
See Section 6.2 "Interrupt Causes and Interrupt Vectors", in CHAPTER 6 "INTERRUPTS" for more information about the allocation of interrupt numbers and interrupt vectors.

### ■ Returning from a Software Interrupt

In the interrupt processing program, when the interrupt return instruction (RETI instruction) is executed, the data saved to the system stack is restored to the dedicated registers and the processing that was being executed before branching for the interrupt is resumed.

## ■ Software Interrupt Operation

**Figure 6.5-1  Software Interrupt Operation**



PS : Processor status
I : Interrupt enable flag
S : Stack flag
IR : Instruction register

1. A software interrupt instruction is (INT instruction) executed.
2. The values of the dedicated registers are saved to the system stack.  Hardware interrupts are masked.  The processing branches to the interrupt vector.
3. The RETI instruction in the user interrupt processing routine terminates the interrupt processing.

**Note:**

When the program bank register (PCB) is "FF$_H$", the vector area of the CALLV instruction overlaps the INT #vct8 instruction table.  When you create software, watch for duplicated addresses of the CALLV and INT #vct8 instructions.

## 6.6 Interrupt of Extended Intelligent I/O Service (EI$^2$OS)

**The extended intelligent I/O service (EI$^2$OS) automatically transfers data between a peripheral function (resource) and memory. When the data transfer terminates, a hardware interrupt is generated.**

### ■ Extended Intelligent I/O Service (EI$^2$OS)

The extended intelligent I/O service (EI$^2$OS) is a type of hardware interrupt. EI$^2$OS transfers data between a peripheral function (resource) and memory. The user should create program to activate and terminate EI$^2$OS but need not create a data transfer program.

#### ❍ Advantages of extended intelligent I/O service (EI$^2$OS)

Compared to data transfer performed by the interrupt processing routine, EI$^2$OS has the following advantages.

- Coding a transfer program is not necessary, reducing program size.
- Since transfer can be activated due to an interrupt cause of a peripheral function (resource), there is no need of polling for a data transfer cause.
- Incrementing of a transfer address can be selected.
- Incrementing or no update can be selected for the I/O register address.

#### ❍ Extended intelligent I/O service (EI$^2$OS) termination interrupt

The processing branches to the interrupt processing routine when data transfer by EI$^2$OS terminates.

An interrupt processing program can determine the EI$^2$OS termination cause by checking the EI$^2$OS status bits (S1 and S0) of the interrupt control register (ICR).

**Reference:**

Interrupt numbers and interrupt vectors are permanently set for each peripheral. See Section 6.2 "Interrupt Causes and Interrupt Vectors", in CHAPTER 6 "INTERRUPTS" for more information.

#### ❍ Interrupt control register (ICR)

This register, which is located in the interrupt controller, activates EI$^2$OS, specifies the EI$^2$OS channel, and displays the EI$^2$OS termination status.

### ❍ Extended intelligent I/O service (EI$^2$OS) descriptor (ISD)

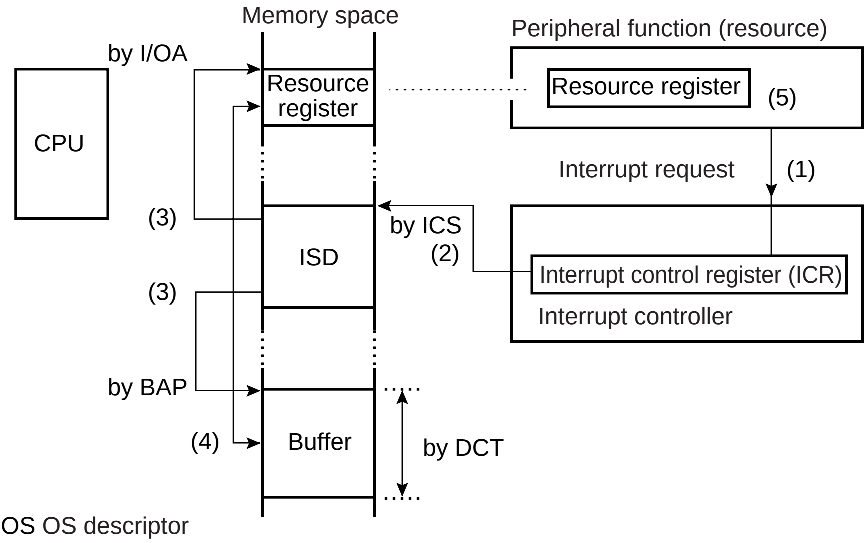
This descriptor, which is located in RAM at 000100$_H$ to 00017F$_H$, is an eight-byte data that retains the transfer mode, resource address, transfer count, and buffer address. The descriptor handles 16 channels. The channel is specified by the interrupt control register (ICR).

**Note:**

When the extended intelligent I/O service (EI$^2$OS) is operating, execution of the CPU program stops.

## ■ Operation of the Extended Intelligent I/O Service (EI$^2$OS)

**Figure 6.6-1 Extended Intelligent I/O Service (EI$^2$OS) Operation**



ISD : EI$^2$OS OS descriptor
I/OA: I/O address pointer
BAP: Buffer address pointer
ICS : EI$^2$OS channel setting bit in the interrupt control register (ICR)
DCT: Data counter

1. A peripheral function (resource) outputs an interrupt request.
2. The interrupt controller selects the EI$^2$OS descriptor in accordance with the setting in the interrupt control register (ICR).
3. The transfer source and transfer destination are read from the descriptor.
4. Transfer is performed between resource and memory.
5. After data transfer is completed, the interrupt request flag bit of the peripheral function (resource) is cleared to "0".

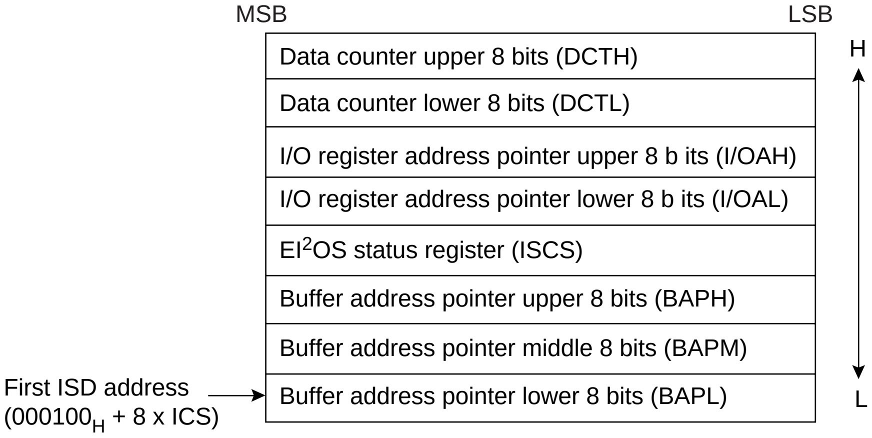
## 6.6.1 Extended Intelligent I/O Service (EI$^2$OS) Descriptor (ISD)

**The extended intelligent I/O service (EI$^2$OS) descriptor (ISD) resides in internal RAM at 000100$_H$ to 00017F$_H$. The ISD consists of 8 bytes x 16 channels.**

### ■ Configuration of the Extended Intelligent I/O Service (EI$^2$OS) Descriptor (ISD)

The ISD consists of 8 bytes x 16 channels.

**Figure 6.6-2 Configuration of EI$^2$OS Descriptor (ISD)**



MSB LSB

Data counter upper 8 bits (DCTH)

Data counter lower 8 bits (DCTL)

I/O register address pointer upper 8 b its (I/OAH)

I/O register address pointer lower 8 b its (I/OAL)

EI$^2$OS status register (ISCS)

Buffer address pointer upper 8 bits (BAPH)

Buffer address pointer middle 8 bits (BAPM)

Buffer address pointer lower 8 bits (BAPL)

MSB : Most significant bit
LSB : Least significant bit

Table 6.6-1 EI$^2$OS descriptor area

| Channel | Descriptor address |
|---|---|
| 0 | 000100$_H$ |
| 1 | 000108$_H$ |
| 2 | 000110$_H$ |
| 3 | 000118$_H$ |
| 4 | 000120$_H$ |
| 5 | 000128$_H$ |
| 6 | 000130$_H$ |
| 7 | 000138$_H$ |
| 8 | 000140$_H$ |
| 9 | 000148$_H$ |
| 10 | 000150$_H$ |
| 11 | 000158$_H$ |
| 12 | 000160$_H$ |
| 13 | 000168$_H$ |
| 14 | 000170$_H$ |
| 15 | 000178$_H$ |

## 6.6.2 Registers of the Extended Intelligent I/O Service (EI$^2$OS) Descriptor (ISD)

**The extended intelligent I/O service (EI$^2$OS) descriptor (ISD) consists of the following registers, which account for a total of eight bytes:**

- **Data counter (DCT: 2 bytes)**
- **I/O register address pointer (I/OA: 2 bytes)**
- **EI$^2$OS status register (ISCS: 1 byte)**
- **Buffer address pointer (BAP: 3 bytes)**

**The initial values of these registers are undefined.**

### ■ Data Counter (DCT)

The DCT is a 16-bit register in which a transfer data byte count can be set. Every time one byte of data is transferred, the counter is decremented by one. EI$^2$OS terminates when the data counter reaches "0000$_H$".

**Figure 6.6-3 Configuration of DCT**

| | DCTH | | | | | | | | DCTL | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
| DCT | B15 | B14 | B13 | B12 | B11 | B10 | B09 | B08 | B07 | B06 | B05 | B04 | B03 | B02 | B01 | B00 | XXXXXXXXXXXXXXXX$_B$ |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

R/W: Read-write
X : Undefined

### ■ I/O Register Address Pointer (IOA)

The IOA is a 16-bit register that indicates the lower address (A15 to A0) of the I/O register used to transfer data to and from the buffer. The upper address (A23 to A16) is 00$_H$. Any I/O from 0000$_H$ to FFFF$_H$ can be specified by address.

**Figure 6.6-4 Configuration of I/O Register Address Pointer (IOA)**

| | I/OAH | | | | | | | | I/OAL | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
| IOA | A15 | A14 | A13 | A12 | A11 | A10 | A09 | A08 | A07 | A06 | A05 | A04 | A03 | A02 | A01 | A00 | XXXXXXXXXXXXXXXX$_B$ |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

R/W: Read-write
X : Undefined

## ■ Extended Intelligent I/O Service (EI$^2$OS) Status Register (ISCS)

The ISCS is an 8-bit register. The ISCS indicates the update/fixed for the buffer address pointer and I/O register address pointer, transfer data format (byte or word), and transfer direction.

**Figure 6.6-5 Configuration of EI$^2$OS Status Register (ISCS)**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | RESV | RESV | RESV | IF | BW | BF | DIR | SE | $XXXXXXXX_B$ |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

| SE | EI$^2$OS termination control bit |
|---|---|
| 0 | Not terminated by a request from the peripheral function. |
| 1 | Terminated by a request from the peripheral function |

| DIR | Data transfer direction specification bit |
|---|---|
| 0 | I/O register address pointer -> buffer address pointer. |
| 1 | Buffer address pointer -> I/O register address pointer |

| BF | BAP update/fixed selection bit |
|---|---|
| 0 | After data transfer, the buffer address pointer is updated. (*1) |
| 1 | After data transfer, the buffer address pointer is not updated. |

| BW | Transfer data length specification bit |
|---|---|
| 0 | Byte |
| 1 | Word |

| IF | IOA update/fixed selection bit |
|---|---|
| 0 | After data transfer, the I/O register address pointer is updated. (*2) |
| 1 | After data transfer, the buffer address pointer is not updated. |

| RESV | Reserved bits |
|---|---|
| 0 must be written to these bits. | |

R/W : Read-write
X : Undefined
*1 : Only the lower 16 bits of the buffer address pointer change.
The buffer address pointer can only be incremented.
*2 : The address pointer can only be incremented.

## ■ Buffer Address Pointer (BAP)

The buffer address pointer (BAP) is a 24-bit register in which a memory address of the data transfer source can be set in the EI$^2$OS operation. Data can be transferred between a 16M-byte memory address and a peripheral function (resource) address because a BAP exists in every channel of EI$^2$OS. If the BAP update/fixed selection bit (BF) of the EI$^2$OS status register (ISCS) is set to "0", only the lower 16 bits (BAPM and BAPL) are incremented and the upper 8 bits (BAPH) are not incremented.

**Figure 6.6-6 Configuration of Buffer Address Pointer (BAP)**

| | bit23 to bit16 | bit15 to bit8 | bit7 to bit0 | Initial value |
|---|---|---|---|---|
| BAP | BAPH | BAPM | BAPL | XXXXXX$_B$ |
| | (R/W) | (R/W) | (R/W) | |

R/W : Read-write
X : Undefined

**Note:**

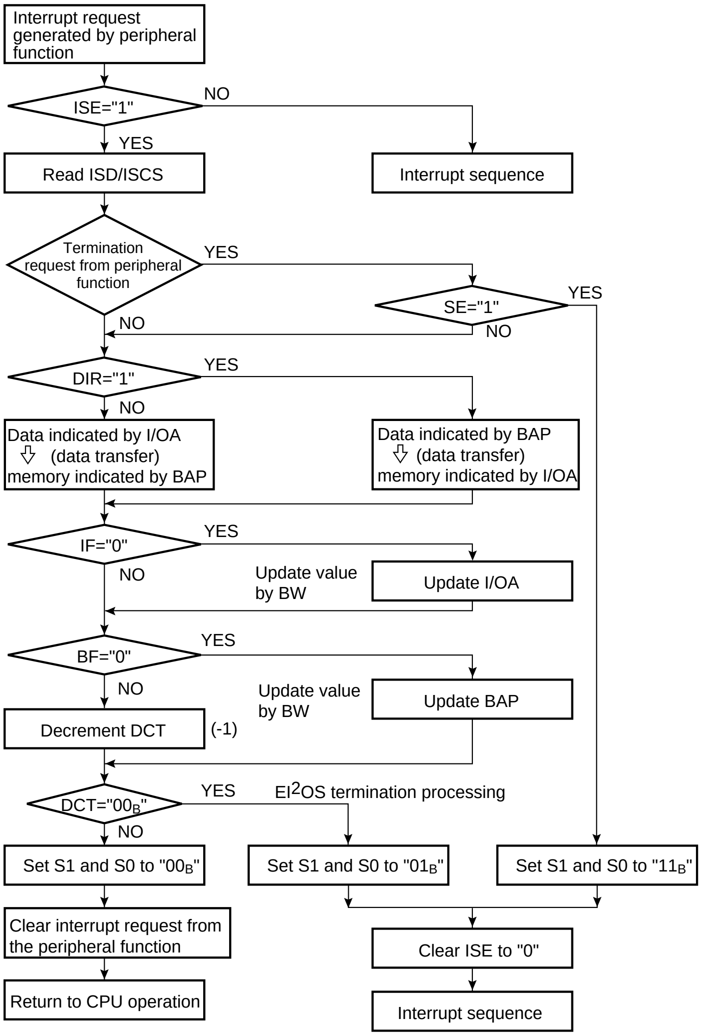
- The maximum transfer count that can be specified by the data counter (DCT) is 65,536 (64 K bytes).
- The area that can be specified by the I/O address pointer (IOA) extends from 000000$_H$ to 00FFFF$_H$.
- The area that can be specified with the buffer address pointer (BAP) extends from 000000$_H$ to FFFFFF$_H$.

## 6.6.3 Operation of the Extended Intelligent I/O Service ($EI^2OS$)

**The CPU uses $EI^2OS$ to transfer data if a peripheral function (resource) outputs an interrupt request while the corresponding interrupt control register (ICR) is set to enable the activation of $EI^2OS$. When the $EI^2OS$ processing terminates, the hardware interrupt processing is performed.**

### ■ Operation Flow of the Extended Intelligent I/O Service ($EI^2OS$)

**Figure 6.6-7 Flow of Extended Intelligent I/O Service ($EI^2OS$) Operation**



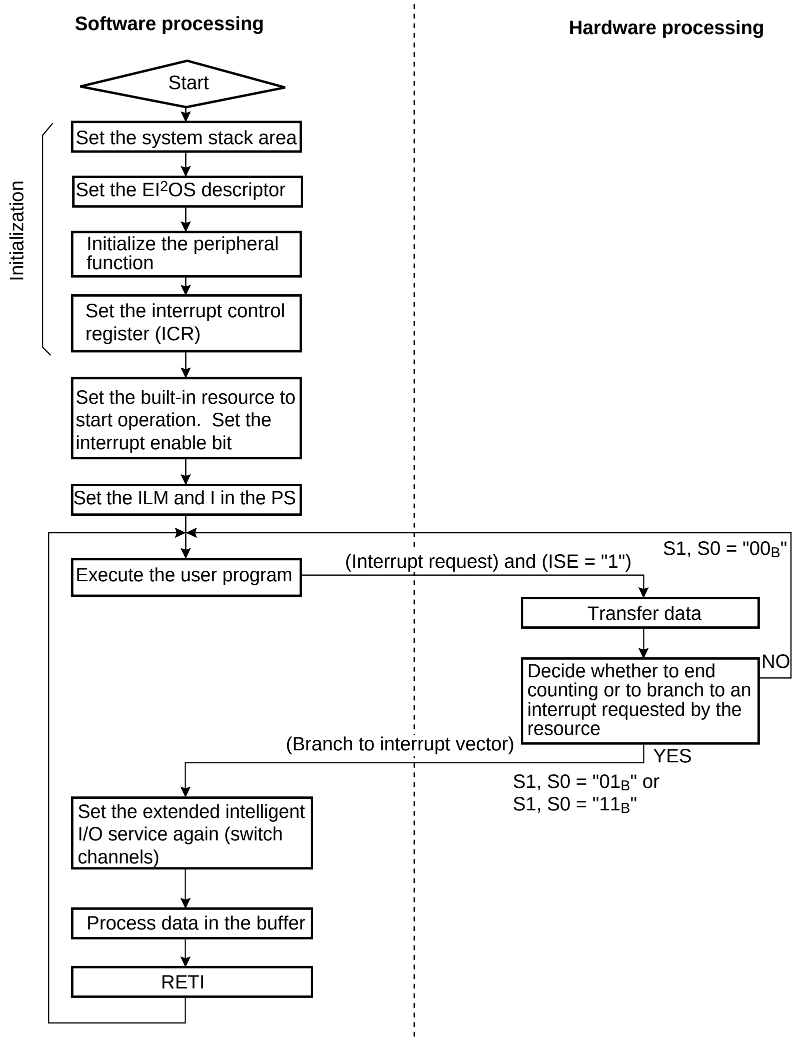
| | |
|---|---|
| ISD | : $EI^2OS$ descriptor |
| ISCS | : $EI^2OS$ status register |
| IF | : I/OA update/fixed selection bit in the $EI^2OS$ status register (ISCS) |
| BW | : Transfer data length specification bit in the $EI^2OS$ status register (ISCS) |
| BF | : BAP update/fixed selection bit in the $EI^2OS$ status register (ISCS) |
| DIR | : Data transfer direction specification bit in the $EI^2OS$ status register (ISCS) |
| SE | : $EI^2OS$ termination control bit in the $EI^2OS$ status register (ISCS) |
| DCT | : Data counter |
| I/OA | : I/O register address pointer |
| BAP | : Buffer address pointer |
| ISE | : $EI^2OS$ enable bit in the interrupt control register (ICR) |
| S1, S0 | : $EI^2OS$ status in the interrupt control register (ICR) |

# 6.6.4 Procedure for Using the Extended Intelligent I/O Service ($EI^2OS$)

**Before the extended intelligent I/O service ($EI^2OS$) can be used, the system stack area, extended intelligent I/O service ($EI^2OS$) descriptor, peripheral function (resource), and interrupt control register (ICR) must be set.**

## ■ Procedure for Using the Extended Intelligent I/O Service ($EI^2OS$)

Figure 6.6-8 Procedure for Using the Extended Intelligent I/O Service ($EI^2OS$)



Software processing
Hardware processing
Start
Initialization
Set the system stack area
Set the EI²OS descriptor
Initialize the peripheral function
Set the interrupt control register (ICR)
Set the built-in resource to start operation. Set the interrupt enable bit
Set the ILM and I in the PS
Execute the user program
(Interrupt request) and (ISE = "1")
S1, S0 = "00B"
Transfer data
Decide whether to end counting or to branch to an interrupt requested by the resource
NO
YES
(Branch to interrupt vector)
S1, S0 = "01B" or
S1, S0 = "11B"
Set the extended intelligent I/O service again (switch channels)
Process data in the buffer
RETI

ISE : $EI^2OS$ enable bit in the interrupt control register (ICR)
S1, S0 : $EI^2OS$ status of the interrupt control register (ICR)

## 6.6.5 Processing Time of the Extended Intelligent I/O Service (EI$^2$OS)

**The time required to process the extended intelligent I/O service (EI$^2$OS) varies with the settings for the EI$^2$OS descriptor (ISD):**

- **Setting in the EI$^2$OS status register (ISCS)**
- **Address pointed to by the I/O register address pointer (I/OA)**
- **Address pointed to by the buffer address pointer (BAP)**
- **External data bus length for external access**
- **Transfer data length**

**Because the hardware interrupt is activated when data transfer by EI$^2$OS terminates, the interrupt handling time is added.**

### ■ Processing Time (one transfer time) of the Extended Intelligent I/O Service (EI$^2$OS)

#### ❍ When data transfer continues

The EI$^2$OS processing time for data transfer continuation is shown in Table 6.6-2 "Extended Intelligent I/O Service Execution Time" based on the EI$^2$OS status register (ISCS) setting.

**Table 6.6-2 Extended Intelligent I/O Service Execution Time**

| EI$^2$OS termination control bit (SE) setting | | Terminates due to termination request from the peripherall | | Ignores termination request from the peripheral | |
|---|---|---|---|---|---|
| **I/OA update/fixed selection bit (IF) setting** | | **Fixed** | **Update** | **Fixed** | **Update** |
| BAP address update/fixed selection bit (BF) setting | Fixed | 32 | 34 | 33 | 35 |
| | Update | 34 | 36 | 35 | 37 |

Unit: Machine cycle (One machine cycle corresponds to one clock cycle of the machine clock (ϕ).

As shown in Table 6.3-3 "Correspondence between the EI$^2$OS Channel Selection Bits and Descriptor Addresses", interpolation is necessary depending on the EI$^2$OS execution condition.

**Table 6.6-3 Data Transfer Interpolation Value for EI$^2$OS Execution Time**

| I/O register address pointer | | | Internal access | | External access | |
|---|---|---|---|---|---|---|
| | | | **B/Even** | **Odd** | **B/Even** | **8/Odd** |
| Buffer address pointer | Internal access | B/Even | 0 | +2 | +1 | +4 |
| | | Odd | +2 | +4 | +3 | +6 |
| | External access | B/Even | +1 | +3 | +2 | +5 |
| | | 8/Odd | +4 | +6 | +5 | +8 |

B: Byte data transfer
8: External bus using the 8-bit word transfer
Even: Even-numbered address word transfer
Odd: Odd-numbered address word transfer

❍ **When the data counter (DCT) count terminates (final data transfer)**

Interrupt handling time is added because the hardware interrupt is activated when data transfer by EI$^2$OS terminates. The EI$^2$OS processing time when counting terminates is calculated by using the equation below. Z in this equation is a correction value for the interrupt handling time.

EI$^2$OS processing time when counting terminates =
EI$^2$OS processing time when data is transferred + $\underbrace{(21 + 6 \text{ x } Z)}_{\text{Interrupt handling time}}$ machine cycles

The interrupt handling time is different for each address pointed to by the stack pointer.

**Table 6.6-4 Interpolation Value (Z) for the Interrupt Handling Time**

| Address pointed to by the stack pointer | Interpolation value (Z) |
|---|---|
| External 8-bit | +4 |
| External even-numbered address | +1 |
| External odd-numbered address | +4 |
| Internal even-numbered address | 0 |
| Internal odd-numbered address | +2 |

### ❍ For termination by a termination request from the peripheral function (resource)

When data transfer by EI$^2$OS is terminated before completion due to a termination request from the peripheral function (resource) (ICR: S1, S0 = $11_B$), the data transfer is not performed and a hardware interrupt is activated. The EI$^2$OS processing time is calculated with the following formula. Z in the formula indicates the interpolation value for the interrupt handling time (Table 6.6-4 "Interpolation Value (Z) for the Interrupt Handling Time").

$$\boxed{\text{EI}^2\text{OS processing time for termination before completion} = 36 + 6 \times Z) \quad \text{machine cycles}}$$

One machine cycle corresponds to one clock cycle of the machine clock ($\phi$).

# 6.7 Exception Processing Interrupt

**In the MB90M405 series, exception processing occurs if an undefined instruction is executed. Exception processing is basically the same as an interrupt. When an exception occurs between instructions, program processing is interrupted and the processing branches to the exception processing routine.**
**Exception processing, occurring due to an unexpected operation, can be used to execute an undefined instruction and detect a CPU runaway status during debugging.**

## ■ Exception Processing

### ❍ Exception processing operation

In the MB90M405 series, the processing branches to the exception processing routine if an instruction undefined in the instruction map is executed.

The following processing is executed before exception processing branches to the interrupt routine:

- The A, DPR, ADB, DTB, PCB, PC, and PS registers are saved to the system stack.
- The I flag of the condition code register (CCR) is cleared to 0, and hardware interrupts are masked.
- The S flag of the condition code register (CCR) is set to 1, and the system stack is activated

The program counter (PC) value saved to the stack is the exact address where the undefined instruction is stored. For 2-byte or longer instruction codes, the code identified as undefined is stored at this address. When the exception factor type must be determined within the exception processing routine, use this PC value.

### ❍ Return from exception processing

If the RETI instruction is used to return control from exception processing, a branch to the exception processing routine occurs again because the PC is pointing to an undefined instruction. Use a software reset or input the "L" level from the $\overline{\text{RST}}$ pin (an external reset).

## 6.8 Stack Operations for Interrupt Processing

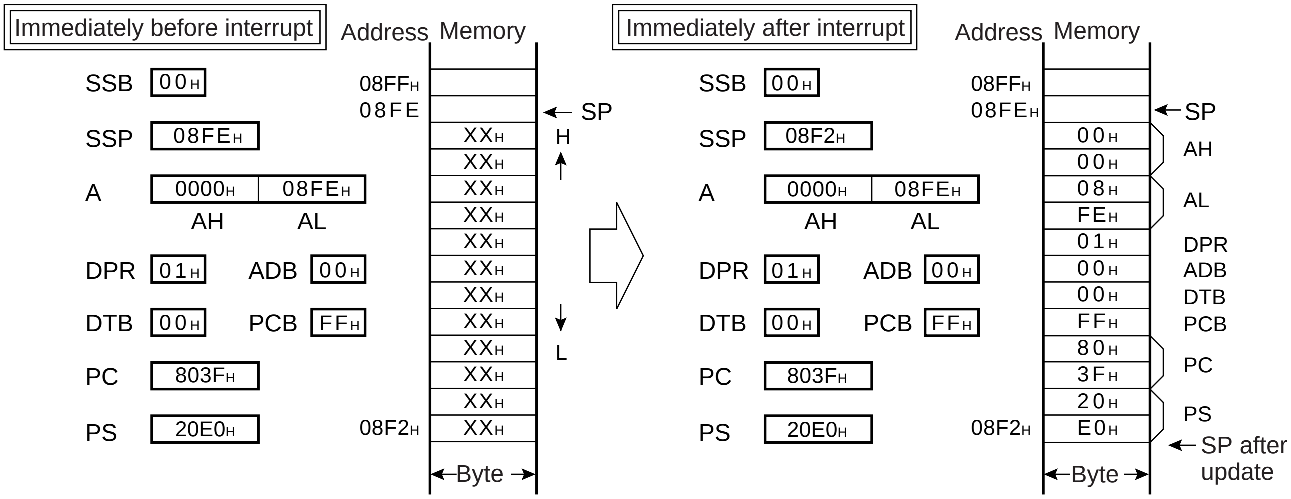
**Once an interrupt is accepted, the contents of the dedicated registers are saved to the system stack before a branch to interrupt processing. Execute the interrupt return instruction after the interrupt processing terminates to restore the values saved on the system stack to the dedicated registers.**

### ■ Stack Operations at the Start of Interrupt Processing

Once an interrupt is accepted, the CPU saves the contents of the current dedicated registers to the system stack in the order given below:

- Accumulator (A)
- Direct page register (DPR)
- Additional data bank register (ADB)
- Data bank register (DTB)
- Program bank register (PCB)
- Program counter (PC)
- Processor status (PS)

**Figure 6.8-1 Stack Operations at the Start of Interrupt Processing**



### ■ Stack Operations on Return from Interrupt Processing

When the interrupt return instruction (RETI) is executed at the termination of interrupt processing, the PS, PC, PCB, DTB, ADB, DPR, and A values are returned from the stack in reverse order from the order they were placed on the stack. The dedicated registers are restored to the status they had immediately before the start of interrupt processing.

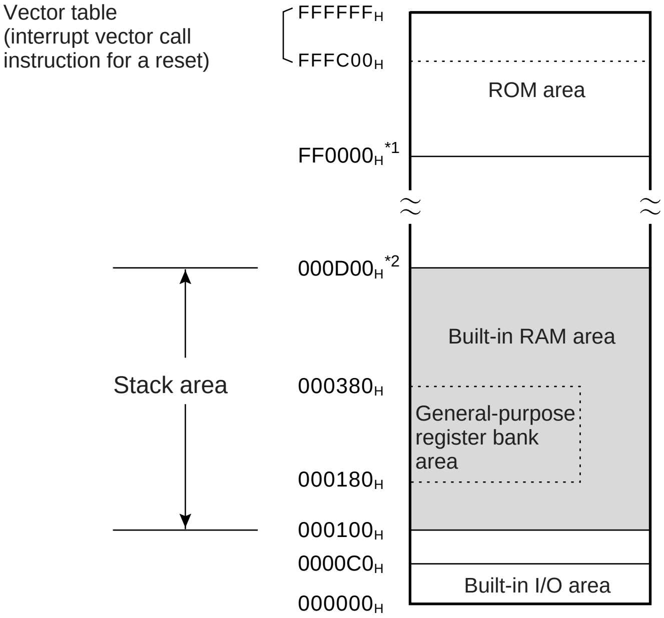
## ■ Stack Area

### ❍ Stack area allocation

The stack area is used for saving and restoring the program counter (PC) when the subroutine call instruction (CALL) and vector call instruction (CALLV) are executed in addition to interrupt processing. The stack area is used for temporary saving and restoring of registers by the PUSHW and POPW instructions.

The stack area is allocated together with the data area in RAM.

**Figure 6.8-2 Stack Area**



*1 The internal ROM is different for each model.
*2 The internal RAM is different for each model.

**Note:**

- Generally set an even-numbered address in the stack pointers (SSP and USP). If an odd-numbered address is set, one extra cycle is required to save data to, and restore data from the stack.
- Allocate the system stack area, user stack area, and data area so that they do not overlap.

### ❍ System stack and user stack

The system stack area is used for interrupt processing. When an interrupt is output, the user area being used is switched to the system stack. Use only the system stack unless the stack space needs to be divided in particular.

# 6.9 Sample Programs for Interrupt Processing

**This section contains sample programs for interrupt processing.**

## ■ Sample Programs for Interrupt Processing

### ❍ Processing specifications

The following is a sample program for an interrupt that uses external interrupt 0 (INT0).

### ❍ Sample coding

```
DDR1     EQU   000011H        ;Port 1 direction register
ENIR     EQU   000028H        ;DTP/interrupt permission register
EIRR     EQU   000029H        ;DTP/interrupt cause register
ELVR     EQU   00002AH        ;Request level setting register
ICR00    EQU   0000B0H        ;Interrupt control register 00
STACK    SSEG                 ;Stack
         RW    100
STACK_T  RW    1
STACK    ENDS
;-------- Main program ----------------------------------------------------------
CODE     CSEG
START:
         MOV   RP,#0              ;General-purpose registers use the first bank.
         MOV   ILM, #07H          ;Sets ILM in PS to level 7.
         MOV   A, #!STACK_T       ;Sets system stack.
         MOV   SSB, A
         MOVW  A, #STACK_T        ;Sets stack pointer, then
         MOVW  SP, A              ;Sets SSP because S flag = 1.
         MOV   DDR1, #00000000B   ;Sets P10/INT0 pin to input.
         OR    CCR, #040H         ;Sets I flag of CCR in PS, enables interrupts.
         MOV   I:ICR00, #00H      ;Sets interrupt level to 0 (highest priority).
         MOV   I:ELVR, #00000001B ;Requests that INT0 be made level H.
         MOV   I:EIRR, #00H       ;Clears INT0 interrupt cause.
         MOV   I:ENIR, #01H       ;Enables INT0 input.
LOOP:    NOP                      ;Dummy loop
         NOP
         NOP
         NOP
         BRA   LOOP               ;Unconditional jump
---------Interrupt program ------------------------------------------------------
ED_INT1:
         MOV   I:EIRR, #00H   ;Acceptance of new INT0 not allowed
         NOP
         NOP
         NOP
         NOP
         NOP
         NOP
         RETI                 ;Return from interrupt
CODE     ENDS
;--------Vector setting----------------------------------------------------------
VECT     CSEG  ABS=0FFH
         ORG   0FFDH          ;Sets vector for interrupt #11 (0BH).
         DSL   ED_INT1
         ORG   0FFDCH         ;Sets reset vector.
         DSL   START
         DB    00H            ;Sets single-chip mode.
VECT     ENDS
         END   START
```

## ■ Processing Specifications of Sample Program for Extended Intelligent I/O Service (EI$^2$OS)

### ❍ Processing Specifications

- This program detects the H level signal input to the INT0 pin and activates the extended intelligent I/O service (EI$^2$OS).
- When the H level is input to the INT0 pin, EI$^2$OS is activated.  Data is transferred from port 0 to the memory at the 3000$_H$ address.
- The number of transfer data bytes is 100 bytes.  After 100 bytes are transferred, an interrupt is generated because EI$^2$OS transfer has terminated.

### ❍ Sample coding

```
DDR1    EQU   000011H         ;Port 1-direction register
ENIR    EQU   000028H         ;DTP/interrupt permission register
EIRR    EQU   000029H         ;DTP/interrupt cause register
ELVR    EQU   00002AH         ;Request level setting register
ICR00   EQU   0000B0H         ;Interrupt control register 00
BAPL    EQU   000100H         ;Lower buffer address pointer
BAPM    EQU   000101H         ;Middle buffer address pointer
BAPH    EQU   000102H         ;Upper buffer address pointer
ISCS    EQU   000103H         ;EI2OS status
I/OAL   EQU   000104H         ;Lower I/O address pointer
I/OAH   EQU   000105H         ;Upper I/O address pointer
DCTL    EQU   000106H         ;Low-order data counter
DCTH    EQU   000107H         ;High-order data counter
ER0     EQU   EIRR:0          ;Definition of external interrupt request flag bit
STACK   SSEG                  ;Stack
        RW    100
STACK_T RW    1
STACK   ENDS
;-------------------Main program-------------------------------------------------
CODE    CSEG
START:
        AND   CCR, #0BFH         ;Clears the I flag of the CCR in the PS and
                                 ;prohibits interrupts.
        MOV   RP, #00            ;Sets the register bank pointer.
        MOV   A, #!STACK_T       ;Sets the system stack.
        MOV   SSB, A
        MOVW  A, #STACK_T        ;Sets the stack pointer, then
        MOVW  SP, A              ;Sets SSP because the S flag = 1.
        MOV   I:DDR1, #00000000B ;Sets the P10/INT0 pin to input.
        MOV   BAPL, #00H         ;Sets the buffer address (003000H).
        MOV   BAPM, #30H
        MOV   BAPH, #00H
        MOV   ISCS, #00010001B   ;No I/O address update, byte transfer, buffer
                                 ;address updated, I/O -> buffer transfer,
                                 ;terminated by the peripheral function.
        MOV   IOAL, #00H         ;Sets the transfer source address
                                 ;(port 0:000000H).
        MOV   IOAH, #00H
        MOV   DCTL, #64H         ;Sets the number of transfer bytes (100 bytes).
        MOV   DCTH, #00H
        MOV   I:ICR00,#00001000B ;EI$^2$OS channel 0, EI$^2$OS enable, interrupt level 0
                                 ;(highest priority)
        MOV   I:ELVR, #00000001B ;Requests that INT0 be made H level.
        MOV   I:EIRR, #00H       ;Clears the INT0 interrupt cause.
        MOV   I:ENIR, #01H       ;Enables INT0 interrupts.
        MOV   ILM, #07H          ;Sets the ILM in the PS to level 7.
        OR    CCR, #040H         ;Sets the I flag of the CCR in the PS and
                                 ;enables interrupts.
         :
LOOP    BRA   LOOP               ;Infinite loop
;---------------Interrupt program------------------------------------------------
WARI    CLRB  ER0                ;Clears interrupt/DTP request flag.
         :
        User  processing         ;Checks EI$^2$OS termination factor,
         :                       ;processes data in buffer, sets EI$^2$OS again.
        RETI
CODE    ENDS
;---------------Vector processing------------------------------------------------
VECT    CSEG  ABS=0FFH
        ORG   0FFD0H             ;Sets vector for interrupt #11 (0BH).
        DSL   WARI
        ORG   0FFDCH             ;Sets reset vector.
        DSL   START
        DB    00H                ;Sets single-chip mode.
VECT    ENDS
        END   START
```

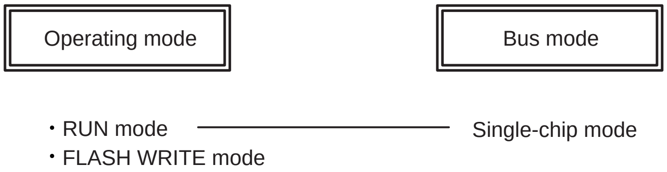
# CHAPTER 7 SETTING A MODE

**This chapter describes the operating modes and the memory access modes of the MB90M405 series.**

# 7.1 Setting a Mode

**Set the mode pin level used after a reset and the mode data in the mode register to determine the operating mode.**

## ■ Mode Setting



Operating mode

Bus mode

- RUN mode
- FLASH WRITE mode

Single-chip mode

## ■ Operating Modes

An operating mode can be specified by the mode pins (MD2 to MD0) and the bus mode setting bits (M1 and M0) of the mode data registers. The microcontroller performs an ordinary start of operation and writes to FLASH memory in the specified operating mode.

**Note:**

Set single-chip mode for the MB90M405 series.

To set single-chip mode, set the MD2 to MD0 pins to "$011_B$" and the M1 and M0 bits of the mode data register to "$00_B$", respectively.

## ■ Bus Mode

The bus mode varies depending on whether the memory into which the reset vector should be read is internal or external. Set the MD2 to MD0 pins and the M1 and M0 bits of the mode data register to specify a bus mode. Set the MD2 to MD0 pins to determine a bus mode in which a reset vector and mode data should be read. Additionally, set the M1 and M0 bits of the mode data register to specify a bus mode.

**Reference:**

The term RUN mode refers to the current mode in which the CPU operates. RUN modes include main clock mode, in which the CPU operates based on the main clock; PLL clock mode, in which the CPU operates based on the PLL clock; and low power consumption mode. For more information, see Chapter 5 "Low Power Consumption Mode".

**Note:**

For the MB90M405 series, specify single-chip mode.

To specify single-chip mode, set the MD2 to MD0 pins to $011_B$ and the bus mode setting bits (M1 and M0) of the mode data register to $00_B$.

# 7.2 Mode Pins (MD2 to MD0)

**Three external pins, MD2 to MD0, are supported as the mode pins. These are used to specify how the reset vector and mode data are fetched.**

## ■ Mode Pins (MD2 to MD0)

Use the mode pins to specify whether a reset vector should be read from external or internal memory. Use the mode data register to specify the external data bus width if a reset vector should be read from external memory.

For a built-in flash memory type, use the mode pins to specify the flash memory write mode in which a program should be written to the built-in flash memory.

**Table 7.2-1 Mode Pin Settings**

| MD2 | MD2 | MD0 | Mode name | Reset vector access area | External data bus width | Remarks |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | Setting not allowed | | | |
| 0 | 0 | 1 | | | | |
| 0 | 1 | 0 | | | | |
| 0 | 1 | 1 | Internal vector mode | Internal memory | Specified in the mode data register | The reset sequence and subsequent sequences are controlled by mode data. |
| 1 | 0 | 0 | Setting not allowed | | | |
| 1 | 0 | 1 | | | | |
| 1 | 1 | 0 | Flash serial write mode (*1) | - | - | - |
| 1 | 1 | 1 | Flash memory mode | - | - | - |

MD2 to MD0: Connect the pins to $V_{SS}$ for 0 and to $V_{CC}$ for 1.

*1: The flash memory serial write cannot be executed just by setting the mode terminal. The settings for other pins must be made as well. For more information, see "Example of a Serial Write Connection".

**Note:**

For the MB90M405 series, specify single-chip mode.

To specify single-chip mode, set the MD2 to MD0 pins to $011_B$ and the bus mode setting bits (M1 and M0) of the mode data register to $00_B$.

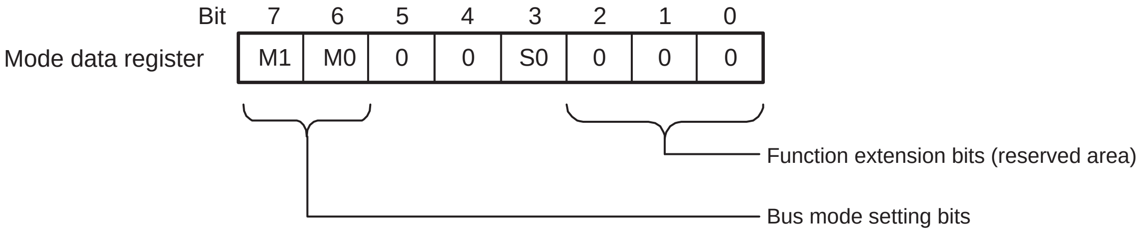
# 7.3 Mode Data

**The mode data register is at memory location FFFFDF$_{H,}$ and is used to specify the operation after a reset sequence.**

## ■ Mode Data

The mode data at address "FFFFDF$_H$" can be fetched to the mode data register while a reset sequence is being executed. The contents of the mode data register can be changed while a reset sequence is being executed. They cannot be changed using an instruction. The mode data setting takes effect after a reset sequence.

**Figure 7.3-1 Mode Data Configuration**



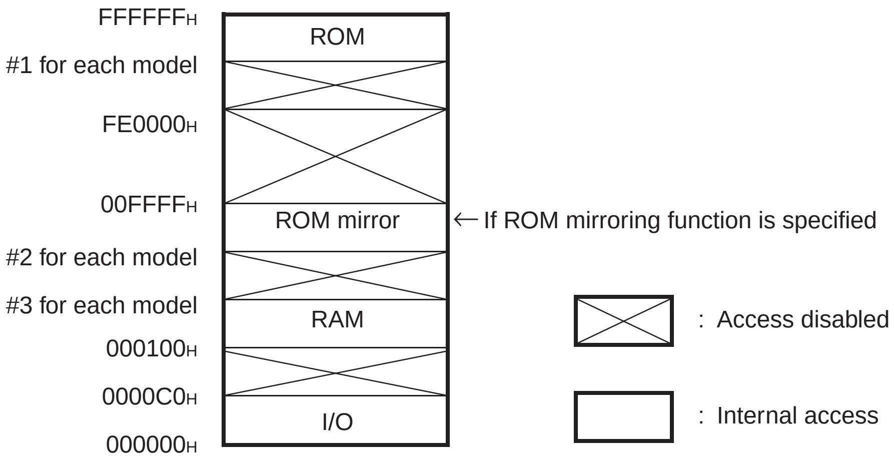
## ■ Bus Mode Setting Bits

The bus mode setting bits specify operating mode after a reset sequence.

**Table 7.3-1 Bus Mode Setting Bits and Functions**

| M1 | M0 | Function | Remarks |
|---|---|---|---|
| 0 | 0 | Single-chip mode | Described in this manual |
| 0 | 1 | (Setting not allowed) | - |
| 1 | 0 | | |
| 1 | 1 | | |

**Figure 7.3-2 Memory Map in Single-Chip Mode**



Note: "#x for each model" is an address that depends on the model. For more information, see the memory map.

## ■ Relationship between Mode Pins and Mode Data

**Table 7.3-2 Relationship between Access Areas and Physical Addresses in Single-chip Mode**

| Mode | MD2 | MD1 | MD0 | M1 | M0 |
|---|---|---|---|---|---|
| Single-chip mode | 0 | 1 | 1 | 0 | 0 |

**Note:**

For the MB90M405 series, specify single-chip mode.

To specify single-chip mode, set the MD2 to MD0 pins to $011_B$ and the bus mode setting bits (M1 and M0) of the mode data register to $00_B$.

# CHAPTER 8 I/O PORTS

**This chapter describes the functions and operations of the MB90M405 series I/O ports.**

# 8.1 Overview of I/O Ports

**A maximum of 26 I/O ports (parallel I/O ports) are available. These ports can also be used as resource I/O pins (I/O pins of peripheral functions).**

## ■ I/O Port Function

I/O ports include port direction registers (DDR) and port data registers (PDR). Each bit in the DDR specifies input or output for a port pin. The PDR specifies the output data for a port pin. If the DDR sets an I/O port pin to input, the level value of the port pin is made ready by reading the PDR. If the DDR sets an I/O port pin to output, the value of the PDR is output to the port pin. The following lists the resources that are also used to function as I/O ports:

- Port 8: Used as an I/O port and also as resource pins (external interrupt input pin, ICU, and UART)
- Port 9: Used as an I/O port and also as resource pins ($I^2C$ and serial I/O ch3)
- Port A: Used as an I/O port and also as resource pins (A/D converter and timing clock output)
- Port B: Used as an I/O port and also as resource pins (A/D converter, serial I/O ch2, external interrupt input pin, and reload timer ch0)

**Table 8.1-1 Functions of Each Port**

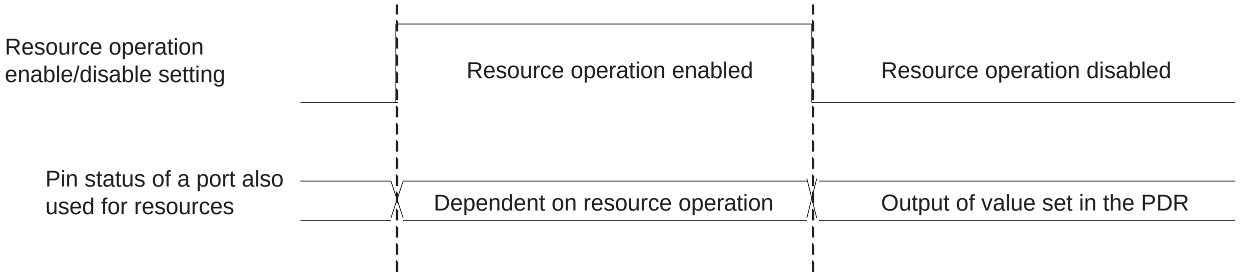
| I/O port | Pin | Input form | Output form | Function | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Port 8 | P80 to P87 | CMOS (hysteresis) | CMOS | I/O port | P87 | P86 | P85 | P84 | P83 | P82 | P81 | P80 |
| | | | | Resource | SO1 | SC1 | SI1 | SO0 | SC0 | SI0 | IC1 | IC0 |
| | | | | | | | | | | | INT1 | INT0 |
| Port 9 | P90/SDA/SO3 to P91/SCL/SC3 | | Nch open drain | I/O port | - | - | - | - | - | - | P91 | P90 |
| | | | | Resource | - | - | - | - | - | - | SCL | SDA |
| | | | | | | | | | | | SC3 | SO3 |
| Port A | PA0/AN0/TMCK to PA7/AN7 | | CMOS | I/O port | PA7 | PA6 | PA5 | PA4 | PA3 | PA2 | PA1 | PA0 |
| | | | | Resource | AN7 | AN6 | AN5 | AN4 | AN3 | AN2 | AN1 | AN0 |
| | | | | | | | | | | | | TMCK |
| Port B | PB0/AN8 to PB7/AN15/INT3 | | | I/O port | PB7 | PB6 | PB5 | PB4 | PB3 | PB2 | PB1 | PB0 |
| | | | | Resource | AN15 | AN14 | AN13 | AN12 | AN11 | AN10 | AN9 | AN8 |
| | | | | | INT3 | INT2 | SO2 | SC2 | SI2 | - | - | - |
| | | | | | | | TO | TIN | | | | |

**Note:**

Ports A and B are also used as analog input pins. When using them as I/O ports, set port A and port B direction registers (DDRA and DDRB), the data registers (PDRA and PDRB), and analog input enable registers 0 and 1 (ADER0 and ADER1) to $00_H$. A reset initializes analog input enable registers 0 and 1 (ADER0 and ADER1) to $FF_H$.

## ■ Operation of an I/O Port Also Used for Resources When DDR Is Set to Port Output

**Figure 8.1-1 Pin Status of a Port Also Used for Resources When Resource Operation is Enabled or Disabled**


Resource operation enable/disable setting
Resource operation enabled
Resource operation disabled
Pin status of a port also used for resources
Dependent on resource operation
Output of value set in the PDR

# 8.2 I/O Port Registers

**This section lists the registers related to I/O port settings.**

## ■ I/O Port Registers

**Table 8.2-1 Port Registers**

| Register | Read/Write | Address | Initial value |
|---|---|---|---|
| Port 8 data register (PDR8) | R/W | $000008_H$ | $XXXXXXXX_B$ |
| Port 9 data register (PDR9) | R/W | $000009_H$ | $XXXXXXXX_B$ |
| Port A data register (PDRA) | R/W | $00000A_H$ | $XXXXXXXX_B$ |
| Port B data register (PDRB) | R/W | $00000B_H$ | $XXXXXXXX_B$ |
| Port 8 direction register (DDR8) | R/W | $000018_H$ | $00000000_B$ |
| Port 9 direction register (DDR9) | R/W | $000019_H$ | $XXXXXX00_B$ |
| Port A direction register (DDRA) | R/W | $00001A_H$ | $00000000_B$ |
| Port B direction register (DDRB) | R/W | $00001B_H$ | $00000000_B$ |
| Analog input enable register 0 (ADER0) | R/W | $00001E_H$ | $11111111_B$ |
| Analog input enable register 1 (ADER1) | R/W | $00001F_H$ | $11111111_B$ |

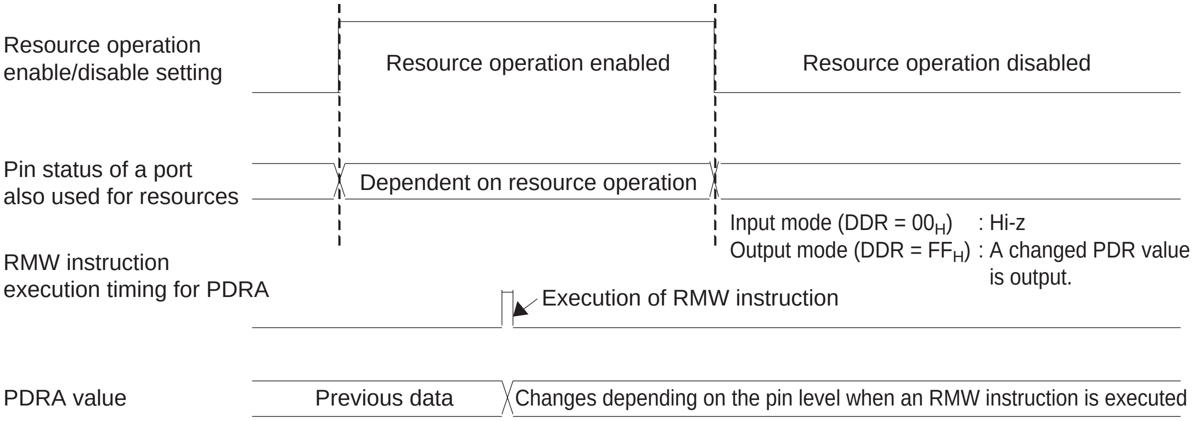
R/W: Read/write enabled
X: Undefined
-: Unused

**Notes:**

- If an RMW instruction is executed for a port data register (PDR) in port input mode, the pin level is read when a READ instruction is executed. Note that the bit value used for the same type of input port other than the one subject to bit operations may be changed.
- If an RMW instruction is executed for a port data register (PDR) that is also used for resources in resource operation mode, the pin level is read for a pin that operates as a resource pin when a READ instruction is executed. Note that the bit value used for the same type of input port other than the one subject to bit operations may be changed.

**Table 8.2-2 What Is Read by READ after RMW Instruction Executed for a PDR**

| | Resource operation enabled | Resource operation disabled |
|---|---|---|
| Port input mode (DDR = $00_H$) | Pin level | Pin level |
| Port output mode (DDR = $FF_H$) | Pin level | PDR value |


Resource operation enable/disable setting
Resource operation enabled
Resource operation disabled
Pin status of a port also used for resources
Dependent on resource operation
Input mode (DDR = 00H) : Hi-z
Output mode (DDR = FFH) : A changed PDR value is output.
RMW instruction execution timing for PDRA
Execution of RMW instruction
PDRA value
Previous data
Changes depending on the pin level when an RMW instruction is executed

# 8.3 Port 8

**Port 8 is an I/O port. This section describes the configuration and registers of Port 8 and provides a block diagram of the pins.**

## ■ Port 8 Configuration

Port 8 consists of the following:

- I/O port pins/resource I/O pins (P80, IC0, INT0 to P87, and SO1)
- Port 8 data register (PDR8)
- Port 8 direction register (DDR8)

## ■ Port 8 Pins

**Table 8.3-1 Port 8 Pins**

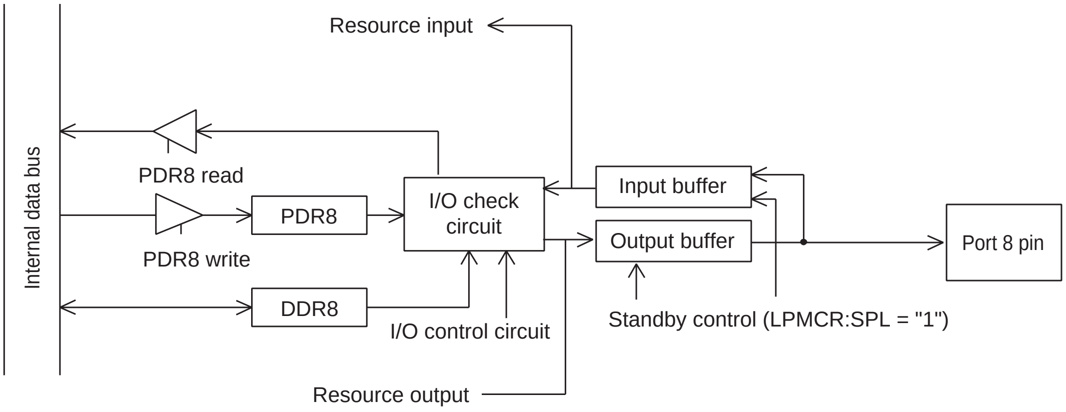
| Port name | Pin name | Port function | | Resource function | | I/O form | | Circuit type |
|---|---|---|---|---|---|---|---|---|
| | | | | | | Input | Output | |
| Port 8 | P80/IC0/INT0 | P80 | I/O port | IC0 | Trigger input for input capture 0 | CMOS (hysteresis) | CMOS | E |
| | | | | INT0 | External interrupt 0 | | | |
| | P81/IC1/INT1 | P81 | | IC1 | Trigger input for input capture 1 | | | |
| | | | | INT1 | External interrupt 1 | | | |
| | P82/SI0 | P82 | | SI0 | Serial data input 0 | | | |
| | P83/SC0 | P83 | | SC0 | Serial clock I/O 0 | | | |
| | P84/SO0 | P84 | | SO0 | Serial data output 0 | | | |
| | P85/SI1 | P85 | | SI1 | Serial data input 1 | | | |
| | P86/SC1 | P86 | | SC1 | Serial clock I/O 1 | | | |
| | P87/SO1 | P87 | | SO1 | Serial data output 1 | | | |

**Note:**

For the circuit type, see Section 1.7 "I/O Circuit Types."

## ■ Block Diagram of the Port 8 Pins

Figure 8.3-1 Block Diagram of the Port 8 Pins



## ■ Port 8 Registers

The Port 8 registers are the Port 8 data register (PDR8) and the Port 8 direction register (DDR8). The bits of each register correspond to the Port 8 pins on a one-to-one basis.

Table 8.3-2 Port 8 Pins and Corresponding Register Bits

| Port name | Register bits and corresponding port pins | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Port 8 | PDR8, DDR8 | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 |
| | Corresponding pin | P87 | P86 | P85 | P84 | P83 | P82 | P81 | P80 |

## 8.3.1 Port 8 Registers (PDR8 and DDR8)

**This section describes the Port 8 registers.**

### ■ Functions of the Port 8 Registers

❍ **Port 8 data register (PDR8)**

The PDR8 register specifies the output value for each Port 8 pin.

❍ **Port 8 direction register (DDR8)**

The DDR8 register specifies the I/O direction for each Port 8 pin.

A pin functions as an output port when the bit corresponding to the pin is set to 1. A pin functions as an input port when the bit corresponding to the pin is set to 0.

**Table 8.3-3 Functions of the Port 8 Registers**

| Register name | Bit value | Read mode | | Write mode | | Address | Initial value |
|---|---|---|---|---|---|---|---|
| | | Input port | Output port | Input port | Output port | | |
| Port 8 data register (PDR8) | 0 | The pin is at the "L" level. | The corresponding bit of PDR8 is set to 0. | The corresponding bit of PDR8 is 0. | The "L" level is output from the pin. | $000008_H$ | $XXXXXXXX_B$ |
| | 1 | The pin is at the "H" level. | The corresponding bit of PDR8 is set to 1. | The corresponding bit of PDR8 is 1. | The "H" level is output from the pin. | | |
| Port 8 direction register (DDR8) | 0 | The corresponding bit of PDR8 is set to 0. | | The pin functions as an input port. | | $000018_H$ | $00000000_B$ |
| | 1 | The corresponding bit of PDR8 is set to 1. | | The pin functions as an output port. | | | |

X: Undefined

# 8.3.2 Operation of Port 8

**This section describes the operation of Port 8.**

## ■ Operation of Port 8

### ❍ When Port 8 is set as an output port in the Port 8 direction register (DDR8)

- The value stored in the Port 8 data register (PDR8) is output to the Port 8 pins.
- When the Port 8 data register (PDR8) is read, the value stored in PDR8 is output.

### ❍ When Port 8 is set as an input port in the Port 8 direction register (DDR8)

- The Port 8 pins have high impedance.
- When a value is set in the Port 8 data register (PDR8), the value is retained but is not output to the pin.
- When the PDR8 register is read, the pin input level (0 for "L" or 1 for "H") is output.

**Note:**

If a read-modify-write instruction (such as the bit set instruction) is used to access the PDR8 register, none of the bits specified for output in the DDR8 register are affected. For a bit specified for input in the DDR8 register, however, the pin input level is written to the PDR8 register. Therefore, to change a bit specified for input to output, first write an output value to the PDR8 register and then specify the DDR8 register as an output port.

### ❍ Port operation after a reset

- When the CPU is reset, the DDR8 and RDR8 registers are initialized to $00_H$ and the Port 8 pins have high impedance.
- The PDR8 register is not initialized when the CPU is reset. To use the PDR8 as an output port, first write an output value to the PDR8 register and then specify the DDR8 register as an output port.

### ❍ Port operation in stop or time-base timer mode

When the port switches to stop mode or time-base timer mode while the pin status setting bit (SPL) of the low-power mode control register (LPMCR) is set to 1, the pins have high impedance regardless of the value in the Port 8 direction register (DDR8). Note that the input buffer is forcibly blocked off to prevent leakage due to an open circuit.

**Table 8.3-4 States of the Port 8 Pins**

| Pin name | Normal operation | Sleep mode | Stop mode or time-base timer mode (SPL = 0) | Stop mode or time-base timer mode (SPL = 1, RDR = 0) | Stop mode or time-base timer mode (SPL = 1, RDR = 1) |
|---|---|---|---|---|---|
| P80 to P87 | I/O port | I/O port | I/O port | Input blocking: Output at Hi-z | Input blocking: Held at the "H" level |
| IC0 | Trigger input for input capture 0 | Trigger input for input capture 0 | Trigger input for input capture 0 | | |
| IC1 | Trigger input for input capture 1 | Trigger input for input capture 1 | Trigger input for input capture 1 | | |
| SI0 | Serial data input 0 | Serial data input 0 | Serial data input 0 | | |
| SC0 | Serial clock I/O 0 | Serial clock I/O 0 | Serial clock I/O 0 | | |
| SO0 | Serial data output 0 | Serial data output 0 | Serial data output 0 | | |
| SI1 | Serial data input 1 | Serial data input 1 | Serial data input 1 | | |
| SC1 | Serial clock I/O 1 | Serial clock I/O 1 | Serial clock I/O 1 | | |
| SO1 | Serial data output 1 | Serial data output 1 | Serial data output 1 | | |
| INT0, INT1 | External interrupt request 0, 1 | External interrupt request 0, 1 | External interrupt request 0, 1 | Input blocking: Output held (Input is enabled when an external interrupt is enabled.) | Input blocking: Output at Hi-z (Input is enabled when an external interrupt is enabled.) |

SPL: Pin status specification bit of the low-power mode control register (LPMCR:SPL)
Hi-z: High impedance

# 8.4 Port 9

**Port 9 is an I/O port that can also be used for resource I/O. Each of the port pins can be switched between a resource and an I/O port according to the value of the corresponding bit. This section describes the configuration and registers of Port 9 and provides a block diagram of the pins. The focus is on I/O ports.**

## ■ Port 9 Configuration

Port 9 consists of the following:

- I/O port pins/resource I/O pins (P90, SDA, SO3 to P91, SCL, and SC3)
- Port 9 data register (PDR9)
- Port 9 direction register (DDR9)

## ■ Port 9 Pins

The I/O pins of Port 9 also function as resource I/O pins. The I/O pins cannot be used as an I/O port when they are being used as resource I/O pins.

**Table 8.4-1 Port 9 Pins**

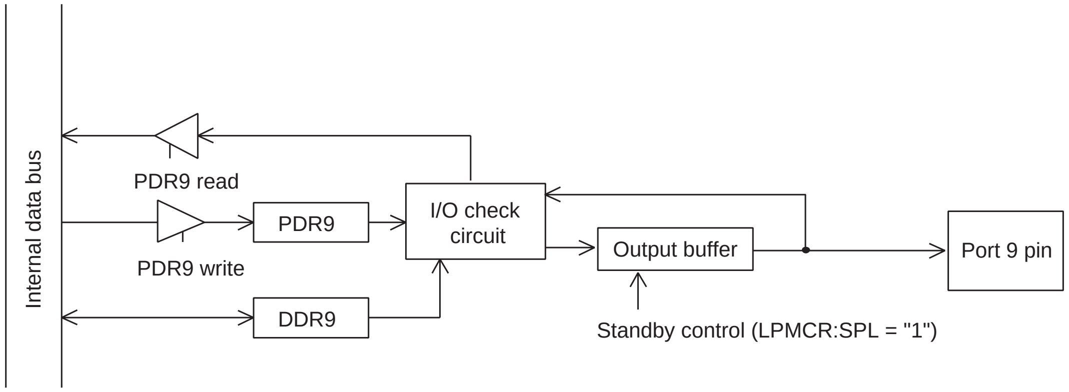
| Port name | Pin name | Port function | | Resource function | | I/O form | | Circuit type |
|---|---|---|---|---|---|---|---|---|
| | | | | | | Input | Output | |
| Port 9 | P90 | P90 | I/O port | SDA/SO3 | $I^2C$ and serial I/O | CMOS (hysteresis) | Nch open drain | G |
| | P91 | P91 | | SCL/SC3 | | | | |

**Note:**

For the circuit type, see Section 1.7 "I/O Circuit Types."

## ■ Block Diagram of the Port 9 Pins

Figure 8.4-1 Block Diagram of the Port 9 Pins



## ■ Port 9 Registers

The Port 9 registers are the Port 9 data register (PDR9) and the Port 9 direction register (DDR9). The bits of each register correspond to the Port 9 pins on a one-to-one basis.

Table 8.4-2 Port 9 Pins and Corresponding Register Bits

| Port name | Register bits and corresponding port pins | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Port 9 | PDR9, DDR9 | bit15 | bit14 | bit13 | bit12 | bit11 | bit10 | bit9 | bit8 |
| | Corresponding pin | - | - | - | - | - | - | P91 | P90 |

# 8.4.1 Port 9 Registers (PDR9 and DDR9)

**This section describes the Port 9 registers.**

## ■ Functions of the Port 9 Registers

### ❍ Port 9 data register (PDR9)

The PDR9 register specifies the output value for each Port 9 pin.

### ❍ Port 9 direction register (DDR9)

The DDR9 register specifies the I/O direction for each Port 9 pin. A pin functions as an output port when the bit corresponding to the pin is set to 1. A pin functions as an input port when the bit corresponding to the pin is set to 0.

**Table 8.4-3 Functions of the Port 9 Registers**

| Register name | Bit value | Read mode | | Write mode | | Address | Initial value |
|---|---|---|---|---|---|---|---|
| | | Input port | Output port | Input port | Output port | | |
| Port 9 data register (PDR9) | 0 | The pin is at the "L" level. | The corresponding bit of PDR9 is set to 0. | The corresponding bit of PDR9 is 0. | The "L" level is output from the pin. | 000009$_H$ | XXXXXXXX$_B$ |
| | 1 | The pin is at the "H" level. | The corresponding bit of PDR9 is set to 1. | The corresponding bit of PDR9 is 1. | Because of Nch open drain, pull-up is required to output the "H" level. | | |
| Port 9 direction register (DDR9) | 0 | The corresponding bit of DDR9 is set to 0. | | Functions as an input port. | | 000019$_H$ | 00000000$_B$ |
| | 1 | The corresponding bit of DDR9 is set to 1. | | Functions as an output port. | | | |

X: Undefined

## 8.4.2 Operation of Port 9

**This section describes the operation of Port 9.**

### ■ Operation of Port 9

❍ **When Port 9 is set as an output port in the Port 9 direction register (DDR9)**

- The value stored in the Port 9 data register (PDR9) is output to the Port 9 pins.
- When Port 9 data register (PDR9) is read, the value stored in the PDR9 is output.

❍ **When Port 9 is set as an input port in the Port 9 direction register (DDR9)**

- The Port 9 pins have high impedance.
- When a value is set in the Port 9 data register (PDR9), the value is retained but is not output to the pin.
- When the PDR9 register is read, the pin input level (0 for "L" or 1 for "H") is output.

**Note:**

If a read-modify-write instruction (such as the bit set instruction) is used to access the PDR9 register, none of the bits specified for output in the DDR9 register are affected. For a bit specified for input in the DDR9 register, however, the pin input level is written to the PDR9 register. Therefore, to change a bit specified for input to output, first write an output value to the PDR9 register and then specify the DDR9 register as an output port.

❍ **Port operation after a reset**

- When the CPU is reset, the DDR9 register is initialized to 00H and the Port 9 pins have high impedance.
- The PDR9 register is not initialized when the CPU is reset. To use the Port 9 as an output port, first write an output value to the PDR9 register and then specify the DDR9 register as an output port.

### ❍ Port operation in stop or time-base timer mode

When the port switches to stop mode or time-base timer mode while the pin status setting bit (SPL) of the low-power mode control register (LPMCR) is set to 1, the pins have high impedance regardless of the value in the Port 9 direction register (DDR9). Note that the input buffer is forcibly blocked off to prevent leakage due to an open circuit.

**Table 8.4-4 States of the Port 9 Pins**

| Pin name | Normal operation | Sleep mode | Stop mode or time-base timer mode (SPL = 0) | Stop mode or time-base timer mode (SPL = 1) |
|---|---|---|---|---|
| P90, P91 | I/O port | I/O port | Input blocking:<br>Output at Hi-z | Input blocking:<br>Held at the "H" level |
| SDA/SO3 | $I^2$C and serial I/O | $I^2$C and serial I/O | | |
| SCL/SC3 | | | | |

SPL: Pin status specification bit of the low-power mode control register (LPMCR:SPL)
Hi-z: High impedance

# 8.5 Port A

**Port A is an I/O port that can also be used for A/D converter analog input. Each of the port pins can be switched between analog input and an I/O port according to the value of the corresponding bit. This section describes the configuration and registers of Port A and provides a block diagram of the pins. The focus is on I/O ports.**

## ■ Port A Configuration

Port A consists of the following:

- I/O port pins and A/D converter analog input pins (PA0, AN0, TMCK to PA7, and AN7)
- Port A data register (PDRA)
- Port A direction register (DDRA)
- Analog input enable register 0 (ADER0)

## ■ Port A Pins

The I/O pins of Port A are also used for A/D converter analog input. The I/O pins cannot be used as an I/O port when they are being used for A/D converter analog input. Conversely, the I/O pins cannot be used for A/D converter analog input when they are being used as an I/O port.

**Table 8.5-1 Port A Pins**

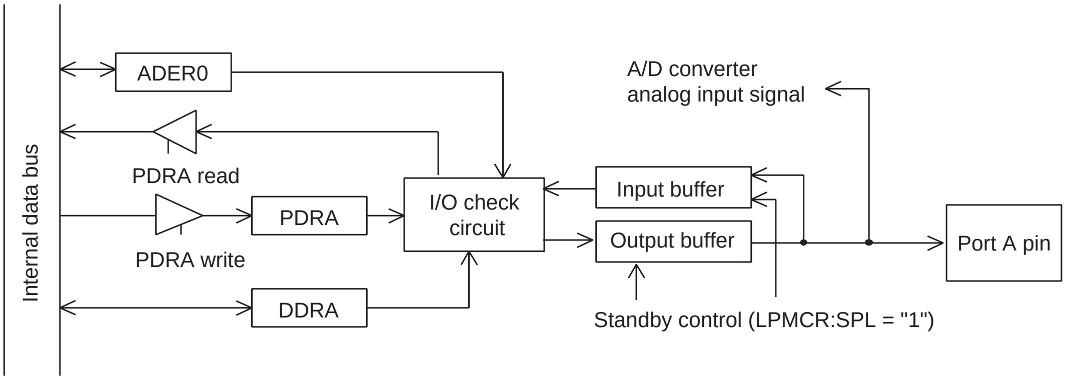
| Port name | Pin name | Port function | | Resource function | | I/O form | | Circuit type |
|---|---|---|---|---|---|---|---|---|
| | | | | | | Input | Output | |
| Port A | PA0/<br>AN0/<br>TMCK | PA0 | I/O port | AN0<br>TMCK | Analog input 0<br>Timing clock output | CMOS (hysteresis) | CMOS | F |
| | PA1/AN1 | PA1 | | AN1 | Analog input 1 | | | |
| | PA2/AN2 | PA2 | | AN2 | Analog input 2 | | | |
| | PA3/AN3 | PA3 | | AN3 | Analog input 3 | | | |
| | PA4/AN4 | PA4 | | AN4 | Analog input 4 | | | |
| | PA5/AN5 | PA5 | | AN5 | Analog input 5 | | | |
| | PA6/AN6 | PA6 | | AN6 | Analog input 6 | | | |
| | PA7/AN7 | PA7 | | AN7 | Analog input 7 | | | |

**Note:**

For the circuit type, see Section 1.7 "I/O Circuit Types."

## ■ Block Diagram of the Port A Pins

Figure 8.5-1 Block Diagram of the Port A Pins



**Note:**

To use a port pin as an input port, set the corresponding bit of the Port A direction register (DDRA) to 0 and the corresponding bit of analog input enable register 0 (ADER0) to 0.

To use a port pin as an analog input pin, set the corresponding bit of the Port A direction register (DDRA) to 0 and the corresponding bit of analog input enable register 0 (ADER0) to 1.

## ■ Port A Registers

The Port A registers are the Port A data register (PDRA), the Port A direction register (DDRA), and analog input enable register 0 (ADER0). The bits of each register correspond to the Port A pins on a one-to-one basis.

**Table 8.5-2 Port A Pins and Corresponding Register Bits**

| Port name | Register bits and corresponding port pins | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Port A | PDRA, DDRA, ADER0 | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 |
| | Corresponding pin | PA7 | PA6 | PA5 | PA4 | PA3 | PA2 | PA1 | PA0 |

## 8.5.1 Port A Registers (PDRA, DDRA and ADER0)

**This section describes the Port A registers.**

### ■ Functions of the Port A Registers

❍ **Port A data register (PDRA)**

The PDRA register specifies the output value for each Port A pin.

❍ **Port A direction register (DDRA)**

The DDRA register specifies the I/O direction for each Port A pin.

A pin functions as an output port when the bit corresponding to the pin is set to 1. A pin functions as an input port when the bit corresponding to the pin is set to 0.

❍ **Analog input enable register 0 (ADER0)**

Each bit of the ADER0 register can be set as an I/O port or as analog input to the A/D converter. A pin is used for analog input when the bit corresponding to the port (pin) is set to 1, and is used as an I/O port when the bit is set to 0.

**Note:**

If a signal at an intermediate level is input when a pin is set as an I/O port, input leakage current flows. To use a pin for analog input, therefore, be sure to set the corresponding bit of the ADER0 register for analog input to the A/D converter.

**Reference:**

When reset, the DDRA register is initialized to $00_H$, the ADER0 register is initialized to $FF_H$, and the A/D converter is set to analog input.

Table 8.5-3 Functions of the Port A Registers

| Register name | Bit value | Read mode | | Write mode | | Address | Initial value |
|---|---|---|---|---|---|---|---|
| | | Input port | Output port | Input port | Output port | | |
| Port A data register (PDRA) | 0 | The pin is at the "L" level. | The corresponding bit of PDRA is set to 0. | The corresponding bit of PDRA is 0. | The "L" level is output from the pin. | 00000A$_H$ | XXXXXXXX$_B$ |
| | 1 | The pin is at the "H" level. | The corresponding bit of PDRA is 1. | The corresponding bit of PDRA is set to 1. | The "H" level is output from the pin. | | |
| Port A direction register (DDRA) | 0 | The corresponding bit of PDRA is set to 0. | | The pin functions as an input port. | | 00001A$_H$ | 00000000$_B$ |
| | 1 | The corresponding bit of PDRA is set to 1. | | The pin functions as an output port. | | | |
| Analog input enable register 0 (ADER0) | 0 | The corresponding bit of ADER0 is set to 0. | | I/O port | | 00001E$_H$ | 11111111$_B$ |
| | 1 | The corresponding bit of ADER0 is set to 1. | | A/D converter analog input 0 to 7 | | | |

X: Undefined

## 8.5.2 Operation of Port A

**This section describes the operation of Port A.**

### ■ Operation of Port A

❍ **When Port A is set as an output port in the Port A direction register (DDRA) and analog input enable register 0 (ADER0)**

- The value stored in the Port A data register (PDRA) is output to the Port A pin.
- When Port A data register (PDRA) is read, the value stored in PDRA is output.

❍ **When Port A is set as an input port in the Port A direction register (DDRA) and analog input enable register 0 (ADER0)**

- The Port A pins have high impedance.
- When a value is set in the Port A data register (PDRA), the value is retained but is not output to the pin.
- When the PDRA register is read, the pin input level (0 for "L" or 1 for "H") is output.
- If a read-modify-write instruction (such as the bit set instruction) is used to access the PDRA register, none of the bits specified for output in the DDRA register are affected. For a bit specified for input in the DDRA register, however, the pin input level is written to the PDRA register. Therefore, to change a bit specified for input to output, first write an output value to the PDRA register and then specify the DDRA register as an output port.

❍ **When Port A is set for analog input to the A/D converter**

To use a port pin for analog input to the A/D converter, set the bit of analog input enable register 0 (ADER0) corresponding to the analog input pin to 1. When the corresponding PDRA bit is read while a pin is set for A/D converter analog input, 0 is read.

❍ **Port operation after a reset**

When the CPU is reset, the DDRA register is initialized to $00_H$ and the ADER0 register is initialized to $FF_H$ so that it can be used for analog input to the A/D converter. To use Port A as an I/O port, set the ADER0 register to $00_H$ to set the port to port I/O mode.

### ❍ Port operation in stop or time-base timer mode

When the port switches to stop mode or time-base timer mode while the pin status setting bit (SPL) of the low-power mode control register (LPMCR) is set to 1, the pins have high impedance regardless of the value in the Port A direction register (DDRA). Note that the input buffer is forcibly blocked off to prevent leakage due to an open circuit.

**Table 8.5-4 States of the Port A pins**

| Pin name | Normal operation | Sleep mode | Stop mode or time-base timer mode (SPL = 0) | Stop mode or time-base timer mode (SPL = 1) |
|---|---|---|---|---|
| PA0 to PA7 | I/O port | I/O port | Input blocking:<br>Out held | Input blocking:<br>Output at Hi-z |
| AN0 to AN7 | Analog input of A/D converter 0 to 7 | Analog input of A/D converter 0 to 7 | | |

SPL: Pin status specification bit of low-power mode control register (LPMCR:SPL)
Hi-z: High impedance

# 8.6 Port B

**Port B is an I/O port that can also be used for A/D converter analog input, a serial interface, and external interrupt input. Each of the port pins can be switched between analog input and an I/O port according to the value of the corresponding bit. This section describes the configuration and registers of Port B and provides a block diagram of the pins. The focus is on I/O ports.**

## ■ Port B Configuration

Port B consists of the following:

- I/O port pins, A/D converter analog input pins, and resource input pins (PB0, AN8 to PB7, AN15, and INT3)
- Port B data register (PDRB)
- Port B direction register (DDRB)
- Analog input enable register 1 (ADER1)

## ■ Port B Pins

The I/O pins of Port B are also used for A/D converter analog input and resource I/O. The I/O pins cannot be used as an I/O port when they are being used for A/D converter analog input or resource I/O. Conversely, the I/O pins cannot be used for A/D converter analog input or resource I/O when they are being used as an I/O port.

**Table 8.6-1 Port B Pins**

| Port name | Pin name | Port function | | Resource function | | I/O form | | Circuit type |
|---|---|---|---|---|---|---|---|---|
| | | | | | | Input | Output | |
| Port B | PB0/AN8 | PB0 | I/O port | AN8 | Analog input 8 | CMOS (hysteresis) | CMOS | F |
| | PB1/AN9 | PB1 | | AN9 | Analog input 9 | | | |
| | PB2/AN10 | PB2 | | AN10 | Analog input 10 | | | |
| | PB3/ AN11/SI2 | PB3 | | AN11<br>SI2 | Analog input 11<br>Serial data input 2 | | | |
| | PB4/ AN12/ SC2/TIN | PB4 | | AN12<br>SC2 | Analog input 12<br>Serial clock output 2 | | | |
| | | | | TIN | Reload timer input 1 | | | |
| | PB5/ AN13/ SO2/TO | PB5 | | AN13 | Analog input 13 | | | |
| | | | | SO2 | Serial data output 2 | | | |
| | | | | TO | Reload timer output 1 | | | |

**Table 8.6-1 Port B Pins (Continued)**

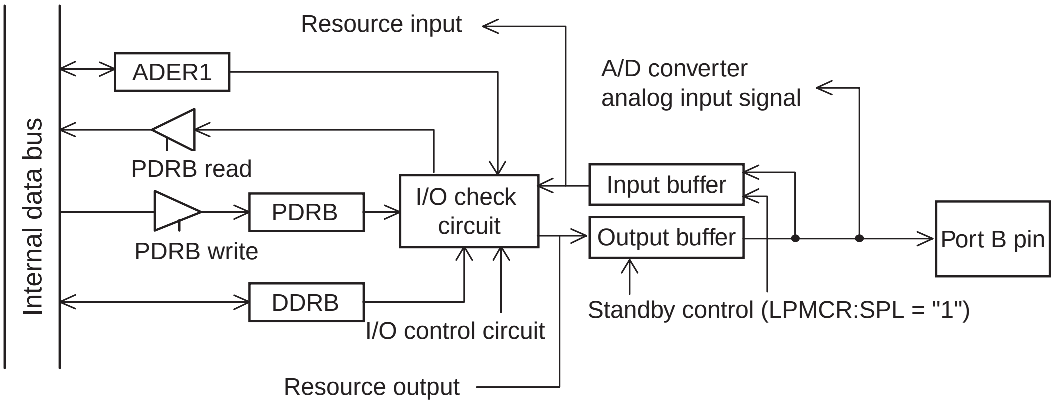
| Port name | Pin name | Port function | | Resource function | | I/O form | | Circuit type |
|---|---|---|---|---|---|---|---|---|
| | | | | | | Input | Output | |
| Port B | PB6/<br>AN14/<br>INT2 | PB6 | I/O port | AN14 | Analog input 14 | CMOS (hysteresis) | CMOS | F |
| | | | | INT2 | External interrupt 2 | | | |
| | PB7/<br>AN15/<br>INT3 | PB7 | | AN15 | Analog input 15 | | | |
| | | | | INT3 | External interrupt 3 | | | |

**Note:**

For the circuit type, see Section 1.7 "I/O Circuit Types."

## ■ Block Diagram of the Port B Pins

**Figure 8.6-1 Block Diagram of the Port B Pins**



**Note:**

To use a port pin as an input port, set the corresponding bit of the Port B direction register (DDRB) to 0 and the corresponding bit of analog input enable register 1 (ADER1) to 0.

To use a port pin as an analog input pin, set the corresponding bit of the Port B direction register (DDRB) to 0 and the corresponding bit of analog input enable register 1 (ADER1) to 1.

## ■ Port B Registers

The Port B registers are the Port B data register (PDRB), Port B direction register (DDRB), and analog input enable register 1 (ADER1). The bits of each register correspond to the Port B pins on a one-to-one basis.

**Table 8.6-2 Port B Pins and Corresponding Register Bits**

| Port name | Register bits and corresponding port pins | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Port B | PDRB, DDRB, DER1 | bit15 | bit14 | bit13 | bit12 | bit11 | bit10 | bit9 | bit8 |
| | Corresponding pin | PB7 | PB6 | PB5 | PB4 | PB3 | PB2 | PB1 | PB0 |

## 8.6.1 Port B Registers (PDRB, DDRB and ADER1)

**This section describes the Port B registers.**

### ■ Functions of the Port B Registers

❍ **Port B data register (PDRB)**

The PDRB register specifies the output value for each Port B pin.

❍ **Port B direction register (DDRB)**

The DDRB register specifies the I/O direction for each Port B pin.

A pin functions as an output port when the bit corresponding to the pin is set to 1.  A pin functions as an input port when the bit corresponding to the pin is set to 0.

❍ **Analog input enable register 0 (ADER1)**

Each bit of the ADER1 register can be set as an I/O port or as analog input to the A/D converter. A pin is used for analog input when the bit corresponding to the port (pin) is set to 1, and is used as an I/O port when the bit is set to 0.

**Note:**

If a signal at an intermediate level is input when a pin is set as an I/O port, input leakage current flows.  To use a pin for analog input, therefore, be sure to set the corresponding bit of the ADER1 register for analog input to the A/D converter.

**Reference:**

When reset, the DDRB register is initialized to $00_H$, the ADER1 register is initialized to $FF_H$, and the A/D converter is set to analog input.

Table 8.6-3 Functions of the Port B Registers

| Register name | Bit value | Read mode | | Write mode | | Address | Initial value |
|---|---|---|---|---|---|---|---|
| | | Input port | Output port | Input port | Output port | | |
| Port B data register (PDRB) | 0 | The pin is at the "L" level. | The corresponding bit of PDRB is set to 0. | The corresponding bit of PDRB is 0. | The "L" level is output from the pin. | $00000B_H$ | $XXXXXXXX_B$ |
| | 1 | The pin is at the "H" level. | The corresponding bit of PDRB is set to 1. | The corresponding bit of PDRB is 1. | The "H" level is output from the pin. | | |
| Port B direction register (DDRB) | 0 | The corresponding bit of DDRB is set to 0. | | The pin functions as an input port. | | $00001B_H$ | $00000000_B$ |
| | 1 | The corresponding bit of DDRB is set to 1. | | The pin functions as an output port. | | | |
| Analog input enable register 1 (ADER1) | 0 | The corresponding bit of ADER1 is set to 0. | | I/O port | | $00001F_H$ | $11111111_B$ |
| | 1 | The corresponding bit of ADER1 is set to 1. | | A/D converter analog input | | | |

X: Undefined

## 8.6.2 Operation of Port B

**This section describes the operation of Port B.**

### ■ Operation of Port B

❍ **When Port B is set as an output port for the Port B direction register (DDRB) and analog input enable register 1 (ADER1)**

- The value stored in the Port B data register (PDRB) is output to the Port B pin.
- When Port B data register (PDRB) is read, the value stored in PDRB is output.

❍ **When Port B is set as an input port in the Port B direction register (DDRB) and analog input enable register 1 (ADER1)**

- The Port B pins have high impedance.
- When a value is set in the Port B data register (PDRB), the value is retained but is not output to the pin.
- When the PDRB register is read, the pin input level (0 for "L" or 1 for "H") is output.

**Note:**

If a read-modify-write instruction (such as the bit set instruction) is used to access the PDRB register, none of the bits specified for output in the DDRB register are affected. For a bit specified for input in the DDRB register, however, the pin input level is written to the PDRB register. Therefore, to change a bit specified for input to output, first write an output value to the PDRB register and then specify the DDRB register as an output port.

❍ **When Port B is set for analog input to the A/D converter**

To use a port pin for analog input to the A/D converter, set the bit of analog input enable register 1 (ADER1) corresponding to the analog input pin to 1. When the corresponding PDRB bit is read while a pin is set for A/D converter analog input, 0 is read.

❍ **Port operation after a reset**

When the CPU is reset, the DDRB register is initialized to $00_H$ and the ADER1 register is initialized to $FF_H$ so that it can be used for analog input to the A/D converter. To use Port B as an I/O port, set the ADER1 register to $00_H$ to set the port to port I/O mode.

### ❍ Port operation in stop or time-base timer mode

When the port switches to stop mode or time-base timer mode while the pin status setting bit (SPL) of the low-power mode control register (LPMCR) is set to 1, the pins have high impedance regardless of the value in the Port B direction register (DDRB). Note that the input buffer is forcibly blocked off to prevent leakage due to an open circuit.

**Table 8.6-4 States of the Port B Pins**

| Pin name | Normal operation | Sleep mode | Stop mode or time-base timer mode (SPL = 0) | Stop mode or time-base timer mode (SPL = 1) |
|---|---|---|---|---|
| PB0 to PB7 | I/O port | I/O port | IInput blocking: Output held | Input blocking: Output at Hi-z |
| AN8 to AN15 | Analog input of A/D converter 8 to 15 | Analog input of A/D converter 8 to 15 | | |
| SI2 | Serial data input 2 | Serial data input 2 | | |
| SC2 | Serial clock I/O 2 | Serial clock I/O 2 | | |
| SO2 | Serial data output 2 | Serial data output 2 | | |
| TIN | Reload timer input 1 | Reload timer input 1 | | |
| TO | Reload timer output 1 | Reload timer output 1 | | |
| INT2, INT3 | External interrupt request 2, 3 | External interrupt request 2, 3 | Input blocking: Output held (Input is enabled when an external interrupt is enabled.) | Input blocking: Output at Hi-z (Input is enabled when an external interrupt is enabled.) |

SPL: Pin status specification bit of low-power mode control register (LPMCR:SPL)
Hi-z: High impedance

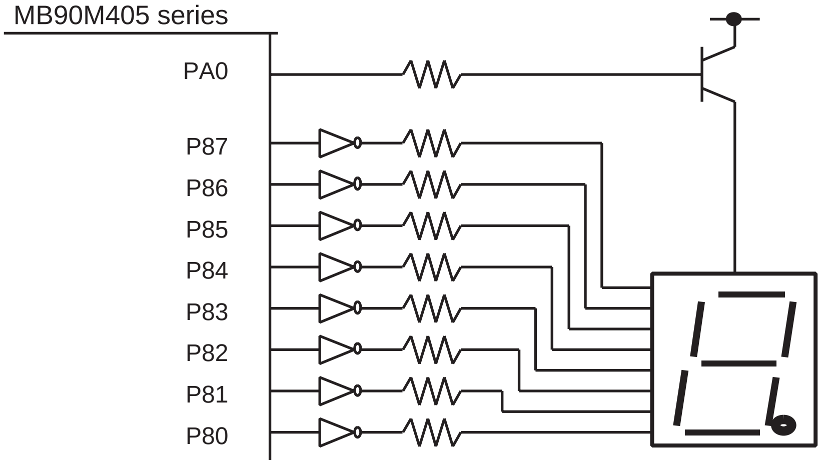
# 8.7 Sample I/O Port Program

**This section provides a sample program that uses I/O ports.**

## ■ Sample I/O Port Program

### ❍ Processing specifications

- Ports 8 and A are used to turn on all segments of a seven-segment (eight-segment if the decimal point is included) LED.
- Pin PA0 corresponds to the anode common pin of the LED, and pins P80 to P87 correspond to the segment pins.



### ❍ Coding example

```
PDR8   EQU       000008H
PDRA   EQU       00000AH
DDR8   EQU       000018H
DDRA   EQU       00001AH
;-------- Main program ----------------------------------------------------------
CODE             CSEG
START:
                                    ; Already initialized
       MOV       I:PDRA, #00000000B  ; Sets PA0 to the "L" level
                                    ; (#xxxxxxx0B)
       MOV       I:DDRA, #11111111B  ; Sets all port-A bits to output mode.
       MOV       I:PDR8, #11111111B  ; Sets all port-8 bits to 1.
       MOV       I:DDR8, #11111111B  ; Sets all port-8 bits to output mode.
CODE             ENDS
;--------------------------------------------------------------------------------
       END       START
```

# CHAPTER 9 SERIAL I/O

**This chapter describes the functions and operations of the serial I/O unit of the MB90M405 series.**

## 9.1 Overview of the Serial I/O Unit

**The serial I/O unit is a serial I/O interface that can transfer data in an 8-bit/2-channel configuration in clock synchronous mode. Moreover, selection between LSB-first and MSB-first transfer for data transfer is possible.**

### ■ Overview of Serial I/O Unit

The two types of serial I/O operating modes are as follows:

❍ **Internal shift clock mode:**

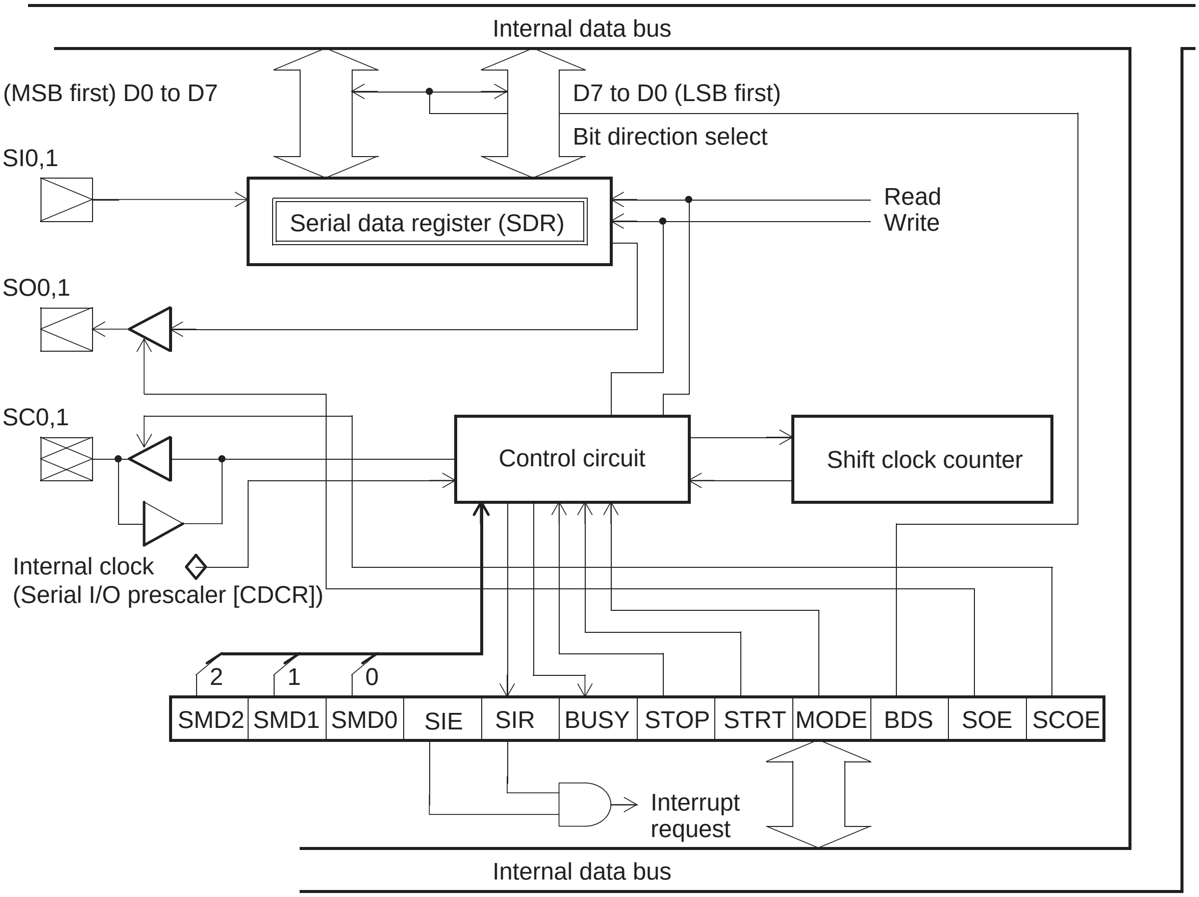
Transfers data in synchronization with the internal clock (communication prescaler).

❍ **External shift clock mode:**

Transfers data in synchronization with the clock input from an external pin (SC). In this mode, general-purpose ports that share the external pin (SC) can perform transfer operations using CPU instructions (based on the timing of execution of the port reversal instruction).

### ■ Block Diagram of the Serial I/O Unit

**Figure 9.1-1 Block Diagram of the Serial I/O Unit**



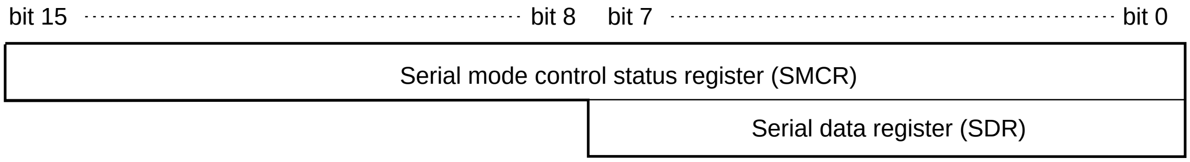
# 9.2 Registers of the Serial I/O Unit

**The operations of the serial I/O unit can be specified using the following registers:**

- **Serial mode control status register (SMCR), higher**
- **Serial mode control status register (SMCR), lower**
- **Serial shift data register (SDR)**

## ■ Registers of the Serial I/O Unit

**Figure 9.2-1 Registers of the Serial I/O Unit**


bit 15
bit 8
bit 7
bit 0
Serial mode control status register (SMCR)
Serial data register (SDR)

## 9.2.1 Serial Mode Control Status Register (SMCR)

**The serial mode control status register (SMCR) controls the operating mode for serial I/O transfer.**

### ■ Serial Mode Control Status Register, Higher (SMCR)

**Figure 9.2-2 Serial Mode Control Status Register, Higher (SMCR)**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | SMD2 | SMD1 | SMD0 | SIE | SIR | BUSY | STOP | STRT | $00000010_B$ |
| | R/W | R/W | R/W | R/W | R/W | R | R/W | R/W | |

| STRT | Start bit |
|---|---|
| 0 | Stop of serial transfer |
| 1 | Start of serial transfer |

| STOP | Stop bit |
|---|---|
| 0 | Normal operation |
| 1 | Stop of transfer |

| BUSY | Transfer status bit |
|---|---|
| 0 | Stopped or serial data register R/W wait |
| 1 | Serial transfer |

| SIR | Serial I/O interrupt request flag bit | |
|---|---|---|
| | Read | Write |
| 0 | No interrupt request exists. | Clears an interrupt request. |
| 1 | An interrupt request exists. | Does not affect operation. |

| SIE | Serial I/O interrupt request enable bit |
|---|---|
| 0 | Disables interrupt requests. |
| 1 | Enables interrupt requests. |

| SMD2 | SMD1 | SMD0 | Serial shift clock mode setting bit | | |
|---|---|---|---|---|---|
| | | | ϕ = 16MHz(div = 8) | ϕ = 8MHz(div = 4) | ϕ = 4MHz(div = 4) |
| 0 | 0 | 0 | 1 MHz | 1 MHz | 500 KMHz |
| 0 | 0 | 1 | 500 KMHz | 500 KMHz | 250 KMHz |
| 0 | 1 | 0 | 125 KMHz | 125 KMHz | 62.5 KMHz |
| 0 | 1 | 1 | 62.5 KMHz | 62.5 KMHz | 31.25 KMHz |
| 1 | 0 | 0 | 31.25 KMHz | 31.25 KMHz | 15.625 KMHz |
| 1 | 0 | 1 | External shift clock mode | | |
| 1 | 1 | 0 | Reserved | | |
| 1 | 1 | 1 | Reserved | | |

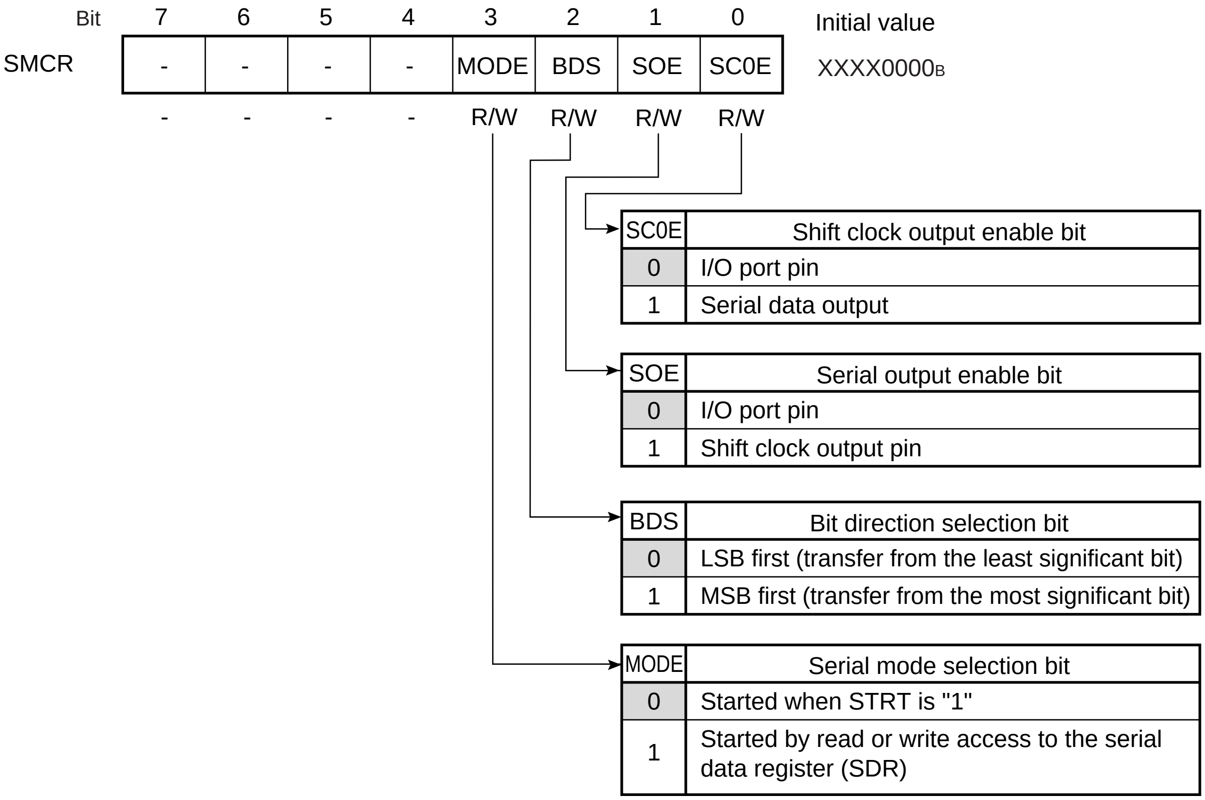
| | | |
|---|---|---|
| R/W | : | Read/write enabled |
| R | : | Read only |
| ▨ | : | Initial value |
| ϕ | : | Machine clock frequency |

**Table 9.2-1 Functional Description of High Order Bits in Serial Mode Control Status Register (SMCR)**

| No. | Bit name | Function |
|---|---|---|
| bit15 to bit13 | SMD2, SMD1, SMD0: Serial shift clock mode setting bits | • Selects a serial shift clock mode.<br>• The settings of the serial shift clock mode setting bits (SMD2, SMD1 and SMD0) and Communication Prescaler Control Register (CDCR0/ CDCR1) determine the transfer speed of the shift clock.<br>• A reset initializes these bits to 000B.<br>• Writing to these bits during transfer is prohibited (Do not write these bits).<br>• One of five internal shift clocks or an external shift clock can be specified. Do not set SMD2 to SMD0 to 110B or 111B because these settings are reserved.<br>• When SCOE is set to "0" for clock selection, shift operations manipulate ports that share the SC pin to allow shift operations for each instruction. |
| bit12 | SIE: Serial I/O interrupt request enable bit | • This bit enables serial I/O interrupt requests.<br>• When this bit is set to "1", setting the serial I/O interrupt request flag bit (SIR) to "1" outputs an interrupt request.<br>• A reset initializes this bit to "0". |
| bit11 | SIR: Serial I/O interrupt request flag bit | • This is a flag bit for an interrupt request to the serial I/O.<br>• This bit is set to "1" when serial data transfer ends.<br>• While the serial I/O interrupt request enable bit (SIE) is "1", set this bit to "1" to output an interrupt request to the CPU.<br>• When the MODE bit is "0", set this bit to "0" to clear the interrupt request.<br>• When the MODE bit is "1", reading or writing the SDR register clears the interrupt flag bit to "0".<br>• Regardless of the value of the MODE bit, a reset or setting of the STOP bit to "1" clears the interrupt flag bit to "0".<br>• Setting this bit to "0" clears the interrupt flag bit to "0".<br>• Setting this bit to "1" does not affect operation.<br>• The read value is always "1". |
| bit10 | BUSY: Transfer status bit | • This bit is "1" while a serial transfer is being executed.<br>• A reset initializes this bit to "0". |
| bit9 | STOP: Stop bit | • This bit forcibly stops serial transfer.<br>• Setting this bit to "1" puts serial transfer into the stopped state with STOP set to "1".<br>• A reset initializes this bit to "1". |
| bit8 | STRT: Start bit | • This bit starts serial transfer.<br>• Setting this bit to "1" in the stopped state starts transfer.<br>• Setting this bit to "1" during serial transfer or in the serial shift register R/W wait state causes the written value to be ignored.<br>• Setting this bit to "0" does not affect operation.<br>• The read value is always "0". |

## ■ Serial Mode Control Status Register Lower (SMCR)

**Figure 9.2-3  Serial Mode Control Status Register Lower (SMCR)**



| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| SMCR | - | - | - | - | MODE | BDS | SOE | SC0E | XXXX0000B |
| | - | - | - | - | R/W | R/W | R/W | R/W | |

| SC0E | Shift clock output enable bit |
|---|---|
| 0 | I/O port pin |
| 1 | Serial data output |

| SOE | Serial output enable bit |
|---|---|
| 0 | I/O port pin |
| 1 | Shift clock output pin |

| BDS | Bit direction selection bit |
|---|---|
| 0 | LSB first (transfer from the least significant bit) |
| 1 | MSB first (transfer from the most significant bit) |

| MODE | Serial mode selection bit |
|---|---|
| 0 | Started when STRT is "1" |
| 1 | Started by read or write access to the serial data register (SDR) |

R/W : Read/write enabled

▨ : Initial value

Table 9.2-2 Functional Description of Lower Bits in Serial Mode Control Status Register (SMCR)

| No. | Bit name | Function |
|---|---|---|
| bit7 to bit4 | -:<br>Undefined bit | • The read value is undefined.<br>• Setting this bit does not affect the transfer operation. |
| bit3 | MODE:<br>Serial mode selection bit | • Use this bit to select the condition for resuming operation from the stopped state.<br>• If this bit is "0", starting is performed by setting STRT to "1".<br>• If this bit is "1", reading or writing the serial data register (SDR) starts the transfer operation.<br>• Rewriting this bit during transfer is prohibited.<br>• A reset initializes this bit to "0".<br>• Set this bit to "1" to start the extended intelligent I/O service. |
| bit2 | BDS:<br>Bit direction selection bit | • This bit selects the serial data transfer direction.<br>• If this bit is "0", data is transferred starting with the least significant bit (LSB first).<br>• If this bit is "1", data is transferred starting with the most significant bit (MSB first).<br>• Set the BDS bit before writing data to the SDR register. |
| bit1 | SOE:<br>Serial output enable bit | • This bit controls the output of the serial I/O output external pins (SO2 and SO3).<br>• If this bit is "0", the pin serves as the I/O port pin.<br>• If this bit is "1", the pin serves as the serial data output pin.<br>• A reset initializes this bit to "0". |
| bit0 | SCOE:<br>Shift clock output enable bit | • This bit controls the output of the shift clock I/O external pins (SC2 and SC3).<br>• If this bit is "0", the pin serves as the I/O port pin.<br>• If this bit is "1", the pin serves as the serial data output pin.<br>• Set this bit to "0" to perform transfer for each instruction in external shift clock mode.<br>• A reset initializes this bit to "0". |

## 9.2.2 Serial Shift Data Register (SDR)

**The serial shift data register (SDR) retains serial I/O transfer data.  Writing to and reading the SDR is prohibited during transfer.**

### ■ Serial Shift Data Register (SDR)

**Figure 9.2-4  Serial Shift Data Register (SDR)**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | $\text{XXXXXXXX}_B$ |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

# 9.3 Communication Prescaler Control Register (CDCR0/ CDCR1)

**The Communication Prescaler Control Register (CDCR0/CDCR1) provides a serial I/O shift clock.**
**The operation clock for serial I/O operation can be obtained by dividing the output of the machine clock. The serial I/O unit is designed to obtain a fixed baud rate for each of the machine clocks in the system. The Communication Prescaler Control Register (CDCR0/CDCR1) controls division of the machine clock.**

## ■ Communication Prescaler Control Register (CDCR0/CDCR1)

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | MD | - | - | - | Reserved | DIV2 | DIV1 | DIV0 | $0XXX0000_B$ |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

**Table 9.3-1 Functional Description of Bits in Communication Prescaler Control Register (CDCR0/ CDCR1)**

| No. | Bit name | Function |
|---|---|---|
| bit15 | MD:<br>Communication prescaler operation enable bit | • Use this bit to enable operation of the communication prescaler.<br>• If this bit is set to "1", the communication prescaler is enabled.<br>• If this bit is set to "0", the communication prescaler is disabled. |
| bit14<br>bit13<br>bit12 | -:<br>Undefined bit | • The read value is undefined.<br>• The value of this bit does not affect operation. |
| bit11 | Reserved:<br>Reserved bit | • Be sure to set this bit to "0". |
| bit10<br>bit9<br>bit8 | DIV2 to DIV0:<br>Division ratio setting bit | • Use this bit to set the division ratio of the machine clock.<br>• For information on the setting values, see Table 9.3-2 "Communication Prescaler." |

**Table 9.3-2  Machine Clock Division Ratio**

| MD | DIV2 | DIV1 | DIV0 | div |
|---|---|---|---|---|
| 0 | - | - | - | Stopped |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 2 |
| 1 | 0 | 1 | 0 | 3 |
| 1 | 0 | 1 | 1 | 4 |
| 1 | 1 | 0 | 0 | 5 |
| 1 | 1 | 0 | 1 | 6 |
| 1 | 1 | 1 | 0 | 7 |
| 1 | 1 | 1 | 1 | 8 |

div:  Machine clock division ratio

**Note:**

If the division ratio has been changed, wait one cycle of the divide-by-two machine clock for clock stabilization before performing transfer.
The CDCR0 is a prescaler for the serial I/O channel 2 (also serve as the UART channel 0).
The CDCR1 is a prescaler for the serial I/O channel 3 (also serve as the UART channel 1).

# 9.4 Operation of the Serial I/O Unit

**The serial I/O unit consists of the serial mode control status register (SMCR) and the shift data register (SDR). The serial I/O unit is used to input and output 8-bit serial data.**

## ■ Operation of the Serial I/O Unit

The following explains how serial data is input and output. The contents of the shift data register are output to the serial output pin (SO pin) in bit serial mode in synchronization with a falling edge of the serial shift clock (external or internal clock). Data is input from the serial input pin (SI pin) in bit serial mode in synchronization with a rising edge of the clock. The shift direction (transfer from the MSB or LSB) can be selected using the bit direction bit (BDS) in the serial mode control status register lower (SMCR).

When transfer ends, the serial I/O unit enters the stopped state or data register R/W wait state according to the setting of the serial mode selection bit (MODE) in the serial mode control status register lower (SMCR). To enter the transfer state from the various states, do the following:

- To return from the stopped state, set the stop bit (STOP) to "0" and the start bit (STRT) to "0" (The settings for STOP and STRT can be made simultaneously).
- To return from the serial data register (SDR) wait state, read or write the data register.

## 9.4.1 Shift Clock

**The shift clock operates in two modes: internal shift clock mode and external shift clock mode. A mode can be specified in the serial mode control status register (SMCR). Switch the mode while the serial I/O unit is stopped. Check for the stopped state by reading the transfer status bit (BUSY).**

### ■ Internal Shift Clock Mode

In internal shift clock mode, the shift clock with a 50% duty cycle can be supplied for synchronous timing output from the SC pin. One-bit data is transferred for each clock. Calculate the transfer rate as follows:

$$\text{transfer-rate (s)} = \frac{\text{div} \times \text{A}}{\text{machine-clock-frequency (Hz)}}$$

A, the divide factor specified in the serial shift clock mode setting bits (SMD0 to SMD2), is 2, 4, 16, or 32.

### ■ External Shift Clock Mode

In external shift clock mode, one-bit data is transferred for each clock in synchronization with the external shift clock that is input from the SC pin. A transfer rate up to 1/5 machine cycles is available. For example, a transfer rate up to 2 MHz is available when one machine cycle is 0.1 μs.

Data can also be transferred for each instruction. Transfer data for each instruction as follows:

1. Select the external shift clock mode and set the shift clock output enable bit (SCOE) in the serial mode control status register (SMCR) to "0".
2. Then, set the direction register for one of the ports that share the ISC pin to "1" to set the port to output mode.

After making the above settings, set the port data register (PDR) to "1" and then "0". The port value that is output to the SC pin is then fetched as the external clock and data is transferred. Start the shift clock at the "H" level.

**Note:**

Writing to the serial mode control register (SMCR) and the serial shift data register (SDR) is prohibited during serial I/O operation.

## 9.4.2 Operating States of the Serial I/O Unit

**The serial I/O unit operates in the following four states:**

- **STOP state**
- **Stopped state**
- **R/W wait state of the SDR register**
- **Transfer state**

### ■ STOP State

When a reset occurs or the stop bit (STOP) in the serial mode control status register (SMCR) is set to "1", the shift counter is initialized and SIR is set to "0". Have the serial I/O unit return from the STOP state by setting STOP to "0" and STRT to "1" (the settings for these bits can be made simultaneously). No transfer operation occurs when the stop bit (STOP) is set to "1" and the start bit (STRT) is set to "1" because STOP has a higher priority than STRT.

### ■ Stopped State

While the serial mode selection bit (MODE) is set to "0", BUSY is set to "0" and SIR is set to "1" in the serial mode control status register (SMCR) at the end of transfer. After that, the counter is initialized and the serial I/O enters the stopped state. Set STRT to "1" to have the serial I/O unit return from the stopped state and resume transfer.

### ■ R/W Wait State of the Serial Data Register

While the serial mode selection bit (MODE) is set to "1", BUSY is set to "0" and SIR is set to "1" in the serial mode control status register (SMCR) at the end of serial transfer. After that, the serial I/O unit enters the SDR register R/W wait state. If the setting in the interrupt enable register is Enabled, this block outputs an interrupt signal.

To have the serial I/O unit return from the R/W wait state and resume transfer, read or write the SDR register to set BUSY to "1".

## ■ Transfer State

While BUSY is set to "1", the serial I/O unit is transferring serial data. Depending on the setting of the serial mode selection bit (MODE), the serial I/O enters either the stopped or the R/W wait state.

**Figure 9.4-1 State Transition Diagram of Serial I/O Operation**

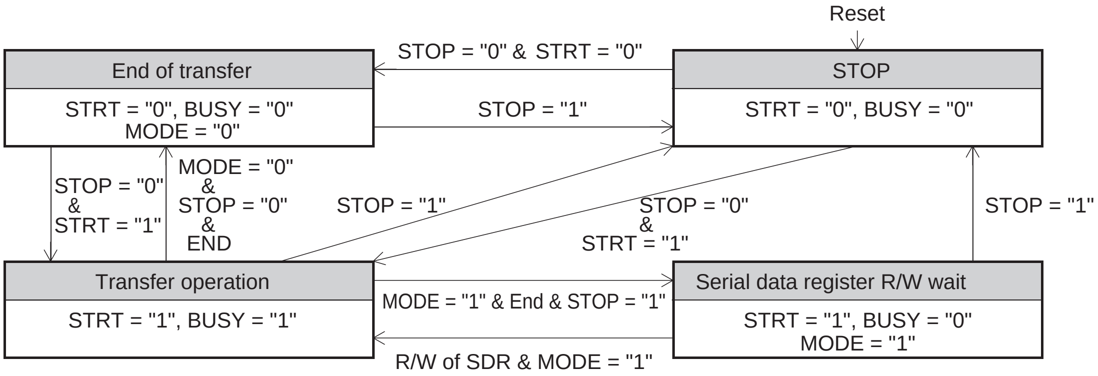
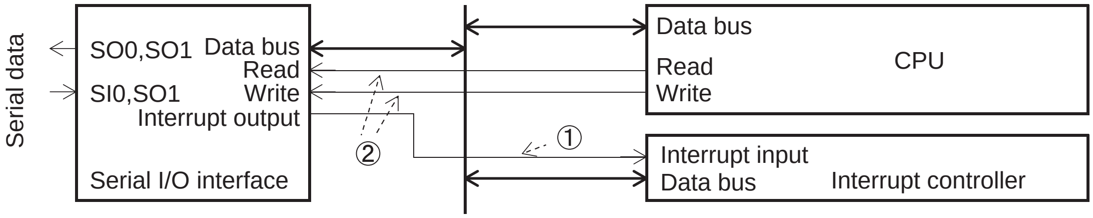


**Figure 9.4-2 Concept of Reading from and Writing to the Serial Data Register**



1. When MODE is set to "1", transfer is completed according to the shift clock counter. Then, SIR is set to "1" and the serial I/O enters the read/write wait state. When the serial I/O interrupt enable bit (SIE) is set to "1", an interrupt signal is generated. However, no interrupt signal is generated if SIE is inactive or transfer is interrupted by setting of the stop bit (STOP) to "1".
2. When the serial shift data register (SDR) is read or written, the interrupt request is cleared and serial transfer is resumed.

## 9.4.3 Start/Stop Timing of Shift Operation

**To start the shift operation, set the stop bit (STOP) to "0" and the start bit (STRT) to "1" in the serial mode control status register (SMCR). The shift operation stops if STOP is set to "1" or when transfer terminates.**

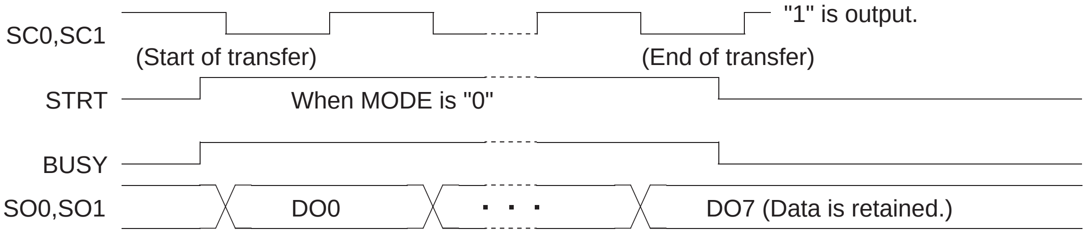
- **Stop by setting STOP to "1": The shift operation stops while SIR remains at "0" regardless of the value the serial mode selection bit (MODE) is set to.**
- **Stop at the end of transfer: The shift operation stops after SIR is set to "1" regardless of the value the serial mode selection bit (MODE) is set to.**

**The transfer status bit (BUSY) is set to "1" in serial transfer state or "0" in stopped or R/W wait state regardless of the value the serial mode selection bit (MODE) is set to. To check the transfer state, read the BUSY bit.**

### ■ Start/Stop Timing of Shift Operation

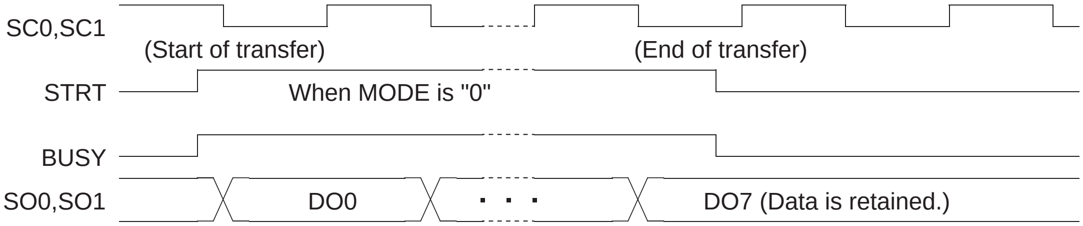
❍ **Internal shift clock mode (LSB first)**

**Figure 9.4-3 Timing of Shift Operation (Internal Clock)**



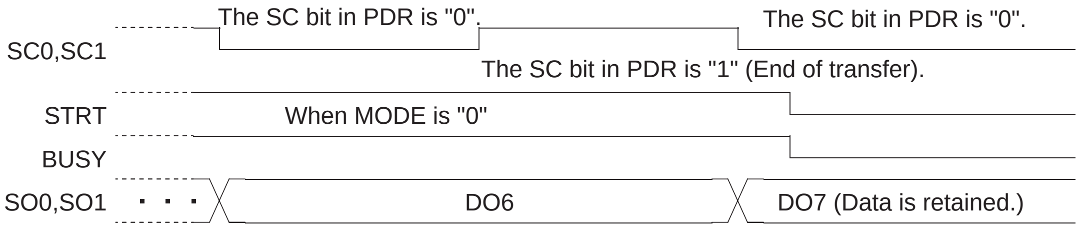
❍ **Internal shift clock mode (LSB first)**

**Figure 9.4-4 Timing of Shift Operation (External Clock)**



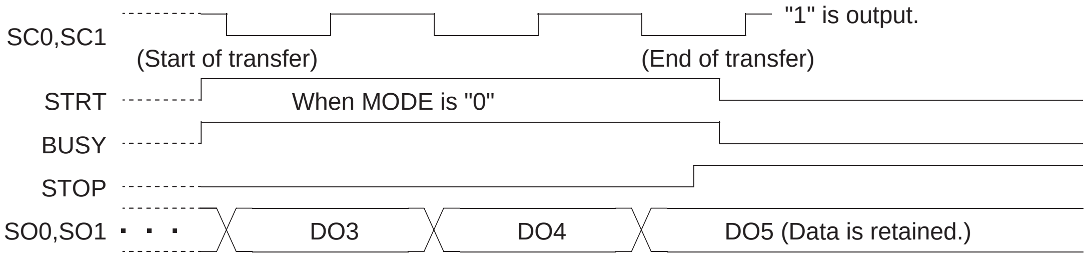
❍ **When shift operation is performed in accordance with instructions in external shift clock mode (LSB first)**

**Figure 9.4-5 Timing of Shift Operation (When a Shift Operation Is Performed in Accordance with Instructions in External Shift Clock Mode)**


SC0,SC1
The SC bit in PDR is "0".
The SC bit in PDR is "0".
The SC bit in PDR is "1" (End of transfer).
STRT
When MODE is "0"
BUSY
SO0,SO1
DO6
DO7 (Data is retained.)

❍ **Stop by setting STOP to "1" (LSB first, internal clock mode)**

**Figure 9.4-6 Stop Timing When the Stop Bit (STOP) Is Set to "1"**



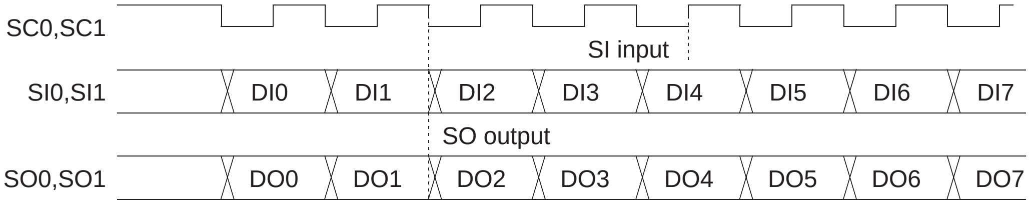
Note: DO7 to DO0 represent output data.

## ■ Serial Data I/O Timings

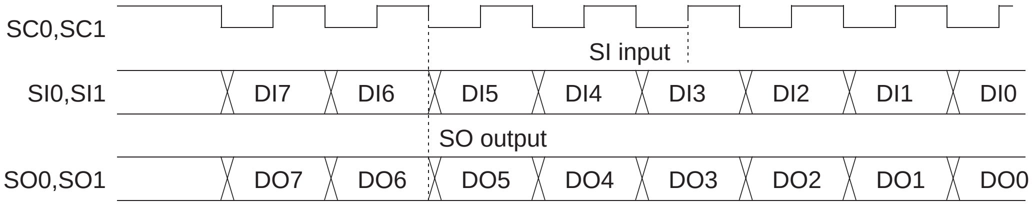
During serial data transfer, data is output from the serial output pin (SO) on a falling edge of the shift clock and data is input from the serial input pin (SI) on a rising or falling edge (determined beforehand).

**Figure 9.4-7 Timing of Serial Data I/O Shift**

○ LSB first (when the BDS bit is "0")



○ MSB first (when the BDS bit is "1")

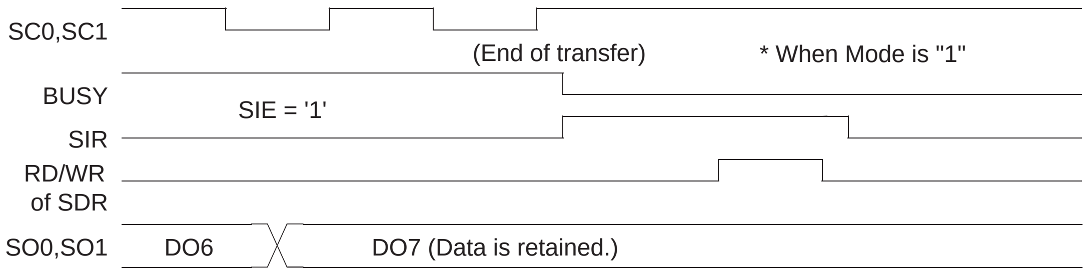


## 9.4.4 Interrupt Function of the Serial I/O Unit

**The serial I/O unit can output interrupt requests to the CPU. If, at the end of data transfer, the serial I/O interrupt request flag bit (SIR), which is an interrupt flag, is set to "1" and the serial I/O interrupt enable bit (SIE) in the serial mode control status register (SMCR) is set to "1", the serial I/O outputs an interrupt request to the CPU.**

### ■ Interrupt Function of the Serial I/O Unit

**Figure 9.4-8 Timing of Serial I/O Interrupt Signal Output**



# CHAPTER 10 TIMEBASE TIMER

**This chapter describes the functions and operation of the timebase timer of the MB90M405 series.**

# 10.1 Overview of the Timebase Timer

**The timebase timer is an 18-bit free-running counter that counts up in synchronization with the main clock. The timer has two functions: An interval timer function that can select one of four intervals and a function for supplying clocks to the oscillation stabilization interval timer and the watchdog timer.**

## ■ Interval Timer Function

The interval timer function repeatedly generates an interrupt request at a given interval.

- An interrupt request is generated when the interval timer bit for the timebase counter overflows.
- The interval timer bit (interval) can be selected from four types.

**Table 10.1-1 Intervals for the Timebase Timer**

| Main clock cycle | Interval cycle |
|---|---|
| 2/HCLK (0.5 µs) | $2^{12}$/HCLK (Approx. 0.97 ms) |
| | $2^{14}$/HCLK (Approx. 3.90 ms) |
| | $2^{16}$/HCLK (Approx. 15.62 ms) |
| | $2^{19}$/HCLK (Approx. 125.00 ms) |

HCLK: Oscillation clock frequency
Values in parentheses are for a 4.194 MHz oscillation clock.

## ■ Clock Supply Function

The clock supply function supplies clocks to the oscillation settling time timer and to some peripheral functions.

**Table 10.1-2 Clock Cycle Time Supplied from the Timebase Timer**

| Clock destination | Clock cycle time | Remarks |
|---|---|---|
| Oscillation setting time | $2^{13}$/HCLK (Approx. 1.95 ms) | Oscillation settling time for ceramic vibrator |
| | $2^{15}$/HCLK (Approx. 7.81 ms) | Oscillation settling time for crystal vibrator |
| | $2^{18}$/HCLK (Approx. 62.50 ms) | |
| Watchdog timer | $2^{12}$/HCLK (Approx. 0.97 ms) | Count-up clock for watchdog timer |
| | $2^{14}$/HCLK (Approx. 3.90 ms) | |
| | $2^{16}$/HCLK (Approx. 15.62 ms) | |
| | $2^{19}$/HCLK (Approx. 125.00 ms) | |

HCLK: Oscillation clock frequency
Values in parentheses occurs during operation of the 4.194 MHz oscillation clock.

**Reference:**

The oscillation settling time is the yardstick because the oscillation cycle time is unstable as soon as oscillation starts.

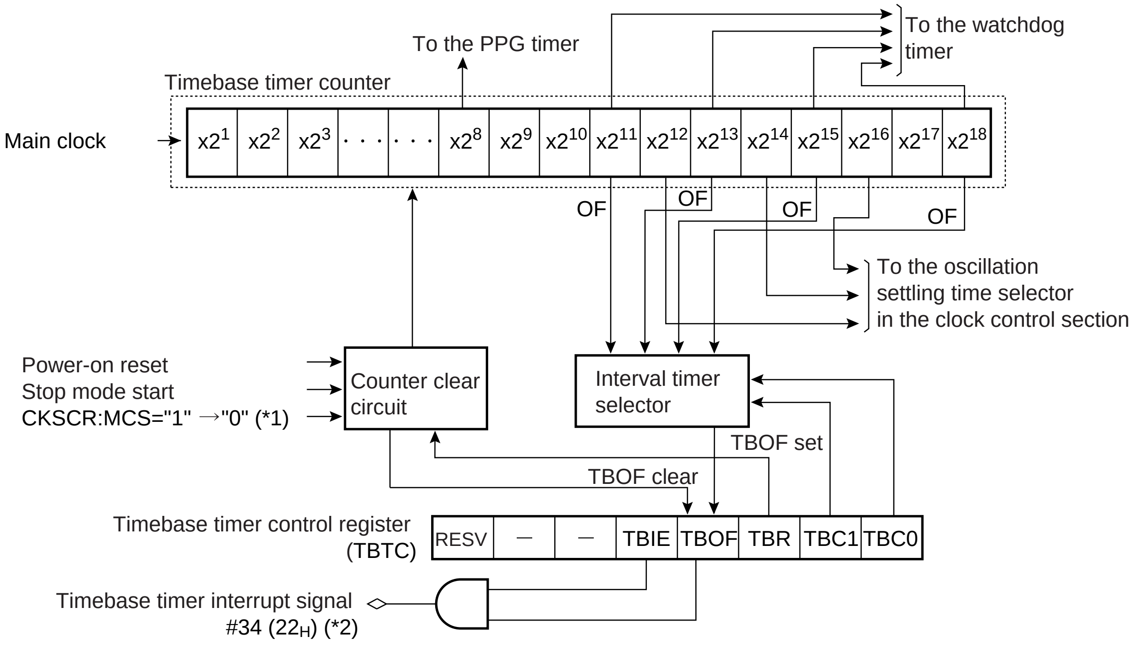
# 10.2 Configuration of the Timebase Timer

**The timebase timer consists of the following four blocks:**

- **Timebase timer counter**
- **Counter clear circuit**
- **Interval timer selector**
- **Timebase timer control register (TBTC)**

## ■ Block Diagram of the Timebase Timer

**Figure 10.2-1 Block Diagram of the Timebase Timer**


To the PPG timer
To the watchdog timer
Timebase timer counter
Main clock
x2^1 x2^2 x2^3 · · · · · · x2^8 x2^9 x2^10 x2^11 x2^12 x2^13 x2^14 x2^15 x2^16 x2^17 x2^18
OF
OF
OF
OF
To the oscillation settling time selector in the clock control section
Power-on reset
Stop mode start
CKSCR:MCS="1" →"0" (*1)
Counter clear circuit
Interval timer selector
TBOF set
TBOF clear
Timebase timer control register (TBTC)
RESV — — TBIE TBOF TBR TBC1 TBC0
Timebase timer interrupt signal #34 (22H) (*2)

- : Undefined bit
OF : Overflow
*1 Switching of the machine clock from the oscillation clock to the PLL clock
*2 Interrupt number

### ❍ Timebase timer counter

An 18-bit up counter that uses the main clock as the count clock

### ❍ Counter clear circuit

Clears the timebase timer counter by setting the timebase timer initialization bit (TBR) of the timebase timer control register (TBTC) to "0", performing a power-on reset, sending the CPU into stop mode (LPMCR: STP = "1"), and changing the machine clock from the main clock to the PLL clock (CKSCR: MCS = "1" -> "0").

### ❍ Interval timer selector

Selects one of four outputs of the timebase timer counter. An overflow of the selected bit becomes an interrupt cause.

- **Timebase timer control register (TBTC)**

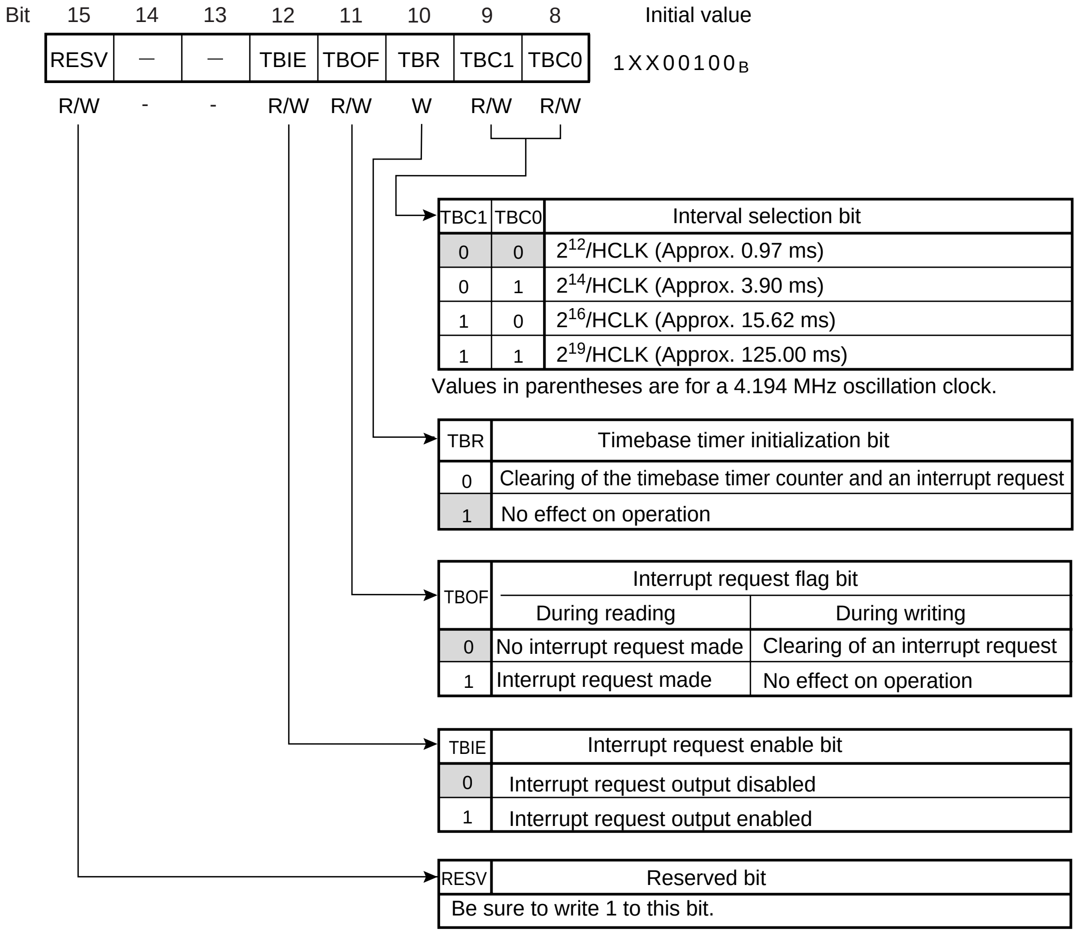
  Selects the interval, clears the timebase timer counter, controls an interrupt request, and checks the status.

# 10.3 Timebase Timer Control Register (TBTC)

**The timebase timer control register (TBTC) selects the interval, clears the timebase timer counter, controls an interrupt request, and checks the status.**

## ■ Timebase Timer Control Register (TBTC)

**Figure 10.3-1 Timebase Timer Control Register (TBTC)**



| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | RESV | — | — | TBIE | TBOF | TBR | TBC1 | TBC0 | $1XX00100_B$ |
| | R/W | - | - | R/W | R/W | W | R/W | R/W | |

| TBC1 | TBC0 | Interval selection bit |
|---|---|---|
| 0 | 0 | $2^{12}$/HCLK (Approx. 0.97 ms) |
| 0 | 1 | $2^{14}$/HCLK (Approx. 3.90 ms) |
| 1 | 0 | $2^{16}$/HCLK (Approx. 15.62 ms) |
| 1 | 1 | $2^{19}$/HCLK (Approx. 125.00 ms) |

Values in parentheses are for a 4.194 MHz oscillation clock.

| TBR | Timebase timer initialization bit |
|---|---|
| 0 | Clearing of the timebase timer counter and an interrupt request |
| 1 | No effect on operation |

| TBOF | Interrupt request flag bit | |
|---|---|---|
| | During reading | During writing |
| 0 | No interrupt request made | Clearing of an interrupt request |
| 1 | Interrupt request made | No effect on operation |

| TBIE | Interrupt request enable bit |
|---|---|
| 0 | Interrupt request output disabled |
| 1 | Interrupt request output enabled |

| RESV | Reserved bit |
|---|---|
| Be sure to write 1 to this bit. | |

| | | |
|---|---|---|
| R/W | : | Read/write |
| W | : | Write only |
| X | : | Undefined |
| - | : | Not used |
| ▨ | : | Initial value |
| HCLK | : | Oscillation clock frequency |

Table 10.3-1 Function Description of Each Bit in the Timebase Timer Control Register (TBTC)

| Bit name | | Function |
|---|---|---|
| bit15 | RESV:<br>Reserved bit | • Be sure to write 1 to this bit. |
| bit14<br>bit13 | Not used | • When read, the value is undefined.<br>• Writing has no effect on operation. |
| bit12 | TBIE:<br>Interrupt request enable bit | • This bit enables interrupt requests.<br>• When this bit and the interrupt request flag bit (TBOF) are 1, an interrupt request is output. |
| bit11 | TBOF:<br>Interrupt request flag bit | • TBOF is a flag bit for an interrupt request.<br>• This bit is set to "1" when the interval timer bit selected for the timebase timer counter overflows.<br>• When the interrupt request enable bit (TBIE) is set to "1", an interrupt request is output.<br>• Set this bit to "0" to clear an interrupt request.<br>• When this bit is set to "1", there is no effect on operation.<br>**Note:**<br>• To set this bit to "0", set the interrupt request enable bit (TBIE) or the interrupt level mask register (ILM) of the processor status (PS) to Disabled.<br>• This bit is cleared to "0" when a transition to stop mode occurs, the timebase timer is cleared due to the timebase timer initialization bit (TBR), or a reset occurs. |
| bit10 | TBR:<br>Timebase timer initialization bit | • Used to clear the timebase timer counter.<br>• When this bit is set to "0", the timebase timer counter is cleared to $00000_H$ and the interrupt request flag bit (TBOF) is cleared to "0".<br>• Writing 1 does not affect operation.<br>**[Reference]**<br>The read value is always 1. |
| bit9<br>bit8 | TBC1, TBC0:<br>Interval selection bit | • Used to select an interval timer cycle.<br>• The bit for the interval timer of the timebase timer counter is specified.<br>• Four types of interval can be selected. |

# 10.4 Timebase Timer Interrupts

**The timebase timer can output an interrupt request when an overflow of the interval counter selected with the interval time setting bit occurs (interval timer function).**

## ■ Timebase Timer Interrupts

The interrupt request flag bit (TBOF) of the timebase timer control register (TBTC) is set to "1" when the timebase timer counter counts up on the main clock and an overflow of the specified interval timer occurs. If the TBOF bit is set to "1" while the interrupt request enable bit is set to Enabled (TBTC:TBIE="1"), an interrupt request (interrupt number #34) is output to the CPU and the interrupt processing routine is executed. In the interrupt processing routine, set the TBOF bit to "0" to clear the interrupt request. When the specified interval timer bit overflows, the TBOF bit is set to "1" regardless of the value in the TBIE bit.

**Reference:**

- The timebase timer cannot use the extended intelligent I/O service (EI$^2$OS).

## ■ Timebase Timer Interrupts and EI$^2$OS

**Table 10.4-1 Timebase Interrupts and EI$^2$OS**

| Interrupt number | Interrupt level setting register | | Vector table address | | | EI$^2$OS |
|---|---|---|---|---|---|---|
| | Register name | Address | Lower | Upper | Bank | |
| #33 ($21_H$) | ICR11 | $0000BB_H$ | $FFFF78_H$ | $FFFF79_H$ | $FFFF7A_H$ | x |

x: Not available

# 10.5 Operation of the Timebase Timer

**This section describes the operation of the interval timer function, the oscillation stabilization interval timer function, and the clock supply function.**

## ■ Operation of the Interval Timer Function (Timebase Timer)

The interval timer function generates an interrupt request for each interval

The stabilization in Figure 10.5-1 "Stabilization of the Timebase Timer" is required to all the timer to operate as an interval timer.

**Figure 10.5-1 Setting of the Timebase Timer**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| TBTC | RESV | — | — | TBIE | TBOF | TBR | TBC1 | TBC0 |
|  | 1 | - | - | ◎ | 0 | 0 | ◎ | ◎ |

◎ : Used
0 : Set 0.
1 : Set 0.
- : Undefined bit

- The timebase timer continues counting up as long as the clock oscillates.
- If the timebase timer counter is cleared (TBTC: TBR = "0"), the counter counts up from "000000000000000000$_B$". When the specified interval timer bit overflows, the interrupt request flag bit (TBOF) of the timebase timer control register (TBTC) is set to "1". If the overflow occurs while the interrupt request enable bit is set to Enabled (TBIE = "1"), interrupt requests are output at every specified interval based on the time when the counter was cleared.
- The interval may become longer than the time set because of timebase timer clearing.

## ■ Oscillation Stabilization Wait Time Timer Function

The timebase timer is also used as the oscillation time timer for oscillation and the PLL clocks.

The timebase timer is also used for the oscillation stabilization interval of the oscillation clock and the PLL clock. The timebase timer counts up the oscillation stabilization interval since it has the value "000000000000000000$_B$" (counter cleared) and until it detects the oscillation stabilization interval. When the timebase timer returns from timebase timer mode to PLL clock mode, the oscillation stabilization interval indicates only the portion from the middle of counting because the timebase timer counter has not been cleared.

**Table 10.5-1 Timebase Timer Counter Clearing and Oscillation Stabilization Wait Time**

| Operation | Counter clear | TBOF clear | Oscillation Stabilization Wait Time |
|---|---|---|---|
| TBTC: Writing of 0 to TBR | O | O | - |
| Power-on reset | O | O | Oscillation clock oscillation stabilization wait time |
| Watchdog reset | O | O | |
| Releasing of stop mode | O | O | Oscillation clock oscillation stabilization wait time (at return to main clock mode) |
| Transition from oscillation clock mode to PLL clock mode (MCS = 1 to 0) | O | O | PLL clock oscillation stabilization wait time |
| Releasing of timebase timer mode | X | X | PLL clock oscillation stabilization wait time (at return to PLL clock mode) |
| Releasing of sleep mode | X | X | - |

O: Available
X: Not available

## ■ Clock Supply Function

The timebase timer supplies clocks to the watchdog timer. Clearing of the timebase counter affects operation of the watchdog timer. For more information, see Section 10.6 "Usage Notes on the Timebase Timer".

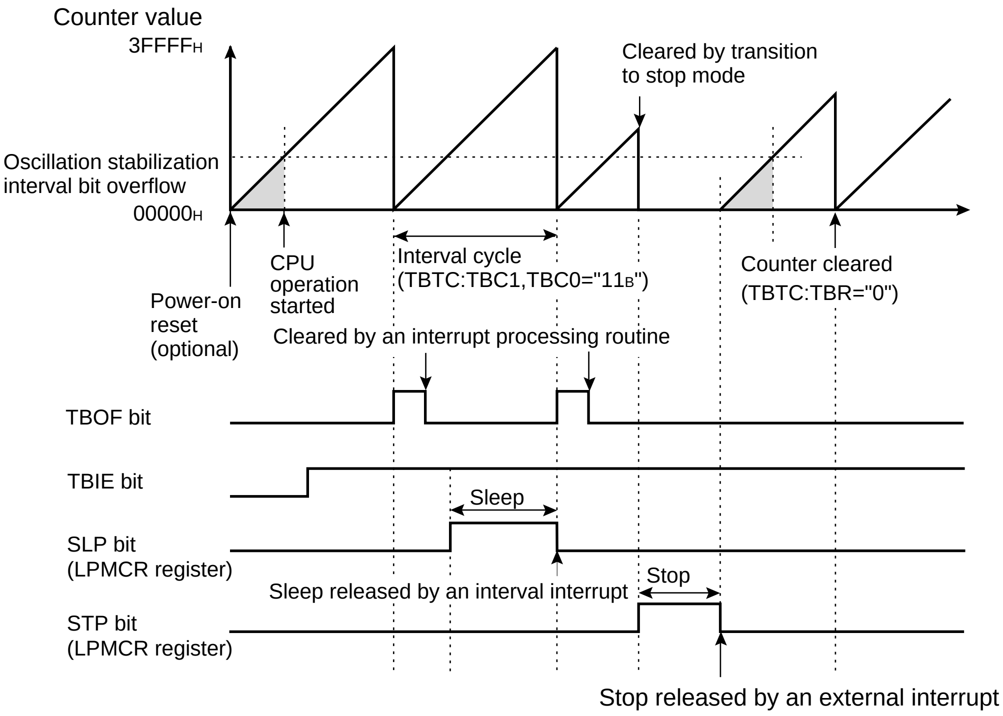
## ■ Timebase Timer Operation Statuses

Figure 10.5-2 "Timebase Timer Operations" shows the following operation statuses.

- When a power-on reset occurs
- When the CPU enters sleep mode while the interval timer function is operating
- When the CPU enters stop mode
- When the timebase timer counter clear request is issued

When the CPU enters stop mode, the timebase timer counter is cleared and the timebase timer counter stops. To release stop mode, use the timebase timer counter to count the oscillation stabilization interval.

**Figure 10.5-2 Timebase Timer Operations**


Counter value
3FFFFH
Oscillation stabilization interval bit overflow
00000H
Cleared by transition to stop mode
Power-on reset (optional)
CPU operation started
Interval cycle (TBTC:TBC1,TBC0="11B")
Counter cleared (TBTC:TBR="0")
Cleared by an interrupt processing routine
TBOF bit
TBIE bit
Sleep
SLP bit (LPMCR register)
Sleep released by an interval interrupt
Stop
STP bit (LPMCR register)
Stop released by an external interrupt

If the interval time setting bits of the timebase timer control register (TBC: TBC, TBC0) is set to "$11_B$" ($2^{19}$/HCLK)

▨ : Oscillation stabilization interval
HCLK : Oscillation clock frequency

## 10.6 Usage Notes on the Timebase Timer

**This section provides notes on how clearing of an interrupt request or clearing of the timebase timer counter affects the functions.**

### ■ Timebase Timer Usage Notes

❍ **Clearing interrupt requests**

Clear the interrupt request flag bit (TBOF) of the timebase timer control register (TBTC) to "0" while the interrupt request enable bit (TBIE) or the interrupt level mask register (ILM) of the processor status (PS) is set to disabled.

❍ **Functions affected by clearing of the timebase timer counter**

- Interval timer function (interval interrupt)
- When the watchdog timer is being used
- Clock output circuit

❍ **Use of the timebase timer as the oscillation settling time timer**

In stop mode in which the operating clock stops, the timebase timer counter is cleared and stopped. When the timebase timer counter is cleared, the clock supplied from it starts to be supplied again from the initial state. As a result, the H level may be shortened or the L level may be prolonged by half a cycle at the maximum. Although the clock for the watchdog timer also starts to be supplied again from the initial state, the watchdog timer operates in normal cycles because the watchdog timer counter is cleared at the same time.

❍ **Notes on peripheral functions to which clocks are supplied from the timebase timer**

At power-on or in stop mode, the source oscillation is stopped. Thus, the timebase timer counter places the oscillation stabilization interval for the operating clock using the clock supplied from the oscillator. Depending on the oscillator type, an appropriate oscillation stabilization interval must be specified.

For more information, see Section 4.5 "Oscillation Stabilization Wait Time".

# CHAPTER 11 WATCHDOG TIMER

**This chapter describes the functions and operations of the watchdog timer of the MB90M405 series.**

# 11.1 Overview of the Watchdog Timer

**The watchdog timer is a 2-bit counter that uses the output of the timebase timer as the count clock. After the watchdog timer is activated, the CPU is reset within a specified interval unless the watchdog timer is cleared.**

## ■ Watchdog Timer Function

The watchdog timer is provided to detect whether a program is running out of control. The watchdog timer, once activated, must continue to be cleared within every specified interval. If the program results in an endless loop and the watchdog timer is not cleared within the minimum time shown in Table 11.1-1 "Intervals for the Watchdog Timer" a watchdog reset is issued to the CPU, sending it into the reset status. Specify the watchdog timer interval in the interval setting bits (WT1, WT0) of the watchdog timer control register (WDTC).

**Table 11.1-1 Intervals for the Watchdog Timer**

| WT1 | WT0 | Interval | | |
|---|---|---|---|---|
| | | **Minimum (*1)** | **Maximum (*1)** | **Oscillation clock cycle count** |
| 0 | 0 | Approx. 3.58 ms | Approx. 4.61 ms | $2^{14} \pm 2^{11}$ cycle |
| 0 | 1 | Approx. 14.33 ms | Approx. 18.3 ms | $2^{16} \pm 2^{13}$ cycle |
| 1 | 0 | Approx. 57.23 ms | Approx. 73.73 ms | $2^{18} \pm 2^{15}$ cycle |
| 1 | 1 | Approx. 458.75 ms | Approx. 589.82 ms | $2^{21} \pm 2^{18}$ cycle |

*1: Value during operation of the 4.19 MHz oscillation clock

For more information on a watchdog timer interval, see Section 11.4 "Operation of the Watchdog Timer".

**Reference:**

The watchdog timer, after being activated, can be stopped using a power-on reset or a reset from the watchdog timer. The watchdog timer can be cleared with an external reset, an internal reset, writing to the watchdog control bit (WTE) of the watchdog timer control register (WDTC), or transition to sleep or stop mode. However, the watchdog function is enabled and not stopped.

**Note:**

The watchdog counter consists of a 2-bit counter that uses the carry signals of the timebase timer as count clocks. Since the watchdog timer uses a carry signal of the timebase timer, the watchdog reset interval time may become longer than the specified time if the timebase timer is cleared.

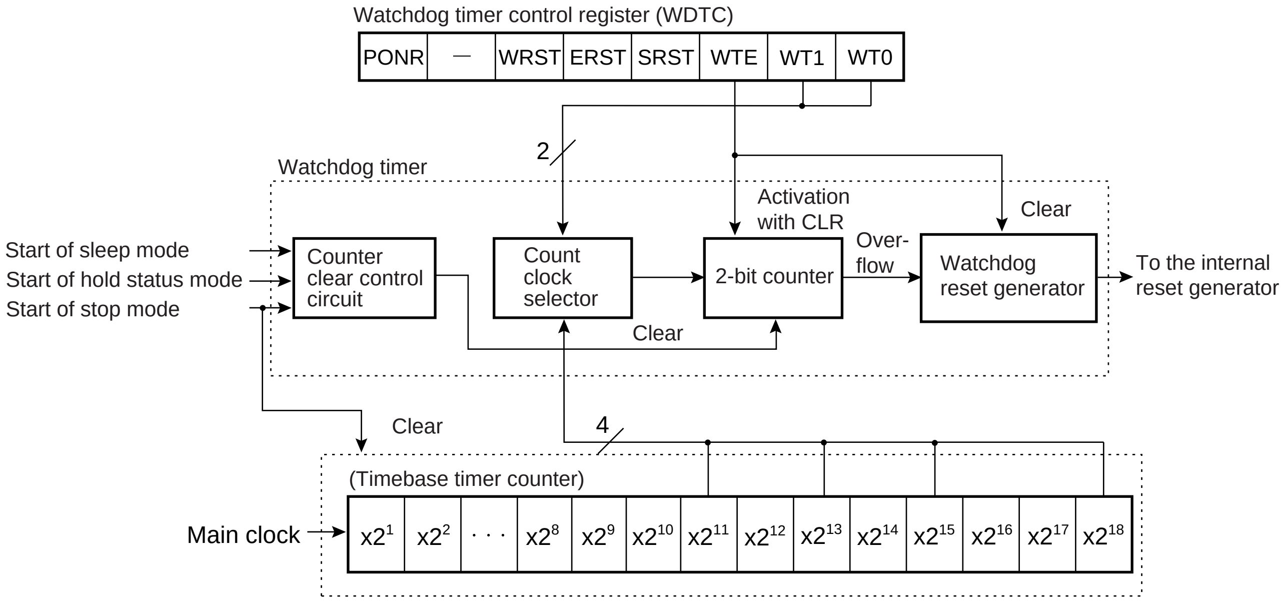
# 11.2 Configuration of the Watchdog Timer

**The watchdog timer consists of the following blocks:**

- **Count clock selector**
- **Watchdog timer (2-bit counter)**
- **Watchdog reset generator**
- **Counter clear control circuit**
- **Watchdog timer control register (WDTC)**

## ■ Block Diagram of the Watchdog Timer

**Figure 11.2-1 Block Diagram of the Watchdog Timer**



- : Undefined bit

❍ **Count clock selector**

Used to select one of the four timebase timer output clocks as the count clock of the watchdog timer. Setting the count clock of the watchdog timer determines a watchdog reset interval.

❍ **Watchdog counter (2-bit counter)**

The watchdog timer is a 2-bit timer that counts the clock specified by the count clock selector.

❍ **Watchdog reset generator**

Used to generate the reset signal by an overflow of the watchdog counter.

❍ **Counter clear circuit**

Used to clear the watchdog counter and to control the operation or stopping of the counter.

❍ **Watchdog timer control register (WDTC)**

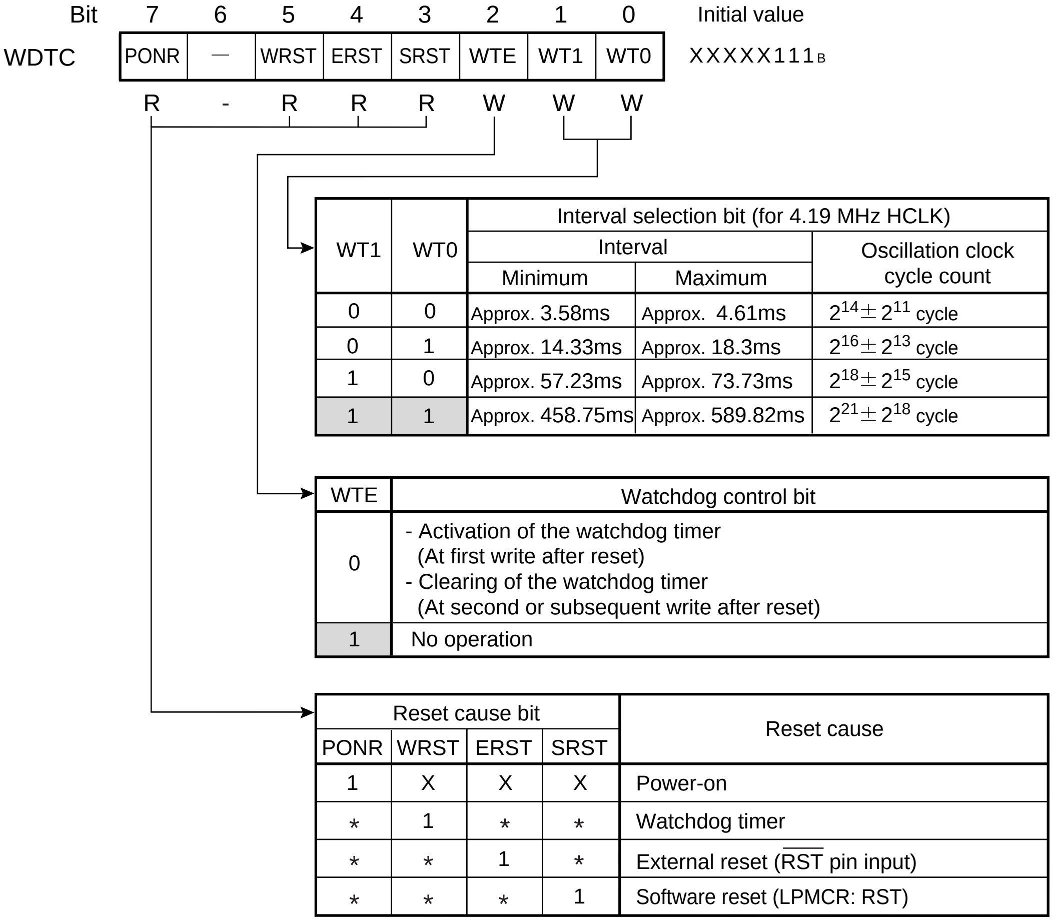
Used to set the interval, activate and clear the watchdog timer, and hold the reset generation cause.

# 11.3 Watchdog Timer Control Register (WDTC)

**The watchdog timer control register (WDTC) is used to set an interval time, start the watchdog timer, clear a setting, and indicate a reset generation cause.**

## ■ Watchdog Timer Control Register (WDTC)

**Figure 11.3-1 Watchdog Timer Control Register (WDTC)**



| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| WDTC | PONR | — | WRST | ERST | SRST | WTE | WT1 | WT0 | XXXXX111B |
| | R | - | R | R | R | W | W | W | |

| WT1 | WT0 | Interval selection bit (for 4.19 MHz HCLK) | | |
|---|---|---|---|---|
| | | Interval | | Oscillation clock cycle count |
| | | Minimum | Maximum | |
| 0 | 0 | Approx. 3.58ms | Approx. 4.61ms | $2^{14} \pm 2^{11}$ cycle |
| 0 | 1 | Approx. 14.33ms | Approx. 18.3ms | $2^{16} \pm 2^{13}$ cycle |
| 1 | 0 | Approx. 57.23ms | Approx. 73.73ms | $2^{18} \pm 2^{15}$ cycle |
| 1 | 1 | Approx. 458.75ms | Approx. 589.82ms | $2^{21} \pm 2^{18}$ cycle |

| WTE | Watchdog control bit |
|---|---|
| 0 | - Activation of the watchdog timer<br>(At first write after reset)<br>- Clearing of the watchdog timer<br>(At second or subsequent write after reset) |
| 1 | No operation |

| Reset cause bit | | | | Reset cause |
|---|---|---|---|---|
| PONR | WRST | ERST | SRST | |
| 1 | X | X | X | Power-on |
| * | 1 | * | * | Watchdog timer |
| * | * | 1 | * | External reset ($\overline{\text{RST}}$ pin input) |
| * | * | * | 1 | Software reset (LPMCR: RST) |

| | | |
|---|---|---|
| R | : | Read only |
| W | : | Write only |
| X | : | Undefined |
| - | : | Undefined bit |
| ▨ | : | Initial value |
| * | : | Retains the previous status. |
| HCLK | : | Oscillation clock frequency |

**Table 11.3-1 Function Description of Each Bit of the Watchdog Timer Control Register (WDTC)**

| Bit name | | Function |
|---|---|---|
| bit7<br>bit5<br>bit4<br>bit3 | PONR, WRST,<br>ERST, SRST:<br>Reset cause bits | • Read-only bits for indicating the reset cause. If more than one reset cause occurs, the bit for each reset cause occurring is set to 1.<br>• These bits are all cleared to 0 after the watchdog timer control register (WDTC) is read.<br>• A power-on reset sets the reset cause flag bit PONR to "1" and sets the reset cause flag bits WRST, ERST, and SRST to an undefined value.<br>• If the PONR bit is set to "1", the contents of the WRST, ERST, and SRST bits should be ignored. |
| bit6 | -:<br>Undefined bit | • The value of this bit is not defined if it is read.<br>• The set value does not affect the operation. |
| bit2 | WTE:<br>Watchdog timer control bit | • The watchdog timer is activated when this bit is set to "0" in the first write operation after a power-on reset or a reset from the watchdog timer.<br>• The watchdog timer is cleared when this bit is set to "0" in write operations after the first after a power-on reset or a reset from the watchdog timer.<br>• Writing 1 does not affect operation. |
| bit1<br>bit0 | WT1, WT0:<br>Interval selection bit | • Used to select the watchdog timer interval.<br>• Data in these bits is valid when the watchdog timer is activated. Data can be written to this bit and the watchdog control bit (WTE) at the same time. Data written to these bits after the watchdog timer is activated is invalid.<br>• These bits are write-only. |

# 11.4 Operation of the Watchdog Timer

**The watchdog timer generates a watchdog reset by an overflow of the watchdog counter.**

## ■ Watchdog Timer Operation

**Figure 11.4-1 Setting of the Watchdog Timer**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| WDTC | PONR | – | WRST | ERST | SRST | WTE | WT1 | WT0 |
| | | | | | | 0 | ◎ | ◎ |

◎ : Used
0 : Set 0.

### ❍ Activating the watchdog timer

- To activate the watchdog timer, set the watchdog control bit (WTE) of the watchdog timer control register (WDTC) to "0" for the first time after a reset from the watchdog timer is generated after a power-on reset. The interval can be set in the interval setting bits (WT1, WT0) of the watchdog timer control register (WDTC).
- The watchdog timer, after being activated, can be stopped using a power-on reset or a reset from the watchdog timer. The watchdog timer cannot be stopped by an external reset, a software reset, writing to the watchdog control bit (WTE) of the watchdog timer control register (WDTC), or transition to sleep or timebase timer mode.

### ❍ Clearing the watchdog timer

- To clear the watchdog timer, set the watchdog control bit (WTE) of the watchdog timer control register (WDTC) to "0".
- The watchdog timer can be cleared also by input of an external reset or an internal reset or transition to sleep mode.
- Transition to timebase timer mode clears and stops the watchdog timer.

### ❍ Intervals for the watchdog timer

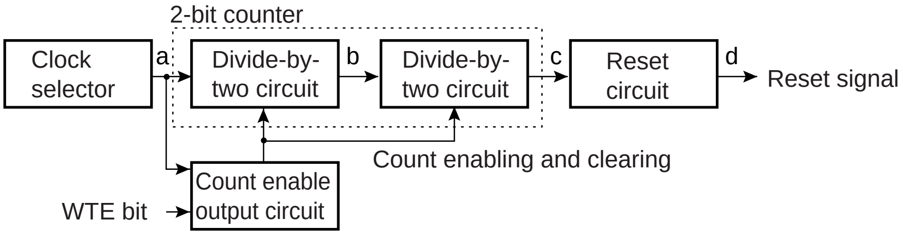
Figure 11.4-2 "Clear Timing and Watchdog Timer Intervals" shows the relationship between the clear timing of the watchdog timer and intervals.

### ❍ Checking a reset cause

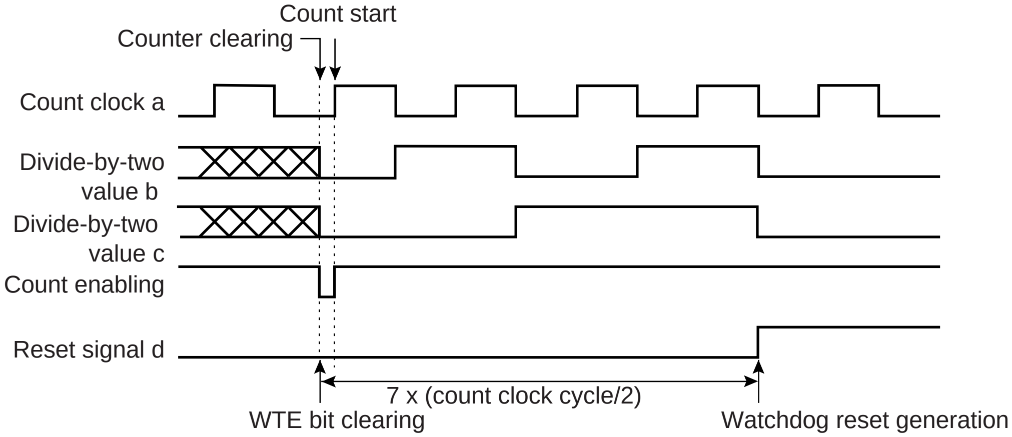
A reset cause can be determined by checking the PONR, WRST, ERST, and SRST bits of the watchdog timer control register (WDTC) after a reset.

**Figure 11.4-2 Clear Timing and Watchdog Timer Intervals**

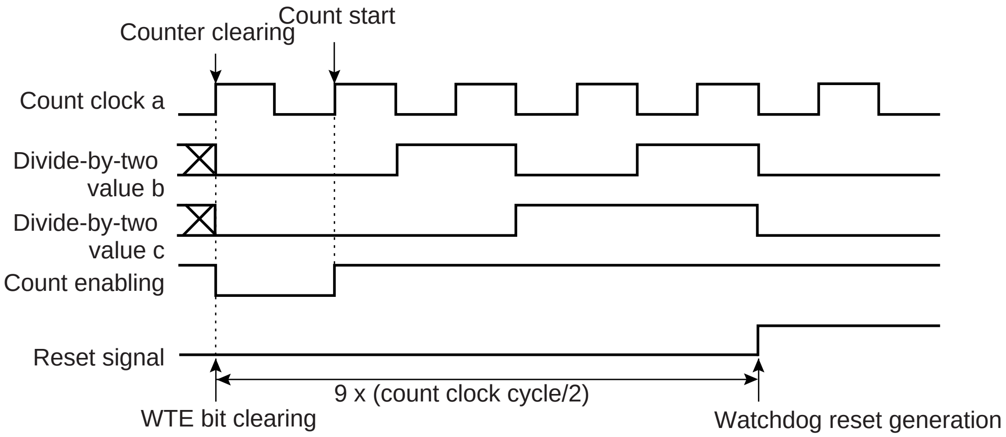
[WDG timer block diagram]



[Minimum interval] When the WTE bit is cleared immediately before the count clock rises:



[Maximum interval] When the WTE bit is cleared immediately after the count clock rises:



# 11.5 Usage Notes on the Watchdog Timer

**Notes on using the watchdog timer are given below.**

## ■ Usage Notes on the Watchdog Timer

❍ **Stopping the watchdog timer**

Once the watchdog timer is activated, it cannot stop until a power-on or watchdog reset occurs.

❍ **Interval setting**

The interval setting becomes valid when the watchdog timer is activated. The interval setting is ignored unless the watchdog timer is activated.

❍ **Interval time**

The interval time of the watchdog timer may become longer than the specified value when the timebase timer is cleared because a carry signal of the timebase timer is used as the count clock.

❍ **Notes on program creation**

If the watchdog timer is repetitiously cleared in a program, the processing time of the program including the interrupt processing must be equal to or less than the minimum interval.

❍ **Watchdog timer operation in timebase timer mode**

In timebase timer mode, the watchdog timer is cleared and stopped. The watchdog timer is restarted when the CPU returns from timebase timer mode to main clock mode or PLL clock mode.

# CHAPTER 12 16-BIT RELOAD TIMER

**This chapter describes the functions and operations of the 16-bit reload timer of the MB90M405 series.**

## 12.1 Overview of the 16-Bit Reload Timer

**The MB90M405 series has three 16-bit reload timer channels. The following clock modes and the following counter operation modes can be specified.**

**Clock modes**

- **Internal clock mode: The timer counts down in synchronization with the internal clock.**
- **Event count mode: The timer counts down in synchronization with the external input pulse.**

**Counter operation modes**

- **Reload mode: The timer repeats counting by reloading the count setting value.**
- **One-shot mode: The timer stops counting when an underflow occurs.**

### ■ 16-bit reload timer operating modes

**Table 12.1-1 16-Bit Reload Timer Operating Modes**

| Clock mode | Counter operation | Operation of 16-bit reload timer |
|---|---|---|
| Internal clock mode | Reload mode | Software trigger operation<br>External trigger operation<br>External gate input operation |
| | One-shot mode | |
| Event count mode (external clock mode) | Reload mode | Software trigger operation |
| | One-shot mode | |

### ■ Internal Clock Mode

The 16-bit reload timer is in internal clock mode if the count clock setting bits (CSL1, CSL0) of the timer control status register (TMCSR) are set to "$00_B$", "$01_B$", or "$10_B$".

For internal clock mode, select one of the following three operations:

#### ❍ Software trigger operation

Counting starts when the software trigger bit (TRG) is set to "1" while the count enable bit (CNTE) of the TMCSR register is set to "1".

#### ❍ External trigger operation

Counting starts when a valid edge (rising, falling, or both edges) of trigger input specified in the operation mode setting bits (MOD2, MOD1, and MOD0) is input to the TIN pins while the CNTE bit of the TMCSR register is set to "1".

#### ❍ External gate input operation

Counting continues while a valid level of gate input ("L" or "H") specified in the operation mode setting bits (MOD2, MOD1, and MOD0) is input to the TIN pins while the CNTE bit of the TMCSR register is set to "1".

## ■ Event Count Mode (External Clock Mode)

The 16-bit reload timer is in event count mode (external clock) if the count clock setting bits (CSL1, CSL0) of the timer control status register (TMCSR) are set to "$11_B$". Counting starts when a valid edge (rising, falling, or both edges) of trigger input specified in the operation mode setting bits (MOD2, MOD1, and MOD0) is input to the TIN pins while the CNTE bit is set to "1". The 16-bit reload timer can also be used as an interval timer if external clock is input periodically.

## ■ Counter Operation

### ❍ Reload mode

Counting starts when an underflow of the 16-bit down counter (change from "$0000_H$" to "$FFFF_H$") loads the values of the 16-bit reload register (TMRLR) into the 16-bit down counter. Since the 16-bit reload timer causes an interrupt request to occur for an underflow condition, it can be used as an interval timer. Every time an underflow occurs, a reversed toggle waveform can be output from the T0 pin.

**Table 12.1-2 Intervals for the 16-Bit Reload Timer**

| Count clock | Count clock period | Interval |
|---|---|---|
| Internal clock | $2^1/\phi$ (0.125 μs) | 0.125 μs to 8.192 ms |
| | $2^3/\phi$ (0.5 μs) | 0.5 μs to 32.768 ms |
| | $2^5/\phi$ (2.0 μs) | 2.0 μs to 131.1 ms |
| External clock | $2^3/\phi$ or more (0.5 μs) | 0.5 μs or more |

$\phi$ : Machine clock
Values in parentheses are for a 16 MHz machine clock.

### ❍ Single-shot mode

An underflow of the 16-bit down counter (change from "$0000_H$" to "$FFFF_H$") stops counting.

**Reference:**

- 16-bit reload timer 0 can be used to create the baud rate of UART.
- 16-bit reload timer 1 can be used to provide a start trigger of the A/D converter.

## ■ 16-Bit Reload Timer Interrupts and EI$^2$OS

An underflow of the 16-bit down counter (change from "$0000_H$" to "$FFFF_H$") outputs an interrupt request.

**Table 12.1-3 16-Bit Reload Timer Interrupts and EI$^2$OS**

| Channel | Interrupt number | Interrupt control register | | Vector table address | | | EI$^2$OS |
|---|---|---|---|---|---|---|---|
| | | Register name | Address | Lower | Upper | Bank | |
| 16-bit reload timer 0 | #23 ($17_H$) | ICR06 | $0000B6_H$ | $FFFFA0_H$ | $FFFFA1_H$ | $FFFFA2_H$ | △ |
| 16-bit reload timer 1 | #24 ($18_H$) | | | $FFFF9C_H$ | $FFFF9D_H$ | $FFFF9E_H$ | |
| 16-bit reload timer 2 | #21 ($15_H$) | ICR05 | $0000B5_H$ | $FFFFA8_H$ | $FFFFA9_H$ | $FFFFAA_H$ | |

△ : Usable when an interrupt cause that shares the ICR is not used.

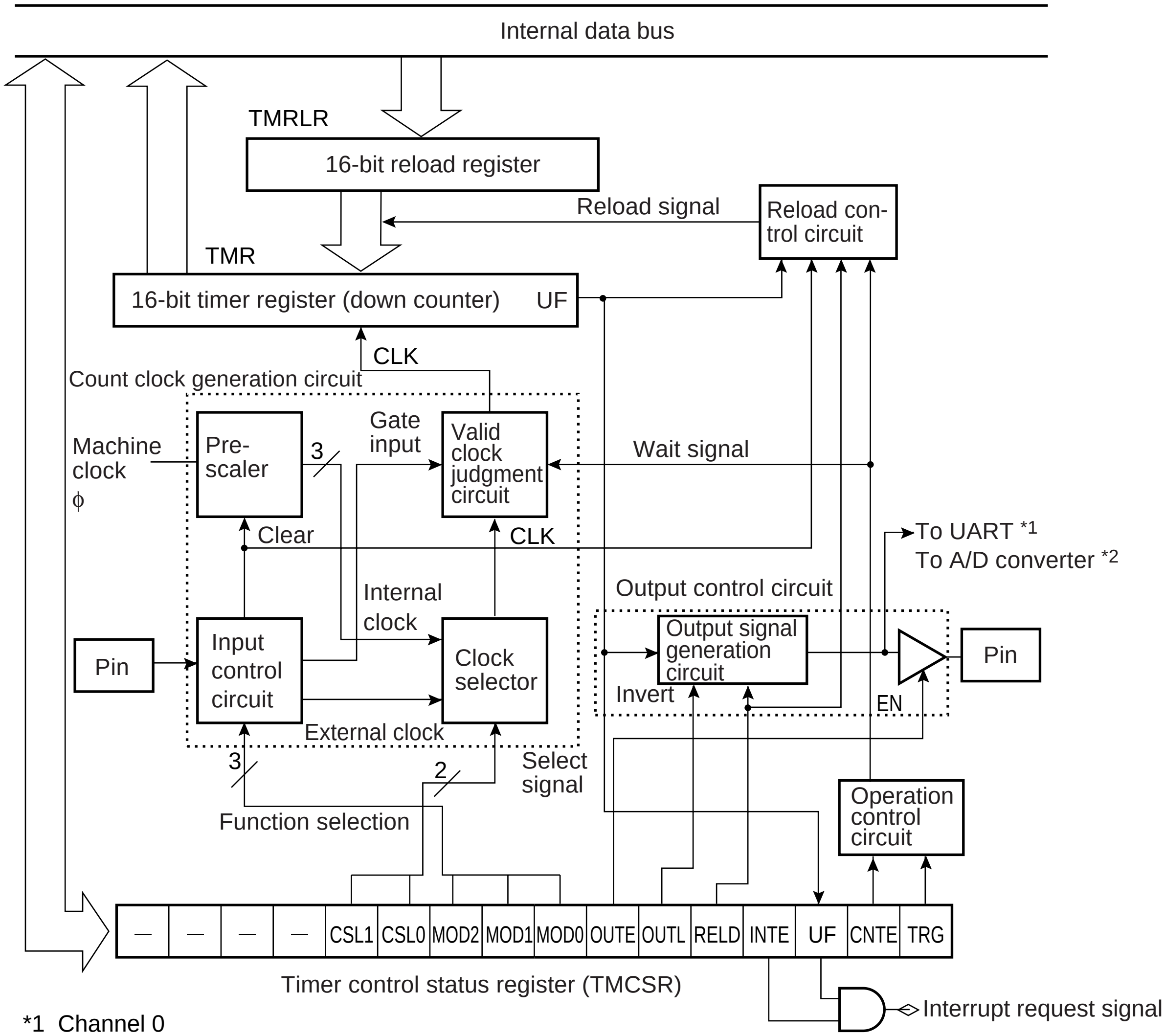
# 12.2 Configuration of the 16-Bit Reload Timer

**Each of the 16-bit reload timers 0 to 2 consists of the following blocks:**

- **Count clock generation circuit**
- **Reload control circuit**
- **Output control circuit**
- **Operation control circuit**
- **16-bit timer register (TMR)**
- **16-bit reload register (TMRLR)**
- **Timer control status register (TMCSR)**

## ■ Block Diagram of the 16-Bit Reload Timer

**Figure 12.2-1 Block Diagram of the 16-Bit Reload Timer**



*1 Channel 0

*2 Channel 1

❍ **Count clock generation circuit**

This circuit generates the count clock for the 16-bit reload timer from the machine clock or external input clock.

❍ **Reload control circuit**

This circuit controls starting of the 16-bit down counter and, if an underflow (change from $0000_H$ to $FFFF_H$) is detected, the load operation for loading a value into the 16-bit down counter.

❍ **Output control circuit**

This circuit controls the inversion of the TO pin output through an underflow of the 16-bit down counter (change from "$0000_H$" to "$FFFF_H$") and enabling and disabling of TO pin output.

❍ **Operation control circuit**

This circuit controls activation and stop of the 16-bit down counter.

❍ **16-bit timer register (TMR)**

This register is a 16-bit down counter. The current counter value is read from this register during a read operation.

❍ **16-bit reload register (TMRLR)**

This register stores a value to be loaded to the 16-bit down counter. The setting value of this register is loaded to the 16-bit down counter, which then starts counting down.

❍ **Timer control status register (TMCSR)**

This register selects the operation mode of the 16-bit reload timer, the count clock, and the operating conditions, enables and disables counting, controls interrupts, and checks the statuses of interrupt requests.

# 12.3 16-Bit Reload Timer Pins

**This section describes the pins of the 16-bit reload timer and provides a pin block diagram.**

## ■ 16-Bit Reload Timer Pins

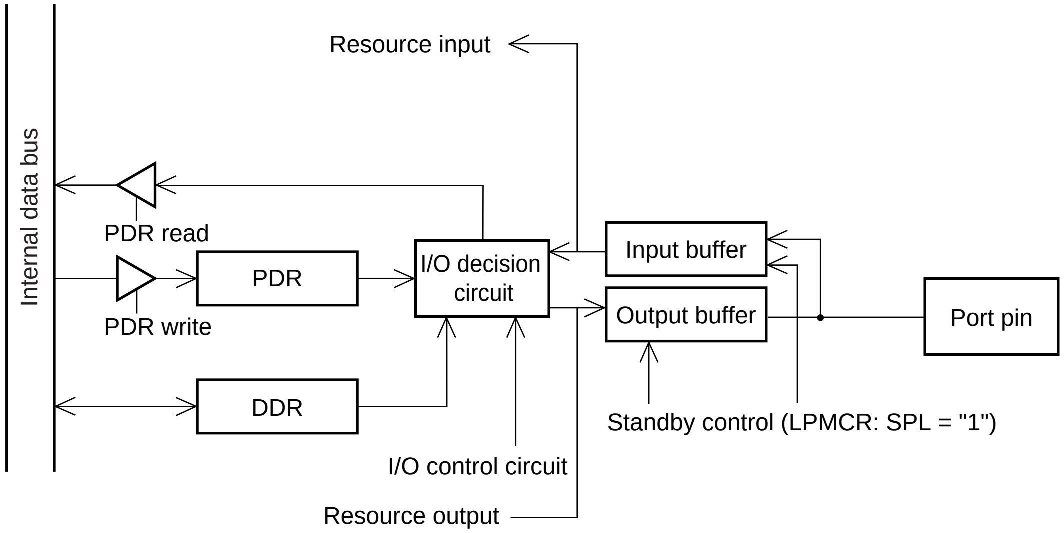
The pins of the 16-bit reload timer are shared with the general-purpose ports.

**Table 12.3-1 16-Bit Reload Timer Pins**

| Pin name | Pin function | I/O format | Pull-up option | Standby control | Settings required for pins |
|---|---|---|---|---|---|
| PB4/TIN0 | Port B input-output/ timer input | CMOS output/ CMOS hysteresis input | Not available | Available | Setting for the input port (DDRB: bit12=0) |
| PB5/TO0 | Port B input-output/ timer output | | | | Setting for timer output enable (TMCR0:OUTE=1) |

## ■ Block Diagram of the 16-Bit Reload Timer Pins

**Figure 12.3-1 Block Diagram of the 16-Bit Reload Timer Pins**



# 12.4 16-Bit Reload Timer Registers

**The 16-bit reload timer registers are as follows.**

## ■ 16-Bit Reload Timer Registers

**Figure 12.4-1 16-Bit Reload Timer Registers**

| bit 15 ···························· bit 8 | bit 7 ···························· bit 0 |
|---|---|
| Timer control status register (TMCSR) | |
| 16-bit timer register/16-bit reload register (TMR/TMRLR) (*1) | |

*1 This register functions as a 16-bit timer register (TMR) during reading, and functions as a 16-bit reload register (TMRLR) during writing.

## 12.4.1 Timer Control Status Register, Higher (TMCSR)

**The timer control status register (TMCSR) is used to select the operating mode and the count clock of the 16-bit reload timer.**

### ■ Timer Control Status Register, Higher (TMCSR)

Figure 12.4-2 Timer Control Status Register, Higher (TMCSR)

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | Initial value |
|---|---|---|---|---|---|---|---|---|---|---|
| | — | — | — | — | CSL1 | CSL0 | MOD2 | MOD1 | MOD0 | XXXX00000B |
| | - | - | - | - | R/W | R/W | R/W | R/W | R/W | |

| MOD2 | MOD1 | MOD0 | Operating mode selection bit (in the internal clock mode) | |
|---|---|---|---|---|
| | | | Input terminal function | Valid edge, level |
| 0 | 0 | 0 | Trigger inhibit | - |
| 0 | 0 | 1 | Trigger input | Rising edge |
| 0 | 1 | 0 | | Falling edge |
| 0 | 1 | 1 | | Both edges |
| 1 | X | 0 | Gate input | "L" level |
| 1 | X | 1 | | "H" level |

| MOD2 | MOD1 | MOD0 | Operating mode selection bit (in the event count mode) | |
|---|---|---|---|---|
| | | | Input terminal function | Valid edge |
| X | 0 | 0 | - | - |
| X | 0 | 1 | Trigger input | Rising edge |
| X | 1 | 0 | | Falling edge |
| X | 1 | 1 | | Both edges |

| CSL1 | CSL0 | Count clock selection bit | |
|---|---|---|---|
| | | Function | Count clock |
| 0 | 0 | Internal clock mode | $2^1/\phi$ (0.125 µs) |
| 0 | 1 | | $2^3/\phi$ (0.5 µs) |
| 1 | 0 | | $2^5/\phi$ (2.0 µs) |
| 1 | 1 | Event count mode | External event input |

| | | |
|---|---|---|
| R/W | : | Read/write |
| - | : | Undefined bit |
| X | : | Indefinite |
| ▨ | : | Initial value |
| $\phi$ | : | Machine clock, the value in the parentheses ( ) indicates the value when the machine clock is operated at 16MHz. |

**Table 12.4-1  Function Description of Each Bit of the Higher of the Timer Control Status Register (TMCSR)**

| Bit name | | Function |
|---|---|---|
| bit15<br>bit14<br>bit13<br>bit12 | Not used<br>Undefined bit | • When these bits are read, their values are undefined.<br>• Writing to these bits has no effect on operation. |
| bit11<br>bit10 | CSL1, CSL0:<br>Count clock selection bits | • These bits select the count clock of the 16-bit reload timer.<br>• When these bits are set to "$00_B$", "$01_B$", and "$10_B$", internal clock mode is selected.<br>• When these bits are set to $11_B$, event count mode is set. |
| bit9<br>bit8<br>bit7 | MOD2, MOD1, MOD0:<br>Operating mode selection bits | • These bits select operation mode.<br>**In internal clock mode:**<br>• The MOD2 bit is used to select input pin functions.<br>• When the MOD2 bit is set to "0", the input pin is used as a trigger input pin.  Whenever a valid edge is input, the value in the 16-bit reload register (TMRLR) is loaded into the counter and counting starts.  The MOD1 and MOD0 bits select the type of valid edge.<br>• When the MOD2 bit is set to "1", the input pin becomes a gate.  Counting continues as long as a valid level specified in the MOD0 bit is input.<br>• The value specified in the MOD1 bit has no effect on operation.<br>**In event count mode:**<br>• The MOD2 bit is not affected.<br>• In event count mode, the input pin becomes a trigger input pin.  Counting starts when a valid edge specified in the MOD1 and MOD0 bits is input. |

## 12.4.2 Timer Control Status Register, Lower (TMCSR)

**The timer control status register (TMCSR) is used to set the operating conditions of the 16-bit reload timer, enable and disable counting, control interrupts, and check the status of interrupt requests.**

### ■ Timer Control Status Register, Lower (TMCSR)

**Figure 12.4-3 Timer Control Status Register, Lower (TMCSR)**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | MOD0* | OUTE | OUTL | RELD | INTE | UF | CNTE | TRG | 00000000B |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

| TRG | Software trigger bit |
|---|---|
| 0 | No change, no effect on other bits |
| 1 | Counting starts after data loading |

| CNTE | Count enable bit |
|---|---|
| 0 | Counting stopped |
| 1 | Counting enabled (wait for the start trigger)) |

| UF | Underflow interrupt request flag bit | |
|---|---|---|
| | During reading | During writing |
| 0 | No interrupt request issued | Clearing the interrupt request |
| 1 | Interrupt request issued | No effect on operation |

| INTE | Underflow interrupt request enable bit |
|---|---|
| 0 | Interrupt request output disabled |
| 1 | Interrupt request output enabled |

| RELD | Reload selection bit |
|---|---|
| 0 | One-shot mode (reload disabled) |
| 1 | Reload mode (reload enabled) |

| OUTL | Pin output level selection bit | |
|---|---|---|
| | In single-shot mode<br>(RELD="0") | In reload mode<br>(RELD="1") |
| 0 | Square wave of H during counting | Toggle output of L when counting is started. |
| 1 | Square wave of L during counting | Toggle output of H when counting is started. |

| OUTE | Timer output enable bit | |
|---|---|---|
| | Pin functions | Registers and pins corresponding to each channel |
| | | TMCR |
| 0 | I/O port | PB5 |
| 1 | Timer output | TO0 |

R/W : Read/write
▭ : Initial value
* : See Section 12.4.1, "Timer Control Status Register, Higher (TMCSR)," for MOD0 (bit 7)

**Table 12.4-2 Function Description of Each Bit of the Lower of the Timer Control StatusRegister (TMCSR)**

| Bit name | | Function |
|---|---|---|
| bit6 | OUTE:<br>Timer output enable bit | • This bit enables or disables output to the timer output pin.<br>• When this bit is set to "0", the pin serves as an I/O port. When this bit is set to "1", the pin serves as a timer output pin. |
| bit5 | OUTL:<br>Pin output level setting bit | • This bit selects the output level of the timer output pin.<br>• The timer output pin outputs a toggle waveform in reload mode or a square wave indicating that counting is in progress in one-shot mode.<br>• Opposite output levels are output from the pin depending on whether this bit is set to "0" or "1". |
| bit4 | RELD:<br>Reload selection bit | • This bit enables reloading.<br>• When this bit is set to "1", the timer is in reload mode. When an underflow of the 16-bit down counter occurs, the value stored in the 16-bit reload register is loaded into the 16-bit down counter and counting continues.<br>• When this bit is set to "0", the timer is in one-shot mode. When an underflow of the 16-bit down counter occurs, counting stops. |
| bit3 | INTE:<br>Underflow interrupt request enable bit | • This bit enables interrupt requests.<br>• When this bit and the interrupt request flag (UF) bit are 1, the timer outputs an interrupt request. |
| bit2 | UF:<br>Underflow interrupt request flag bit | • This is a flag bit for an interrupt request.<br>• This bit is set to "1" when an underflow of the 16-bit down counter occurs.<br>• An interrupt request is output when this bit is set to "1" while the underflow interrupt request enable bit (INTE) is set to "1".<br>• Setting this bit to "0" clears an interrupt request.<br>• Setting this bit to "1" has no effect on operation.<br>• This bit is also cleared when EI$^2$OS is activated. |
| bit1 | CNTE:<br>Count enable bit | • This bit enables or disables counting.<br>• When this bit is set to "1", the counter is placed in trigger standby mode. Counting starts when the software trigger bit (TRG) is set to "1" or a valid edge (rising, falling, or both edges) of trigger input specified in the operation mode setting bits (MOD2, MOD1, and MOD0) is input to the TIN pins.<br>• When this bit is set to "0", counting stops. |
| bit0 | TRG:<br>Software trigger bit | • This bit starts the interval timer function or counter function with software.<br>• When this bit is set to "1" while the count enable bit (CNTE) is set to "1", the value stored in the 16-bit reload register is loaded into the 16-bit down counter and counting starts.<br>• When this bit is set to "0", there is no effect on operation.<br>• The read value is "0". |

## 12.4.3 16-bit Timer Register (TMR)

**The 16-bit timer register (TMR) is always able to read the count value from the 16-bit down counter.**

### ■ 16-bi t Timer Register (TMR)

**Figure 12.4-4 16-Bit Timer Register (TMR)**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | $XXXXXXXX_B$ |
| | R | R | R | R | R | R | R | R | |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | $XXXXXXXX_B$ |
| | R | R | R | R | R | R | R | R | |

R: Read only
X: Undefined

The 16-bit timer register is a 16-bit down counter.

When the software trigger bit (TRG) is set to "1" while the count enable bit (CNTE) of the timer control status register (TMCSR) is set to "1" or a valid edge (rising, falling, or both edges) of trigger input specified in the operation mode setting bits (MOD2, MOD1, and MOD0) is input to the TIN pins, the values stored in the 16-bit reload register (TMRLR) are loaded into the 16-bit down counter and counting starts. This register holds the value of the 16-bit timer register (TMR) while counting is stopped (TMCSR: CNTE = "0").

**Notes:**

- Be sure to use a word transfer instruction (MOVW A, $003A_H$) to read data from the 16-bit timer register (TMR).
- Although 16-bit timer register (TMR) is read-only and the 16-bit reload register (TMRLR) is write-only, they are placed at the same address. Thus, writing a value to the 16-bit timer register has no effect on this register because the value is written to the 16-bit reload register.

## 12.4.4 16-bit Reload Register (TMRLR)

**The 16-bit reload register (TMRLR) sets a reload value in the 16-bit down counter. The value written to this register is loaded into the down counter, and the value is counted down.**

### ■ 16-Bit Reload Register (TMRLR)

**Figure 12.4-5 16-Bit Reload Register (TMRLR)**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | $XXXXXXXX_B$ |
| | W | W | W | W | W | W | W | W | |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | $XXXXXXXX_B$ |
| | W | W | W | W | W | W | W | W | |

W: Write only
X: Undefined

When a value is written to the 16-bit reload register (TMRLR), counting must be stopped (TMCSR:CNTE="0") regardless of the operating mode of the 16-bit reload register. While the count enable bit (CNTE) of the timer control status register (TMCSR) is set to "1", a value stored in the 16-bit reload register is loaded into the 16-bit down counter and countdown starts in one of two cases: the software trigger bit (TRG) is set to "1" or a valid edge (rising, falling, or both edges) of the trigger input specified in the operation mode setting bits (MOD2, MOD1, and MOD0) is input to the TIN pins.

In reload mode, when an underflow of the 16-bit down counter occurs (change from "$0000_H$" to "$FFFF_H$"), a value stored in the 16-bit reload register (TMRLR) is loaded into the 16-bit down counter and countdown continues. In one-shot mode, when an underflow of the 16-bit down counter occurs, the 16-bit down counter stops at the value "$FFFF_H$".

**Notes:**

- Counting must be stopped (TMCSR: CNTE = "0") when a value is written to the 16-bit reload register (TMRLR).
- Use a word transfer instruction (MOVW $003A_H$, A) to write a value to the 16-bit reload register (TMRLR).
- Although the 16-bit reload register (TMRLR) is write-only and the 16-bit timer register (TMR) is read-only, they are placed at the same address. Since different values are written to, and read from these registers, none of the INC/DEC and other instructions that result in read-modify-write (RMW) operations can be used.

# 12.5 16-Bit Reload Timer Interrupts

**The 16-bit reload timer outputs an interrupt request when an underflow of the 16-bit down counter occurs. The 16-bit reload timer also supports the extended intelligent I/O service (EI$^2$OS).**

## ■ 16-Bit Reload Timer Interrupts

Table 12.5-1 Interrupt Control Bits and Interrupt Causes of the 16-Bit Reload Timer

| | 16-bit reload timer 0 | 16-bit reload timer 1 |
|---|---|---|
| Interrupt request flag bit | TMCSR0: UF | TMCSR1: UF |
| Interrupt request enable bit | TMCSR0: INTE | TMCSR1: INTE |
| Interrupt cause | Underflow of the 16-bit down counter (TMR0) | Underflow of the 16-bit down counter (TMR1) |

In the 16-bit reload timer, the underflow interrupt request flag bit (UF) of the timer control status register (TMCSR) is set to "1" when an underflow of the 16-bit down counter occurs (change from "0000$_H$" to "FFFF$_H$"). If, at this time, the underflow interrupt request enable bit is set to Enabled (TMSCR: INTE = "1"), an interrupt request is output.

## ■ 16-Bit Reload Timer Interrupts and EI$^2$OS

Table 12.5-2 16-Bit Reload Timer Interrupts and EI$^2$OS

| Channel | Interrupt number | Interrupt control register | | Vector table address | | | EI$^2$OS |
|---|---|---|---|---|---|---|---|
| | | Register name | Address | Lower | Upper | Bank | |
| 16-bit reload timer 0 | #23 (17$_H$) | ICR06 | 0000B6$_H$ | FFFFA0$_H$ | FFFFA1$_H$ | FFFFA2$_H$ | △ |
| 16-bit reload timer 1 | #24 (18$_H$) | | | FFFF9C$_H$ | FFFF9D$_H$ | FFFF9E$_H$ | |
| 16-bit reload timer 2 | #21 (15$_H$) | ICR05 | 0000B5$_H$ | FFFFA8$_H$ | FFFFA9$_H$ | FFFFAA$_H$ | |

△ : Usable when an interrupt cause that shares the ICR is not used.

## ■ EI$^2$OS Function of the 16-Bit Reload Timer

The 16-bit reload timer allows the use of the extended intelligent I/O service (EI$^2$OS) when an underflow of the 16-bit down counter occurs (change from "0000$_H$" to "FFFF$_H$").

# 12.6 Operation of the 16-Bit Reload Timer

**This section describes the 16-bit reload timer settings and counter operating status.**

## ■ 16-Bit Reload Timer Settings

### ❍ Internal clock mode setting

The setting shown in Figure 12.6-1 "Internal Clock Mode Setting" is required to operate this timer as an interval timer.

**Figure 12.6-1 Internal Clock Mode Setting**

| | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TMCSR | — | — | — | — | CSL1 | CSL0 | MOD2 | MOD1 | MOD0 | OUTE | OUTL | RELD | INTE | UF | CNTE | TRG |
| | | | | | Other than 11 | | ◎ | ◎ | ◎ | ◎ | ◎ | ◎ | ◎ | ◎ | 1 | ◎ |

| | |
|---|---|
| TMRL | Set the initial value (reload value) of the counter. |

◎: Used
1 : Set 1.

### ❍ Event counter mode setting

he setting shown in Figure 12.6-2 "Event Counter Mode Setting" is required to operate this timer as an event counter.

**Figure 12.6-2 Event Counter Mode Setting**

| | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TMCSR | — | — | — | — | CSL1 | CSL0 | MOD2 | MOD1 | MOD0 | OUTE | OUTL | RELD | INTE | UF | CNTE | TRG |
| | | | | | 1 | 1 | ◎ | ◎ | ◎ | ◎ | ◎ | ◎ | ◎ | ◎ | 1 | ◎ |

| | |
|---|---|
| TMRL | Set the initial value (reload value) of the counter. |

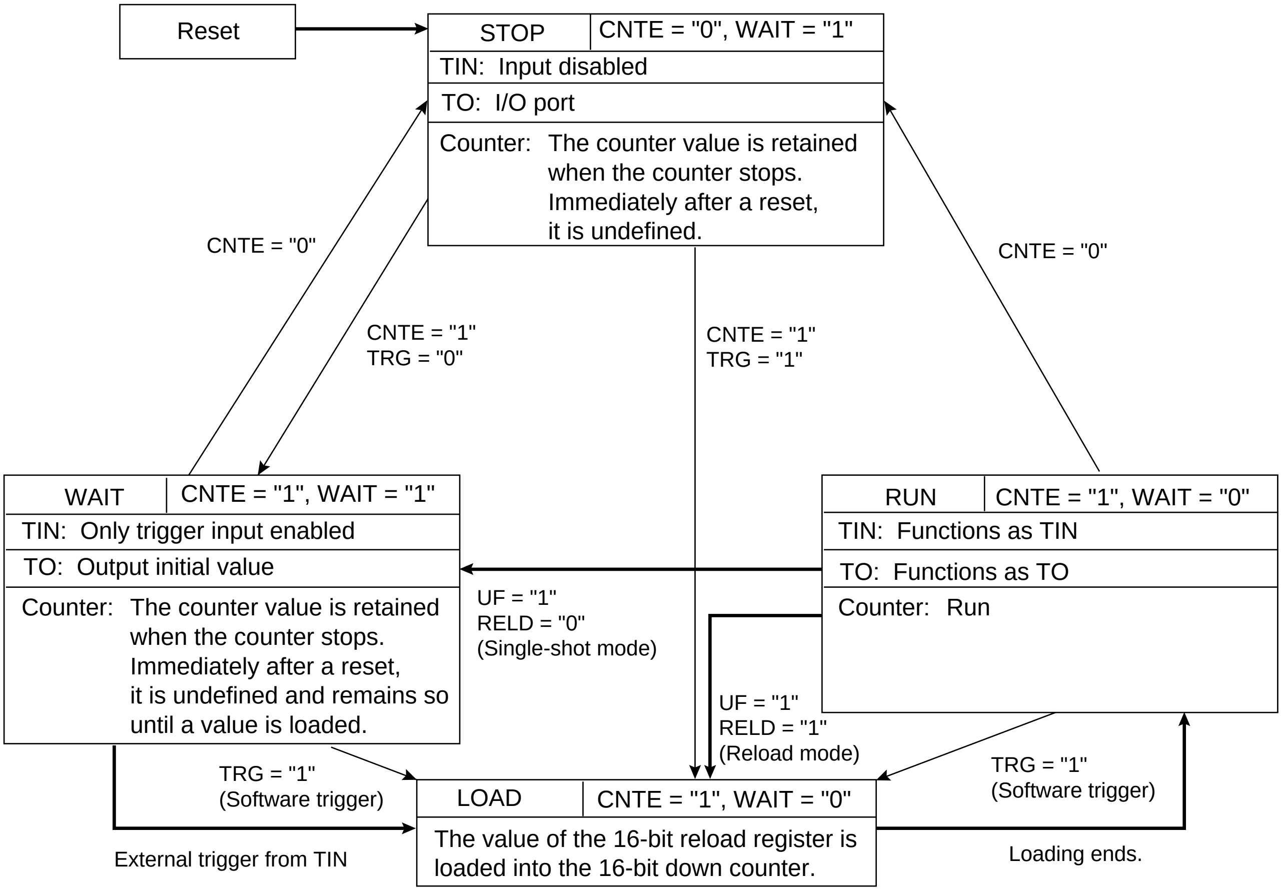
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| DDRB | | | | | | | | |
| | | | | △ | | | | |

◎ : Used
1 : Set 1
△ : Set the corresponding to the pin used to 0.

## ■ Counter Operating Status

The 16-bit down counter status is determined by the count enable bit (CNTE) values of the timer control status register (TMCSR) and the internal trigger wait signal value (WAIT). Figure 12.6-3 "Counter Status Transition" shows the relationship between the count enable bit (CNTE) values and the internal trigger wait signal (WAIT) values in the stop status (STOP status), trigger wait status (WAIT status), and running status (RUN status)

**Figure 12.6-3 Counter Status Transition**



━━▶ : State transition by hardware
──▶ : State transition by register access
WAIT : Wait signal (internal signal)
TRG : Software trigger bit of timer control status register (TMCSR)
CNTE : Count enable bit of timer control status register (TMCSR)
UF : Underflow interrupt request flag bit of timer control status register (TMCSR)
RELD : Reload selection bit of timer control status register (TMCSR)

## 12.6.1 Internal Clock Mode (Reload Mode)

**The 16-bit reload timer count downs the 16-bit down counter in synchronization with the internal count clock and outputs an interrupt request when an underflow occurs (change from "0000$_H$" to "FFFF$_H$"). It can also output a toggle waveform from the timer output pin.**

### ■ Operation in Internal Clock Mode (Reload Mode)

While the count enable bit (CNTE) of the timer control status register (TMCSR) is set to "1", a value stored in the 16-bit reload register (TMRLR) is loaded into the 16-bit down counter, and then countdown starts in one of the following cases: the software trigger bit (TRG) is set to "1", or a valid edge (rising, falling, or both edges) of the trigger input specified in the operation mode setting bits (MOD2, MOD1, and MOD0) is input to the TIN pins. When the CNTE bit and the software trigger bit are set to "1" simultaneously, countdown starts as soon as counting is enabled.

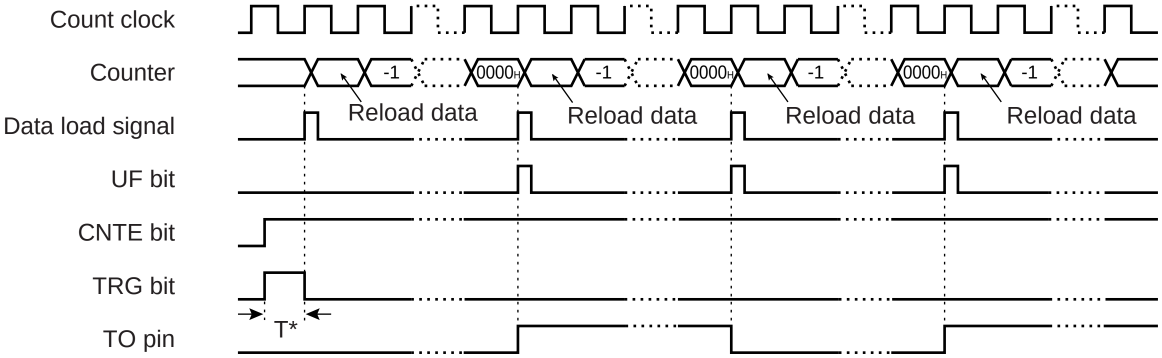
When an underflow of the 16-bit down counter occurs (change from "0000$_H$" to "FFFF$_H$"), a value stored in the 16-bit reload register (TMRLR) is loaded into the 16-bit down counter and countdown continues. An interrupt request is output to the CPU when an underflow of the 16-bit down counter occurs while the underflow interrupt request flag bit (UF) of the timer control status register (TMCSR) is "1" and the underflow interrupt request enable bit (INTE) is "1".

The timer can also output from the TO pin a toggle waveform, which is inverted for each underflow.

#### ❍ Software trigger operation

Counting starts when the software trigger bit (TRG) is set to "1" while the count enable bit (CNTE) of the timer control status register (TMCSR) is set to "1".

**Figure 12.6-4 Count Operation in Reload Mode (Software Trigger Operation)**

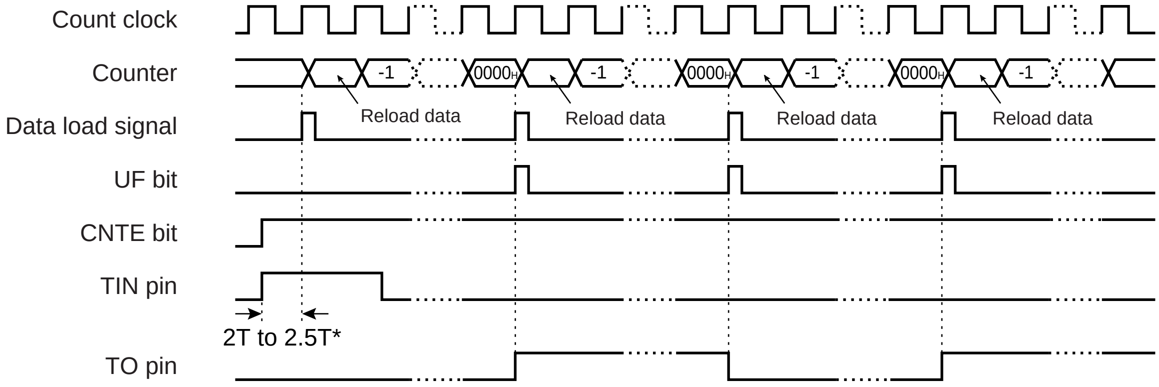


T: Machine cycle (one cycle of the machine clock)
* It takes 1T time from trigger input to loading of the reload data.

### ❍ External trigger operation

Counting starts when a valid edge (rising, falling, or both edges) of trigger input specified in the operation mode setting bits (MOD2, MOD1, and MOD0) is input to the TIN pins while the count enable bit (CNTE) of the timer control status register (TMCSR) is set to "1".

**Figure 12.6-5 Count Operation in Reload Mode (External Trigger Operation)**



T: Machine cycle (one cycle of the machine clock)
* It takes 2T to 2.5T time from external trigger input to loading of the reload data.

**Note:**

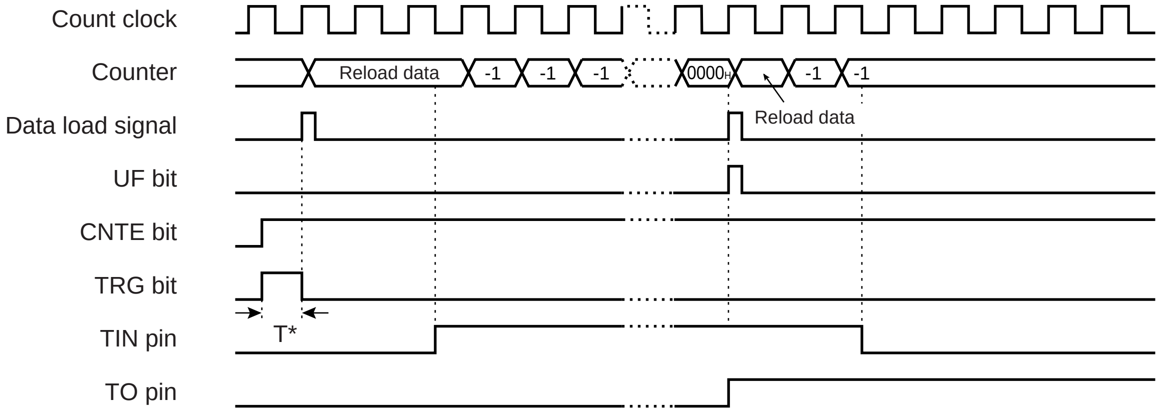
Specify 2/ϕ (ϕ : machine clock frequency) or more for the width of a trigger pulse to be input to the TIN pin.

### ❍ Gate input operation

Counting starts when the software trigger bit (TRG) is set to "1" while the count enable bit (CNTE) of the timer control status register (TMCSR) is set to "1".

Counting continues while a valid level of gate input ("L" or "H") specified in the operation mode setting bits (MOD2, MOD1, and MOD0) is input to the TIN pin.

**Figure 12.6-6 Count Operation in Reload Mode (Software Trigger, Gate Input Operation)**



T: Machine cycle (one cycle of the machine clock)
* It takes 1T time from trigger input to loading of the reload data.

**Note:**

Specify 2/ϕ (ϕ : machine clock frequency) or more for the width of a gate input pulse to be input to the TIN pin.

## 12.6.2 Internal Clock Mode (Single-shot Mode)

**The 16-bit reload timer count downs the 16-bit down counter in synchronization with the internal count clock and outputs an interrupt request when an underflow occurs (change from "0000$_H$" to "FFFF$_H$"). It can also output a square waveform from the T0 pin.**

### ■ Internal Clock Mode (Single-Shot Mode)

When the software trigger bit (TRG) is set to "1" while the count enable bit (CNTE) of the timer control status register (TMCSR) is set to "1" or a valid edge (rising, falling, or both edges) of trigger input specified in the operation mode setting bits (MOD2, MOD1, and MOD0) is input to the TIN pins, a value stored in the 16-bit reload register (TMRLR) is loaded into the 16-bit down counter and countdown starts. When both the CNTE bit and the software trigger bit (TMCSR: TRG) are set to "1", countdown starts as soon as counting is enabled.

When an underflow of the 16-bit down counter occurs (change from "0000$_H$" to "FFFF$_H$"), the 16-bit down counter stops counting at the value "FFFF$_H$".

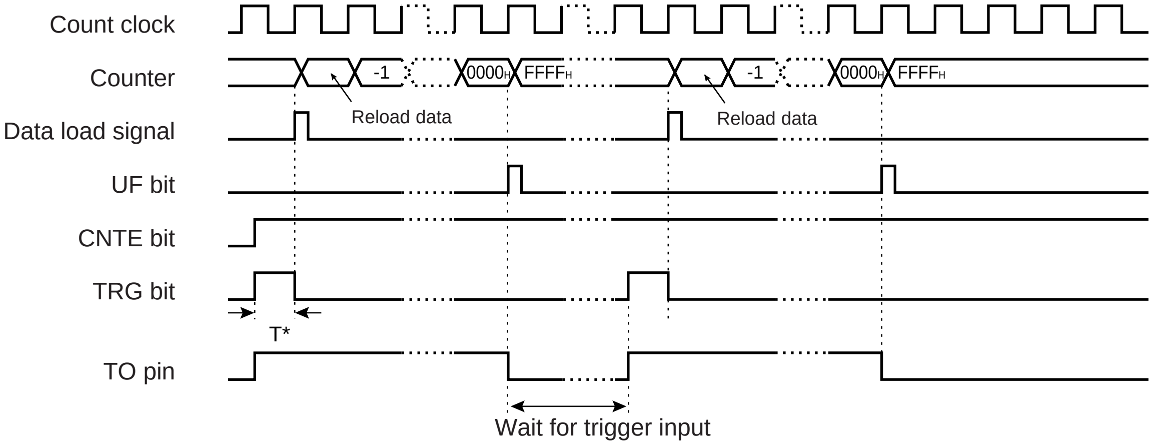
An interrupt request is output when an underflow of the 16-bit down counter occurs while the underflow interrupt request flag bit (UF) of the timer control status register (TMCSR) is set to "1" and the underflow interrupt request enable bit (INTE) is set to "1".

The timer can also output from the TO pin a rectangular waveform indicating that counting is in progress.

#### ❍ Software trigger operation

Counting starts when the software trigger bit (TRG) is set to "1" while the count enable bit (CNTE) of the timer control status register (TMCSR) is set to "1".

**Figure 12.6-7 Count Operation in Single-Shot Mode (Software Trigger Operation)**

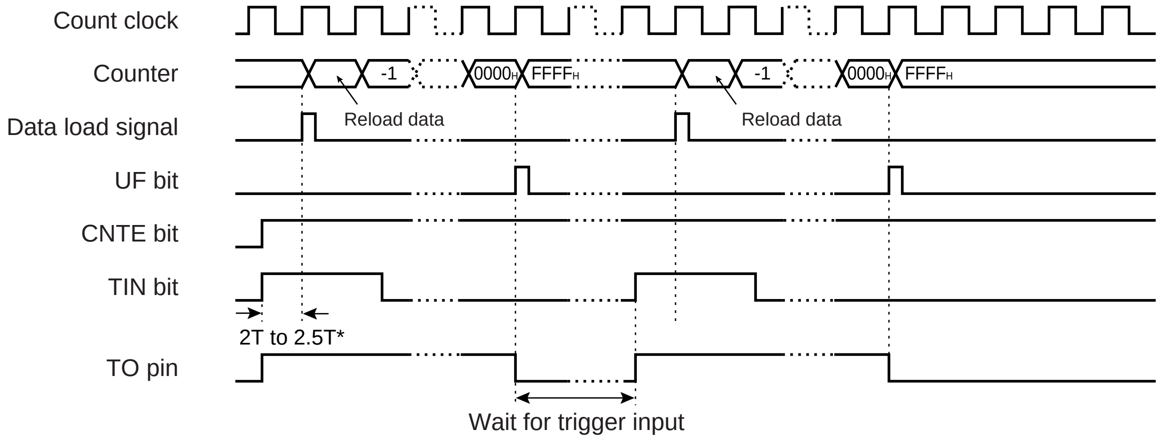


T: Machine cycle (one cycle of the machine clock)
* It takes 1T time from trigger input to loading of the reload data.

### ❍ External trigger input operation

Counting starts when a valid edge (rising, falling, or both edges) of trigger input specified in the operation mode setting bits (MOD2, MOD1, and MOD0) is input to the TIN pins while the count enable bit (CNTE) of the timer control status register (TMCSR) is set to "1".

**Figure 12.6-8 Count Operation in Single-Shot Mode (External Trigger Operation)**



T: Machine cycle (one cycle of the machine clock)

* It takes 2T to 2.5T time from external trigger input to loading of the reload data.

**Note:**

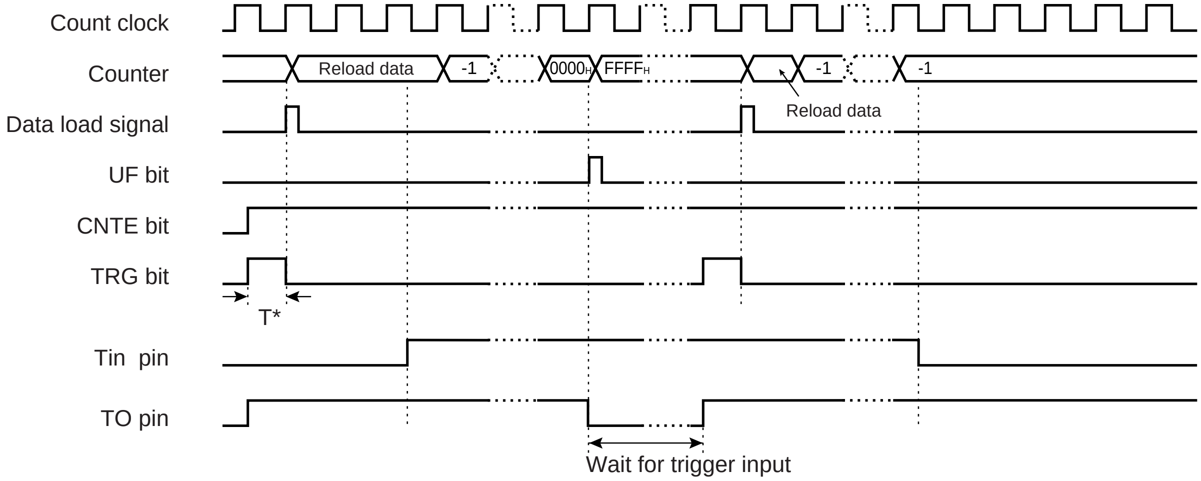
Specify 2/ϕ or more for the width of the trigger pulse input to the TIN pin.

### ❍ External Gate input operation

Counting starts when the software trigger bit (TRG) is set to "1" while the count enable bit (CNTE) of the timer control status register (TMCSR) is set to "1".

Counting continues while a valid level of gate input ("L" or "H") specified in the operation mode setting bits (MOD2, MOD1, and MOD0) is input to the TIN pin.

**Figure 12.6-9 Count Operation in Single-Shot Mode (Software Trigger Gate Input Operation)**



T: Machine cycle (one cycle of the machine clock)

* It takes 1T time from trigger input to loading of the reload data.

**Note:**

Specify 2/ϕ (ϕ : machine clock frequency) or more for the width of a gate input pulse to be input to the TIN pin.

## 12.6.3 Event Count Mode

**The 16-bit reload timer count downs the 16-bit down counter every time it detects a valid edge of pulse input to the TIN pin and outputs an interrupt request when an underflow occurs (change from "0000$_H$" to "FFFF$_H$"). It can also output a toggle or square waveform from the T0 pin.**

### ■ Event Count Mode

When the software trigger bit (TRG) is set to "1" while the count enable bit (CNTE) of the timer control status register (TMCSR) is set to "1", a value stored in the 16-bit reload register (TMRLR) is loaded into the 16-bit down counter and countdown occurs every time a valid edge (rising, falling, or both edges) of pulse input to the TIN pin (external count clock) is detected. When both the CNTE bit and the software trigger bit (TRG) are set to "1", countdown starts as soon as counting is enabled.

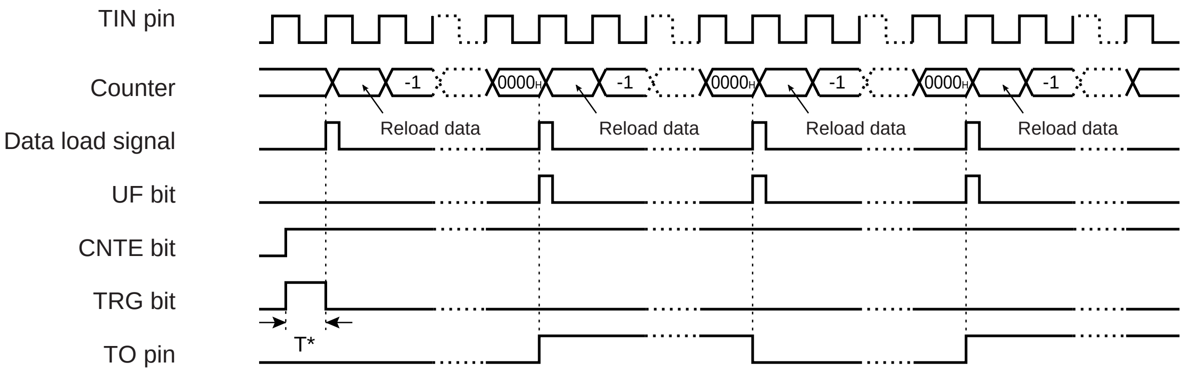
#### ❍ Operation in reload mode

When an underflow of the 16-bit down counter occurs (change from "0000$_H$" to "FFFF$_H$"), a value stored in the 16-bit reload register (TMRLR) is loaded into the 16-bit down counter and countdown continues.

An interrupt request is output to the CPU when an underflow of the 16-bit down counter (change from 0000$_H$ to FFFF$_H$) occurs while the underflow interrupt request flag bit (UF) of the timer control status register (TMCSR) is set to "1" and the underflow interrupt request enable bit (INTE) is set to "1".

The timer can also output from the TO pin a toggle waveform, which is inverted for each underflow.

**Figure 12.6-10 Count Operation in Reload Mode (Event Count Mode)**



T: Machine cycle (one cycle of the machine clock)
* It takes 1T time from trigger input to loading of the reload data.

**Note:**

Specify $2^2/\phi$ or more for the H and L widths of the pulse input to the TIN pin.

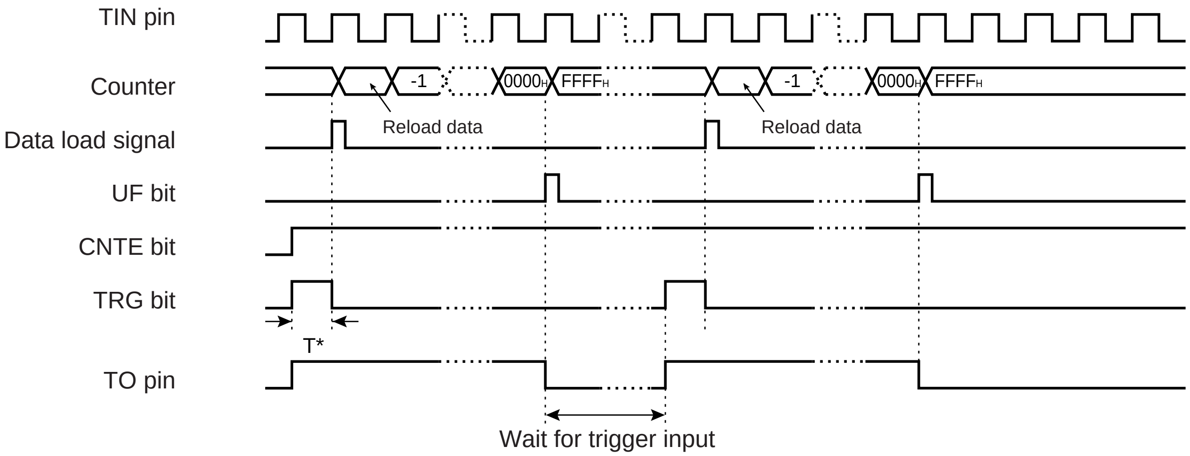
### ❍ Operation in single-shot mode

When an underflow of the 16-bit down counter occurs (change from "$0000_H$" to "$FFFF_H$"), the 16-bit down counter stops counting at the value "$FFFF_H$".

An interrupt request is output to the CPU when an underflow of the 16-bit down counter (change from $0000_H$ to $FFFF_H$) occurs while the underflow interrupt request flag bit (UF) of the timer control status register (TMCSR) is "1" and the underflow interrupt request enable bit (INTE) is "1".

The timer can also output from the TO pin a rectangular waveform indicating that count.

**Figure 12.6-11 Counter Operation in Single-Shot Mode (Event Count Mode)**



T: Machine cycle (one cycle of the machine clock)
* It takes 1T time from trigger input to loading of the reload data.

**Note:**

Specify $2^2/\phi$ or more for the H and L widths of the pluse input to the TIN pin.

# 12.7 Usage Notes on the 16-Bit Reload Timer

**Notes on using the 16-bit reload timer are given below.**

## ■ Usage Notes on the 16-Bit Reload Timer

### ❍ Notes on using a program for setting

- Write a value to the 16-bit reload register (TMRLR) when counting stops (TMCSR: CNTE = 0). Also, a value can be read from the 16-bit timer register (TMR) even during counting, but always be sure to use a word transfer instruction (MOVW A, dir, etc.).
- Counting must be stopped (TMCSR: CNTE = "0") when the count clock setting bits (CSL1 and CSL0) of the timer control register (TMCSR) are changed.

### ❍ Notes about interrupts

- When the UF bit of the timer control status register (TMCSR) is set to 1 and an interrupt request is enabled (TMCSR: INTE = 1), control cannot be returned from interrupt processing. Always clear the UF bit.
- The 16-bit reload timer shares the interrupt control register with the 8/16-bit PPG timers. Therefore, if multiple interrupts of the same level are output, the interrupt with a smaller interrupt vector number has precedence.

# CHAPTER 13 16-BIT I/O TIMER

**This chapter describes the functions and operations of the 16-bit I/O timer of the MB90M405 series.**

# 13.1 Overview of the 16-Bit I/O Timer

**The 16-bit I/O timer, which is based on a 16-bit free-running timer, can output two independent waveforms and measure an input pulse width and an external clock cycle.**

## ■ 16-bit Free-Running Timer (One Channel)

The 16-bit free-running timer consists of a 16-bit up counter (timer data register [TCDT]), timer control status register (TCCS), and prescaler.

The counter output value of the 16-bit free-running timer is used as the basic time (base timer) for output compare and input capture.

### ❍ Counter operation clock (one of four types)

Internal clock types: $\phi/4$, $\phi/16$, $\phi/32$, $\phi/64$

$\phi$: Machine clock frequency

### ❍ Interrupt

An interrupt is issued to the CPU when a counter overflow occurs or the counter value matches the value of compare register 0.

### ❍ Initialization

The counter value is initialized to $0000_H$ when a reset occurs, the software reset bit is cleared to "0", or the value of compare register 0 matches the count value of the free-running timer.

## ■ Output Compare (One Channel)

The output compare module consists of a 16-bit compare register and a control register. When the value of the 16-bit free-running timer matches the compare register value, an interrupt is issued to the CPU.

## ■ Input Capture (Two Channels)

The input capture module consists of a capture register and a control register, each of which corresponds to two independent external input pins. The capture register stores the value of the 16-bit free-running timer. It can also issue an interrupt to the CPU when an edge of the signal input from an external pin is detected.

- The edge to be detected (rising, falling, or both edges) of an external input signal can be selected.
- Two input capture modules can operate independently.
- An interrupt can be issued upon detection of a valid edge of an external input signal.

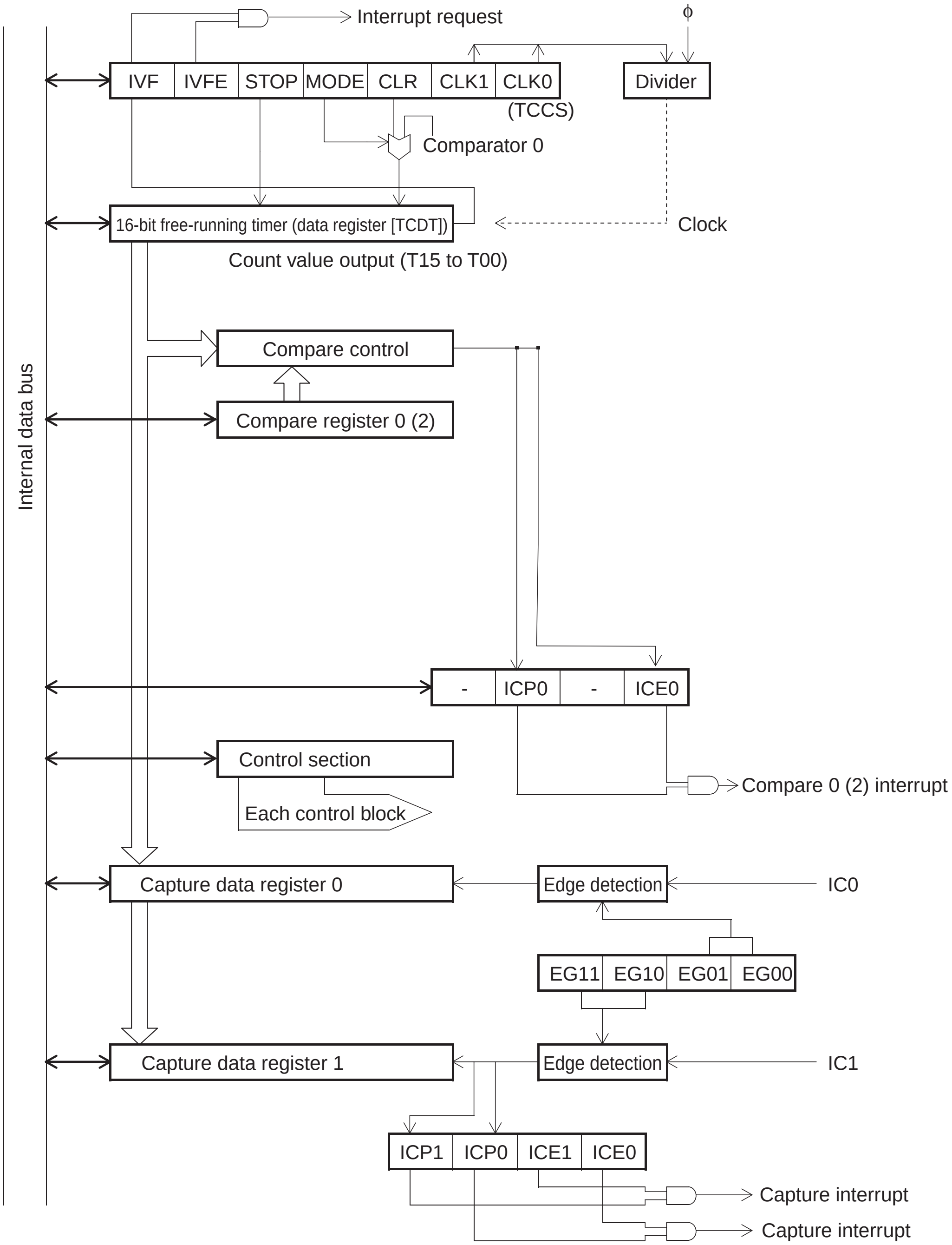
  An input capture interrupt can be used to start the extended intelligent I/O service.

# 13.2 16-Bit I/O Timer Block Diagram

**Figure 13.2-1 "16-Bit I/O Timer Block Diagram" shows a block diagram of the 16-bit I/O timer.**

## ■ 16-Bit I/O Timer Block Diagram

**Figure 13.2-1 16-Bit I/O Timer Block Diagram**



# 13.3 16-Bit I/O Timer Registers

**The 16-bit I/O timer has the following six types of registers:**

- **Timer counter data register (TCDT)**
- **Timer counter control status register (TCCS)**
- **Output compare register (OCCP0)**
- **Output compare control status register (OCS0)**
- **Input capture data register (IPC0 and IPC1)**
- **Input capture control status register (ICS01)**

## ■ 16-Bit I/O Timer Registers

**Figure 13.3-1 16-Bit I/O Timer Registers**

| bit15 ··· bit8 | bit7 ··· bit0 |
|---|---|
| Timer counter data register upper (TCDT) | Timer counter data register lower (TCDT) |
| | Timer counter control status register (TCCS) |
| Output compare register upper (OCCP) | Output compare register lower (OCCP) |
| | Output compare control status register (OCS0) |
| Input capture data register upper (IPC) | Input capture data register lower (IPC) |
| | Input capture control status register (ICS01) |

## 13.3.1 16-Bit Free-Running Timer Registers (TCDT and TCCS)

**The 16-bit free-running timer has the following registers:**

- **Timer counter data register (TCDT)**
- **Timer counter control status register (TCCS)**

### ■ Timer Counter Data Register (TCDT)

The count value of the 16-bit free-running timer can be read from the timer counter data register (TCDT). A reset clears the counter value to $0000_B$. You can set the timer value via the timer counter data register. The setting must be made in stop state (STOP="1").

The 16-bit free-running timer is initialized by any of the following events:

- Reset
- Initialization due to the clear bit (CLR) of the control status register
- Initialization due to a match between compare register 0 for output compare and the timer counter value (mode setting is required)

**Figure 13.3-2 Timer Counter Data Register (TCDT)**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | T15 | T14 | T13 | T12 | T11 | T10 | T09 | T08 | $00000000_B$ |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | T07 | T06 | T05 | T04 | T03 | T02 | T01 | T00 | $00000000_B$ |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

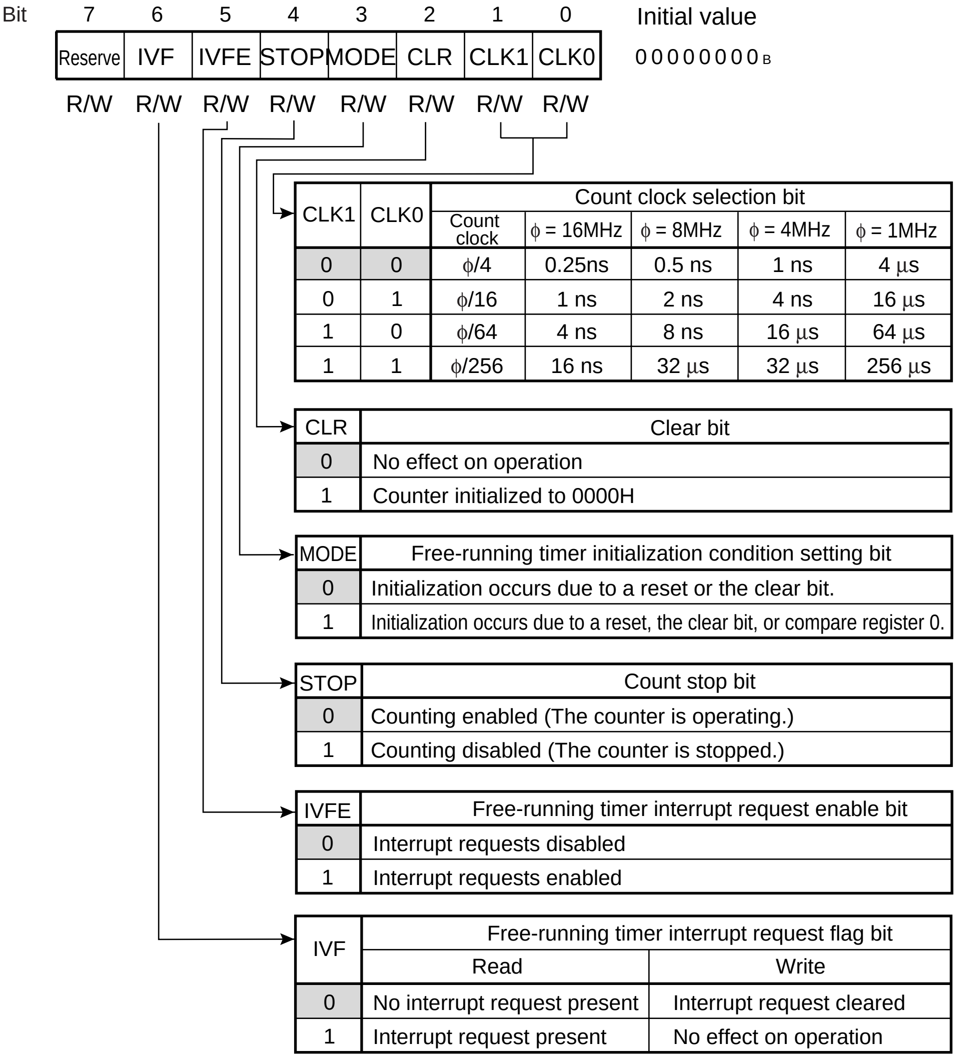
R/W : Read/write enabled

**Note:**

The timer counter data register (TCDT) requires word access.

## ■ Timer Counter Control Status Register (TCCS)

**Figure 13.3-3 Timer Counter Control Status Register (TCCS)**



| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | Reserve | IVF | IVFE | STOP | MODE | CLR | CLK1 | CLK0 | 00000000B |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

| CLK1 | CLK0 | Count clock selection bit | | | | |
|---|---|---|---|---|---|---|
| | | Count clock | ϕ = 16MHz | ϕ = 8MHz | ϕ = 4MHz | ϕ = 1MHz |
| 0 | 0 | ϕ/4 | 0.25ns | 0.5 ns | 1 ns | 4 µs |
| 0 | 1 | ϕ/16 | 1 ns | 2 ns | 4 ns | 16 µs |
| 1 | 0 | ϕ/64 | 4 ns | 8 ns | 16 µs | 64 µs |
| 1 | 1 | ϕ/256 | 16 ns | 32 µs | 32 µs | 256 µs |

| CLR | Clear bit |
|---|---|
| 0 | No effect on operation |
| 1 | Counter initialized to 0000H |

| MODE | Free-running timer initialization condition setting bit |
|---|---|
| 0 | Initialization occurs due to a reset or the clear bit. |
| 1 | Initialization occurs due to a reset, the clear bit, or compare register 0. |

| STOP | Count stop bit |
|---|---|
| 0 | Counting enabled (The counter is operating.) |
| 1 | Counting disabled (The counter is stopped.) |

| IVFE | Free-running timer interrupt request enable bit |
|---|---|
| 0 | Interrupt requests disabled |
| 1 | Interrupt requests enabled |

| IVF | Free-running timer interrupt request flag bit | |
|---|---|---|
| | Read | Write |
| 0 | No interrupt request present | Interrupt request cleared |
| 1 | Interrupt request present | No effect on operation |

R/W : Read/write enabled
▨ : Initial value
ϕ : Machine clock frequency

**Table 13.3-1 Functions of the Timer Counter Control Status Register (TCCS) Bits**

| Bit name | | Function |
|---|---|---|
| bit 7 | Reserved:<br>Reserved bit | • Be sure to set this bit to "0". |
| bit 6 | IVF:<br>Free-running timer interrupt request flag bit | • This bit is a flag bit for an interrupt request.<br>• This bit is set to "1" if the counter value of the 16-bit free-running timer overflows.<br>• An interrupt is output if this bit is set to "1" while the free-running timer interrupt enable bit (IVFE) is "1".<br>• An interrupt request is cleared if this bit is set to "0".<br>• Setting this bit to "1" does not effect operation.<br>• Read-modify-write instructions return "1" for this bit. |
| bit 5 | IVFE:<br>Free-running timer interrupt request enable bit | • This bit enables interrupt requests.<br>• An interrupt is output if the free-running timer interrupt request flag bit (IVF) is set to "1" while this bit is "1". |
| bit 4 | STOP:<br>Count stop bit | • This bit stops the 16-bit free-running timer counter.<br>• If this bit is set to "0", the 16-bit free-running timer counter runs.<br>• If this bit is set to "1", the 16-bit free-running timer counter stops.<br>**Note:**<br>• When the 16-bit free-running timer counter stops, the output compare operation also stops. |
| bit 3 | MODE:<br>Free-running timer initialization condition setting bit | • This bit sets the initialization condition of the 16-bit free-running timer counter value.<br>• If this bit is set to "0", a reset or setting the clear bit (CLR="1") initializes the counter value to $0000_H$.<br>• If this bit is set to "1", a reset, setting the clear bit (CLR="1"), or a match between the 16-bit free-running timer counter value and the compare clear register (CPCLR) value initializes the counter value to $0000_H$.<br>**Note:**<br>• The counter value is cleared at the next counting operation after the detection of an initialization condition as specified in the MODE bit. |
| bit 2 | CLR:<br>Clear bit | • The CLR bit clears the counter value to $0000_H$ while the 16-bit free-running timer counter is running.<br>• If this bit is set to "0", operation is not affected.<br>• If this bit is set to "1", the counter value is cleared to $0000_H$.<br>• The read value of this bit is always "0".<br>**Note:**<br>• To clear the counter value to $0000_H$ while the 16-bit free-running timer counter is stopped (STOP="1"), set the timer data register (TCDT) to $0000_H$. |
| bit 1<br>bit 0 | CLK1 and CLK0:<br>Count clock selection bits | • These bits select the count clock of the 16-bit free-running timer.<br>• Set these bits while output compare and input capture are stopped, since the count clock changes as soon as these bits are set. |

## 13.3.2 Output Compare Registers (OCCP0 and OCS0)

**The output compare unit uses the following registers:**

- **Output compare register (OCCP0)**
- **Output compare control status register (OCS0)**

### ■ Output Compare Register (OCCP0)

The output compare register (OCCP0) is a 16-bit register whose value is compared with the 16-bit free-running timer. Because the initial value of this register is undefined, set an initial value before enabling timer operation. When the output compare register value matches the 16-bit free-running timer value, a compare signal that sets the output compare interrupt request flag bit to "1" is generated.

**Figure 13.3-4 Output Compare Register (OCCP0)**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | C15 | C14 | C13 | C12 | C11 | C10 | C09 | C08 | XXXXXXXX$_B$ |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
| | C07 | C06 | C05 | C04 | C03 | C02 | C01 | C00 | XXXXXXXX$_B$ |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

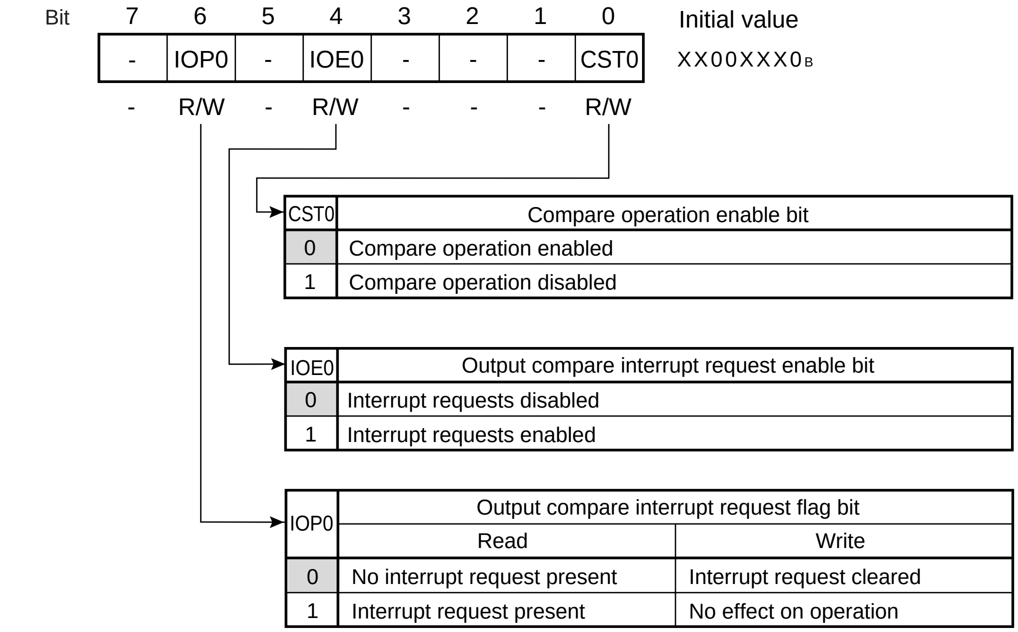
R/W : Read/write enabled
X : Undefined

**Note:**

The output compare register (OCCP0) requires word access.

## ■ Output Compare Control Status Register (OCS0)

**Figure 13.3-5 Output Compare Control Status Register (OCS0)**



| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
|  | - | IOP0 | - | IOE0 | - | - | - | CST0 | XX00XXX0$_B$ |
|  | - | R/W | - | R/W | - | - | - | R/W |  |

| CST0 | Compare operation enable bit |
|---|---|
| 0 | Compare operation enabled |
| 1 | Compare operation disabled |

| IOE0 | Output compare interrupt request enable bit |
|---|---|
| 0 | Interrupt requests disabled |
| 1 | Interrupt requests enabled |

| IOP0 | Output compare interrupt request flag bit |  |
|---|---|---|
|  | Read | Write |
| 0 | No interrupt request present | Interrupt request cleared |
| 1 | Interrupt request present | No effect on operation |

R/W : Read/write enabled
X : Undefined
- : Undefined bit
▨ : Initial value

**Table 13.3-2 Functions of the Output Compare Control Status Register (OCS0) Bits**

| Bit name | | Function |
|---|---|---|
| bit 7 | -:<br>Undefined bit | • The value read from this bit is undefined.<br>• Setting this bit to a new value does not affect operation. |
| bit 6 | IOP0:<br>Output compare interrupt request flag bit | • This bit is a flag bit for interrupt requests.<br>• This bit is set to "1" if the compare register (OCCP) value matches the value of the 16-bit free-running timer counter.<br>• An interrupt is output to the CPU if this bit is set to "1" while the output compare interrupt enable bit (IOE0) is "1".<br>• An interrupt request is cleared if this bit is set to "0".<br>• Setting this bit to "1" does not affect operation.<br>• Read-modify-write instructions return "1" for this bit. |
| bit 5 | -:<br>Undefined bit | • The value read from this bit is undefined.<br>• Setting this bit to a new value does not affect operation. |
| bit 4 | IOE0:<br>Output compare interrupt request enable bit | • This bit enables interrupt requests.<br>• An interrupt request is output to the CPU if the output compare interrupt request flag bit (IOP0) is set to "1" while this bit is "1". |
| bit 3<br>bit 2 | -:<br>Undefined bit | • The value read from this bit is undefined.<br>• Setting this bit to a new value does not affect operation. |
| bit 1<br>bit 0 | CST1, CST0:<br>Compare operation enable bit | • This bit enables a compare operation between the output compare register (OCCP0) value and the 16-bit free-running timer counter value.<br>• Set the output compare register (OCCP0) value before enabling the compare operation.<br>• If this bit is set to "1", the compare operation is enabled. |

**Note:**

Since output compare is synchronized with the 16-bit free-running timer clock, the compare operation stops when the 16-bit free-running timer stops (TCCS:STOP="1").

## 13.3.3 Input Capture Registers (IPC0/IPC1 and ICSSS0)

**The input capture unit has the following registers:**

- **Input capture data registers (IPC0/IPC1)**
- **Input capture control status register (ICS01)**

### ■ Input Capture Data Registers (IPC0/IPC1)

The input capture data registers (IPC0/IPC1) retain the value of the 16-bit free-running timer when a valid edge of the corresponding external pin input waveform is detected.

**Figure 13.3-6 Input Capture Data Registers (IPC0/IPC1)**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | CP15 | CP14 | CP13 | CP12 | CP11 | CP10 | CP09 | CP08 | XXXXXXXX$_B$ |
| | R | R | R | R | R | R | R | R | |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | CP07 | CP06 | CP05 | CP04 | CP03 | CP02 | CP01 | CP00 | XXXXXXXX$_B$ |
| | R | R | R | R | R | R | R | R | |

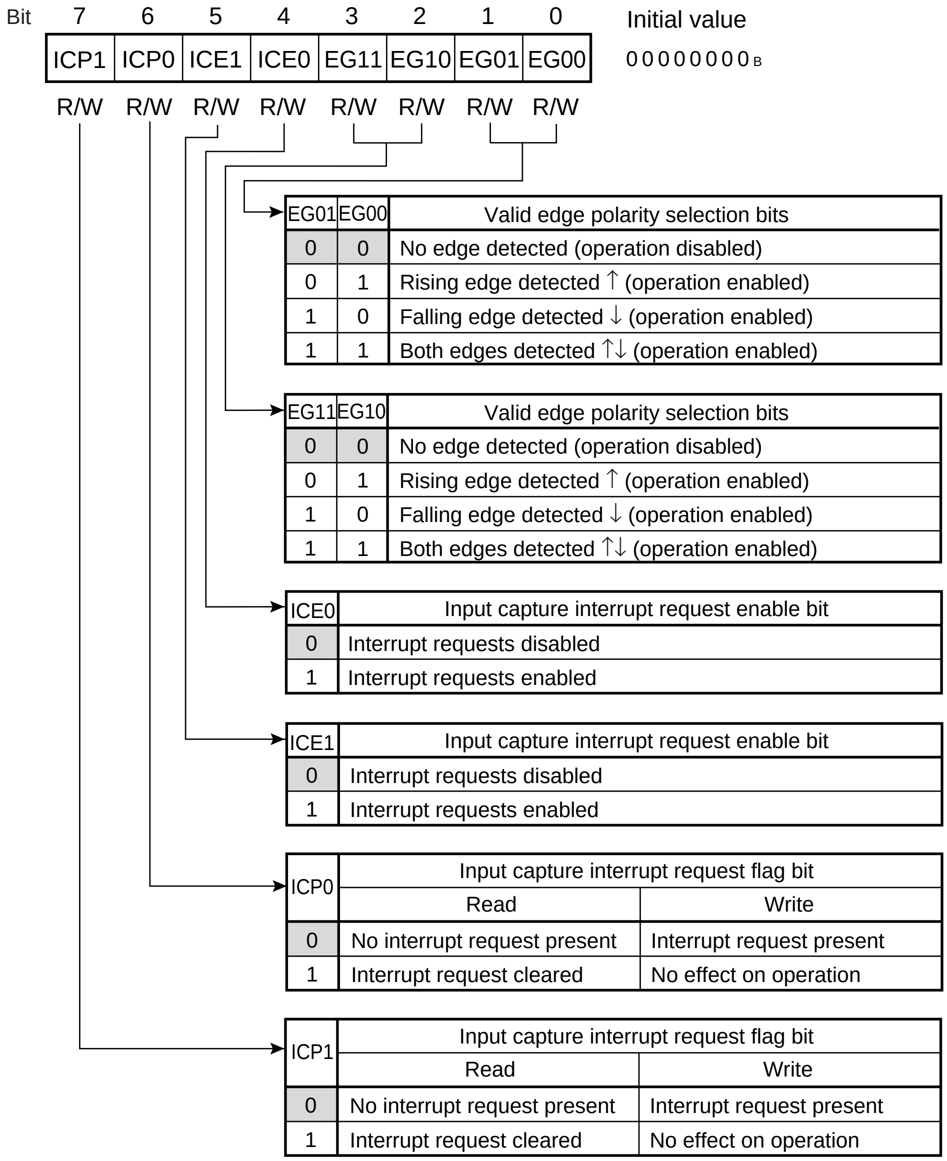
R : Read only
X : Undefined

**Note:**

The input capture data registers (IPC0/IPC1) require word access. Writing to these registers is disabled.

## ■ Input Capture Control Status Register (ICS01)

**Figure 13.3-7 Input Capture Control Status Register (ICS01)**



| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | ICP1 | ICP0 | ICE1 | ICE0 | EG11 | EG10 | EG01 | EG00 | 00000000B |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

| EG01 | EG00 | Valid edge polarity selection bits |
|---|---|---|
| 0 | 0 | No edge detected (operation disabled) |
| 0 | 1 | Rising edge detected ↑ (operation enabled) |
| 1 | 0 | Falling edge detected ↓ (operation enabled) |
| 1 | 1 | Both edges detected ↑↓ (operation enabled) |

| EG11 | EG10 | Valid edge polarity selection bits |
|---|---|---|
| 0 | 0 | No edge detected (operation disabled) |
| 0 | 1 | Rising edge detected ↑ (operation enabled) |
| 1 | 0 | Falling edge detected ↓ (operation enabled) |
| 1 | 1 | Both edges detected ↑↓ (operation enabled) |

| ICE0 | Input capture interrupt request enable bit |
|---|---|
| 0 | Interrupt requests disabled |
| 1 | Interrupt requests enabled |

| ICE1 | Input capture interrupt request enable bit |
|---|---|
| 0 | Interrupt requests disabled |
| 1 | Interrupt requests enabled |

| ICP0 | Input capture interrupt request flag bit | |
|---|---|---|
| | Read | Write |
| 0 | No interrupt request present | Interrupt request present |
| 1 | Interrupt request cleared | No effect on operation |

| ICP1 | Input capture interrupt request flag bit | |
|---|---|---|
| | Read | Write |
| 0 | No interrupt request present | Interrupt request present |
| 1 | Interrupt request cleared | No effect on operation |

R/W : Read/write enabled
▨ : Initial value

**Note:**

The input capture data register (ICS01) requires byte access.

Table 13.3-3 Functions of the Input Capture Control Status Register (ICS01) Bits

| Bit name | | Function |
|---|---|---|
| bit 7<br>bit 6 | ICP1, ICP0:<br>Input capture interrupt request flag bit | • This bit is a flag bit for interrupt requests.<br>• This bit is set to "1" if a valid edge of an external input pin is detected.<br>• An interrupt request is output if this bit is set to "1" while the input capture interrupt enable bits (ICE1 and ICE0) are "1".<br>• An interrupt request is cleared if this bit is set to "0".<br>• Setting this bit to "1" does not affect operation.<br>• Read-modify-write instructions return "1" for this bit. |
| bit 5<br>bit 4 | ICE1, ICE0:<br>Input capture interrupt request enable bit | • This bit enables interrupt requests.<br>• An interrupt request is output if the input capture interrupt request flag bits (ICP1, ICP0) are set to "1" while this bit is "1". |
| bit 3<br>to<br>bit 0 | EG11, EG10, EG01, EG00:<br>Valid edge polarity selection bits | • These bits select the polarity of a valid edge of the input waveform and enable or disable the input capture operation.<br>• The input capture operation is enabled if these bits are set to $01_B$ to $11_B$. The polarity of a valid edge of the input waveform can be selected as rising edge, falling edge, or both edges.<br>• If these bits are set to $00_B$, input capture operation and edge detection are disabled. |

# 13.4 16-Bit Free-Running Timer Operations

**The 16-bit free-running timer starts counting from the counter value $0000_B$ after a reset is released. The counter value is used as the reference time for16-bit output compare and 16-bit input capture.**

## ■ 16-Bit Free-Running Timer Operations

A counter value is cleared in the following cases:

- The counter value overflows.
- The counter value matches the value of output compare register 0. (A mode must be set.)
- The clear bit (CLR) of the timer counter control status register (TCCS) is set to "1".
- The timer counter data register (TCDC) is set to $0000_B$ during a stop.
- A reset occurs.

An interrupt can be output to the CPU if the free-running timer overflows or if the counter is cleared when the free-running timer value matches the value of compare register 0 (A mode must be set for the compare match interrupt).

**Figure 13.4-1 Counter Cleared by an Overflow**

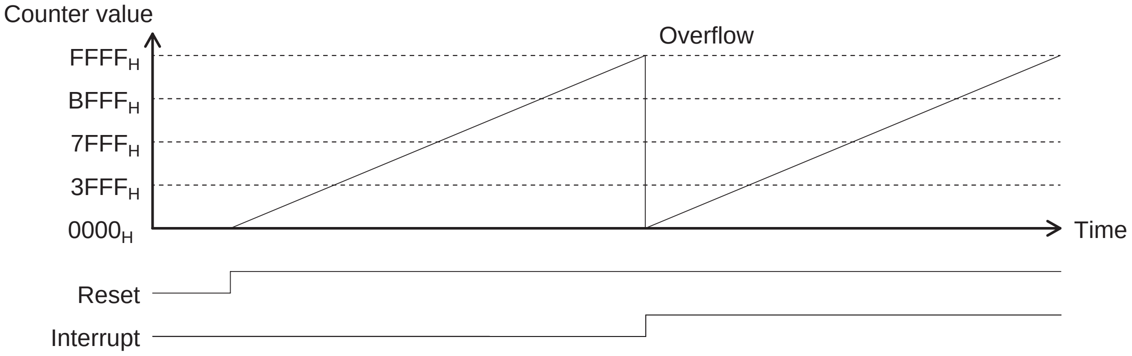
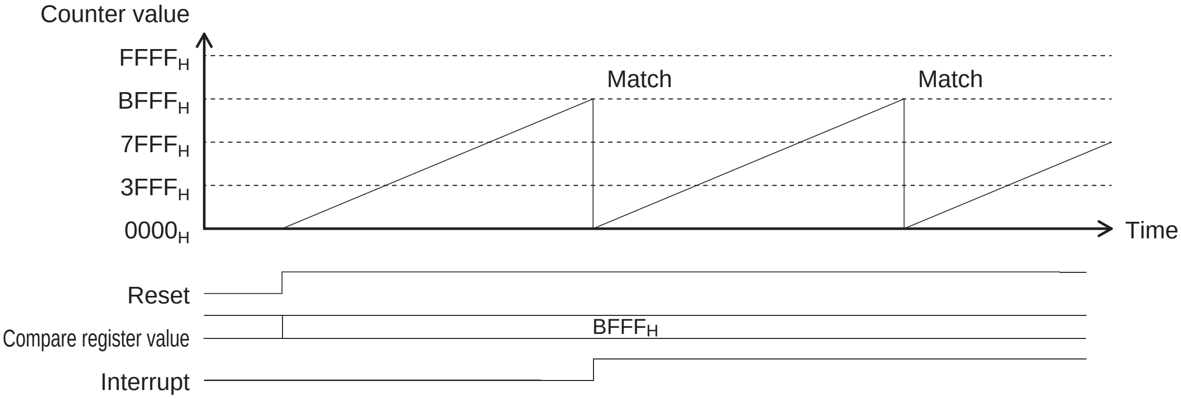


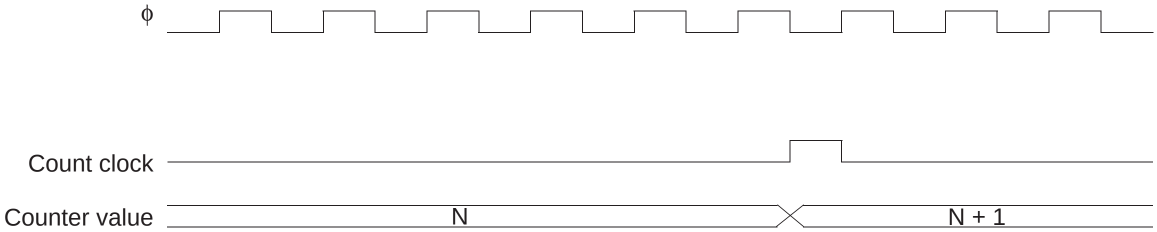
**Figure 13.4-2 Counter Cleared by a Compare Match with Output Compare Register 0 Value**



## ■ 16-bit Free-Running Timer Count Timing

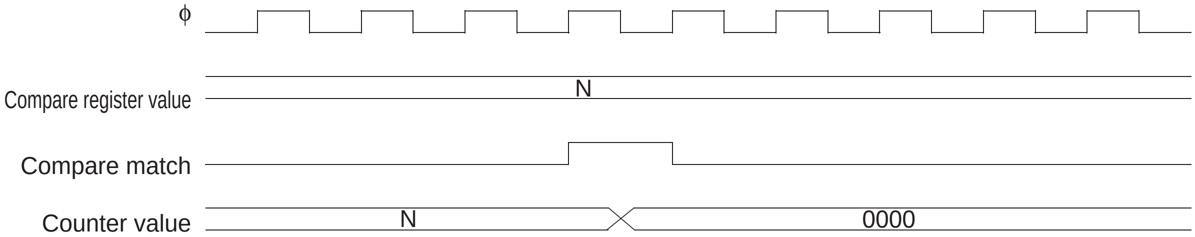
The 16-bit free-running timer counts up based on the specified internal clock.

**Figure 13.4-3 Count Timing of the Free-Running Timer**



The counter can be cleared using a reset, software clear, or match with compare register 0. A reset or software clears the counter immediately. A match with compare register 0 clears the counter in synchronization with the count timing.

**Figure 13.4-4 Clear Timing of the Free-Running Timer (Match with Compare Register 0)**



# 13.5 16-Bit Output Compare Operations

**A 16-bit output compare compares the specified compare register value with the value of the 16-bit free-running timer. If a match occurs, the interrupt request flag bit is set to "1".**

## ■ 16-bit Output Compare Operations

Output compare outputs an interrupt when the free-running timer value matches the specified compare register value and a compare match signal is generated.

**Figure 13.5-1 Compare Operation When Compare Register Rewritten**

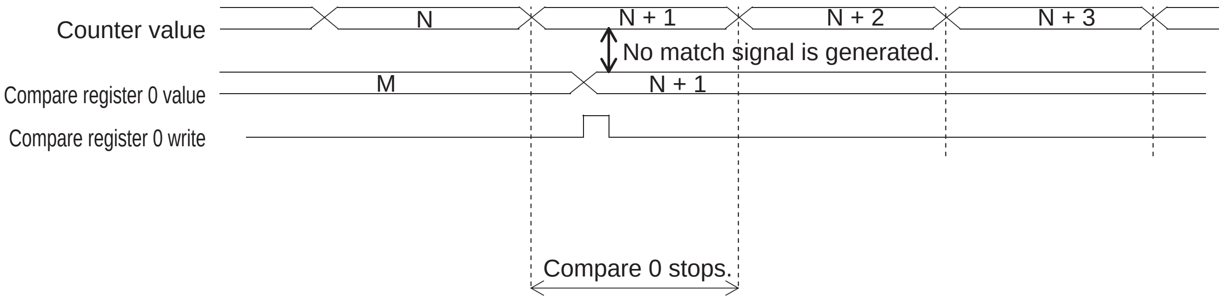
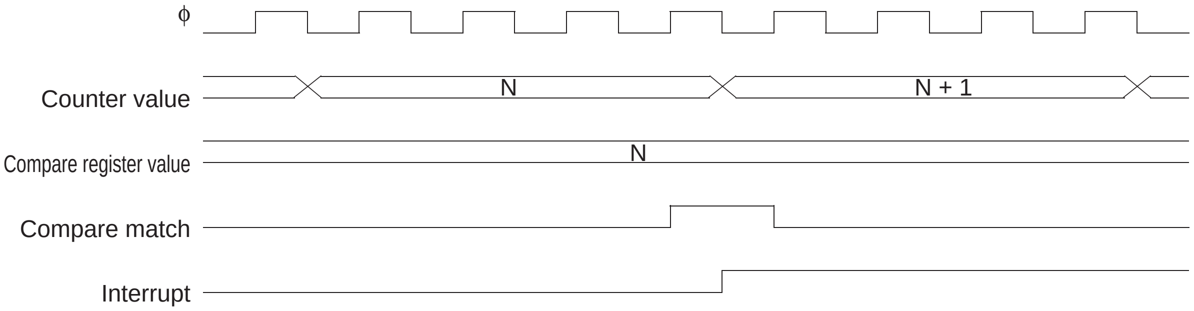


**Figure 13.5-2 Interrupt Timing of Output Compare**

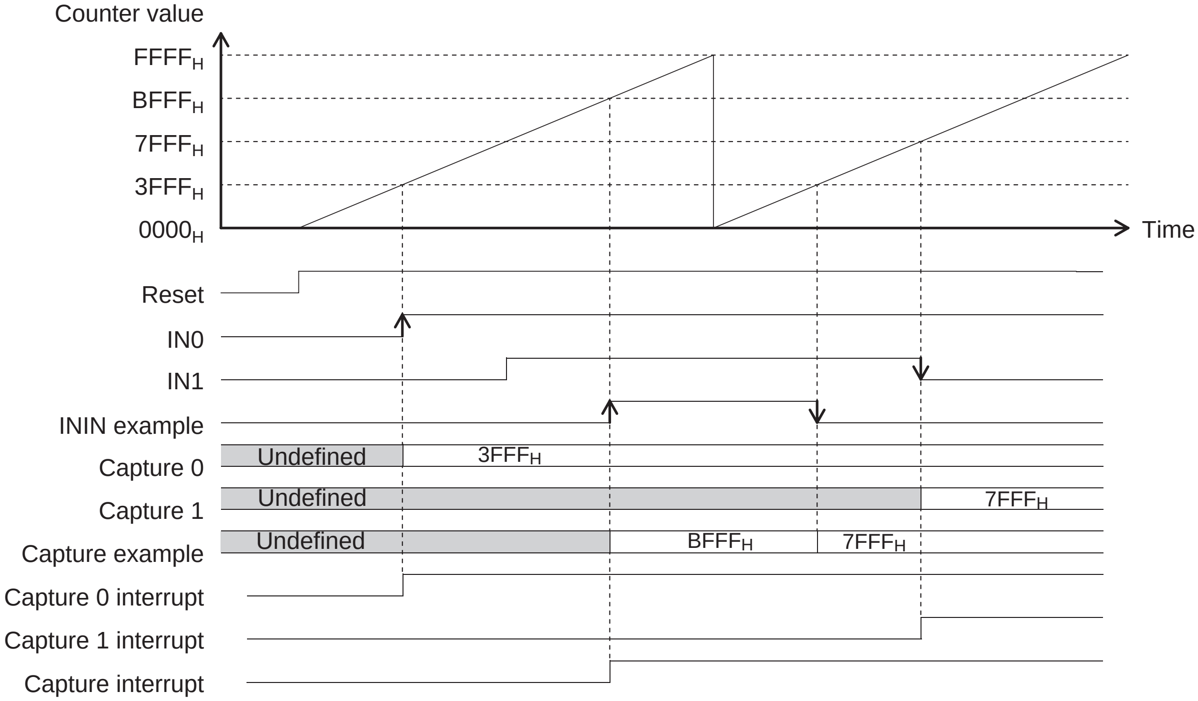


# 13.6 16-Bit Input Capture Operations

**The 16-bit input capture can capture the 16-bit free-running timer value in the capture register for outputting an interrupt when the specified valid edge is detected.**

## ■ 16-bit Input Capture Operations

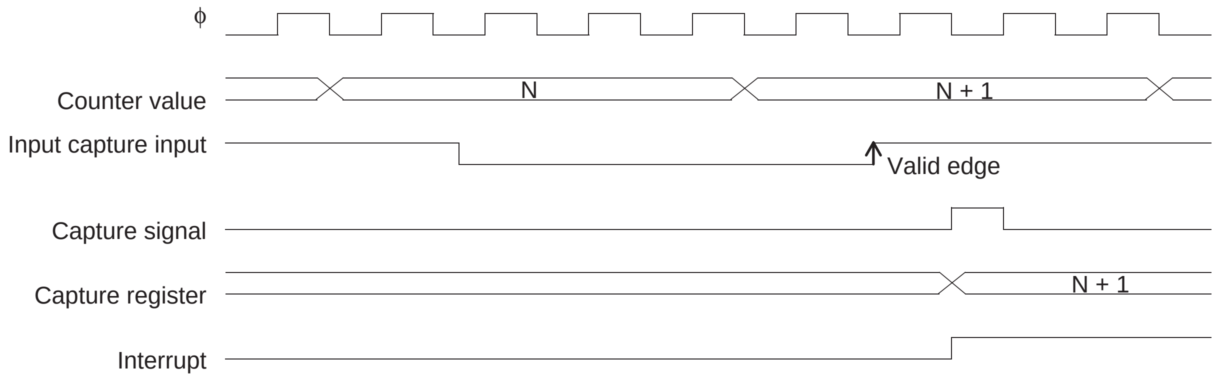
**Figure 13.6-1 Example of Input Capture Timing**



Capture0 = Rising edge
Capture1 = Falling edge
Capture example = Both edges (example)

## ■ Input Capture Input Timing

**Figure 13.6-2  Capture Timing for Input Signals**



# CHAPTER 14 UART

**This chapter describes the functions and operations of the MB90M405 series UART.**

## 14.1 Overview of UART

**UART is a general-purpose serial data communication interface for performing synchronous or asynchronous (start-stop synchronization) communication with external devices. The UART has a bidirectional communication function (normal mode), additionally the master-slave communication function (multiprocessor mode) is only available for the master system.**

### ■ UART Functions

#### ❍ UART Functions

UART is a general-purpose serial data communication interface for transmitting serial data to and receiving data from another CPU and peripheral devices. It has the functions listed in Table 14.1-1 "UART Functions".

**Table 14.1-1 UART Functions**

| | Function |
|---|---|
| Data buffer | Full-duplex, double buffering |
| Transfer mode | • Clock synchronous (using start and stop bits)<br>• Clock asynchronous (start-stop synchronization) |
| Baud rate | • Up to 2MHz (when the machine clock is operated at 16MHz)<br>• A dedicated baud rate generator is provided.<br>• Baud rate by an external clock (clock input through the SCK pins)<br>• Internal clock (internal clocks supplied from 16-bit reload timer 0 can be used.)<br>• The baud rate can be selected from a total of eight types |
| Data length | • 7 bits (in asynchronous normal mode only)<br>• 8 bits |
| Signal mode | Non-return to zero (NRZ) |
| Reception error detection | • Framing error<br>• Overrun error<br>• Parity error (cannot be detected in multiprocessor mode.) |
| Interrupt request | • Reception interrupt (reception completion and reception error detection)<br>• Transmission interrupt (transmission completion)<br>• Extended intelligent I/O service (EI$^2$OS) is available for both transmission and reception interrupts. |
| Master-slave communication function (multiprocessor mode) | One-to-n communication (one master to n slaves) can be performed. (This function is supported only for the master system.) |

**Note:**

During clock synchronous transfer, start and stop bits are not added so only data is transferred in UART.

**Table 14.1-2 UART Operation Mode**

| Operation mode | | Data length | | Synchronization on mode | Stop bit length |
|---|---|---|---|---|---|
| | | When parity is disabled | When parity is enabled | | |
| 0 | Normal mode | 7 or 8 bits | | Asynchronous | 1 or 2 bits (*2) |
| 1 | Multiprocessor | 8+1 (*1) | - | Asynchronous | |
| 2 | Normal mode | 8 | - | Synchronous | None |

-: Setting not possible
*1: "+1" indicates the address/data selection bit (A/D) for communication control.
*2: During reception, only one stop bit can be detected.

## ■ UART Interrupt and EI$^2$OS

**Table 14.1-3 UART Interrupt and EI$^2$OS**

| Interrupt cause | Interrupt number | Interrupt control register | | Vector table address | | | EI$^2$OS |
|---|---|---|---|---|---|---|---|
| | | Register name | Address | Lower | Upper | Bank | |
| UART0 reception interrupt | #35(23$_H$) | ICR12 | 0000BC$_H$ | FFFF70$_H$ | FFFF71$_H$ | FFFF72$_H$ | ◎ |
| UART0 transmission interrupt | #36(24$_H$) | | | FFFF6C$_H$ | FFFF6D$_H$ | FFFF6E$_H$ | △ |
| UART1 reception interrupt | #39(27$_H$) | ICR14 | 0000BE$_H$ | FFFF60$_H$ | FFFF61$_H$ | FFFF62$_H$ | ◎ |
| UART1 transmission interrupt | #40(28$_H$) | | | FFFF5C$_H$ | FFFF5D$_H$ | FFFF5E$_H$ | △ |

◎ : Provided with a function that detects a UART reception error and stops EI$^2$OS
△ : Usable when ICR12 and ICR14 or interrupt causes that share an interrupt vector are not used

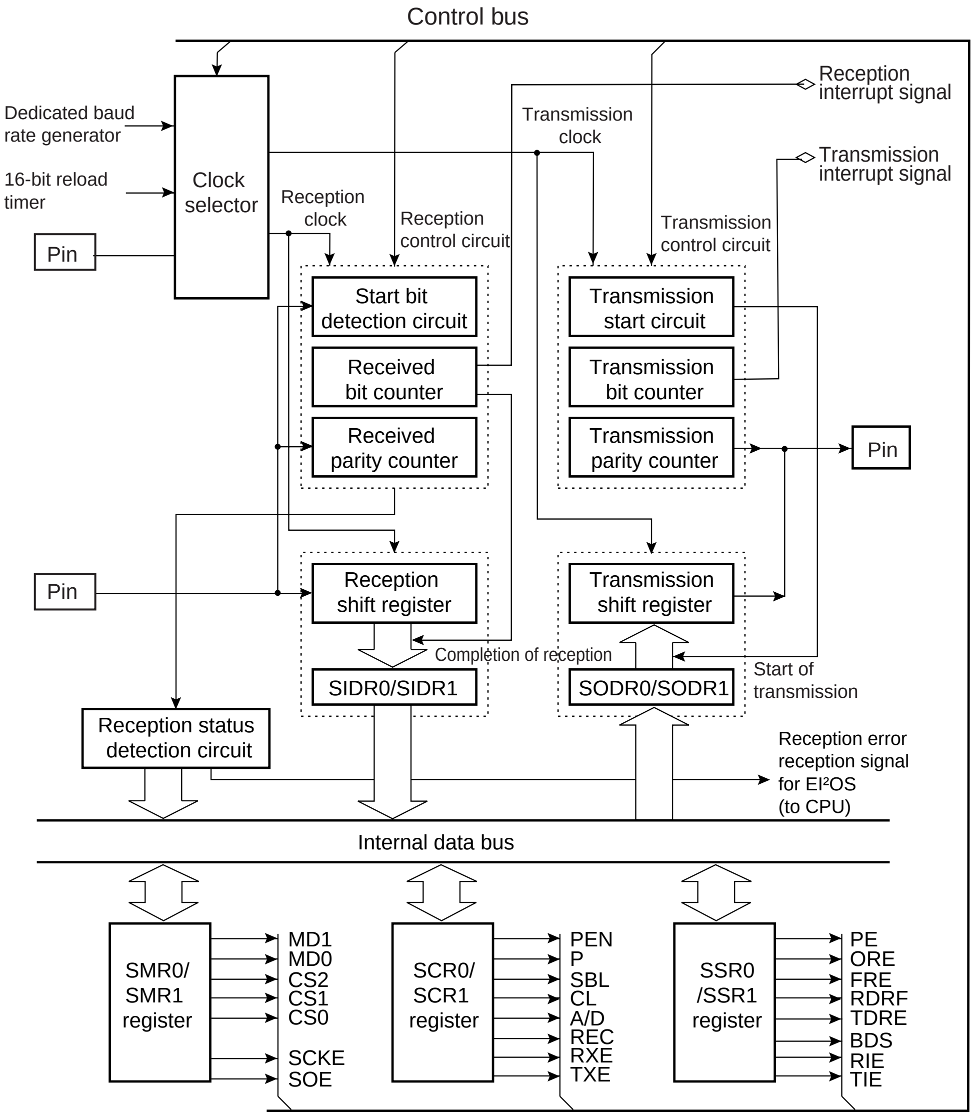
## 14.2 Configuration of UART

**UART consists of the following 11 blocks:**

- **Clock Selector**
- **Reception Control Circuit**
- **Transmission Control Circuit**
- **Reception Status Detection Circuit**
- **Reception Shift Register**
- **Transmission Shift Register**
- **Mode Control Register (SMR0/1)**
- **Control Register (SCR0/1)**
- **Status Register (SSR0/1)**
- **Input Data Register (SIDR0/1)**
- **Output Data Register (SODR0/1)**

### ■ Block Diagram of UART

**Figure 14.2-1 Block Diagram of UART**



### ❍ Clock Selector

The clock selector selects the sending/receiving clock from the dedicated baud rate generator, external input clock (clock input through the SCK0/SCK1 pins), and internal clock (clock supplied from the 16-bit reload timer).

### ❍ Reception Control Circuit

The reception control circuit consists of a received bit counter, start bit detection circuit, and received parity counter.The received bit counter counts receive data bits. When reception of one data item for the specified data length is complete, the received bit counter generates a reception interrupt request. The start bit detection circuit detects start bits from the serial input signal. When the circuit detects a start bit, it writes data in the SIDR0/1 register by shifting at the specified transfer rate. The received parity counter calculates the parity of the receive data.

### ❍ Transmission Control Circuit

The transmission control circuit consists of a transmission bit counter, transmission start circuit, and transmission parity counter.The transmission bit counter counts transmission data bits. When transmission of one data item of the specified data length is complete, the transmission bit counter generates a transmission interrupt request. The transmission start circuit starts transmission when send data is written to the output data register (SODR0/SODR1). The transmission parity counter generates the parity bits for data when transmitting data with parity.

### ❍ Reception Shift Register

The reception shift register fetches receive data input from the SIN0/1 pin, shifting the data bit by bit. When reception is complete, the reception shift register transfers receive data to the SIDR0/1 register.

### ❍ Transmission Shift Register

The transmission shift register transfers data written to the SODR0/1 register to itself and outputs the data to the SOT0/1 pin, shifting the data bit by bit.

### ❍ Mode Control Register (SMR0/1)

The mode register performs the operations of the operation mode setting, baud rate clock setting, serial clock I/O control, and output enable setting of serial data to pins.

### ❍ Control Register (SCR0/1)

The control register performs the operations of the parity presence/absence setting, parity setting, stop bit length/data length settings, frame data format setting in operation mode 1, clearing of received error flag bits, and enable/disable setting of send/receive operations.

### ❍ Status Register (SSR0/1)

The status register performs the operations of status check for transmission/reception and errors, transfer direction setting of serial data, and enable/disable setting of send/receive interrupt requests.

### ❍ Input Data Register (SIDR0/1)

This register retains receive data.

### ❍ Output Data Register (SODR0/1)

This register sets transmission data. Data written to this register is converted to serial data and output.

# 14.3 UART Pins

**This section describes the UART pins and provides a pin block diagram.**

## ■ UART Pins

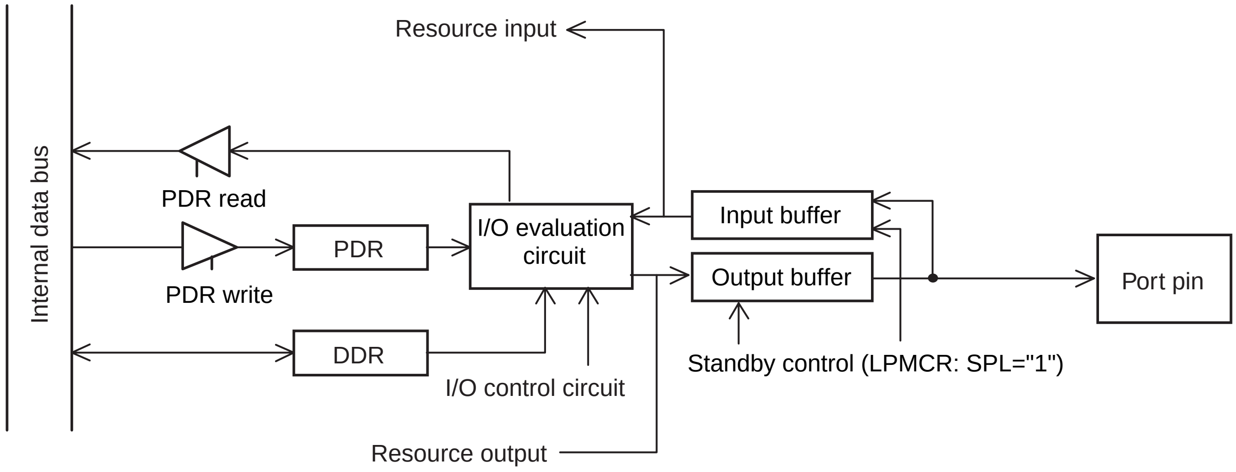
The UART pins also serve as I/O ports.

**Table 14.3-1 UART Pins**

| Pin name | Pin function | I/O format | Pull-up | Standby control | Setting required to use pin |
|---|---|---|---|---|---|
| SI0 | Port I/O or serial data input | CMOS output and CMOS hysteresis input | Nothing | Provided | Set as an input port (DDR8: bit2 = 0) |
| SO0 | Port I/O or serial data output | | | | Set to serial data output enable mode (SMR0: SOE = 1) |
| SC0 | Port I/O or serial clock input/output | | | | Set as an input port (DDR8: bit3 = 0) |
| | | | | | Set to serial clock output enable mode (SMR0:SCKE = 1) |
| SI1 | Port I/O or serial data input | | | | Set as an input port (DDR8: bit5 = 0) |
| SO1 | Port I/O or serial data output | | | | Set to serial data output enable mode (SMR1: SOE = 1) |
| SC1 | Port I/O or serial clock input/output | | | | Set as an input port (DDR8: bit6 = 0) |
| | | | | | Set to serial clock output enable mode (SMR1:SCKE = 1) |

## ■ Block Diagram of UART Pins

**Figure 14.3-1 Block Diagram of UART Pins**

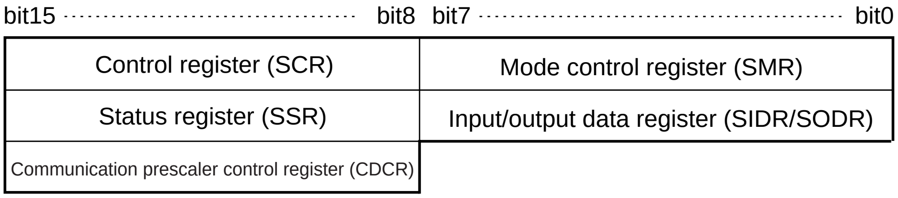


# 14.4 UART Registers

**The following figure shows the UART registers.**

## ■ UART Registers

**Figure 14.4-1 UART Registers**



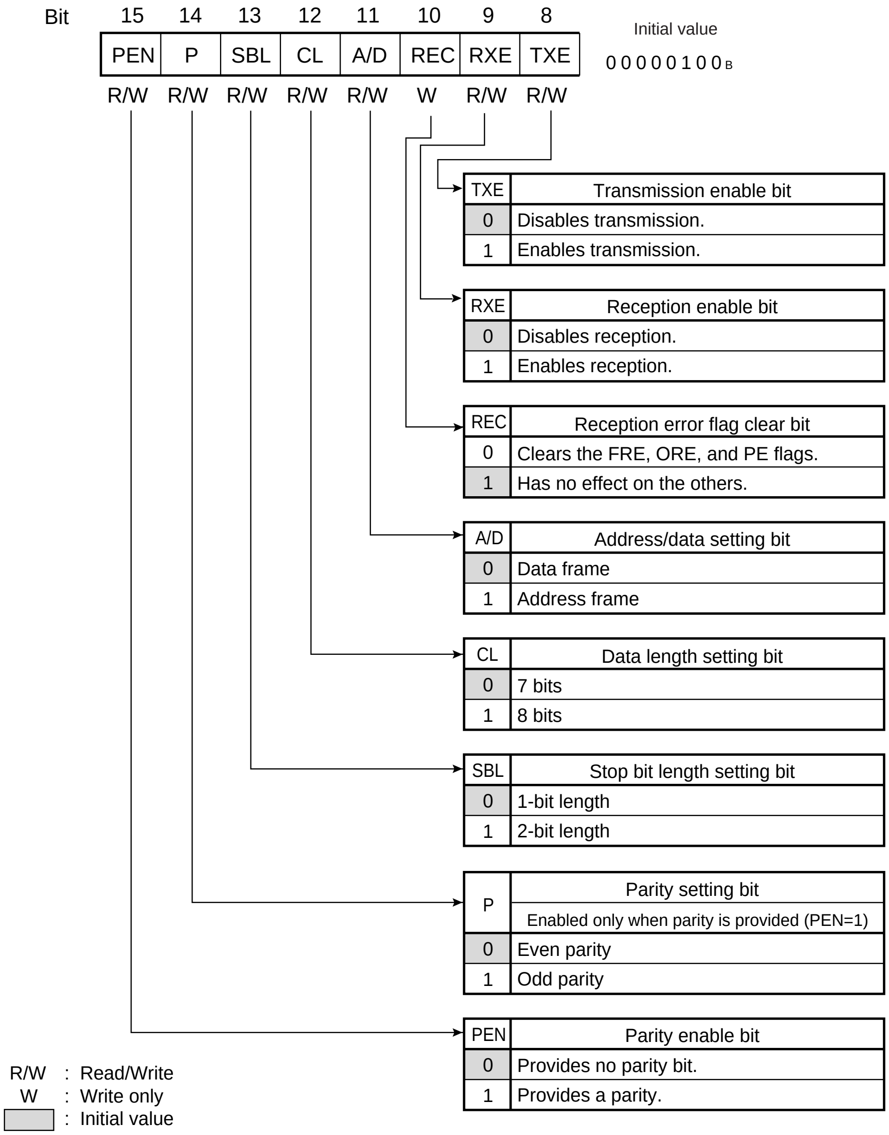
| bit15 ······ bit8 | bit7 ······ bit0 |
|---|---|
| Control register (SCR) | Mode control register (SMR) |
| Status register (SSR) | Input/output data register (SIDR/SODR) |
| Communication prescaler control register (CDCR) | |

## 14.4.1 Control Register (SCR0/SCR1)

**The control register (SCR0/SCR1) is a register for performing the operations of the parity presence/absence setting, parity setting, stop bit length/data length settings, frame data format setting in operation mode 1, clearing of received error flag bits, and enable/disable setting of send/receive operations.**

### ■ Control Register (SCR0/SCR1)

**Figure 14.4-2 Control Register (SCR0/SCR1)**



| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | PEN | P | SBL | CL | A/D | REC | RXE | TXE |
| | R/W | R/W | R/W | R/W | R/W | W | R/W | R/W |

Initial value
$00000100_B$

| TXE | Transmission enable bit |
|---|---|
| 0 | Disables transmission. |
| 1 | Enables transmission. |

| RXE | Reception enable bit |
|---|---|
| 0 | Disables reception. |
| 1 | Enables reception. |

| REC | Reception error flag clear bit |
|---|---|
| 0 | Clears the FRE, ORE, and PE flags. |
| 1 | Has no effect on the others. |

| A/D | Address/data setting bit |
|---|---|
| 0 | Data frame |
| 1 | Address frame |

| CL | Data length setting bit |
|---|---|
| 0 | 7 bits |
| 1 | 8 bits |

| SBL | Stop bit length setting bit |
|---|---|
| 0 | 1-bit length |
| 1 | 2-bit length |

| P | Parity setting bit<br>Enabled only when parity is provided (PEN=1) |
|---|---|
| 0 | Even parity |
| 1 | Odd parity |

| PEN | Parity enable bit |
|---|---|
| 0 | Provides no parity bit. |
| 1 | Provides a parity. |

R/W : Read/Write
W : Write only
▨ : Initial value

Table 14.4-1 Functions of Bits for Control Register (SCR0/SCR1)

| Bit name | | Function |
|---|---|---|
| bit15 | PEN: Parity enable bit | This bit selects whether to add a parity bit during transmission in serial data input-output mode or to detect it during reception.<br>**Note:**<br>No parity can be used in operation modes 1 and 2. Therefore, fix this bit to 0. |
| bit14 | P: Parity selection bit | This bit specifies the odd parity/even parity.<br>**Note:**<br>Valid only when parity presence (PEN="1") is selected. |
| bit13 | SBL: Stop bit length selection bit | This bit selects the length of the stop bits or the frame end mark of send data in asynchronous transfer mode.<br>**Note:**<br>During reception, only the first bit of the stop bits is detected. |
| bit12 | CL: Data length selection bit | This bit specifies the length of send and receive data.<br>**Note:**<br>Seven bits can be selected in operation mode 0 (asynchronous) only. Be sure to select eight bits (CL=1) in operation mode 1 (multiprocessor mode) and operation mode 2 (synchronous). |
| bit11 | A/D: Address/ data selection bit | • Specify the data format of a frame to be sent or received in multiprocessor mode (mode 1).<br>• Select usual data when this bit is 0, and select address data when the bit is 1. |
| bit10 | REC: Reception error flag clear bit | • This bit clears the FRE, ORE, and PE flags of the status register (SSR0/1).<br>• Write 0 to this bit to clear the FRE, ORE, and PE flag. Writing 1 to this bit has no effect on the others.<br>**Note:**<br>If UART is active and a reception interrupt is enabled, clear the REC bit only when the FRE, DRE, or PE flag indicates 1. |
| bit9 | RXE: Reception enable bit | • This bit controls UART reception.<br>• When this bit is 0, reception is disabled. When it is 1, reception is enabled.<br>**Note:**<br>If reception operation is disabled during reception, reception of data currently being received is finished and the received data is stored in the input data register (SIDR0/SIDR1), and then the reception operation is stopped. |

**Table 14.4-1 Functions of Bits for Control Register (SCR0/SCR1) (Continued)**

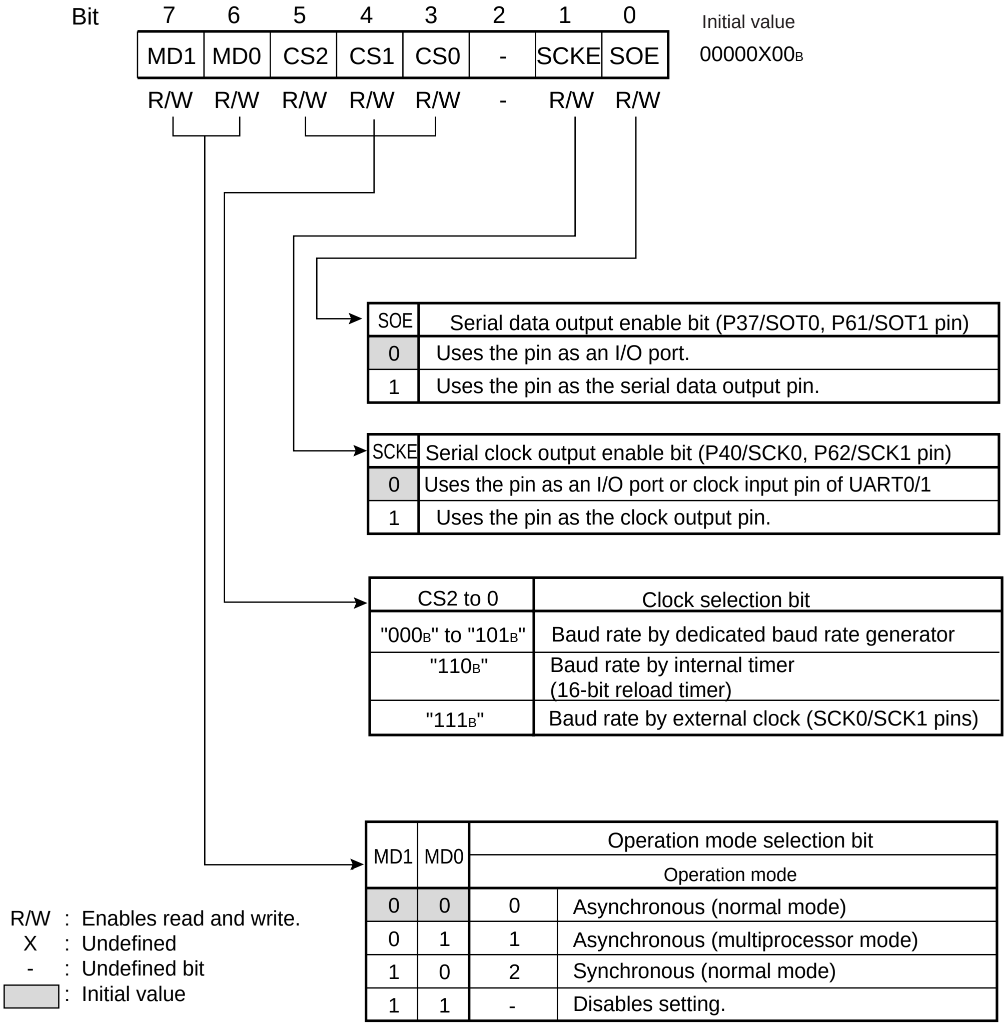
| Bit name | | Function |
|---|---|---|
| bit8 | TXE:<br>Transmission enable bit | • This bit controls UART transmission.<br>• When this bit is 0, transmission is disabled. When the bit is 1, transmission is enabled.<br>**Note:**<br>When transmission operation is disabled during transmission, wait until there is no data in the send data buffers (SODR0/1) before stopping the transmission operation.<br>If transmission operation is disabled during transmission, transmission operation is stopped after all data in the output data register (SODR0/SODR1) is sent out. When setting "0", write data to the output data register (SODR0/SODR1) and then wait for at least 1/16 period of the baud rate for the clock asynchronous transfer mode and the same period as the baud rate for the clock synchronous transfer mode. |

## 14.4.2 Mode Register (SMR0/SMR1)

**The mode register (SMR0/SMR1) is a register for performing the operations of the operation mode setting, baud rate clock setting, serial clock I/O control, and output enable setting of serial data to pins.**

### ■ Mode Control Register (SMR0/SMR1)

Figure 14.4-3 Mode Control Register (SMR0/SMR1)



| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | MD1 | MD0 | CS2 | CS1 | CS0 | - | SCKE | SOE |
| | R/W | R/W | R/W | R/W | R/W | - | R/W | R/W |

Initial value $00000X00_B$

| SOE | Serial data output enable bit (P37/SOT0, P61/SOT1 pin) |
|---|---|
| 0 | Uses the pin as an I/O port. |
| 1 | Uses the pin as the serial data output pin. |

| SCKE | Serial clock output enable bit (P40/SCK0, P62/SCK1 pin) |
|---|---|
| 0 | Uses the pin as an I/O port or clock input pin of UART0/1 |
| 1 | Uses the pin as the clock output pin. |

| CS2 to 0 | Clock selection bit |
|---|---|
| "$000_B$" to "$101_B$" | Baud rate by dedicated baud rate generator |
| "$110_B$" | Baud rate by internal timer (16-bit reload timer) |
| "$111_B$" | Baud rate by external clock (SCK0/SCK1 pins) |

| MD1 | MD0 | Operation mode selection bit | |
|---|---|---|---|
| | | Operation mode | |
| 0 | 0 | 0 | Asynchronous (normal mode) |
| 0 | 1 | 1 | Asynchronous (multiprocessor mode) |
| 1 | 0 | 2 | Synchronous (normal mode) |
| 1 | 1 | - | Disables setting. |

R/W : Enables read and write.
X : Undefined
- : Undefined bit
▨ : Initial value

**Table 14.4-2 Functions of Bits for Mode Register (SMR0/SMR1)**

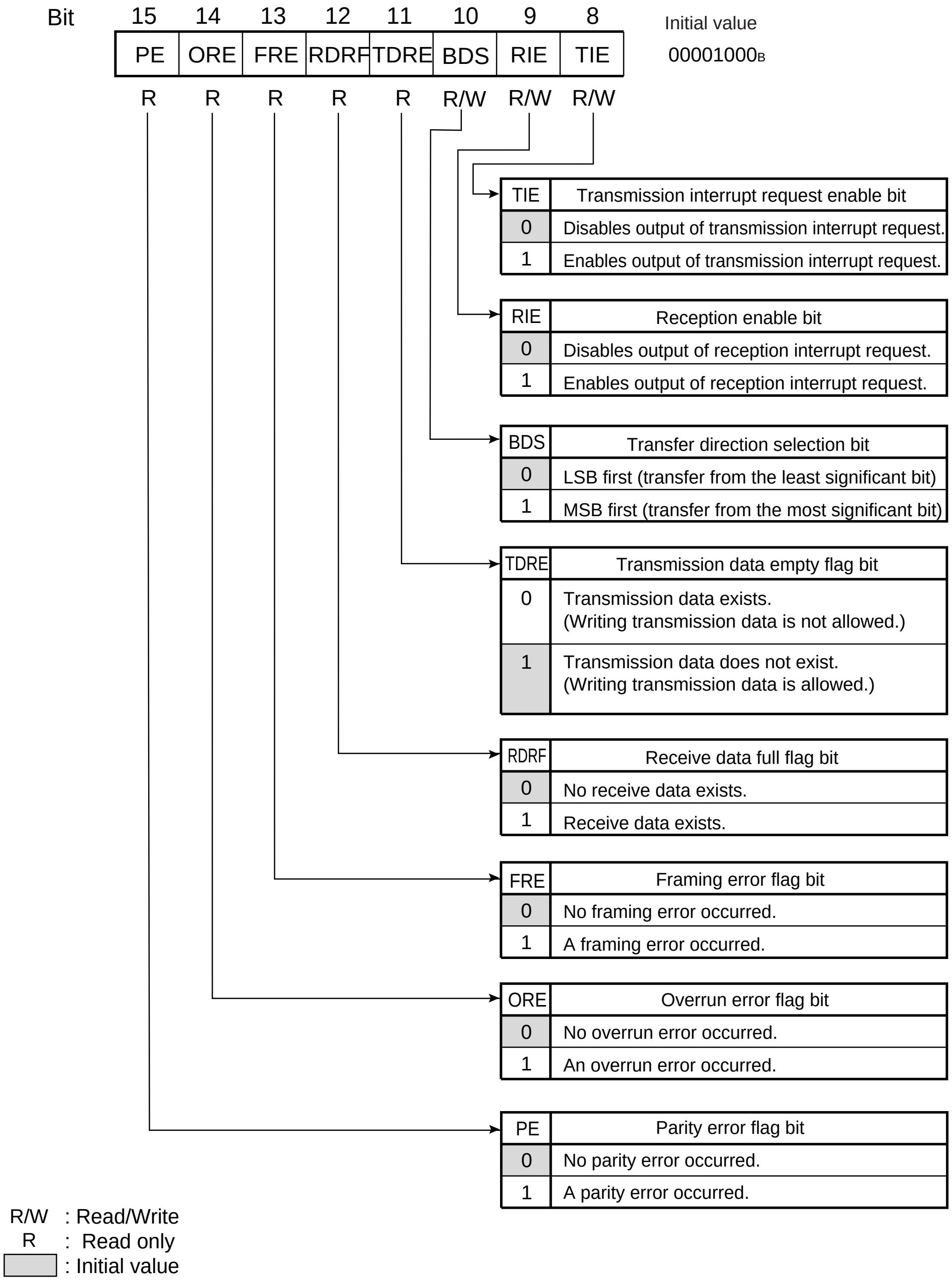
| Bit name | | Function |
|---|---|---|
| bit7<br>bit6 | MD1 and MD0:<br>Operation mode selection bits | These bits select an operation mode.<br>**Note:**<br>Operation mode 1 (multiprocessor mode) can be used only from the master system during master-slave communication. UART cannot be used from the slave system because it has no address/data detection function during reception. |
| bit5<br>bit4<br>bit3 | CS2 to CS0:<br>Clock selection bits | • This bit selects a baud rate clock source. When the dedicated baud rate generator is selected, the baud rate is determined at the same time.<br>• When the dedicated baud rate generator is selected, eight baud rates can be selected: 6 types for the dedicated baud rate generator selected, 1 type for the internal timer selected, and 1 type for the external clock selected.<br>• Clock input can be selected from external clocks (SCK0/SCK1 pin input), the internal clock (16-bit reload timer0), and the dedicated baud rate generator. |
| bit2 | -:<br>Undefined bit | • When this bit is read, the value is undefined.<br>• The value set to this bit does not affect operation. |
| bit1 | SCKE:<br>Serial clock output enable bit | • This bit controls the serial clock input-output ports.<br>• When this bit is 0, the SC pins operate as input-output ports or serial clock input pins. When this bit is 1, the pins operate as serial clock output pins.<br>**Note:**<br>- To use the SC pins as serial clock input (SCKE="0") pins, set them as input pins. Also, select an external clock (SMR0/SMR1: CS2 to CS0="$111_B$") using the clock setting bits.<br>- To use the SC pins as serial clock output (SCKE="1"), select the dedicated baud rate generator (SMR0/SMR1:CS2 to CS0=$000_B$ to $101_B$) or internal clock (SMR0/SMR1:CS2 to CS0=$110_B$).<br>**Reference:**<br>When the SCK0/1 pin is assigned to serial clock output (SCKE=1), it functions as the serial clock output pin regardless of the status of the input-output ports. |
| bit0 | SOE:<br>Serial data output enable bit | • This bit enables the output of serial data.<br>• When this bit "0", the SO pins operate as an I/O port.<br>• When this bit is "1", the SO pins operate as a serial data output pin.<br>**Reference:**<br>When the serial data output (SOE="1") is selected, the pins operate as a serial data output pin regardless of the state of the I/O port. |

### 14.4.3 Status Register (SSR0/SSR1)

**The status register (SSR0/SSR1) is a register for performing the operations of status check for transmission/reception and errors, transfer direction setting of serial data, and enable/disable setting of send/receive interrupts.**

#### ■ Status Register (SSR0/SSR1)

Figure 14.4-4 Status Register (SSR0/SSR1)



| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | PE | ORE | FRE | RDRF | TDRE | BDS | RIE | TIE | 00001000B |
| | R | R | R | R | R | R/W | R/W | R/W | |

| TIE | Transmission interrupt request enable bit |
|---|---|
| 0 | Disables output of transmission interrupt request. |
| 1 | Enables output of transmission interrupt request. |

| RIE | Reception enable bit |
|---|---|
| 0 | Disables output of reception interrupt request. |
| 1 | Enables output of reception interrupt request. |

| BDS | Transfer direction selection bit |
|---|---|
| 0 | LSB first (transfer from the least significant bit) |
| 1 | MSB first (transfer from the most significant bit) |

| TDRE | Transmission data empty flag bit |
|---|---|
| 0 | Transmission data exists.<br>(Writing transmission data is not allowed.) |
| 1 | Transmission data does not exist.<br>(Writing transmission data is allowed.) |

| RDRF | Receive data full flag bit |
|---|---|
| 0 | No receive data exists. |
| 1 | Receive data exists. |

| FRE | Framing error flag bit |
|---|---|
| 0 | No framing error occurred. |
| 1 | A framing error occurred. |

| ORE | Overrun error flag bit |
|---|---|
| 0 | No overrun error occurred. |
| 1 | An overrun error occurred. |

| PE | Parity error flag bit |
|---|---|
| 0 | No parity error occurred. |
| 1 | A parity error occurred. |

R/W : Read/Write
R : Read only
▨ : Initial value

**Table 14.4-3 Functions of Bits for Status Register (SSR0/SSR1)**

| No. | Bit name | Function |
|---|---|---|
| bit15 | PE:<br>Parity error flag bit | • This bit is set to "1" when a parity error occurs during reception.<br>• This bit is cleared to "0" when "0" is written to the reception error flag clear bit (REC) of the control register (SCR0/SCR1).<br>• When this bit is set to "1" while the reception interrupt request enable bit (RIE) is 1, a reception interrupt request is output.<br>• When this bit is set to "1", data in the input data register (SIDR0/SIDR1) will be invalid. |
| bit14 | ORE:<br>Overrun error flag bit | • This bit is set to "1" when an overrun error occurs during reception.<br>• This bit is cleared to "0" when "0" is written to the reception error flag clear bit (REC) of the control register (SCR0/SCR1).<br>• When this bit is set to "1" while the reception interrupt request enable bit (RIE) is 1, a reception interrupt request is output.<br>• When this bit is set to "1", data in the input data register (SIDR0/SIDR1) will be invalid. |
| bit13 | FRE:<br>Framing error flag bit | • This bit is set to "1" when a framing error occurs during reception.<br>• This bit is cleared to "0" when "0" is written to the reception error flag clear bit (REC) of the control register (SCR0/SCR1).<br>• When this bit is set to "1" while the reception interrupt request enable bit (RIE) is 1, a reception interrupt request is output.<br>• When this bit is set to "1", data in the input data register (SIDR0/SIDR1) will be invalid. |
| bit12 | RDRF:<br>Receive data full flag bit | • This bit indicates the status of the input data register (SIDR0/SIDR1).<br>• This bit is set to "1" when the receive data is stored in the input data register (SIDR0/SIDR1).<br>• This bit is cleared to "0" when the input data register (SIDR0/SIDR1) is read.<br>• When the reception interrupt request enable bit (RIE) is set to 1 while this bit is "1", a reception interrupt request is output. |
| bit11 | TDRE: Transmission data empty flag bit | • This bit indicates the status of the output data register (SODR0/SODR1).<br>• This bit is cleared to "0" when transmission data is written to the output data register (SODR0/SODR1).<br>• This bit is set to "1" when data is sent after loading it into the transmission shift register.<br>• When the transmission interrupt request enable bit (TIE) is set to 1 while this bit is "1", a transmission interrupt request is output.<br>**Note:**<br>"1" is set in the initial state. |
| bit10 | BDS:<br>Transfer direction selection bit | • This bit specifies the transfer direction of serial data.<br>• When this bit is set to "0", transfer starts with the lowest-order bit (LSB first).<br>• When this bit is set to "1", transfer starts with the highest-order bit (MSB first).<br>**Note:**<br>The upper bits and lower bits of data are exchanged during reading from or writing to the serial data register. Written data therefore becomes invalid if the bit direction selection bit (BDS) is rewritten after the data was written to the output data register (SODR0/SODR1). |

Table 14.4-3 Functions of Bits for Status Register (SSR0/SSR1) (Continued)

| No. | Bit name | Function |
|---|---|---|
| bit9 | RIE:<br>Reception interrupt request enable bit | • This bit enables a reception interrupt request.<br>• When the reception data full flag enable bit (RDRF) is set to "1" or one of the reception error flag bits (PE, ORE, and FRE) is set to "1" while this bit is "1", a reception interrupt request is output. |
| bit8 | TIE:<br>Transmission interrupt request enable bit | • This bit enables a transmission interrupt request.<br>• When the transmission data empty flag bit (TDRE) is set to "1" while this bit is "1", a transmission interrupt request is output. |

## 14.4.4 Input Data Register (SIDR0/SIDR1) and Output Data Register (SODR0/SODR1)

**The input data register (SIDR0/SIDR1) is a serial data reception register. The output data register (SODR0/SODR1) is a serial data transmission register.**

### ■ Input Data Register (SIDR0/SIDR1)

**Figure 14.4-5 Input Data Register (SIDR0/SIDR1)**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | XXXXXXXX$_B$ |
| | R | R | R | R | R | R | R | R | |

R : Read only
X : Indefinite

The input data register (SIDR0/SIDR1) is a register to store receive data. The serial data signal transmitted to the SI0/SI1 pin is converted in the shift register and then stored in the input data register (SIDR0/SIDR1). When the data length is 7 bits long in operation mode 0, bit7 (D7) becomes invalid. When receive data is stored in this register, the reception data full flag bit (RDRF) of the status register (SSR0/SSR1) is set to "1". If the reception interrupt request output is enabled (SSR0/SSR1: RIE="1"), a reception interrupt is output.

Read the input data register (SIDR0/SIDR1) when the reception data full flag bit (RDRF) of the status register (SSR0/SSR1) is set to "1". The reception data full flag bit (RDRF) is cleared to "0" when the input data register (SIDR0/SIDR1) is read. When a reception error occurs (one of SSR0/SSR1: PE, ORE, FRE is "1"), data in the input data register (SIDR0/SIDR1) becomes invalid.

## ■ Output Data Register (SODR0/SODR1)

**Figure 14.4-6 Output Data Register (SODR0/SODR1)**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | $XXXXXXXX_B$ |
| | W | W | W | W | W | W | W | W | |

W : Write only
X : Indefinite

When data to be transmitted is written to the output data register (SODR0/SODR1), if transmission is enabled, the send data is transferred to the transmission shift register, converted into serial data, and then transmitted from the serial data output pin (SO0/SO1 pin). When the data length is 7 bits long in operation mode 0, bit7 (D7) becomes invalid.

When the transmission data is written to the output data register, the transmission data empty flag bit (TDRE) of the status register (SSR0/SSR1) is cleared to "0". When transfer to the transmission shift register is completed, the status register is set to "1". If the transmission data empty flag bit (TDRE) is "1", the next send data can be written. When the transmission data empty flag bit (TDRE) is set to "1" while the transmission interrupt request output is enabled (SSR0/SSR1: TIE="1"), a transmission interrupt is output. When a transmission interrupt is output, write the next transmission data after the transmission data empty flag bit (TDRE) is set to "1".

**Note:**

The output data register (SODR0/SODR1) is a write-only register and the input data register (SIDR0/SIDR1) is a read-only register. These registers are located at the same address, and so the read value is different from the write value. Therefore, instructions that perform a read-modify-write (RMW) operation such as the INC/DEC instruction cannot be used.

## 14.4.5 Communication Prescaler Control Register (CDCR0/ CDCR1)

**The communication prescaler control register (CDCR0/CDCR1) is a register that controls the division of machine clocks.**

### ■ Communication Prescaler Control Register (CDCR0/CDCR1)

The operation clocks of UART can be obtained by dividing machine clocks. UART is designed to obtain certain baud rates for various machine cycles. Output from the communication prescaler is used for the operation clocks of serial I/O.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|
| MD | - | - | - | RESV | DIV2 | DIV1 | DIV0 | $0XXX0000_B$ |
| R/W | - | - | - | R/W | R/W | R/W | R/W | |

R/W : Read/write
X : Undefined
- : Undefined bit

**Table 14.4-4 Functions of Bits for Communication Prescaler Control Register (CDCR0/CDCR1)**

| No. | Bit name | Function |
|---|---|---|
| bit15 | MD:<br>Communication prescaler operation enable bit | • This bit enables a communication prescaler operation.<br>• If this bit is "1", the communication prescaler operates.<br>• If this bit is "0", the communication prescaler stops. |
| bit14<br>bit13<br>bit12 | -:<br>Undefined bit | • When these bits are read, values are undefined.<br>• Values set to these bits do not affect operation. |
| bit11 | RESV:<br>Reservation bit | • Always set "0". |
| bit10<br>bit9<br>bit8 | DIV2 to DIV0:<br>Frequency division ratio setting bit | • These bits are used to set the frequency division ratio of the machine clock.<br>• For the set values, see Table 14.4-5 "Machine Clock Division Ratio". |

Table 14.4-5 Machine Clock Division Ratio

| MD | DIV2 | DIV1 | DIV0 | div |
|---|---|---|---|---|
| 0 | - | - | - | Stopped |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 2 |
| 1 | 0 | 1 | 0 | 3 |
| 1 | 0 | 1 | 1 | 4 |
| 1 | 1 | 0 | 0 | 5 |
| 1 | 1 | 0 | 1 | 6 |
| 1 | 1 | 1 | 0 | 7 |
| 1 | 1 | 1 | 1 | 8 |

div: Machine clock division ratio

**Note:**

If a division ratio is changed, wait two time cycles as clock stabilization time before starting communication.
The CDCR0 is a prescaler for the UART channel 0 (also serve as the serial I/O channel 2).
The CDCR1 is a prescaler for the UART channel 1 (also serve as the serial I/O channel 3).

# 14.5 UART Interrupts

**UART uses both reception and transmission interrupts and outputs an interrupt request for either of the following causes:**

- **A reception interrupt is output when receive data is set to the input data register (SIDR0/SIDR1) or a reception error occurs.**
- **A transmission interrupt is output when send data is transferred from the output data register (SODR0/SODR1) to the transmission shift register.**

**The extended intelligent I/O service (EI$^2$OS) is available for these interrupts.**

## ■ UART Interrupts

**Table 14.5-1 Interrupt Control Bits and Interrupt Causes of UART**

| Reception/ transmission | Interrupt request flag bit | Operation mode | | | Interrupt cause | Interrupt cause enable bit | When interrupt request flag is cleared |
|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | | | |
| Reception | RDRF | O | O | O | Loading receive data into buffers (SIDR0/1) | SSR0/SSR1: RIE | Receive data is read. |
| | ORE | O | O | O | Overrun error | | 0 is written to the reception error flag clear bit (SCR0/ 1:REC). |
| | FRE | O | O | X | Framing error | | |
| | PE | O | X | X | Parity error | | |
| Transmission | TDRE | O | O | O | Send data transfer from the output data register (SODR0/SODR1) completed | SSR0/ SSR1:TIE | Transmission data is written |

O: Used
X: Not used

### ❍ Reception Interrupt

In reception mode, "1" is set to the reception data full flag bit (RDRF) or one of the reception error flag bits (ORE, FRE, PE) in the status register (SSR0/SSR1) when data reception is completed, or an overrun error, framing error, or parity error occurs. If the reception interrupt is enabled (SSR0/SSR1: RIE="1"), a reception interrupt request is output.

The reception data full flag bit (RDRF) of the status register (SSR0/SSR1) is cleared to "0" when the input data register (SIDR0/SIDR1) is read. All reception error flag bits (PE, ORE, FRE) of the status register (SSR0/SSR1) are cleared to "0" when the reception error flag clear bit (REC) of the status register (SCR0/SCR1) is set to "0".

### ❍ Transmission Interrupt

The transmission data empty flag bit (TDRE) of the status register (SSR0/SSR1) is set to "1" when send data is transferred from the output data register (SODR0/SODR1) to the transfer shift register. If the transmission interrupt is enabled (SSR0/SSR1: TIE="1"), a transmission interrupt request is output.

## ■ UART Interrupts and EI$^2$OS

Table 14.5-2 UART Interrupts and EI$^2$OS

| Interrupt cause | Interrupt number | Interrupt control register | | Vector table address | | | EI$^2$OS |
|---|---|---|---|---|---|---|---|
| | | Register name | Address | Lower | Upper | Bank | |
| UART0 reception interrupt | #35(23$_H$) | ICR12 | 0000BC$_H$ | FFFF70$_H$ | FFFF71$_H$ | FFFF72$_H$ | ◎ |
| UART0 transmission interrupt | #36(24$_H$) | | | FFFF6C$_H$ | FFFF6D$_H$ | FFFF6E$_H$ | △ |
| UART1 reception interrupt | #39(27$_H$) | ICR14 | 0000BE$_H$ | FFFF60$_H$ | FFFF61$_H$ | FFFF62$_H$ | ◎ |
| UART1 transmission interrupt | #40(28$_H$) | | | FFFF5C$_H$ | FFFF5D$_H$ | FFFF5E$_H$ | △ |

◎ : Provided with a function that detects a UART reception error and stops EI$^2$OS
△ : Usable when interrupt causes that share the ICR12 and ICR14 or the interrupt vectors are not used

## ■ UART EI$^2$OS Functions

UART has a circuit for operating EI$^2$OS, which can be started up for either reception or transmission interrupts.

### ❍ For Reception

EI$^2$OS can be used regardless of the status of other resources.

### ❍ For Transmission

UART shares the interrupt registers (ICR14) with the UART reception interrupts. Therefore, EI$^2$OS can be started up only when no UART reception interrupts are used.

## 14.5.1 Reception Interrupt Generation and Flag Set Timing

**The following are reception interrupt causes: completion of reception (SSR0/SSR1: RDRF="1") and occurrence of a reception error (one of SSR0/SSR1: PE, ORE, and FRE is "1").**

### ■ Reception Interrupt Generation and Flag Set Timing

Receive data is stored in the input data register (SIDR0/SIDR1) and the reception data full flag bit (RDRF) of the status register (SSR0/SSR1) is set to "1" when a stop bit is detected (in operation mode 0 or 1) or the last bit of data (D7) is detected (in operation mode 2). When a reception error occurs, one of the reception error flags (PE, ORE, FRE) is set to "1". If any of the reception error flag bits in each operation mode is set to "1", the value contained in the input data register (SIDR0/SIDR1) becomes invalid.

#### ❍ Operation Mode 0 (Asynchronous, Normal Mode)

The reception data full flag bit (RDRF) of the status register (SSR0/SSR1) is set to "1" when a stop bit is detected. If a reception error is detected, "1" is set to one of the reception error flag bits (PE, ORE, FRE).

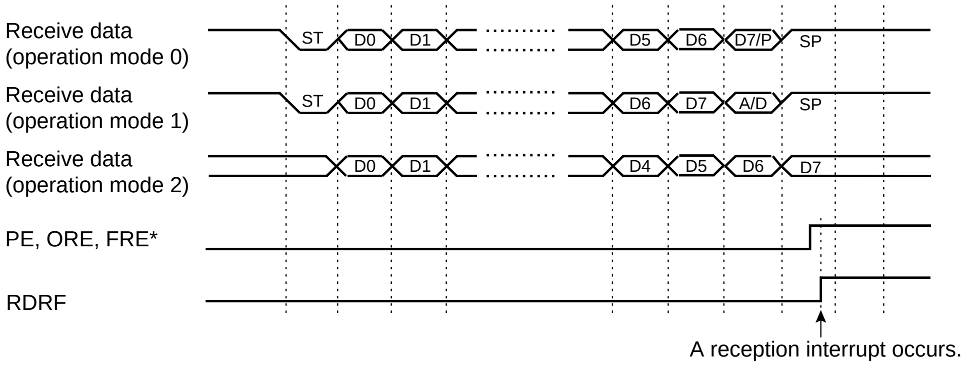
#### ❍ Operation Mode 1 (Asynchronous, Multiprocessor Mode)

The reception data full flag bit (RDRF) of the status register (SSR0/SSR1) is set to "1" when a stop bit is detected. If a reception error is detected, "1" is set to either of the reception error flag bits (ORE, FRE). Parity errors cannot be detected.

#### ❍ Operation Mode 2 (Synchronous, Normal Mode)

The reception data full flag bit (RDRF) of the status register (SSR0/SSR1) is set to "1" when the last bit (D7) of receive data is detected. If a reception error is detected, "1" is set to the reception error flag bit (ORE). Parity and framing errors cannot be detected.

**Figure 14.5-1 Reception Operation and Flag Set Timing**



| | | |
|---|---|---|
| * | : | The PE flag cannot be used in mode 1.<br>The PE and PRE flags cannot be used in mode 2. |
| ST | : | Start bit |
| SP | : | Stop bit |
| A/D | : | Mode 2 (multiprocessor mode) address/data selection bit |

### ❍ Reception Interrupt Output Timing

When the reception data full flag bit (RDRF) of the status register (SSR0/SSR1) or the one of the reception error flag bits (PE, ORE, FRE) is set to "1" while the reception interrupt is enabled (SSR0/SSR1: RIE="1"), a reception interrupt request is output.

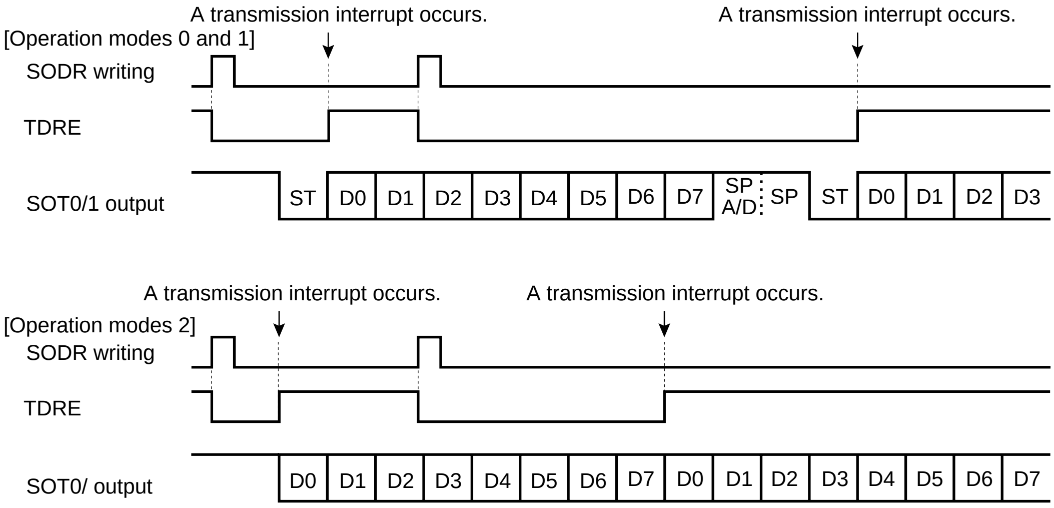
## 14.5.2 Transmission Interrupt Generation and Flag Set Timing

**A transmission interrupt is output when the output data register (SODR0/SODR1) is ready to accept writing of the next data item.**

### ■ Transmission Interrupt Output and Flag Set Timing

The transmission data empty flag bit (TDRE) of the status register (SSR0/SSR1) is set to "1" when data written to the output data register (SODR0/SODR1) is transferred to the transmission shift register and the next piece of data is ready to be written. The transmission data empty flag bit (TDRE) is cleared to "0" when send data is written to the output data register (SODR0/SODR1).

**Figure 14.5-2 Transmission Operation and Flag Set Timing**



A transmission interrupt occurs.
A transmission interrupt occurs.
[Operation modes 0 and 1]
SODR writing
TDRE
SOT0/1 output
ST
D0
D1
D2
D3
D4
D5
D6
D7
SP
A/D
SP
ST
D0
D1
D2
D3
A transmission interrupt occurs.
A transmission interrupt occurs.
[Operation modes 2]
SODR writing
TDRE
SOT0/ output
D0
D1
D2
D3
D4
D5
D6
D7
D0
D1
D2
D3
D4
D5
D6
D7

| | |
|---|---|
| ST | : Start bit |
| D0 to D7 | : Data bit |
| SP | : Stop bit |
| A/D | : Address/data selection bit |

#### ❍ Transmission Interrupt Request Output Timing

When the transmission data empty flag bit (TDRE) of the status register (SSR0/SSR1) is set to "1" while the transmission interrupt is enabled (SSR0/SSR1: TIE="1"), a transmission interrupt request is output.

**Note:**

Because the transmission data empty flag bit (TDRE) of the status register (SSR0/SSR1) is set to "1" in the initial state, a transmission interrupt request is output when the transmission interrupt is enabled (SSR0/SSR1: TIE="1"). The transmission data empty flag bit (TDRE) is a read-only bit. Accordingly, the only way to clear it is to write new data to the output data register (SODR0/SODR1). Specify carefully the timing for enabling a transmission interrupt.

# 14.6 UART Baud Rates

**One of the following can be selected as the UART transmitting/receiving block:**

- **Dedicated baud rate generator**
- **Internal clock (16-bit reload timer 0)**
- **External clock (clock input to the SCK pin)**

## ■ UART Baud Rate Selection

The baud rate selection circuit is designed as shown below. One of the following three types of baud rates can be selected:

### ❍ Baud Rates Determined Using the Dedicated Baud Rate Generator

UART has an internal dedicated baud rate generator. One of six baud rates can be selected using the mode control register (SMR0/1).

An asynchronous or clock synchronous baud rate is selected using the machine clock and CS2 to CS0 bits of the mode control register (SMR0/1).

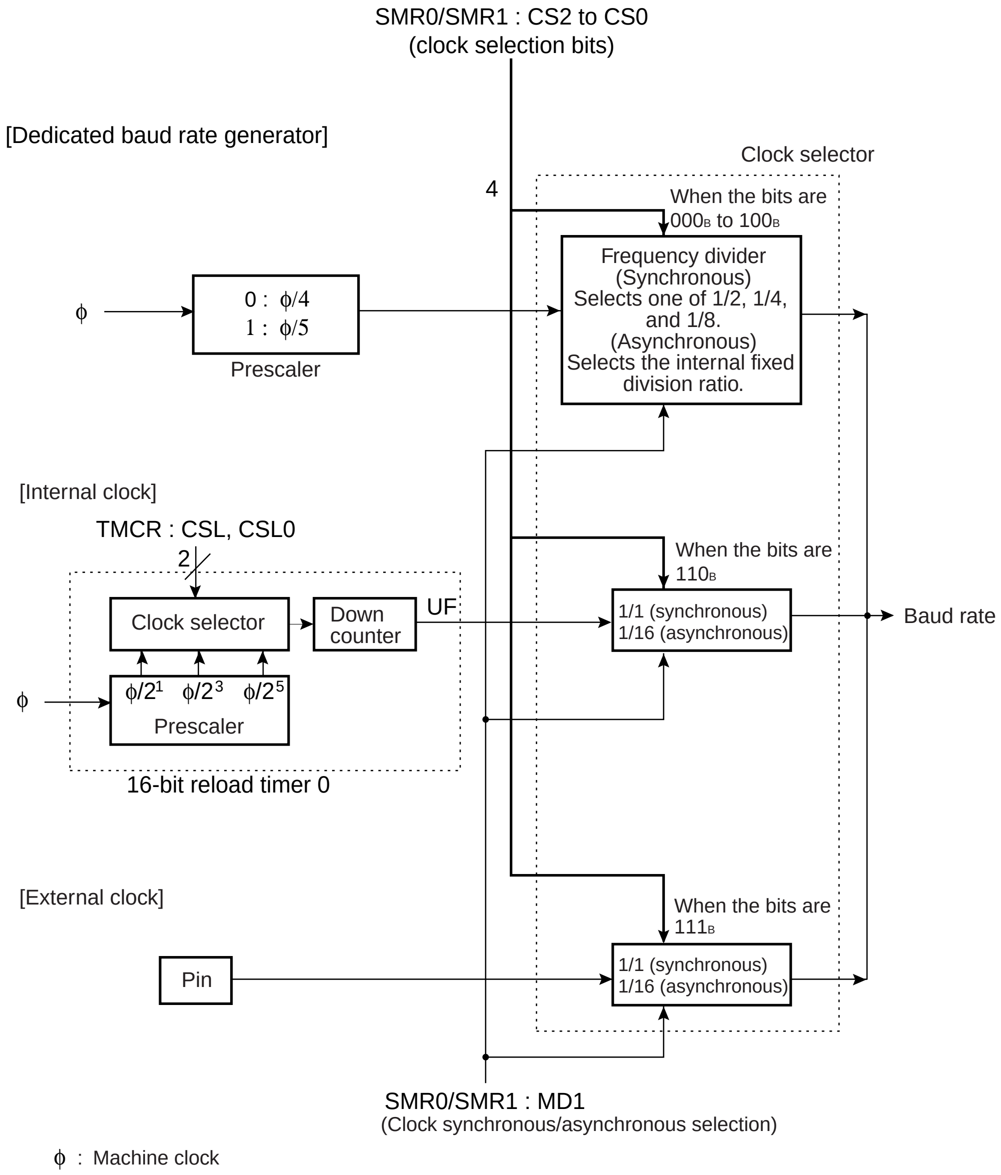
### ❍ Baud Rates Determined Using the Internal Timer

The internal clock supplied from 16-bit reload timer is used as is (synchronous) or by dividing it by 16 (asynchronous) for the baud rate. Any baud rate can be set by setting the reload value.

### ❍ Baud Rates Determined Using the External Clock

The clock input from the UART clock input pin (SCK) is used as is as the baud rate in synchronous mode or as the divide-by-16 input clock in asynchronous mode. Note, however, that the frequency of the input external clock must be 2 MHz or less.

### Figure 14.6-1 Baud Rate Selection Circuit



## 14.6.1 Baud Rates Determined Using the Dedicated Baud Rate Generator

**This section describes the baud rates that can be set when the clock from the dedicated baud rate generator is selected as the UART transfer clock.**

### ■ Baud Rates Determined Using the Dedicated Baud Rate Generator

When the clock setting bits of the mode register (SMR0/SMR1) are set to "$000_B$-$101_B$", the baud rate is set by the dedicated baud rate generator.

When the transfer clock is generated by the dedicated baud rate generator, the machine clock is divided by the machine clock prescaler and then divided by using the transfer clock division ratio set by the clock selector. The machine clock division ratios are common to the asynchronous and synchronous baud rates, but the transfer clock division ratio is set by the clock setting bits (CS2 to CS0) of the mode register (SMR0/SMR1) separately for the asynchronous and synchronous baud rates.

The actual transfer ratio can be calculated by using the following formulas:

asynchronous baud rate = $\phi$ / (prescaler division ratio) / (asynchronous transfer clock division ratio)

synchronous baud rate = $\phi$ / (prescaler division ratio) / (synchronous transfer clock division ratio)

$\phi$ : Machine clock frequency

#### ❍ Division Ratios for the Prescaler (Common to Asynchronous and Synchronous Baud Rates)

As listed in Table 14.6-1 "Setting of Each Division Ratio by the Machine Clock Prescaler", the machine clock division ratio is set by the division ratio setting bits (DIV2 to DIV0) of the communication prescaler control register (CDCR0/CDCR1).

**Table 14.6-1 Setting of Each Division Ratio by the Machine Clock Prescaler**

| MD | DIV2 | DIV1 | DIV0 | div |
|---|---|---|---|---|
| 0 | - | - | - | Stopped |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 2 |
| 1 | 0 | 1 | 0 | 3 |
| 1 | 0 | 1 | 1 | 4 |
| 1 | 1 | 0 | 0 | 5 |
| 1 | 1 | 0 | 1 | 6 |
| 1 | 1 | 1 | 0 | 7 |
| 1 | 1 | 1 | 1 | 8 |

div: Machine clock division ratio

### ❍ Synchronous Transfer Clock

The synchronous baud rate is set, as listed in Table 14.6-2 "Synchronous Baud Rate Division Ratio Setting", by the clock setting bits (CS2 to CS0) of the mode register (SMR0/SMR1).

**Table 14.6-2 Synchronous Baud Rate Division Ratio Setting**

| CS2 | CS1 | CS0 | Division ratio for CLK synchronization | Calculation formula | SC0/SC1 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 16 Mbps | (ϕ / div)/1 | (ϕ / div)/1 |
| 0 | 0 | 1 | 8 Mbps | (ϕ / div)/2 | (ϕ / div)/2 |
| 0 | 1 | 0 | 4 Mbps | (ϕ / div)/4 | (ϕ / div)/4 |
| 0 | 1 | 1 | 2 Mbps | (ϕ / div)/8 | (ϕ / div)/8 |
| 1 | 0 | 0 | 1 Mbps | (ϕ / div)/16 | (ϕ / div)/16 |
| 1 | 0 | 1 | 500 kbps | (ϕ / div)/32 | (ϕ / div)/32 |

This calculation assumes a machine cycle frequency ϕ = 16 MHz and div = 4.

### ❍ Asynchronous Transfer Clock Division Ratios

Asynchronous baud rates is selected using the CS2 to CS0 bits of the mode control register (SMR0/1) as listed in Table 14.6-3 "Asynchronous Baud Rate Division Ratio Setting".

**Table 14.6-3 Asynchronous Baud Rate Division Ratio Setting**

| CS2 | CS1 | CS0 | Asynchronus (start-stop synchronization) | Calculation formula | SC0/SC1 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 76,923 bps | (ϕ / div)/(8 x 13 x 2) | (ϕ / div)/(13 x 1) |
| 0 | 0 | 1 | 38,461 bps | (ϕ / div)/(8 x 13 x 4) | (ϕ / div)/(13 x 2) |
| 0 | 1 | 0 | 19,230 bps | (ϕ / div)/(8 x 13 x 8) | (ϕ / div)/(13 x 4) |
| 0 | 1 | 1 | 9,615 bps | (ϕ / div)/(8 x 13 x 16) | (ϕ / div)/(13 x 8) |
| 1 | 0 | 0 | 500 kbps | (ϕ / div)/(8 x 2 x 2) | (ϕ / div)/2 |
| 1 | 0 | 1 | 250 kbps | (ϕ / div)/(8 x 2 x 4) | (ϕ / div)/4 |

However, the baud rate is calculated based on the values of ϕ (machine clock) = 16MHz and div (machine clock division ratio) = 1.

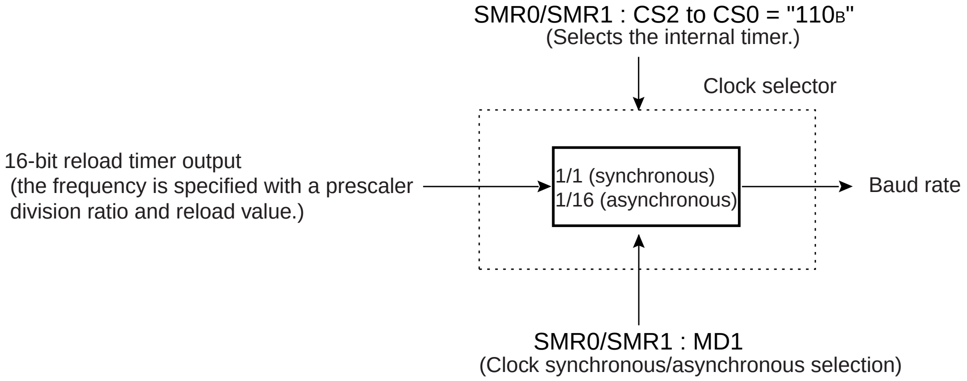
## 14.6.2 Baud Rates Determined Using the Internal Timer

**This section describes the settings used when the internal clock supplied from 16-bit reload timer is selected as the UART transfer clock. It also shows the baud rate calculation formulas.**

### ■ Baud Rates Determined Using the Internal Timer (16-bit Reload Timer 0)

If the clock setting bits (CS2 to CS0) of the mode register (SMR0/SMR1) are set to "$110_B$", the baud rate is set by the internal clock. The baud rate can be set by specifying the prescaler division ratio and reload value of the 16-bit reload timer.

**Figure 14.6-2 Baud Rate Selection Circuit for the Internal Timer (16-bit Reload Timer 0)**


SMR0/SMR1 : CS2 to CS0 = "110B"
(Selects the internal timer.)
Clock selector
16-bit reload timer output
(the frequency is specified with a prescaler
division ratio and reload value.)
1/1 (synchronous)
1/16 (asynchronous)
Baud rate
SMR0/SMR1 : MD1
(Clock synchronous/asynchronous selection)

#### ❍ Baud Rate Calculation Formulas

Asynchronous baud rate = ($\phi$ / N)/(16x2x (n+1)) bps

Synchronous baud rate = ($\phi$ / N)/(2x (n+1)) bps

$\phi$: Machine clock

N: Division ratio for the prescaler of 16-bit reload timer 0 ($2^1$, $2^3$, or $2^5$)

n: Reload value for 16-bit reload timer 0 (0 to 65535)

### ❍ Examples of Setting Reload Values (Machine Clock: 7.3728 MHz)

Table 14.6-4 Baud Rates and Reload Values

| Baud rate | Reload value (n) | | | |
|---|---|---|---|---|
| | Clock asynchronous (start-stop synchronization) | | Clock synchronous | |
| | N=$2^1$(machine cycle divided by 2) | N=$2^3$ (machine cycle divided by 8) | N=$2^1$(machine cycle divided by 2) | N=$2^3$ (machine cycle divided by 8) |
| 38400 | 2 | - | 47 | 11 |
| 19200 | 5 | - | 95 | 23 |
| 9600 | 11 | 2 | 191 | 47 |
| 4800 | 23 | 5 | 383 | 95 |
| 2400 | 47 | 11 | 767 | 191 |
| 1200 | 95 | 23 | 1535 | 383 |
| 600 | 191 | 47 | 3071 | 767 |
| 300 | 383 | 95 | 6143 | 1535 |

N: Division ratio for the prescaler of 16-bit reload timer 0

-: Setting not allowed

## 14.6.3 Baud Rates Determined Using the External Clock

**This section describes the settings used when the external clock is selected as the UART transfer clock. It also shows the baud rate calculation formulas.**

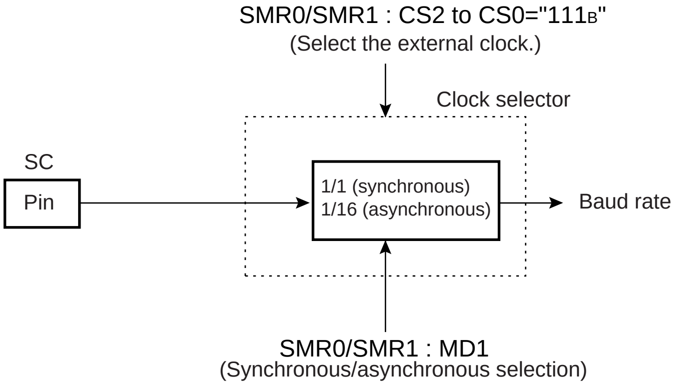
### ■ Baud Rates Determined Using the External Clock

The following three settings are required to select the baud rate determined by using the external clock:

- Write $111_B$ to the CS2 to CS0 bits of the mode control register (SMR0/1) to select the baud rate determined by using the external clock input.
- Set the SC pin to be used as an input port.
- Write 0 to the SCKE bit of the mode control register (SMR0/1) to set the pin as an external clock input pin.

As shown in Figure 14.6-3 "Baud Rate Selection Circuit for the External Clock", a baud rate is selected using the external clock input from the SC pin. To change the baud rate, the external input clock cycle must be changed because the internal division ratio is fixed.

**Figure 14.6-3 Baud Rate Selection Circuit for the External Clock**


SMR0/SMR1 : CS2 to CS0="111B"
(Select the external clock.)
Clock selector
SC
Pin
1/1 (synchronous)
1/16 (asynchronous)
Baud rate
SMR0/SMR1 : MD1
(Synchronous/asynchronous selection)

### ❍ Baud Rate Calculation Formulas

asynchronous baud rate = f/16 bps

synchronous baud rate = f bps

f: External clock (up to 2 MHz)

# 14.7 Operation of UART

**UART operates in operation modes 0 and 2 for normal bidirectional serial communication and in operation mode 1 for master-slave communication.**

## ■ Operation of UART

### ❍ Operation modes

There are three UART operation modes: modes 0 to 2. As listed in Table 14.7-1 "UART Operation Mode", an operation mode can be selected according to the inter-CPU connection method and data transfer mode.

**Table 14.7-1 UART Operation Mode**

| Operation mode | | Data length | | Synchronization mode | Stop bit length |
|---|---|---|---|---|---|
| | | When parity is disabled | When parity is enabled | | |
| 0 | Normal mode | 7 or 8 bits | | Asynchronous | 1 or 2 bits*2 |
| 1 | Multiprocessor mode | 8+1*1 | - | Asynchronous | |
| 2 | Normal mode | 8 | - | Synchronous | None |

-: Setting not possible

*1: "+1" indicates the address/data selection bit (A/D) for communication control.

*2: During reception, only one stop bit can be detected.

**Note:**

Operation mode 1 of UART is used only from the master system during master-slave connection.

### ❍ Inter-CPU Connection Method

One-to-one connection (normal mode) and master-slave connection (multiprocessor mode) can be selected. For either connection method, the data length, whether to enable parity, and the synchronization method must be common to all CPU. Select an operation mode as follows:

- In the one-to-one connection method, operation mode 0 or 2 must be used in the two CPUs. Select operation mode 0 for asynchronous transfer mode and operation mode 2 for synchronous transfer mode.
- Select operation mode 1 for the master-slave connection method and use it from the master system. Select "When parity is disabled" for this connection method.

### ❍ Synchronization Method

Asynchronous mode (start-stop synchronization) or clock synchronous mode can be selected in any operation mode.

### ❍ Signal Method

UART can treat data only in non-return to zero (NRZ) format.

### ❍ Operation Enable

UART controls both transmission and reception using the control register (SCR0/SCR1) with its transmission enable bit (TXE) and reception enable bit (RXE). If any of the operations is disabled during operation, the following will occur:

- If reception operation is disabled during reception (data is input to the reception shift register), finish frame reception and store the received data in the input data register (SIDR0/1). Then stop the reception operation.
- If the transmission operation is disabled during transmission (data is output from the transmission shift register), wait until there is no data in the output data register (SODR0/1) before stopping the transmission operation.
- When mode 1 is selected for URAT, the received data bit 9 is ignored.

## 14.7.1 Operation in Asynchronous Mode (Operation Modes 0 and 1)

**When UART is used in operation mode 0 (normal mode) or operation mode 1 (multiprocessor mode), asynchronous transfer mode is selected.**

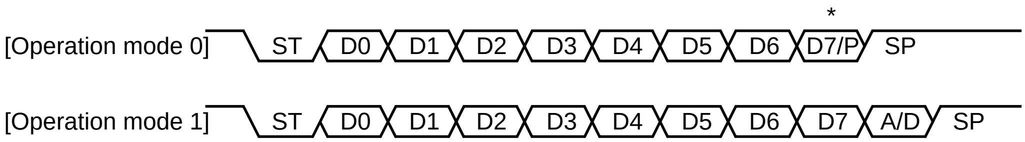
### ■ Operation in Asynchronous Mode

#### ❍ Transfer Data Format

Transfer data always starts with the start bit ("L" level) and ends with the stop bit ("H" level). The data of the specified data bit length is transferred in "LSB first."

- In operation mode 0, data without a parity bit is always seven bits long and data with a parity bit is always eight bits long.
- In operation mode 1, the data is fixed to eight bits with an address/data setting bit (A/D) bit instead of a parity bit.

**Figure 14.7-1 Transfer Data Format (Operation Modes 0 and 1)**



| | | |
|---|---|---|
| * | : | D7(bit7)........When no parity bit is added<br>P(parity)........When a parity bit is added |
| ST | : | Start bit |
| SP | : | Stop bit |
| A/D | : | Address/data setting bit of operation mode 1 (multiprocessor mode) |

#### ❍ Transmission Operation

When the transmission data empty flag bit (TDRE) of the status register (SSR0/SSR1) is set to "1", send data is written to the output data register (SODR0/SODR1). If the transmission is enabled (SCR0/SCR1: TXE="1"), data is sent.

When the send data is transferred to the transmission shift register and transmission is started, the transmission data empty flag bit (TDRE) of the status register (SSR0/SSR1) is set to "1" again so that the next piece of send data can be set. If the transmission interrupt request output is enabled (SSR0/SSR1: TIE="1"), a transmission interrupt request is output so that send data is set to the output data register (SODR0/SODR1). The transmission data empty flag bit (TDRE) is cleared to "0" after writing the send data to the output data register (SODR0/SODR1).

### ❍ Reception Operation

If the reception is enabled (SCR0/SCR1: RXE="1"), reception operation is always performed. When a start bit is detected, one frame of data is received in accordance with the data format specified in the control register (SCR0/SCR1). When reception of one frame is completed, if a reception error occurs, one of the reception error flag bits (PE, ORE, FRE) of the status register (SSR0/SSR1) is set to "1" and then the reception data full flag bit (RDRF) is set to "1". If the reception interrupt request output is enabled (SSR0/SSR1: RIE="1"), a reception interrupt request is output. Check each of the reception error flag bits (PE, ORE, FRE) of the status register (SSR0/SSR1). If no error occurred in reception, read the input data register (SIDR0/SIDR1). If an error has occurred, take the appropriate measures to deal with the error. The reception data full flag bit (RDRF) is cleared to "0" after reading receive data from the input data register (SIDR0/SIDR1).

### ❍ Stop Bit

One or two stop bits can be set for transmission. The receiving side always evaluates the first one bit.

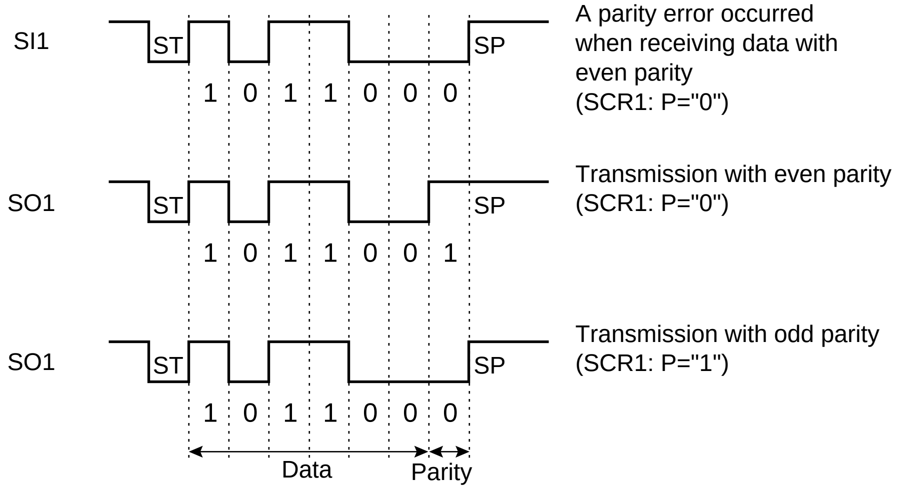
### ❍ Error Detection

- In operation mode 0, parity, overrun, and framing errors can be detected.
- In operation mode 1, overrun and framing errors can be detected, but parity errors cannot be detected.

### ❍ Parity

Parity can be used only in operation mode 0. The parity presence/absence can be set in the parity enable bit (PEN) of the control register (SCR0/SCR1) and even parity/odd parity can be set in the parity setting bit (P). Parity cannot be used in operation mode 1.

**Figure 14.7-2 Transmission Data when Parity is Enabled**


SI1
ST
SP
1 0 1 1 0 0 0
A parity error occurred when receiving data with even parity (SCR1: P="0")
SO1
ST
SP
1 0 1 1 0 0 1
Transmission with even parity (SCR1: P="0")
SO1
ST
SP
1 0 1 1 0 0 0
Transmission with odd parity (SCR1: P="1")
Data
Parity

ST: Start bit
SP: Stop bit
Note: Parity cannot be used in operation modes 1 and 2.

# 14.7.2 Operation in Synchronous Mode (Operation Mode 2)

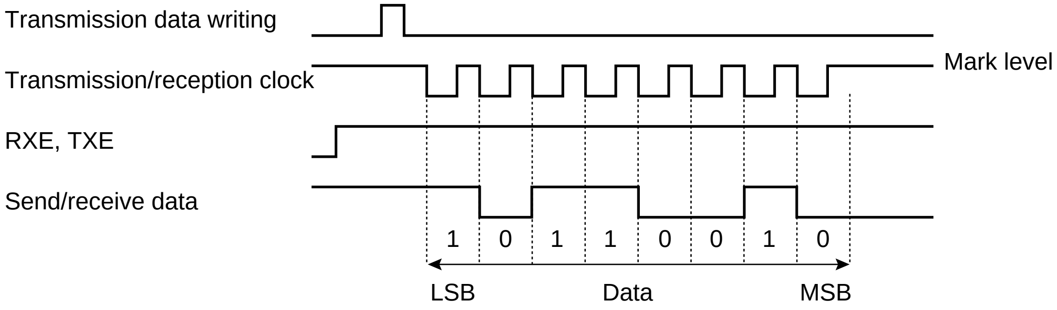
**When UART is used in operation mode 2 (normal mode), the synchronous transfer mode is selected.**

## ■ Operation in Synchronous Mode (Operation Mode 2)

### ❍ Transfer Data Format

In synchronous mode, 8-bit data is transferred in LSB first mode.

**Figure 14.7-3 Transfer Data Format (Operation Mode 2)**



### ❍ Clock Supply

In clock synchronous (I/O extended serial) mode, as many clocks as the number of transmission/reception bits need to be supplied.

- If the internal clock is selected, a data reception synchronous clock is generated when data is received.
- If an external clock is selected, a clock for exactly one byte needs to be supplied from an external source after making sure that the output data register (SODR0/SODR1) on the sending side UART contains data (SSR0/SSR1: TDRE="0"). The mark level "H" must be retained before transmission starts and after it is completed.

### ❍ Error Detection

Only overrun errors can be detected, and parity and framing errors cannot be detected.

### ❍ Initialization

The following shows the values to be set to use synchronous mode:

**[Mode register (SMR0/SMR1)]**

MD1, MD0: Set "$10_B$".

CS2, CS1, CS0: Specify the clock input of the clock selector.

SCKE: Set "1" for the dedicated baud rate generator or the internal clock. Set "0" for the clock output or an external clock.

SOE: Set "1" for transmission. Set "0" for reception.

**[Control register (SCR0/SCR1)]**

PEN: Set "0".

P, SBL, A/D: These bits make no sense.

CL: Set "1" (8-bit data).

REC: Set "0" (Clear all error flags for initialization).

RXE, TXE: Set "1" to either of the bits.

**[Status register (SSR0/SSR1)]**

RIE: Set "1" to use interrupts and "0" to not use interrupts.

TIE: Set "1" to use interrupts and "0" to not use interrupts.

### ❍ Communication Start

Write data to the output data register (SODR0/SODR1) start communication. Note that the receiving side must also write send data to the output data register (SODR0/SODR1) to start communication.

### ❍ Communication End

The reception data full flag bit (RDRF) of the status register (SSR0/SSR1) is set to "1" when transmission or reception of one frame of data is completed. When receiving data, check the overrun error flag bit (ORE) to see if the communication has been normally executed.

## 14.7.3 Bidirectional Communication Function (Normal Mode)

**In operation mode 0 or 2, serial bidirectional communication (one-to-one connection) is available. Select operation mode 0 for asynchronous communication and operation mode 2 for synchronous communication.**

### ■ Bidirectional Communication Function

The settings shown in Figure 14.7-4 "Settings for UART1 Operation Mode 0" are required to operate UART1 in normal mode (operation mode 0 or 2).

**Figure 14.7-4 Settings for UART1 Operation Mode 0**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SCR1, SMR1 | PEN | P | SBL | CL | AD | REC | RXE | TXE | MD1 | MD0 | CS2 | CS1 | CS0 | - | SCKE | SOE |
| Mode 0 → | ◎ | ◎ | ◎ | ◎ | × | 0 | ◎ | ◎ | 0 | 0 | ◎ | ◎ | ◎ | - | ◎ | ◎ |
| Mode 2 → | 0 | × | × | 1 | × | 0 | ◎ | ◎ | 1 | 0 | ◎ | ◎ | ◎ | - | ◎ | ◎ |

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7–0 |
|---|---|---|---|---|---|---|---|---|---|
| SSR1,SIDR0/SIDR1 | PE | ORE | FRE | RDRF | TDRE | BDS | RIE | TIE | Set conversion data (during writing).<br>Retain receive data (during reading). |
| Mode 0 → | ◎ | ◎ | ◎ | ◎ | ◎ | ◎ | ◎ | ◎ | |
| Mode 2 → | × | ◎ | × | ◎ | ◎ | ◎ | ◎ | ◎ | |

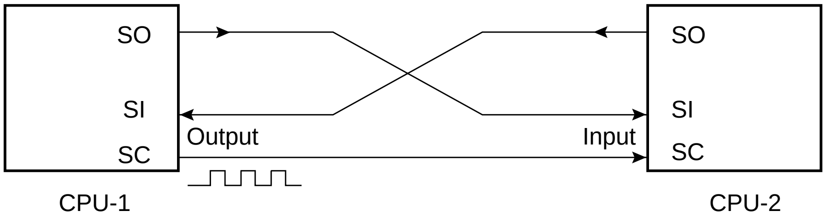
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| DDR | | | | | | | | |
| | | | | | | △ | | △ |

◎ : bit used
× : Bit not used
1 : Set 1.
0 : Set 0.
△ : Set 0 to use an input pin.

### ❍ Inter-CPU Connection

As shown in Figure 14.7-5 "Connection Example of UART1 Bidirectional Communication", interconnect two CPU.

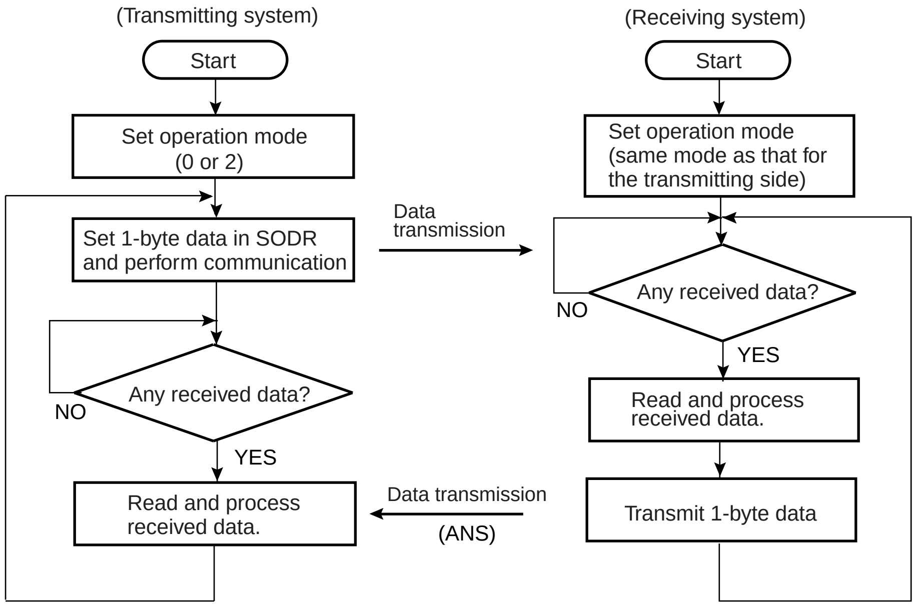
**Figure 14.7-5 Connection Example of UART1 Bidirectional Communication**



### ❍ Communication Procedure

Communication starts from the transmitting system when transmission data has been prepared. An ANS is returned periodically (byte by byte in this example) when the receiving system receives transmission data.

### Figure 14.7-6 Example of Bidirectional Communication Flowchart


(Transmitting system)
(Receiving system)
Start
Set operation mode (0 or 2)
Set 1-byte data in SODR and perform communication
Any received data?
NO
YES
Read and process received data.
Data transmission
Start
Set operation mode (same mode as that for the transmitting side)
Any received data?
NO
YES
Read and process received data.
Transmit 1-byte data
Data transmission
(ANS)

## 14.7.4 Master-slave Communication Function (Multiprocessor Mode)

**With UART, communication with multiple CPU connected in master-slave mode is available. However, UART can be used only from the master system.**

### ■ Master-slave Communication Function

The settings shown in Figure 14.7-7 "Settings for UART1 Operation Mode 1" are required to operate UART1 in multiprocessor mode (operation mode 1).

**Figure 14.7-7 Settings for UART1 Operation Mode 1**

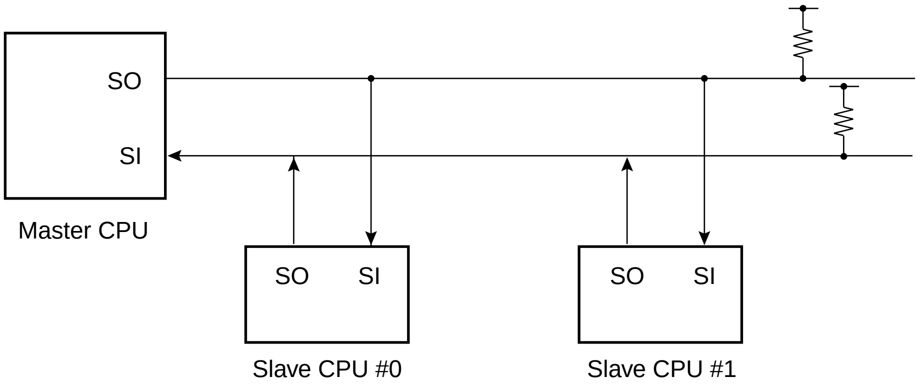
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SCR1, SMR1 | PEN | P | SBL | CL | AD | REC | RXE | TXE | MD1 | MD0 | CS2 | CS1 | CS0 | - | SCKE | SOE |
| | 0 | × | ◎ | 1 | ◎ | 0 | ◎ | ◎ | 0 | 1 | ◎ | ◎ | ◎ | - | 0 | ◎ |
| SSR1,<br>SIDR1, SODR1 | PE | ORE | FRE | RDRF | TDRE | BDS | RIE | TIE | Set transmission data (during writing).<br>Retain receive data (during reading). | | | | | | | |
| | × | ◎ | ◎ | ◎ | ◎ | ◎ | ◎ | ◎ | | | | | | | | |
| DDR | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | △ | | △ |

◎ : Bit used
× : Bit not used
1 : Set 1.
0 : Set 0.
△ : Set 0 to use an input pin.

#### ❍ Inter-CPU Connection

As shown in Figure 14.7-8 "Connection Example of UART1 Master-Slave Communication", a communication system consists of one master CPU and multiple slave CPU connected to two communication lines. UART1 can be used only from the master CPU.

**Figure 14.7-8 Connection Example of UART1 Master-Slave Communication**



### ❍ Function Selection

Select the operation mode and data transfer mode for master-slave communication as shown in Table 14.7-2 "Selection of the Master-Slave Communication Function".

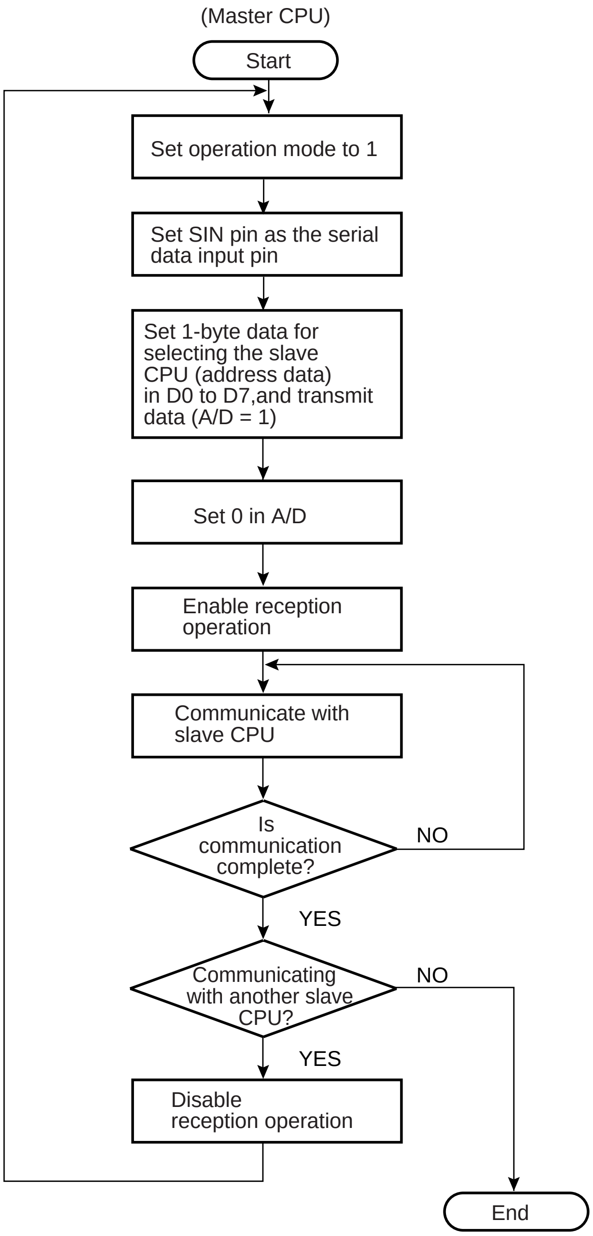
**Table 14.7-2 Selection of the Master-Slave Communication Function**

| | Operation mode | | Data | Parity | Synchronization method | Stop bit |
|---|---|---|---|---|---|---|
| | Master CPU | Slave CPU | | | | |
| Address transmission and reception | Operation mode 1 | - | A/D="1"<br>+<br>8-bit address | None | Asynchronous | 1 or 2 bits |
| Data transmission and reception | | | A/D="0"<br>+<br>8-bit address | | | |

### ❍ Communication Procedure

When the master CPU transmits address data, communication starts. The A/D bit in the address data is set to 1, and the communication destination slave CPU is selected. Each slave CPU checks the address data using a program. When the address data indicates the address assigned to a slave CPU, the slave CPU communicates with the master CPU (ordinary data).

### Figure 14.7-9 Master-Slave Communication Flowchart


(Master CPU)
Start
Set operation mode to 1
Set SIN pin as the serial data input pin
Set 1-byte data for selecting the slave CPU (address data) in D0 to D7,and transmit data (A/D = 1)
Set 0 in A/D
Enable reception operation
Communicate with slave CPU
Is communication complete?
NO
YES
Communicating with another slave CPU?
NO
YES
Disable reception operation
End

# 14.8 Notes on Using UART

## Notes on using UART are given below.

### ■ Notes on Using UART

#### ❍ Enabling Operations

UART has the transmission enable bit (TXE) and reception enable bit (RXE) in the control register (SCR0/SCR1). Both transmission and reception operations need to be enabled before starting transfer because the transmission/reception enable bits (TXE/RXE) are set to "0" in the initial state. The transfer can also be canceled by disabling the transmission/reception as required.

#### ❍ Communication Mode Setting

Set the communication mode while the system is not operating. If the mode is set during transmission or reception, the transmission or reception data is not guaranteed.

#### ❍ Synchronous Mode

UART clock synchronous mode (operation mode 2) uses clock control (I/O extended serial) mode, in which start and stop bits are not added to the data.

#### ❍ Transmission Interrupt Enabling Timing

The default (initial value) of the transmission data empty flag bit (SSR0/1: TDRE) is 1 (no transmission data and transmission data write enable state). A transmission interrupt request is generated as soon as the transmission interrupt requests are enabled (SSR0/1: TIE=1). Be sure to set the TIE flag to 1 after setting the transmission data.

#### ❍ Reception in Operation Mode 1 (Multiprocessor Mode)

In operation mode 1 (multiprocessor mode) of UART, the 9-bit receive operation cannot be performed.

# CHAPTER 15 DTP/EXTERNAL INTERRUPT CIRCUIT

**This chapter describes the functions and operations of the DTP/external interrupt circuit of the MB90M405 series.**

# 15.1 Overview of the DTP/External Interrupt Circuit

**The data transfer peripheral (DTP)/external interrupt circuit detects interrupt request input from an external interrupt input pin in order to generate an interrupt request.**

## ■ DTP/External Interrupt Functions

The DTP/external interrupt circuit function outputs an interrupt request when it detects an edge or level signal input to the external interrupt input pins.

When an interrupt request is accepted by the CPU and the extended intelligent I/O service (EI$^2$OS) is enabled, the automatic data transfer (DTP function) is performed by EI$^2$OS before branching to the interrupt processing routine. If EI$^2$OS is disabled, the automatic data transfer (DTP function) by EI$^2$OS is not activated; instead, direct branching to the interrupt processing routine takes place.

**Table 15.1-1 Overview of the DTP/External Interrupt Circuit**

| | External interrupt function | DTP function |
|---|---|---|
| Input pins | Four (P80/INT0, P81/INT1, PB6/INT2, and PB7/INT3) | |
| Interrupt cause | By using the request level setting register (ELVR), the detection level or edge type can be set for each pin. | |
| | Input of the "L" level/"H" level | Input of the rising edge/falling edge |
| Interrupt number | #11 ($0B_H$), #13 ($0D_H$), #16 ($10_H$) | |
| Interrupt control | The output of interrupt requests is enabled and disabled using the DTP/interrupt enable register (ENIR). | |
| Interrupt flag | Interrupt causes are stored in the DTP/interrupt cause register (EIRR). | |
| Processing selection | EI$^2$OS is disabled (ICR: ISE = 0). | EI$^2$OS is enabled (ICR: ISE = 1). |
| Processing | The circuit branches to an external interrupt processing routine. | Branching to the interrupt processing routine after automatic data transfer by EI$^2$OS |

ICR: Interrupt control register

## ■ Interrupt of the DTP/external interrupt circuit and EI$^2$OS

Table 15.1-2 Interrupt of the DTP/External Interrupt Circuit and EI$^2$OS

| Channel | Interrupt number | Interrupt control register | | Vector table address | | | EI$^2$OS |
|---|---|---|---|---|---|---|---|
| | | Register name | Address | Lower | Upper | Bank | |
| INT0 | #11 (0B$_H$) | ICR00 | 0000B0$_H$ | FFFFD0$_H$ | FFFFD1$_H$ | FFFFD2$_H$ | O |
| INT1 | #13 (0D$_H$) | ICR01 | 0000B1$_H$ | FFFFC8$_H$ | FFFFC9$_H$ | FFFFCA$_H$ | |
| INT2 | #16 (10$_H$) | ICR02 | 0000B2$_H$ | FFFFBC$_H$ | FFFFBD$_H$ | FFFFBE$_H$ | |
| INT3 | | | | | | | |

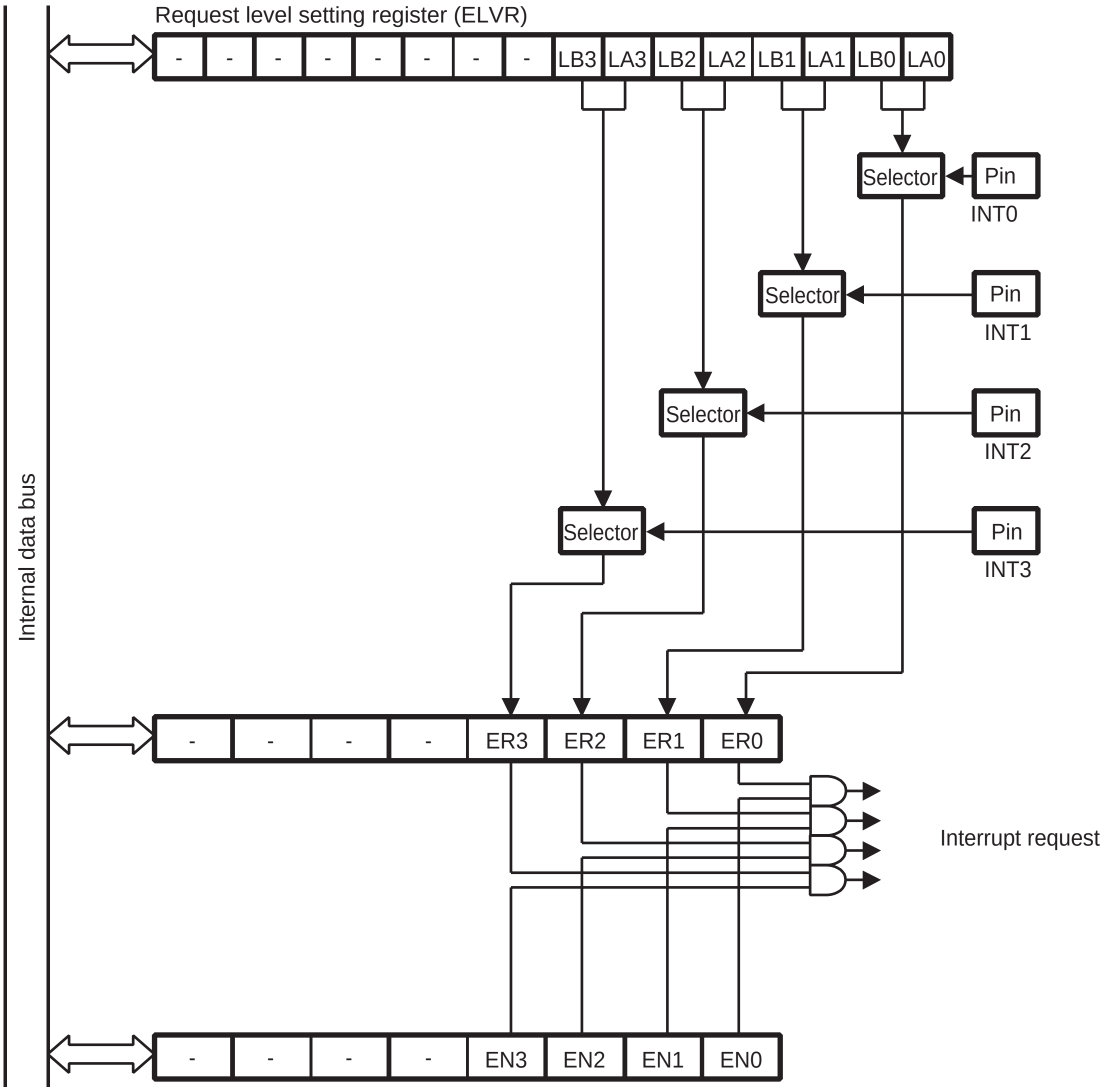
# 15.2 Configuration of the DTP/External Interrupt Circuit

**The DTP/external interrupt circuit consists of the following blocks:**

- **DTP/interrupt input detection circuit**
- **Request level setting register (ELVR)**
- **DTP/interrupt cause register (EIRR)**
- **DTP/interrupt enable register (ENIR)**

## ■ Block Diagram of the DTP/External Interrupt Circuit

**Figure 15.2-1 Block Diagram of the DTP/External Interrupt Circuit**



❍ **DTP/external interrupt input detection circuit**

Upon detecting a match of the signal input to an external interrupt input pin and the level or edge specified in the request level setting register (ELVR), the DTP/external interrupt cause flag bit (EIRR: ER3 to ER0) corresponding to the external interrupt input pin is set to "1".

❍ **Request level setting register (ELVR)**

This register sets the detection condition (level or edge) of interrupt requests for each external interrupt input pin.

❍ **DTP/interrupt cause register (EIRR)**

This register retains and clears interrupt causes.

❍ **DTP/interrupt enable register (ENIR)**

This register enables/disables interrupt requests for each external interrupt input pin.

## 15.3 DTP/External Interrupt Circuit Pins

**This section describes the DTP/external interrupt circuit pins and provides a pin block diagram.**

### ■ DTP/External Interrupt Circuit Pins

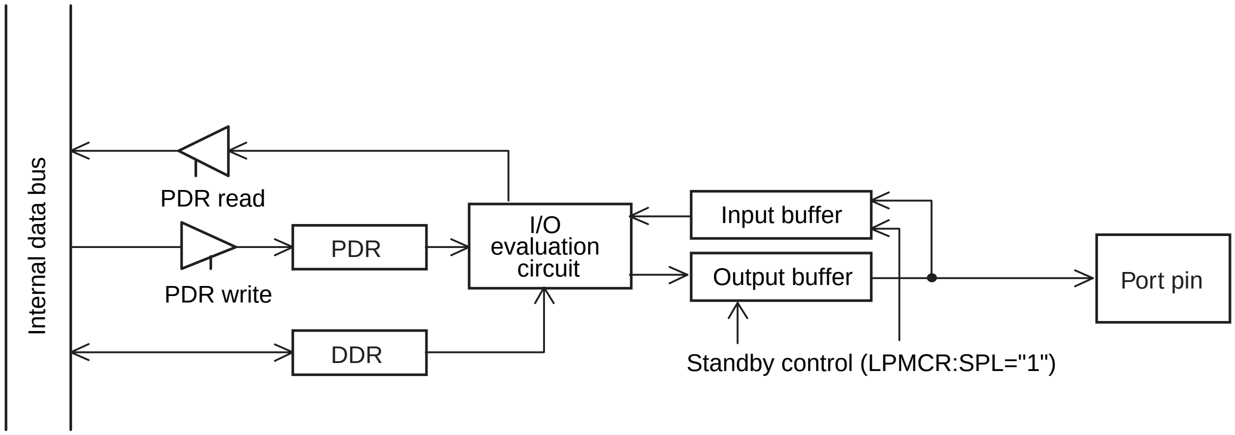
The DTP/external interrupt circuit pins are also used as I/O ports.

**Table 15.3-1 DTP/External Interrupt Circuit Pins**

| Pin name | Function | I/O format | Pull-up resistor | Standby control | Setting required to use pins |
|---|---|---|---|---|---|
| INT0 | Port 8 input-output/external interrupt input | CMOS output/ CMOS hysteresis input | Not provided | Not provided | Set the pin as an input port (DDR8: bit0 = 0) |
| INT1 | | | | | Set the pin as an input port (DDR8: bit9 = 0) |
| INT2 | Port B input-output/external interrupt input | | | | Set the pin as an input port (DDRB: bit16 = 0) |
| INT3 | | | | | Set the pin as an input port (DDRB: bit17 = 0) |

### ■ Block Diagram of the DTP/External Interrupt Circuit Pins

**Figure 15.3-1 Block Diagram of the DTP/External Interrupt Circuit Pins**



# 15.4 DTP/External Interrupt Circuit Registers

**This section describes lDTP/external interrupt circuit registers.**

## ■ DTP/External Interrupt Circuit register

**Figure 15.4-1 DTP/External Interrupt Circuit Registers**

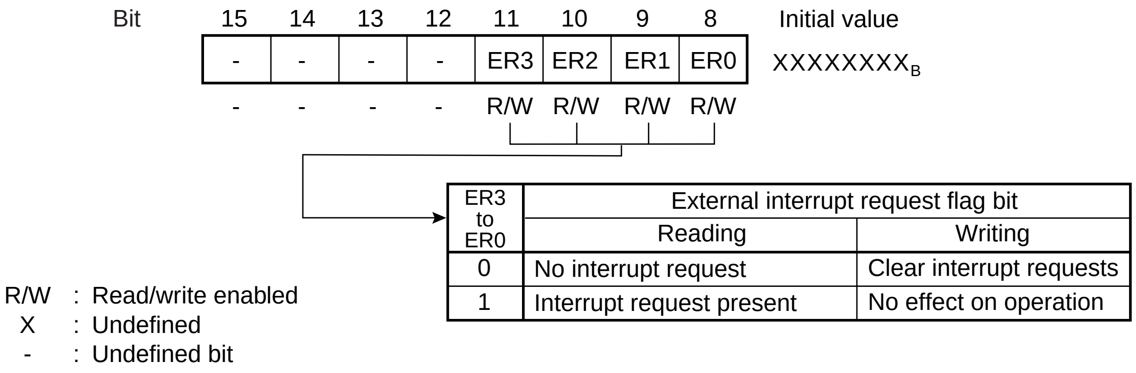
| bit15 ······ bit8 | bit7 ······ bit0 |
|---|---|
| DTP/interrupt cause register (EIRR) | DTP/interrupt enable register (ENIR) |
| Request level setting register (ELVR) | |

# 15.4.1 DTP/Interrupt Cause Register (EIRR)

**The DTP/interrupt cause register (EIRR) stores and clears interrupt causes.**

## ■ DTP/Interrupt Cause Register (EIRR)

**Figure 15.4-2  DTP/Interrupt Cause Register (EIRR)**



| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | - | - | - | - | ER3 | ER2 | ER1 | ER0 | XXXXXXXX$_B$ |
| | - | - | - | - | R/W | R/W | R/W | R/W | |

| ER3<br>to<br>ER0 | External interrupt request flag bit | |
|---|---|---|
| | Reading | Writing |
| 0 | No interrupt request | Clear interrupt requests |
| 1 | Interrupt request present | No effect on operation |

R/W : Read/write enabled
X : Undefined
- : Undefined bit

**Table 15.4-1 Function Description of Each Bit of the DTP/Interrupt Cause Register (EIRR)**

| Bit name | | | Function |
|---|---|---|---|
| bit15 to bit12 | ER7: to ER4: | External interrupt request flag bit | • Setting of any of these bits is invalid because the external input pins are INT0 through INT3. |
| bit11 | ER3: | | • This bit is a flag that requests an interrupt.<br>• This bit is set to "1" when the level or edge signal set in the external interrupt request detection condition setting bit (LB3, LA3) of the request level setting register (ELVR) is detected in the external interrupt input pin (INT3).<br>• When this bit is set to "1" while the external interrupt request enable bit (EN3) of the DTP/external interrupt enable register (ENIR) is set to "1", an interrupt request is output.<br>• When this bit is "0", the interrupt request is cleared.<br>• When this bit is "1", operation is not affected. |
| bit10 | ER2: | | • This bit is a flag that requests an interrupt.<br>• This bit is set to "1" when the level or edge signal set in the external interrupt request detection condition setting bit (LB2, LA2) of the request level setting register (ELVR) is detected in the external interrupt input pin (INT2).<br>• When this bit is set to "1" while the external interrupt request enable bit (EN2) of the DTP/external interrupt enable register (ENIR) is set to "1", an interrupt request is output.<br>• When this bit is "0", the interrupt request is cleared.<br>• When this bit is "1", operation is not affected. |
| bit9 | ER1: | | • This bit is a flag that requests an interrupt.<br>• This bit is set to "1" when the level or edge signal set in the external interrupt request detection condition setting bit (LB1, LA1) of the request level setting register (ELVR) is detected in the external interrupt input pin (INT1).<br>• When this bit is set to "1" while the external interrupt request enable bit (EN1) of the DTP/external interrupt enable register (ENIR) is set to "1", an interrupt request is output.<br>• When this bit is "0", the interrupt request is cleared.<br>• When this bit is "1", operation is not affected. |
| bit8 | ER0: | | • This bit is a flag that requests an interrupt.<br>• This bit is set to "1" when the level or edge signal set in the external interrupt request detection condition setting bit (LB0, LA0) of the request level setting register (ELVR) is detected in the external interrupt input pin (INT0).<br>• When this bit is set to "1" while the external interrupt request enable bit (EN0) of the DTP/external interrupt enable register (ENIR) is set to "1", an interrupt request is output.<br>• When this bit is "0", the interrupt request is cleared.<br>• When this bit is "1", operation is not affected. |

**Reference:**

When the extended intelligent I/O service (EI$^2$OS) is activated as a DTP function, the corresponding external interrupt request flag bit (ER3 to ER0) is cleared to "0" when the transfer of one piece of data is completed.

**Note:**

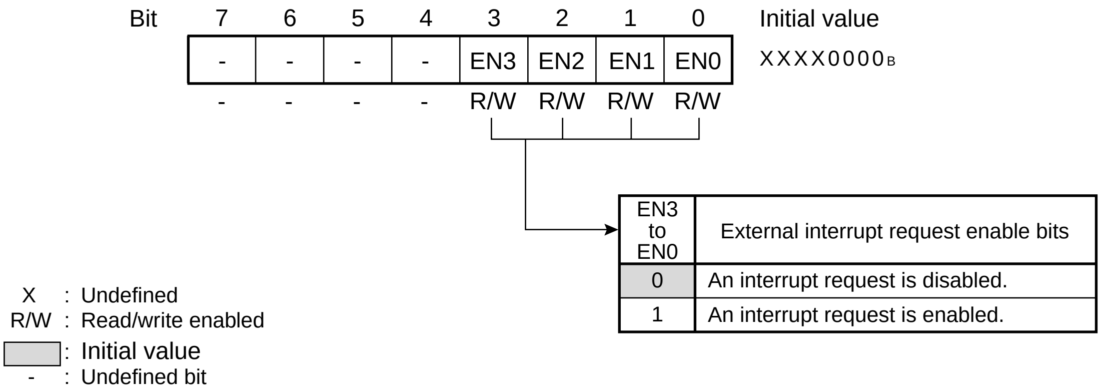
Reading by read-modify-write type instructions always returns "1".
If multiple external interrupt request outputs are enabled (ENIR: EN3 to EN0=1), only the bits for which the CPU accepts an interrupt (bits for which "1" was set in ER3 to ER0) are cleared. No other bits must be cleared unconditionally.

## 15.4.2 DTP/Interrupt Enable Register (ENIR)

**The DTP/interrupt enable register (ENIR) enables/disables an external interrupt request for each external interrupt pin (INT7 to INT0).**

### ■ DTP/Interrupt Enable Register (ENIR)

**Figure 15.4-3 DTP/Interrupt Enable Register (ENIR)**



| EN3<br>to<br>EN0 | External interrupt request enable bits |
|---|---|
| ▨ 0 | An interrupt request is disabled. |
| 1 | An interrupt request is enabled. |

**Table 15.4-2 Function Description of Each Bit of the DTP/Interrupt Enable Register (ENIR)**

| Bit name | | | Function |
|---|---|---|---|
| bit7<br>to<br>bit4 | EN7:<br>to<br>EN4: | External Interrupt request enable bit | • Setting of any of these bits is invalid because the external input pins are INT0 through INT3. |
| bit3 | EN3: | | • This bit enables an interrupt request.<br>• When the external interrupt request flag bit (ER3) of the DTP/interrupt cause register (EIRR) is set to "1" while this bit is set to "1", an interrupt request is output. |
| bit2 | EN2: | | • This bit enables an interrupt request.<br>• When the external interrupt request flag bit (ER2) of the DTP/interrupt cause register (EIRR) is set to "1" while this bit is set to "1", an interrupt request is output. |
| bit1 | EN1: | | • This bit enables an interrupt request.<br>• When the external interrupt request flag bit (ER1) of the DTP/interrupt cause register (EIRR) is set to "1" while this bit is set to "1", an interrupt request is output. |
| bit0 | EN0: | | • This bit enables an interrupt request.<br>• When the external interrupt request flag bit (ER0) of the DTP/interrupt cause register (EIRR) is set to "1" while this bit is set to "1", an interrupt request is output. |

**Reference:**

To use an external interrupt input pin that also serves as an I/O port, set the bit that also serves the corresponding I/O port of the port direction register (DDR) to "0" to use the pin as an input port.

The states of the external interrupt input pins can be read directly using the port data register (PDR) regardless of the states of the external interrupt request enable bits (ENIR: EN3 to EN0).

External interrupt request flag bits (ER3 to ER0) of the DTP/interrupt cause register (EIRR) are set to "1" regardless of the values of the external interrupt request enable bits (ENIR: EN3 to EN0) when a DTP/external interrupt request signal is detected.

**Table 15.4-3 Correspondence between the DTP/Interrupt Control Registers (EIRR and ENIR) and Each Channel**

| DTP/external interrupt pin | Interrupt number | External interrupt request flag bit | External interrupt request enable bit |
|---|---|---|---|
| INT3 | #16 ($10_H$) | ER3 | EN3 |
| INT2 | | ER2 | EN2 |
| INT1 | #13 ($0D_H$) | ER1 | EN1 |
| INT0 | #11 ($0B_H$) | ER0 | EN0 |

# 15.4.3 Request Level Setting Register (ELVR)

**The request level setting register (ELVR) sets the detection condition (level or edge) for interrupt requests for each external interrupt input pin.**

## ■ Request Level Setting Register (ELVR)

Figure 15.4-4 Request Level Setting Register (ELVR)

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | - | - | - | - | - | - | - | - | LB3 | LA3 | LB2 | LA2 | LB1 | LA1 | LB0 | LA0 | (Upper) XXXX0000B<br>(Lower) XXXX0000B |
| | - | - | - | - | - | - | - | - | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

| LB7 to LB0 | LA7 to LA0 | External interrupt request detection selection bits |
|---|---|---|
| ▨ 0 | ▨ 0 | The L level is to be detected. |
| 0 | 1 | The H level is to be detected. |
| 1 | 0 | A rising edge is to be detected. |
| 1 | 1 | A falling edge is to be detected. |

X : Undefined
R/W : Read/write enabled
▨ : Initial value
- : Undefined bit

Table 15.4-4 Function Description of Each Bit of the Request Level Setting Register (ELVR)

| Bit name | | | Function |
|---|---|---|---|
| Bit15 to bit8 | LB7: to LB4:<br>LA7: to LA4: | External interrupt request detection condition setting bit | • Setting of any of these bits is invalid because the external input pins are INT0 through INT3. |
| bit7 | LB3: | | • This bit is used to set the detection condition (level or edge) for interrupt requests from the signal input to the external interrupt input pin (INT3). |
| bit6 | LA3: | | |
| bit5 | LB2: | | • This bit is used to set the detection condition (level or edge) for interrupt requests from the signal input to the external interrupt input pin (INT2). |
| bit4 | LA2: | | |
| bit3 | LB1: | | • This bit is used to set the detection condition (level or edge) for interrupt requests from the signal input to the external interrupt input pin (INT1). |
| bit2 | LA1: | | |
| bit1 | LB0: | | • This bit is used to set the detection condition (level or edge) for interrupt requests from the signal input to the external interrupt input pin (INT0). |
| bit0 | LA0: | | |

**Reference:**

When the detection signal set in the request level setting register (ELVR) is input to an external interrupt input pin, the external interrupt request flag bit (EIRR: ER7 to ER0) of the corresponding pin is set to "1" regardless of the setting in the DTP/interrupt enable register (ENIR).

**Table 15.4-5 Correspondence between Request Level Setting Register (ELVR) and Each Channel**

| External interrupt input pin | Interrupt number | Bit name |
|---|---|---|
| INT3 | #16 ($10_H$) | LB3, LA3 |
| INT2 | | LB2, LA2 |
| INT1 | #13 ($0D_H$) | LB1, LA1 |
| INT0 | #11 ($0B_H$) | LB0, LA0 |

# 15.5 Operation of the DTP/External Interrupt Circuit

**The DTP/external interrupt circuit provides the external interrupt function and the DTP function. This section describes the settings required for each function and the operation of the circuit.**

## ■ Setting the DTP/External Interrupt Circuit

Figure 15.5-1 "DTP/External Interrupt Circuit" shows the settings required to operate the DTP/ external interrupt circuit.

**Figure 15.5-1 DTP/External Interrupt Circuit**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ICR | ICS3 | ICS2 | ICS1 | ICS0 | ISE | IL2 | IL1 | IL0 | ICS3 | ICS2 | ICS1 | ICS0 | ISE | IL2 | IL1 | IL0 | |
| | | | | | 0 | ◆ | ◆ | ◆ | | | | | 0 | ◆ | ◆ | ◆ | For the external interrupt function |
| | ◆ | ◆ | ◆ | ◆ | 1 | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | 1 | ◆ | ◆ | ◆ | For the DTP function |
| EIRR<br>/ENIR | | | | | ER3 | ER2 | ER1 | ER0 | | | | | EN3 | EN2 | EN1 | EN0 | |
| | | | | | ◆ | ◆ | ◆ | ◆ | | | | | ◇ | ◇ | ◇ | ◇ | |
| ELVR | | | | | | | | | LB3 | LA3 | LB2 | LA2 | LB1 | LA1 | LB0 | LA0 | |
| | | | | | | | | | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | |
| DDR | PB7 | PB6 | | | | | | | | | | | | | P81 | P80 | |
| | △ | △ | | | | | | | | | | | | | △ | △ | |

◆ : Used
◇ : Set the bit corresponding to the bit used to 1.
△ : Set the bit corresponding to the bit used to 0.
0 : Specifies 0.
1 : Specifies 1.

Set the DTP/external interrupt circuit registers in accordance with the following procedure because an interrupt request may be generated erroneously when setting them.

1. Set the DTP/interrupt enable register (ENIR) to "$00_H$" to disable interrupt requests.
2. Set the interrupt detection condition to the external interrupt request detection condition setting bit (LB3 to LB0, LA3 to LA0) corresponding to the external interrupt input pin of the request level setting register (ELVR).
3. Set the external interrupt request flag bit (ER3 to ER0) corresponding to the external interrupt input pin of the DTP/interrupt cause register (EIRR) to "0" to clear the interrupt request.
4. To enable an interrupt request, set the external interrupt request enable bit (EN3 to EN0) corresponding to the external interrupt input pin of the DTP/interrupt enable register (ENIR) to "1".

### ❍ Switching between the external interrupt function and the DTP function

Switching between the external interrupt function and the DTP function is set by the $EI^2OS$ enable bit (ISE) of the interrupt control register (ICR) corresponding to the interrupt cause to be used. When the $EI^2OS$ enable bit (ISE) is set to "1", the extended intelligent I/O service ($EI^2OS$) is enabled, operating as the DTP function. When the $EI^2OS$ enable bit (ISE) is set to "0", the extended intelligent I/O service ($EI^2OS$) is disabled, and the register operates as the external interrupt function.

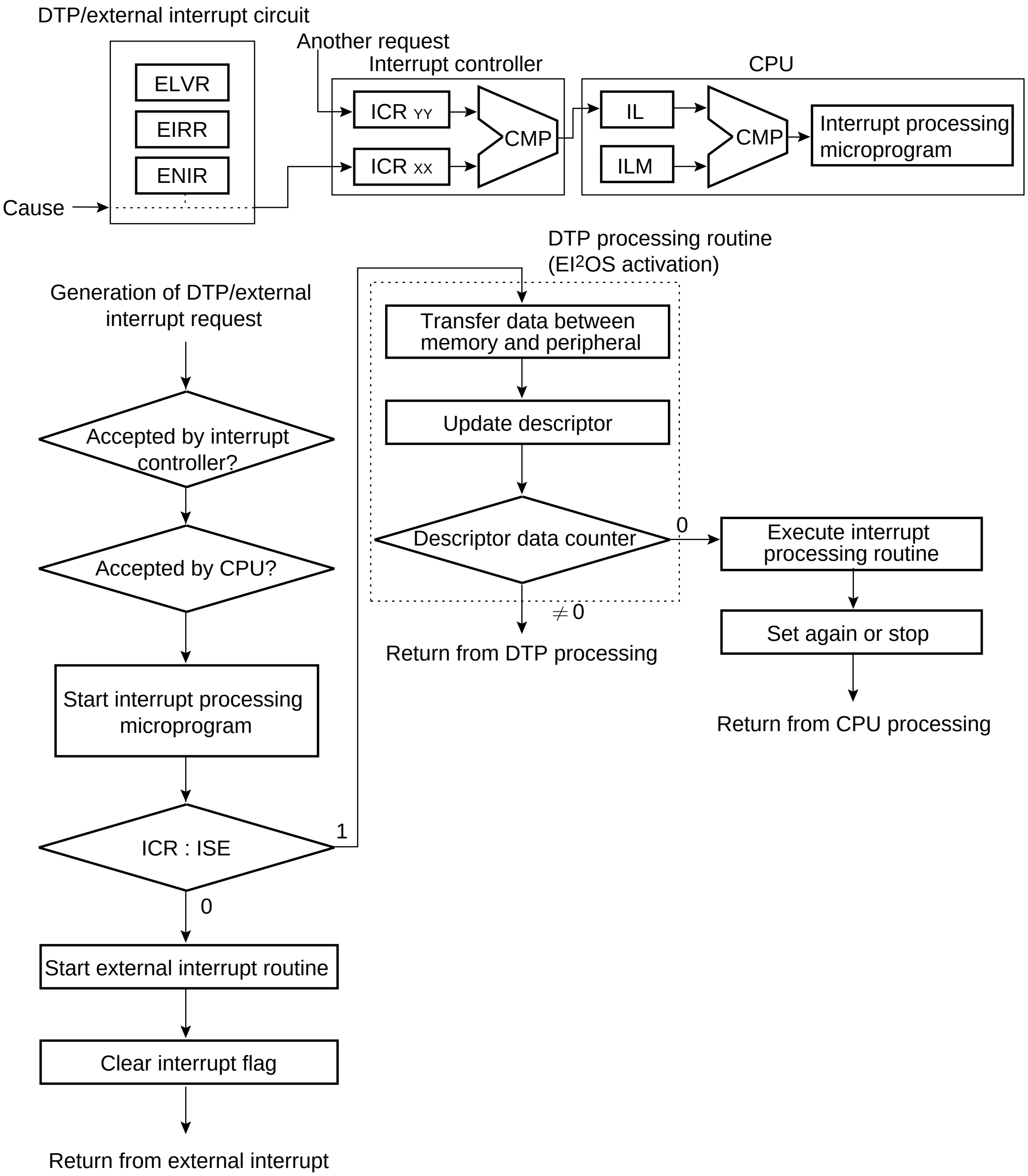
## ■ Operation of the DTP/External Interrupt Circuit

**Table 15.5-1 Control Bit and Interrupt Cause of the DTP/External Interrupt Circuit**

| | DTP/external interrupt circuit |
|---|---|
| External interrupt request flag bit | EIRR: ER3 to ER0 |
| External interrupt request enable bit | ENIR: EN3 to EN0 |
| Interrupt cause | Input of an effective edge or level to pin INT3 to INT0 |

The DTP/external interrupt circuit outputs an interrupt request to the interrupt controller when, after operations are set to the request level setting register (ELVR), DTP/interrupt cause register (EIRR), and DTP/interrupt enable register (ENIR), the detection condition set in the request level setting register (ELVR) is input to the corresponding external interrupt input pin. When the $EI^2OS$ enable bit (ICR: ISE) of the interrupt control register is "0", interrupt processing is performed. When the $EI^2OS$ enable bit (ICR: ISE) of the interrupt control register is "1", interrupt processing is performed after the extended intelligent I/O service processing (DTP processing) is executed.

**Figure 15.5-2 Operation of the DTP/External Interrupt Circuit**


DTP/external interrupt circuit
ELVR
EIRR
ENIR
Cause
Another request
Interrupt controller
ICR YY
ICR XX
CMP
CPU
IL
ILM
CMP
Interrupt processing microprogram
DTP processing routine (EI2OS activation)
Generation of DTP/external interrupt request
Accepted by interrupt controller?
Accepted by CPU?
Start interrupt processing microprogram
ICR : ISE
1
0
Start external interrupt routine
Clear interrupt flag
Return from external interrupt
Transfer data between memory and peripheral
Update descriptor
Descriptor data counter
0
≠ 0
Return from DTP processing
Execute interrupt processing routine
Set again or stop
Return from CPU processing

## 15.5.1 External Interrupt Function

**The DTP/external interrupt circuit has an external interrupt function that outputs an interrupt request when the input signal is input to an external interrupt input pin.**

### ■ External Interrupt Function

When the detection condition (level or edge) set in the request level setting register (ELVR) is input to an external interrupt input pin, the external interrupt request flag bit (ER3 to ER0) corresponding to the pin of the DTP/external interrupt cause register (EIRR) is set to "1". If the external interrupt request enable bit (EN3 to EN0) corresponding to the external interrupt input pin of the DTP/external enable register (ENIR) is set to "1", an interrupt request is output to the interrupt controller. The interrupt controller determines the interrupt level (ICR: IL2 to IL0) of interrupt requests from peripheral functions (resources) and priorities when interrupt requests are output simultaneously. The CPU determines whether to accept an interrupt request based on the interrupt level mask register (PS: ILM) and interrupt enable flag (PS: CCR: I). When the CPU accepts an interrupt request, it performs interrupt processing before branching to the interrupt processing routine. In the interrupt processing program, set the corresponding external interrupt request flag bit (ER3 to ER0) to "0" and clear the interrupt request before returning from the interrupt using an interrupt return instruction.

**Note:**

When the interrupt processing program is activated, be sure to set the external interrupt request flag bit (EIRR: EN3 to EN0) that caused the activation to "0". It is not possible to return from an interrupt while the external interrupt request flag bit (EIRR: EN3 to EN0) is set to "1".

## 15.5.2 DTP Function

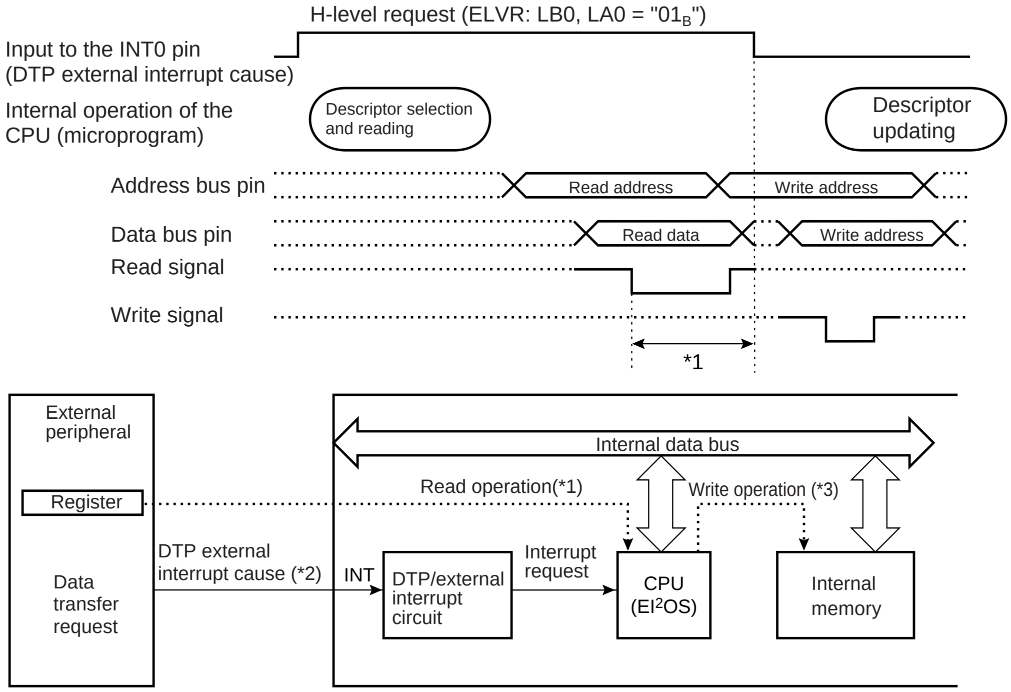
**The DTP/external interrupt circuit has a DTP (Data Transfer Peripheral) function that detects the data transfer request signal input to an external interrupt input pin from external peripheral devices and activates the extended intelligent I/O service.**

### ■ Operation of the DTP Function

The DTP function is a function for detecting the data transfer request signal input to an external interrupt input pin from external peripheral devices to automatically transfer data between memory and the peripheral devices.

The extended intelligent I/O service is activated by the external interrupt function. The operation of the DTP function is the same as that of the external interrupt function until an interrupt request is accepted by the CPU. If the EI$^2$OS operation is enabled (ICR: ISE="1"), EI$^2$OS is activated when an interrupt request is accepted to start data transfer. When the data transfer is completed, the descriptor is updated and the external interrupt request flag bit (EIRR: ER3 to ER0) is cleared to "0" to operate as the external interrupt function again. When the transfer by EI$^2$OS is completed, branching to the interrupt processing routine pointed to by the vector address of the external interrupt occurs. Peripheral devices that are externally connected should remove the cause input of the data transfer request signal (DTP cause) within three machine cycles after the first transfer started.

**Figure 15.5-3 Example of Interfacing with External Peripheral Devices**



*1 Three machine cycles
*2 Must be removed within three machine cycles of transfer.
*3 If the extended intelligent I/O service is in peripheral → memory transfer mode.

## 15.6 Usage Notes on the DTP/External Interrupt Circuit

**Notes on the signal to be input to the DTP/external interrupt circuit, release from standby mode, and interrupts are given below.**

### ■ Usage Notes on the DTP/External Interrupt Circuit

#### ❍ Conditions for external peripherals using the DTP function

To support the DTP function, peripheral devices that are externally connected must be able to automatically clear data transfer requests after transfer is carried out. If an externally connected peripheral device continues to output a transfer request longer than three machine cycles after the CPU started the transfer operation, the DTP/external interrupt circuit interprets the request as another transfer request and performs the data transfer operation again.

#### ❍ Input polarities of external interrupts

- If the request level setting register (ELVR) is set for edge detection, the pulse width of at least three machine cycles is required from the point of change of the input level to detect the input of an edge that is to become an interrupt request.
- If the request level setting register (ELVR) is set for level detection, and the level for interrupt request is input, the cause flip-flop in the DTP/interrupt cause register (EIRR) is set to "1" and retains the cause, as shown in Figure 15.6-1 "Clearing the Cause Retention Circuit When a Level is Specified". Thus, even if the interrupt cause is removed, the request to the interrupt controller remains active. To cancel the request to the interrupt controller, set the external interrupt request flag bits (EIRR: ER3 to ER0) to "0" to clear the cause flip-flop to "0", as shown in Figure 15.6-2 "DTP/External Interrupt Cause and Interrupt Request When the Output of Interrupt Requests is Enabled".

**Figure 15.6-1 Clearing the Cause Retention Circuit When a Level is Specified**

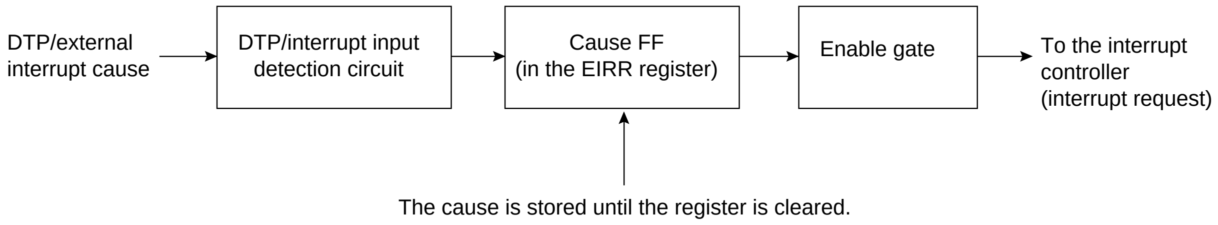
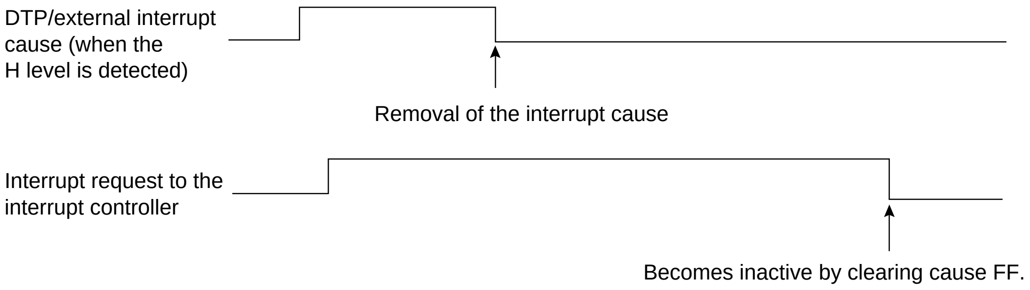

DTP/external interrupt cause
DTP/interrupt input detection circuit
Cause FF (in the EIRR register)
Enable gate
To the interrupt controller (interrupt request)
The cause is stored until the register is cleared.

**Figure 15.6-2 DTP/External Interrupt Cause and Interrupt Request When the Output of Interrupt Requests is Enabled**


DTP/external interrupt cause (when the H level is detected)
Removal of the interrupt cause
Interrupt request to the interrupt controller
Becomes inactive by clearing cause FF.

### ❍ Notes about interrupts

When the external interrupt function is used to branch to an interrupt processing routine, it is not possible to return from the interrupt processing program if an external interrupt request flag bit (EIRR: ER3 to ER0) is "1" and an external interrupt request enable bit (ENIR: EN3 to EN0) is "1". Be sure to clear the external interrupt request flag bits (EIRR: ER3 to ER0) to "0" in the processing program. (When using the DTP function, the external interrupt request flag bits (EIRR: ER3 to ER0) are cleared to "0" by EI$^2$OS.)

If the register is set for level detection, it is not possible to return from the interrupt processing program when the level signal of an interrupt request is input to an external interrupt input pin (INT3 to INT0) because, even if an external interrupt request flag bit (EIRR: ER3 to ER0) is set to "0", it is set to "1" again. Thus, disable an interrupt request or remove the level signal for an interrupt request.

# CHAPTER 16 $I^2C$ INTERFACE

**This chapter describes the functions and operations of the $I^2C$ interface of the MB90M405 series.**

## 16.1 Overview of the I$^2$C Interface

**The I$^2$C interface operates as a master or slave device on the I$^2$C bus at the serial I/O port that supports an inter-IC bus.**

### ■ Features of the I$^2$C Interface

MB90M405 series microcontrollers use one channel for the I$^2$C built-in interface.

The I$^2$C interface has the following features:

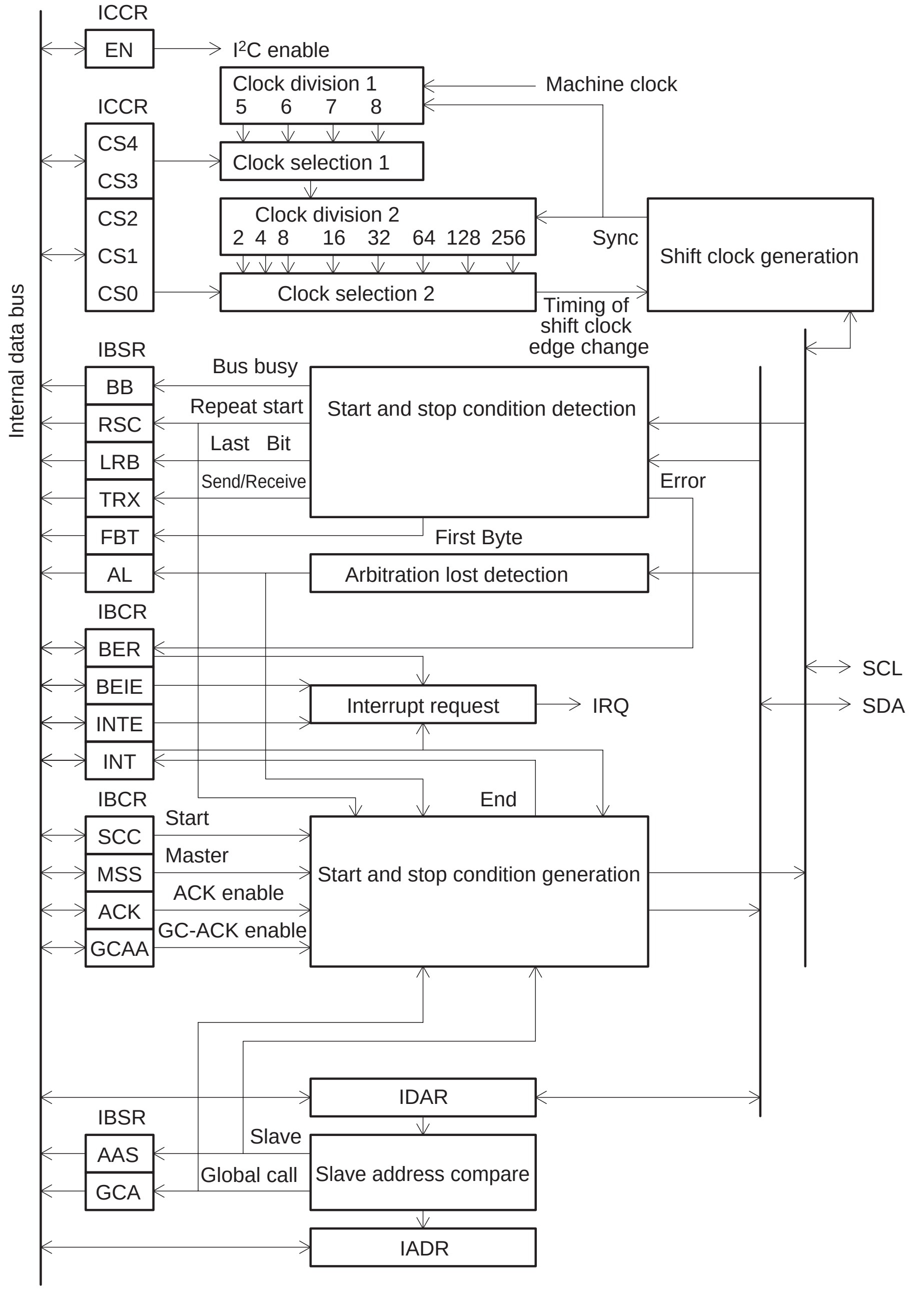
- Master/slave transmission
- Arbitration function
- Clock synchronization function
- Function for detecting a slave address and a general call address
- Function for detecting the transfer direction
- Function for repeated generation and detection of the start condition
- Bus error detection function
- Support of a transfer rate up to 100 kbps

# 16.2 Block Diagram and Configuration of the I$^2$C Interface

**Figure 16.2-1 "Block Diagram of the I$^2$C Interface" shows a block diagram of the I$^2$C interface. Figure 16.2-2 "Configuration of the I$^2$C Interface" shows the configuration of the I$^2$C interface.**

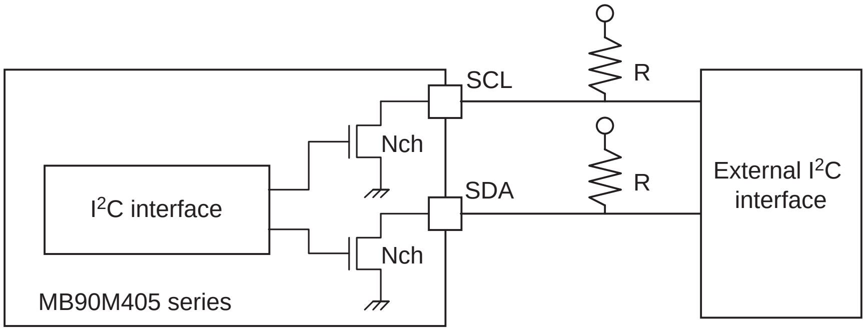
## ■ Block Diagram of the I$^2$C Interface

**Figure 16.2-1 Block Diagram of the I$^2$C Interface**



## ■ Configuration of the I$^2$C Interface

**Figure 16.2-2 Configuration of the I$^2$C Interface**



# 16.3 $I^2C$ Interface Registers

**The $I^2C$ interface uses the following six types of registers:**

- **$I^2C$ status register (IBSR)**
- **$I^2C$ control register (IBCR)**
- **$I^2C$ clock control register (ICCR)**
- **$I^2C$ address register (IADR)**
- **$I^2C$ data register (IDAR)**
- **$I^2C$ port select register (ISEL)**

## ■ $I^2C$ Interface Registers

**Figure 16.3-1 $I^2C$ Interface Registers**

| bit15 ··· bit8 | bit7 ··· bit0 |
|---|---|
| $I^2C$ port select register (ISEL) | $I^2C$ status register (IBSR) |
| $I^2C$ control register (IBCR) | |
| | $I^2C$ clock control register (ICCR) |
| $I^2C$ address register (IADR) | |
| | $I^2C$ data register (IDAR) |

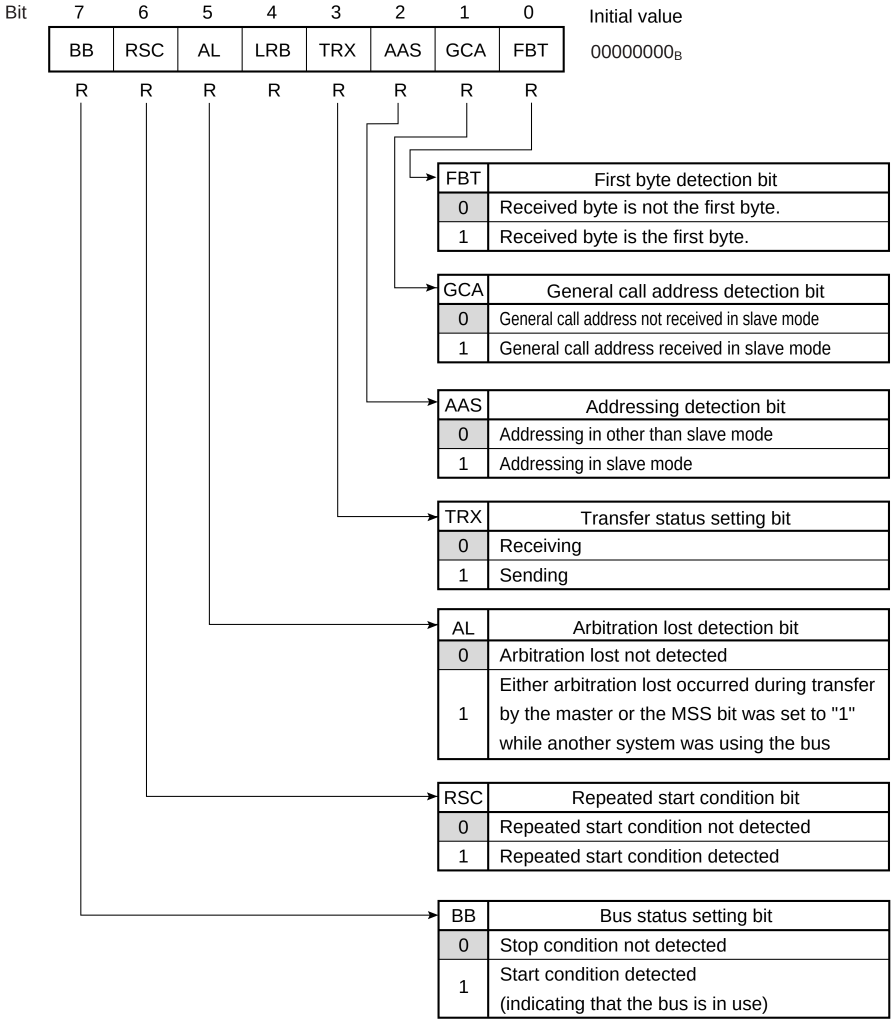
## 16.3.1 I$^2$C Status Register (IBSR)

**The I$^2$C status register (IBSR) has the following functions:**

- **Detection of a repeated start condition**
- **Detection of arbitration lost**
- **Storage of acknowledgements**
- **Detection of the first byte**
- **Detection of addressing**
- **Detection of the general call address**
- **Data transfer**

### ■ I$^2$C Status Register (IBSR)

**Figure 16.3-2 I$^2$C Status Register (IBSR)**



| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | BB | RSC | AL | LRB | TRX | AAS | GCA | FBT | 00000000$_B$ |
| | R | R | R | R | R | R | R | R | |

| FBT | First byte detection bit |
|---|---|
| 0 | Received byte is not the first byte. |
| 1 | Received byte is the first byte. |

| GCA | General call address detection bit |
|---|---|
| 0 | General call address not received in slave mode |
| 1 | General call address received in slave mode |

| AAS | Addressing detection bit |
|---|---|
| 0 | Addressing in other than slave mode |
| 1 | Addressing in slave mode |

| TRX | Transfer status setting bit |
|---|---|
| 0 | Receiving |
| 1 | Sending |

| AL | Arbitration lost detection bit |
|---|---|
| 0 | Arbitration lost not detected |
| 1 | Either arbitration lost occurred during transfer by the master or the MSS bit was set to "1" while another system was using the bus |

| RSC | Repeated start condition bit |
|---|---|
| 0 | Repeated start condition not detected |
| 1 | Repeated start condition detected |

| BB | Bus status setting bit |
|---|---|
| 0 | Stop condition not detected |
| 1 | Start condition detected<br>(indicating that the bus is in use) |

R : Read only

▨ : Initial value

**Table 16.3-1 Functions of the I$^2$C Status Register (IBSR) Bits**

| Bit name | | Function |
|---|---|---|
| bit7 | BB:<br>Bus status bit | • This bit shows the I$^2$C bus status.<br>• "0" is read from this bit if the stop condition is detected.<br>• "1" is read from this bit if the start condition is detected. |
| bit6 | RSC:<br>Repeated start condition detection bit | • This bit shows whether the repeated start condition is detected.<br>• "1" is read from this bit if the start condition is detected again while the bus is in use.<br>• This bit is cleared to "0" if the start condition or the stop condition is detected during the bus stop status while the INT bit is set to "0" and addressing is not in slave mode. |
| bit5 | AL:<br>Arbitration lost detection bit | • This bit shows whether arbitration lost is detected.<br>• "1" is read from this bit if arbitration lost has occurred during transfer by the master or if the MSS bit is set to "1" while another system is using the bus.<br>• This bit is cleared to "0" if the INT bit is set to "0". |
| bit4 | LRB:<br>Acknowledge storage bit | • This bit stores an acknowledgement from the receiving system.<br>• This bit is cleared if a start or stop condition is detected. |
| bit3 | TRX:<br>Transfer status bit | • This bit shows the data transfer send or receive status<br>• "0" is read from this bit during receiving<br>• "1" is read from this bit during sending. |
| bit2 | AAS:<br>Address detection bit | • This bit shows whether addressing is detected.<br>• "1" is read from this bit if addressing was in slave mode.<br>• This bit is cleared to "0" if the start or stop condition is detected. |
| bit1 | GCA:<br>General call address detection bit | • This bit shows whether the general call address ($00_H$) is detected.<br>• "1" is read from this bit if the general call address is received in slave mode.<br>• This bit is cleared to "0" if the start or stop condition is detected. |
| bit0 | FBT:<br>First byte detection bit | • This bit shows whether the first byte is detected.<br>• Read operations return "1" for this bit if the received byte is the first byte (address data).<br>• Even if this bit has been set to "1" due to detection of a start condition, it will be cleared to "0" if the INT bit is set to "0" or addressing was in other than slave mode. |

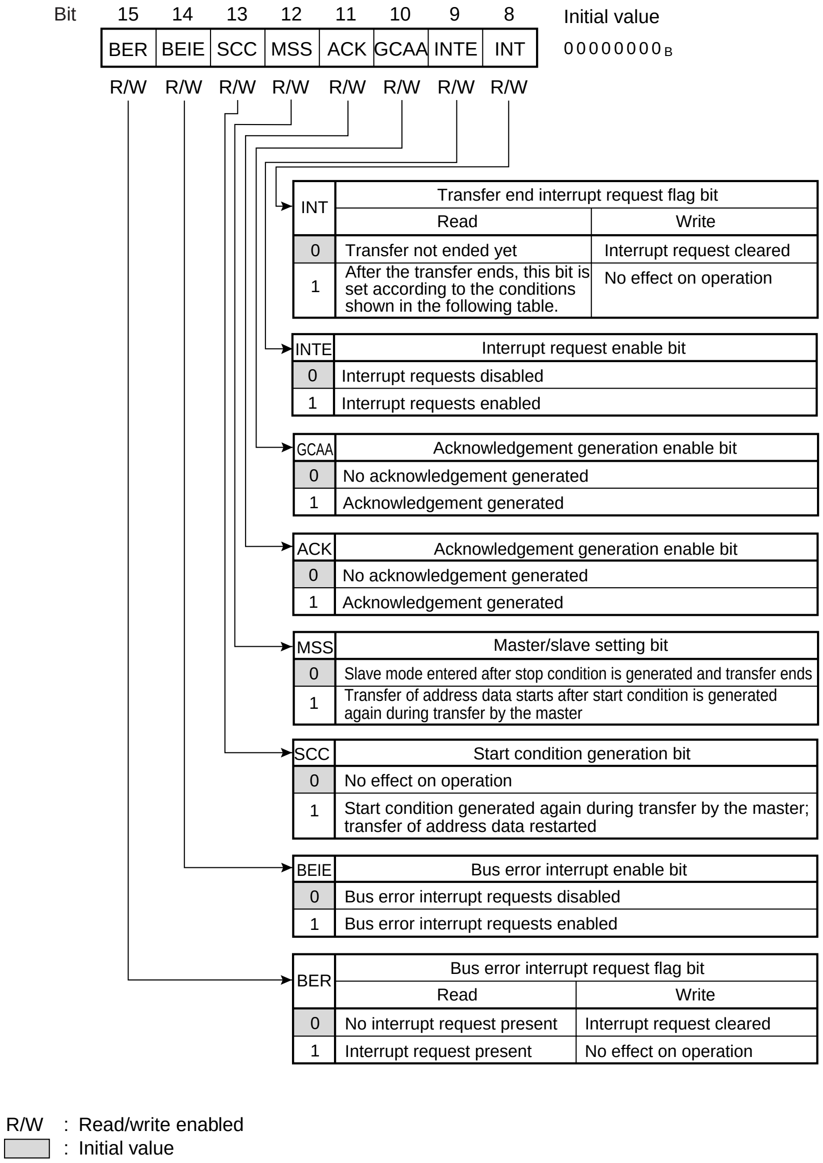
# 16.3.2 I$^2$C Control Register (IBCR)

**The I$^2$C control register (IBCR) has the following functions:**

- **Interrupt request and interrupt enable**
- **Generation of the start condition**
- **Setting a system as the master or slave**
- **Enabling of generation of acknowledgements**

## ■ I$^2$C Control Register (IBCR)

**Figure 16.3-3 I$^2$C Control Register (IBCR)**



| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | BER | BEIE | SCC | MSS | ACK | GCAA | INTE | INT | 00000000B |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

| INT | Transfer end interrupt request flag bit | |
|---|---|---|
| | Read | Write |
| 0 | Transfer not ended yet | Interrupt request cleared |
| 1 | After the transfer ends, this bit is set according to the conditions shown in the following table. | No effect on operation |

| INTE | Interrupt request enable bit |
|---|---|
| 0 | Interrupt requests disabled |
| 1 | Interrupt requests enabled |

| GCAA | Acknowledgement generation enable bit |
|---|---|
| 0 | No acknowledgement generated |
| 1 | Acknowledgement generated |

| ACK | Acknowledgement generation enable bit |
|---|---|
| 0 | No acknowledgement generated |
| 1 | Acknowledgement generated |

| MSS | Master/slave setting bit |
|---|---|
| 0 | Slave mode entered after stop condition is generated and transfer ends |
| 1 | Transfer of address data starts after start condition is generated again during transfer by the master |

| SCC | Start condition generation bit |
|---|---|
| 0 | No effect on operation |
| 1 | Start condition generated again during transfer by the master; transfer of address data restarted |

| BEIE | Bus error interrupt enable bit |
|---|---|
| 0 | Bus error interrupt requests disabled |
| 1 | Bus error interrupt requests enabled |

| BER | Bus error interrupt request flag bit | |
|---|---|---|
| | Read | Write |
| 0 | No interrupt request present | Interrupt request cleared |
| 1 | Interrupt request present | No effect on operation |

R/W : Read/write enabled
▨ : Initial value

**Table 16.3-2 Functions of the $I^2C$ Control Register (IBCR) Bits**

| Bit name | | Function |
|---|---|---|
| bit15 | BER:<br>Bus error interrupt request flag bit | • This is a bus error interrupt request flag bit.<br>• An interrupt request is output if this bit is set to "1" while the bus error interrupt request enable bit (BEIE) is "1".<br>• If this bit is set to "1", the EN bit of the CCR register is cleared, the $I^2C$ interface is stopped, and data transfer is halted.<br>• If this bit is set to "0", the interrupt request is cleared.<br>• Setting this bit to "1" does not effect operation. |
| bit14 | BEIE:<br>Bus error interrupt request enable bit | • This is a bus error interrupt enable bit.<br>• An interrupt is generated if the bus error interrupt request flag bit (BER) is set to "1" while this bit is "1". |
| bit13 | SCC:<br>Start condition generation bit | • This is a start condition generation bit.<br>• If this bit is set to "1", the start condition is generated again during transfer by the master and address data transfer is started.<br>• The read value is always "0". |
| bit12 | MSS:<br>Master/slave setting bit. | • This bit sets either master or slave mode.<br>• If this bit is set to "0", slave mode is entered after the stop condition is generated and the transfer ends.<br>• If this bit is set to "1", address data transfer is started after master mode is entered and the start condition is generated again.<br>• This bit is cleared to "0" and slave mode starts if arbitration lost occurs during transfer by the master. |
| bit11 | ACK:<br>Acknowledgement generation enable bit | • This bit enables generation of an acknowledgement when data is received.<br>• If this bit is set to "1", an acknowledgement is generated.<br>• This bit is invalid if address data is received in slave mode. |
| bit10 | GCAA:<br>Acknowledgement generation enable bit | • This bit enables generation of an acknowledgement when the general call address is received.<br>• If this bit is set to "1", an acknowledgement is generated. |
| bit9 | INTE:<br>Interrupt request enable bit | • This bit enables interrupts.<br>• An interrupt is generated if the transfer end interrupt request flag bit (INT) is set to "1" while this bit is set to "1". |
| bit8 | INT:<br>Transfer end interrupt request flag bit | • This is the transmission end interrupt request flag bit.<br>• If this bit is set to "1", the SCL line is kept at the "L" level.<br>• If this bit is set to "0", the interrupt request is cleared, the SCL line is released, and the next byte is transferred.<br>• This bit is cleared to "0" if the start or stop condition is generated in master mode. |

## ■ Notes on Competition among the SCC, MSS and INT Bits

When simultaneous writing to the SCC, MSS, and INT bits occurs, there is competition for transmission of the next byte or generation of the start or stop condition. The priority is determined as explained below.

### 1) Transmission of the next byte or generation of the stop condition

When the INT bit is set to "0" and the MSS bit is set to "0", the "0" setting of the MSS bit takes precedence, and the stop condition is generated.

### 2) Transmission of the next byte or generation of the start condition

When the INT bit is set to "0" and the SCC bit is set to "1", the "1" setting of the SCC bit takes precedence, and the start condition is generated.

### 3) Generation of the start or stop condition

Setting the SCC bit to "1" and the MSS bit to "0" at the same time is prohibited.

## 16.3.3 $I^2C$ Clock Control Register (ICCR)

**The $I^2C$ clock control register (ICCR) has the following functions:**

- **Enabling operation of the $I^2C$ interface**
- **Setting the frequency of the serial clock**

### ■ $I^2C$ Clock Control Register (ICCR)

**Figure 16.3-4 Clock Control Register (ICCR)**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | - | - | EN | CS4 | CS3 | CS2 | CS1 | CS0 | XX0XXXXX$_B$ |
| | - | - | R/W | R/W | R/W | R/W | R/W | R/W | |

| EN | $I^2C$ interface operation enable bit |
|---|---|
| 0 | Operation disabled |
| 1 | Operation enabled |

R/W : Read/write enabled
X : Undefined
▨ : Initial value

**Table 16.3-3 Functions of the $I^2C$ Clock Control Register (ICCR) Bits**

| Bit name | | Function |
|---|---|---|
| bit7<br>bit6 | -:<br>Undefined bit | • The read value of this bit is undefined.<br>• The value set for this bit does not affect operation. |
| bit5 | EN:<br>$I^2C$ interface operation enable bit | • This bit enables operation of the $I^2C$ interface.<br>• If this bit is set to "0", the bits of the $I^2C$ bus status register (IBSR) and the $I^2C$ bus control register (IBCR), except for the BER and BEIE bits, are cleared.<br>• This bit is cleared to "0" if the BER bit is set to "1". |
| bit4<br>to<br>bit0 | CS4 to CS0:<br>Serial clock frequency setting bits | • These bits set the frequency of the serial clock<br>• The frequency of the shift clock (fsck) is set as follows:<br>$\text{fsck} = \dfrac{\phi}{m \times n + 4}$<br>$\phi$: Machine clock frequency<br>• For information on the serial clock frequency setting value, see Table 16.3-5 "Serial Clock Frequency Settings (CS2 to CS0)". |

**Table 16.3-4 Serial Clock Frequency Settings (CS4 and CS3)**

| m | CS4 | CS3 |
|---|---|---|
| 5 | 0 | 0 |
| 6 | 0 | 1 |
| 7 | 1 | 0 |
| 8 | 1 | 1 |

**Table 16.3-5 Serial Clock Frequency Settings (CS2 to CS0)**

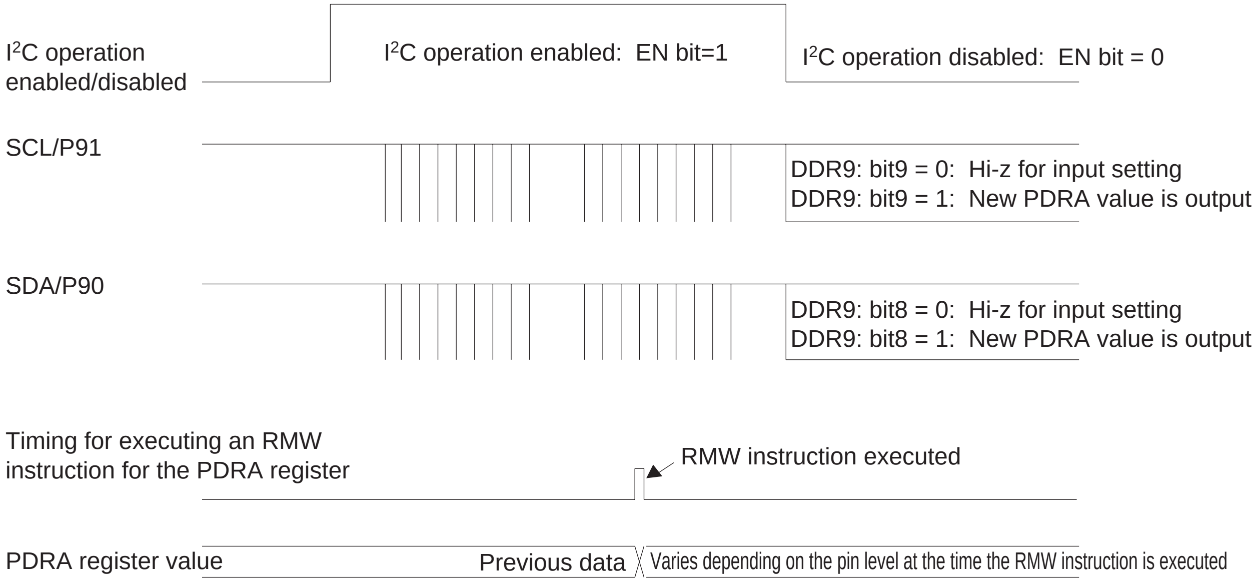
| n | CS2 | CS1 | CS0 |
|---|---|---|---|
| 4 | 0 | 0 | 0 |
| 8 | 0 | 0 | 1 |
| 16 | 0 | 1 | 0 |
| 32 | 0 | 1 | 1 |
| 64 | 1 | 0 | 0 |
| 128 | 1 | 0 | 1 |
| 256 | 1 | 1 | 0 |
| 512 | 1 | 1 | 1 |

If, for example, $m = 5$ and $n = 32$ are selected when $\phi = 16$ MHz, the resulting serial clock frequency is 97.561 kHz.

**Notes:**

- The addition of four cycles is the minimum overhead required to check whether the output level of the SCL pin has changed. More cycles are required if the rising edge delay on the SCPL pin is greater or the clock period on the slave device is prolonged.
- According to the setting of the $I^2C$ operation enable bit (EN bit), the output from the $I^2C$ common port pin varies as follows:
  - When the EN bit is set to 1 (operation is enabled): An $I^2C$ output signal is output from the SDA/D90 and SCL/P91 pins regardless of the setting values (input or output settings) of bit 8 of DDR9 bit 9 of DDR9.
  - When the EN bit is set to 0 (operation is disabled): The P90 and P91 setting values of the PDR9 register are output from the SDA/D90 and SCL/P91 pins if bit 8 of the DDR9 is "1" and bit 9 of DDR9 is also "1" (output setting).
- While the $I^2C$ is in operation, execution of an RMW instruction for the port data register (PDR9) reflecting the setting of the $I^2C$ pin reads the pin level into bit 8 and bit 9 of PDR9 during reads operations. Accordingly, note that the values of bit 8 and bit 9 of PDR9 may change depending on the level of the P91/SCL and P90/SDA pins.

**Figure 16.3-5 Change Timing for the I$^2$C Common Port**


I²C operation enabled/disabled
I²C operation enabled: EN bit=1
I²C operation disabled: EN bit = 0
SCL/P91
DDR9: bit9 = 0: Hi-z for input setting
DDR9: bit9 = 1: New PDRA value is output
SDA/P90
DDR9: bit8 = 0: Hi-z for input setting
DDR9: bit8 = 1: New PDRA value is output
Timing for executing an RMW instruction for the PDRA register
RMW instruction executed
PDRA register value
Previous data
Varies depending on the pin level at the time the RMW instruction is executed

## 16.3.4 I$^2$C Address Register (IADR)

**The I$^2$C address register (IADR) specifies the slave address.**

### ■ I$^2$C Address Register (IADR)

**Figure 16.3-6 I$^2$C Address Register (IADR)**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
|  | - | A6 | A5 | A4 | A3 | A2 | A1 | A0 | XXXXXXXX$_B$ |
|  | - | R/W | R/W | R/W | R/W | R/W | R/W | R/W |  |

R/W : Read/write enabled
X : Undefined

This register specifies the slave address. In slave mode, received address data is compared with the DAR register, and if a match occurs, an acknowledgement is sent to the master.

## 16.3.5 I$^2$C Data Register (IDAR)

**The I$^2$C data register (IDAR) is used for serial transfer.**

### ■ I$^2$C Data Register (IDAR)

**Figure 16.3-7 I$^2$C Data Register (IDAR)**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | $XXXXXXXX_B$ |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

R/W : Read/write enabled
X : Undefined

The data register is used for serial transfer. Data is transferred starting from the MSB. The data output value is "1" if data is being received (TRX="0").

The writing side of this register is double-buffered. While the bus is in use (BB="1"), the write data is loaded into the register for serial transfer for each byte is transferred. During reading, data is read directly from the register for serial transfer. Thus, the received data is valid only while the INT bit is set.

# 16.3.6 I$^2$C Port Select Register (ISEL)

**The I$^2$C port select register (ISEL) contains the settings for the I$^2$C.**

## ■ I$^2$C Port Select Register (ISEL)

**Figure 16.3-8 I$^2$C Port Select Register (ISEL)**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | - | - | - | - | - | - | - | SEL | XXXXXXX0$_B$ |
| | - | - | - | - | - | - | - | R/W | |

| | | |
|---|---|---|
| R/W | : | Read/write enabled |
| X | : | Undefined |
| - | : | Undefined bit |

**Table 16.3-6 Functions of the I$^2$C Port Select Register (ISEL) Bits**

| Bit name | | Function |
|---|---|---|
| bit15 to bit9 | -:<br>Undefined bit | • The value read from this bit is undefined.<br>• Setting this bit to a new value does not affect operation. |
| bit8 | SEL:<br>I$^2$C setting bit | • Set this bit to use the I$^2$C pin functions.<br>• If this bit is set to "0", the port or UART ch.3 is enabled.<br>• If this bit is set to "1" and the I$^2$C interface operation enable bit (EN) of the I$^2$C clock control register (ICCR) is set to "1", the pins become the I$^2$C I/O pins.<br>**Note:**<br>Be sure to set this bit to "1" to use the I$^2$C interface. |

## 16.4 Operation of the $I^2C$ Interface

**The $I^2C$ bus, which serves as a bidirectional bus line for communications, consists of a serial data line (SDA) and a serial clock line (SCL). As the $I^2C$ interface, SDA and SCL can be used as open drain I/O pins (SDA and SCL), enabling wired logic. The input withstand voltage is 5 V (Typ.).**

### ■ Start Condition

If the MSS bit is set to "1" while the bus is free (BB="0" and MSS="1"), the $I^2C$ interface enters master mode and generates the start condition at the same time. In master mode, you can generate the start condition again by setting the SCC bit to "1" even though the bus line is in use (BB="1").

The start condition is generated for either of the following conditions:

1. The MSS bit is set to "1" while the bus is free (MSS="0", BB="0", INT="0", AL="0").
2. The SCC bit is set to "1" in an interrupt state in bus master mode (MSS="1", BB="1", INT="1", AL="0").

If the MSS bit is set to "1" while the bus is being used by another system (idle state), the AL bit is set to "1". Under conditions other than (1) and (2), the "1" setting in the MSS and SCC bits is ignored.

### ■ Stop Condition

In master mode, setting the MSS bit to "0" generates a stop condition, causing the $I^2C$ interface to enter slave mode.

The stop condition is generated under the following condition:

- The MSS bit is set to "0" in an interrupt state in bus master mode (MSS="1", BB="1", INT="1", AL="0").

Under conditions other than the above, an MSS bit setting of "0" is ignored.

### ■ Addressing

If, in master mode, a start condition is generated, the BB bit is set to "1", the TRX bit is set to "1", and the contents of the IDAR register are output starting from the MSB. If, after the address data is sent, an acknowledgement is received from the slave, bit 0 of the send data (IDAR bit 0 after sending) is inverted and stored in the TRX bit.

If, in slave mode, the start condition is generated, the BB bit is set to "1", the TRX bit is cleared to "0", and the send data from the master is received in the IDAR register. After the address data is received, the IDAR and IADR registers are compared. If the register values match, AAS is set to "1" and an acknowledgement is sent to the master. Then, bit 0 of the receive data (IDAR bit 0 after receiving) is stored in the TRX bit.

## ■ Arbitration

Arbitration occurs if data is sent in master mode and another master sends data at the same time. If the send data is "1" and the data on the SDA line is at the "L level", the sender assumes that it has lost the arbitration and sets the AL bit to "1". The AL bit is set to "1" also if the start condition is generated while the bus is in use.

If the AL bit is set to "1", both the MSS and TRX bits are set to "0", causing the I$^2$C interface to enter slave receive mode.

## ■ Acknowledgement

An acknowledgement is sent from the receiver to the sender. While data is being received, the ACK bit is used to set whether an acknowledgement should be used. While data is being sent, an acknowledgement is stored in the LRB bit.

If, in slave send mode, no acknowledgement is received from the master receiver, the TRX bit is set to "0", causing the I$^2$C interface to enter slave receive mode. Thus, the master can generate the stop condition when the slave frees the SCL line.

## ■ Bus Error

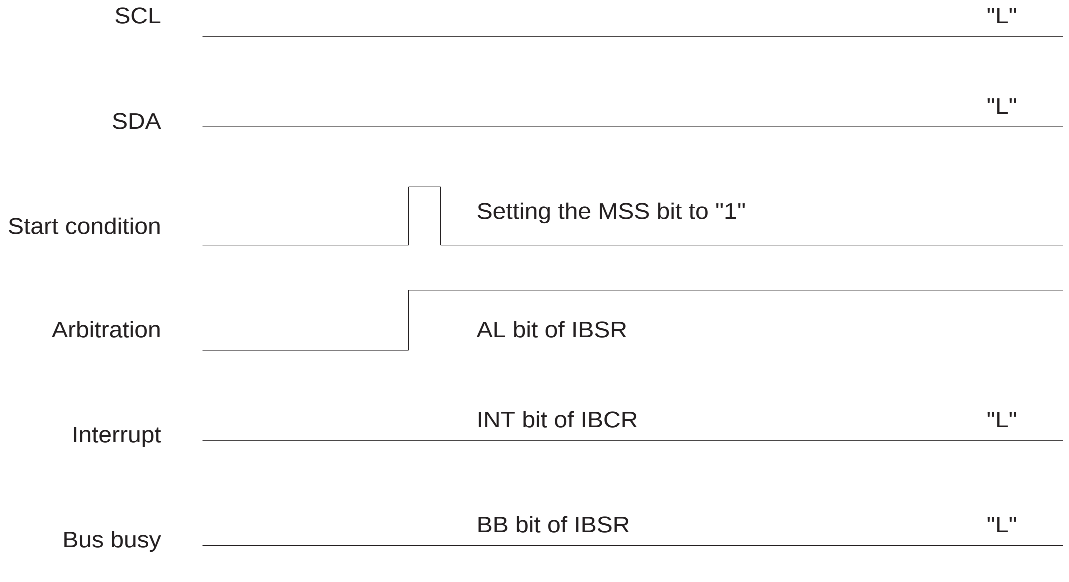
If any of the following conditions occurs, a bus error is assumed and the I$^2$C interface is stopped.

- A basic specification violation is detected on the I$^2$C bus during data transfer (including the ACK bit).
- The stop condition is detected in master mode.
- A basic specification violation is detected on the I$^2$C bus when the bus is idle.

## ■ Execution of Start Condition Generation Instruction While SDA=LOW and SCL=LOW

When the start condition generation instruction (which writes 1 to the MSS bit) is executed while SDA=LOW and SCL=LOW, this results in BB=0 and AL=1. In this case, the transfer end interrupt request flag (INT bit) is not set because the transfer is not completed. Consequently, detect this status by monitoring the BB and AL bits from the program.

**Figure 16.4-1 Change Timing for Flags When the Start Condition Generation Instruction is Executed While SDA=LOW and SCL=LOW**



## 16.4.1 Transfer Flow of the I$^2$C Interface

**Figure 16.4-2 "One-byte Transfer Flow from the Master to the Slave" shows the flow of a one-byte transfer from the master to the slave. Figure 16.4-3 "One-byte Transfer Flow from the Slave to the Master" shows the flow of a one-byte transfer from the slave to the master.**

### ■ Transfer Flow of the I$^2$C Interface

**Figure 16.4-2 One-byte Transfer Flow from the Master to the Slave**

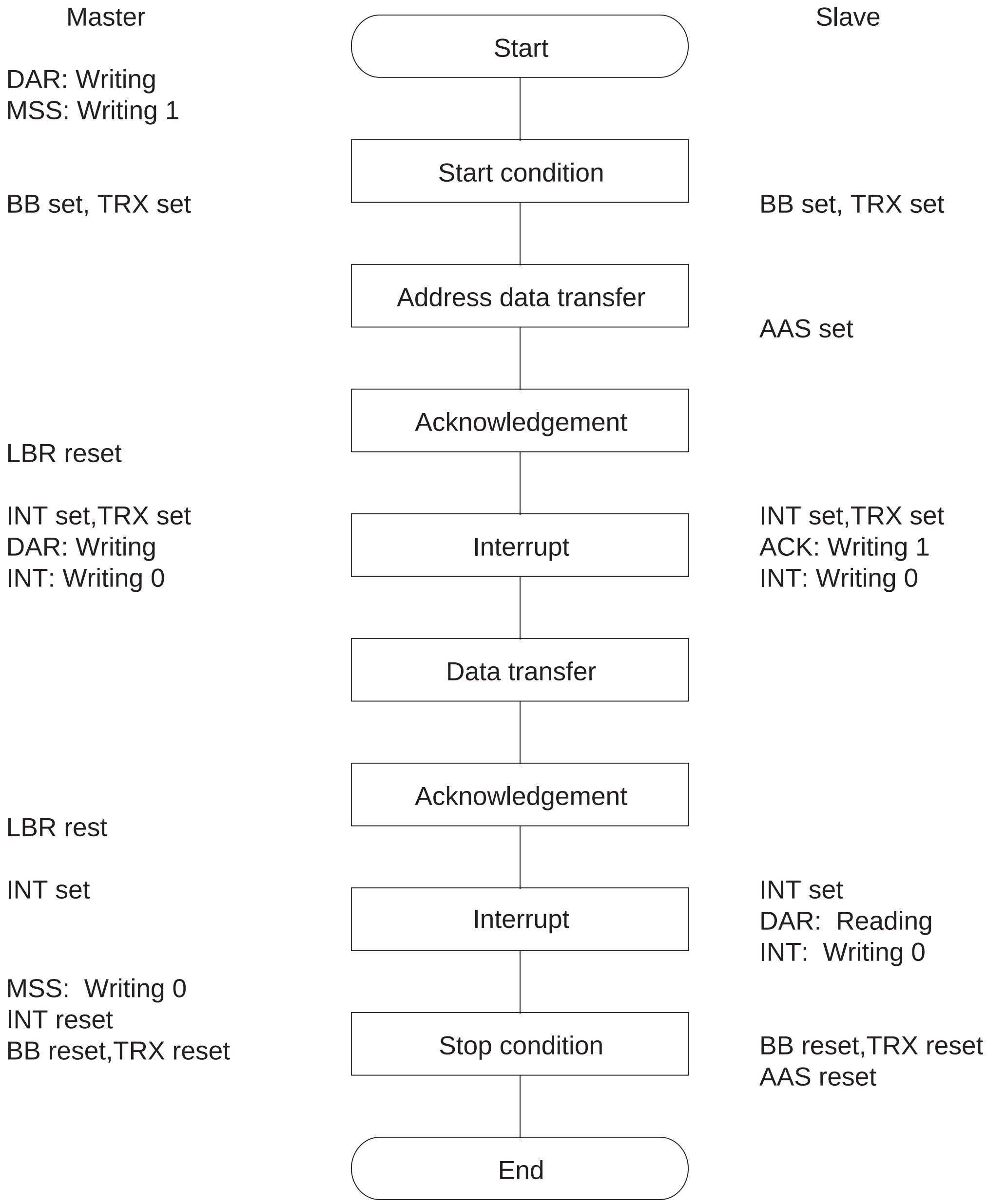
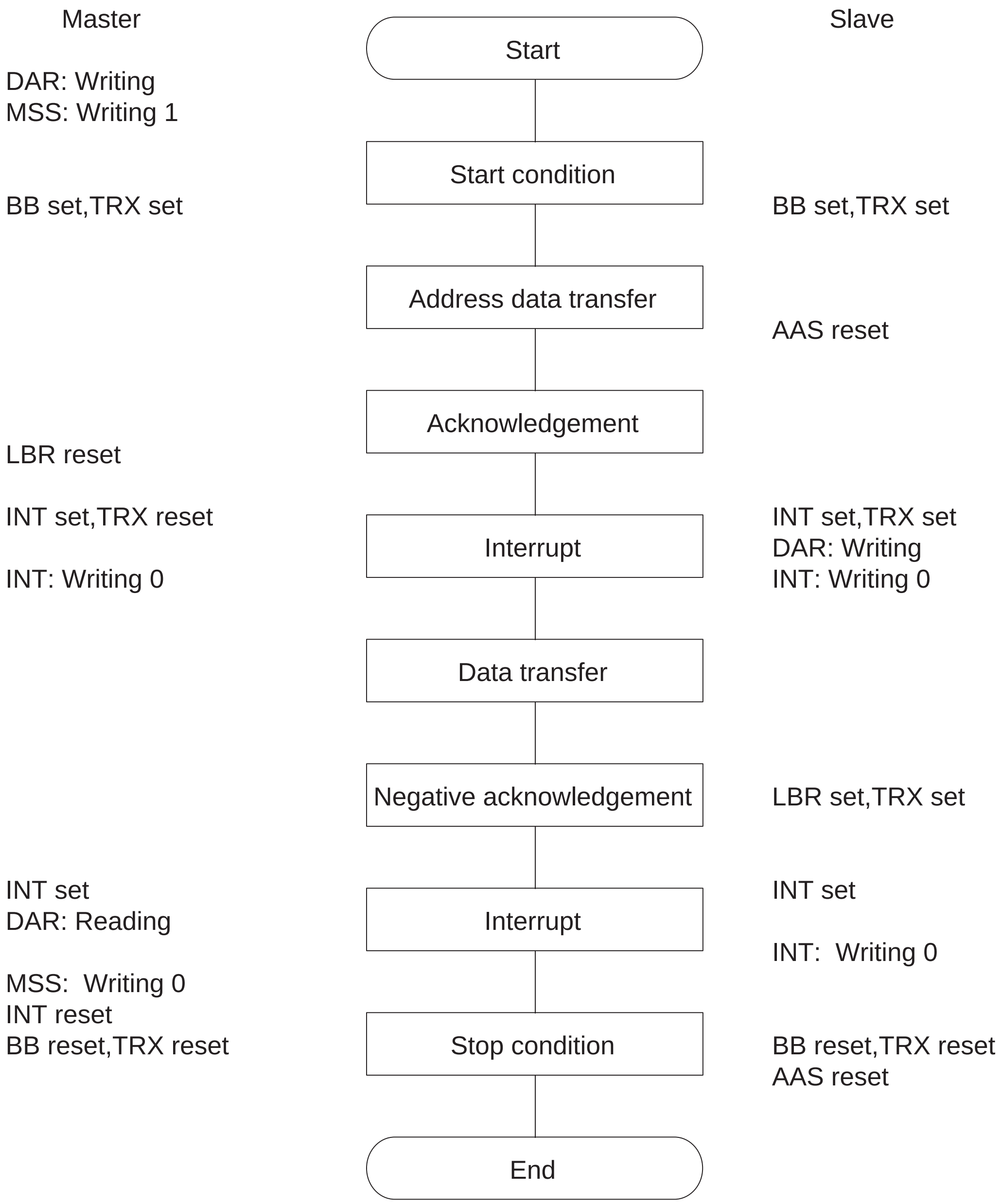


**Figure 16.4-3 One-byte Transfer Flow from the Slave to the Master**

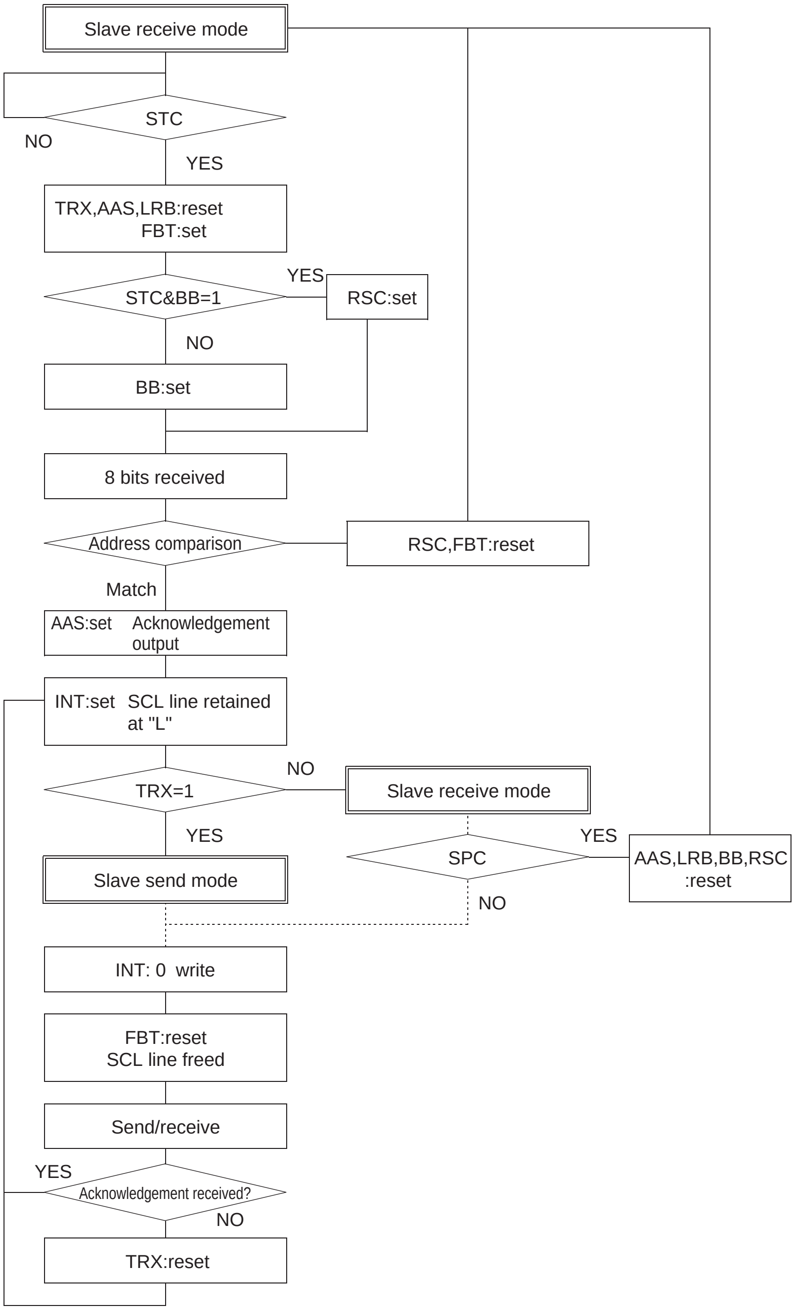


## 16.4.2 Mode Flow of the I$^2$C Interface

**Figure 16.4-4 "I$^2$C Mode Flow" shows the flow of mode transitions for the I$^2$C interface.**

### ■ Flow of I$^2$C Interface Mode Transitions

**Figure 16.4-4 I$^2$C Mode Flow**



## 16.4.3 Operation Flow of the I$^2$C Interface

**Figure 16.4-5 "Operation Flow of the Master Send/Receive Program (with Interrupts) for the I$^2$C Interface" shows the operation flow of a master send/receive program (with interrupts) for the I$^2$C interface. Figure 16.4-6 "Operation Flow of the Slave Program (with Interrupts) for the I$^2$C Interface" shows the operation flow of the slave program (with interrupts) for the I$^2$C interface.**

### ■ Operation Flow of the I$^2$C Interface

**Figure 16.4-5 Operation Flow of the Master Send/Receive Program (with Interrupts) for the I$^2$C Interface**

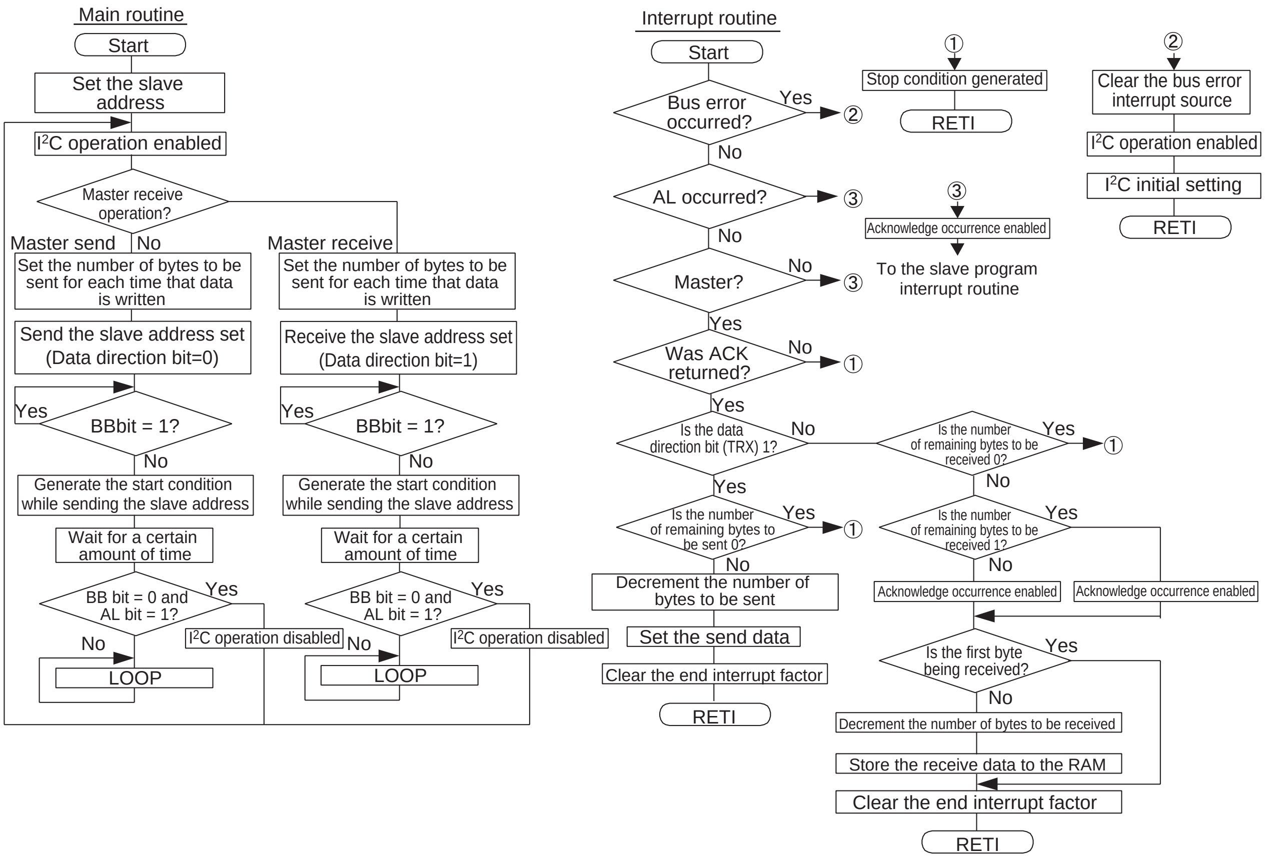
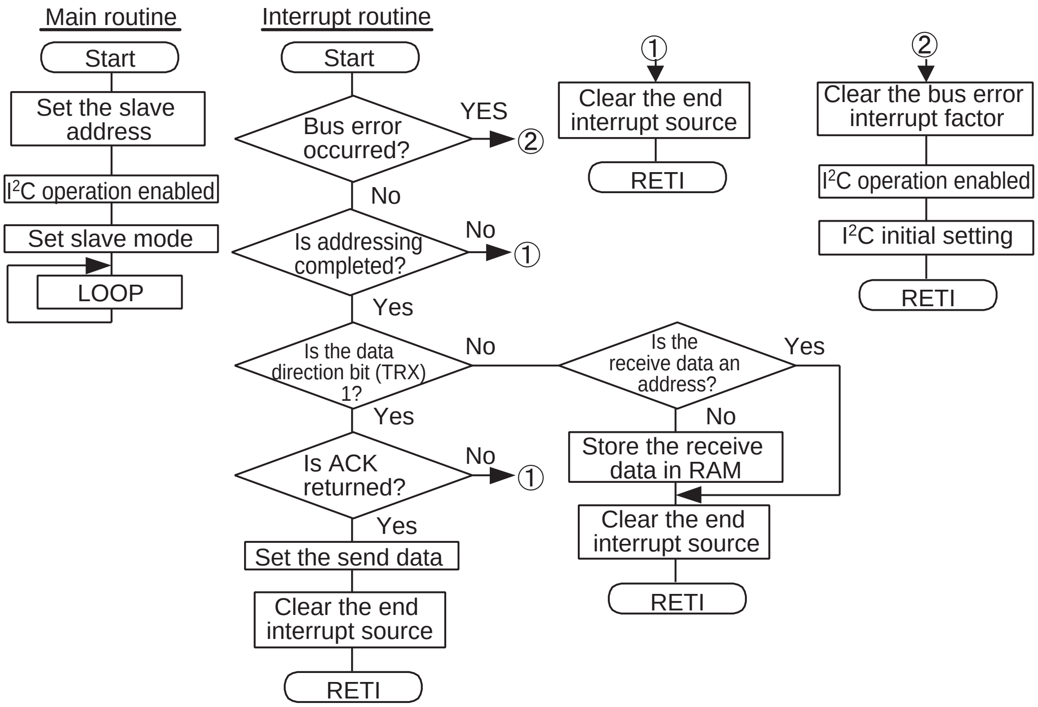

Main routine
Start
Set the slave address
I2C operation enabled
Master receive operation?
Master send
No
Master receive
Set the number of bytes to be sent for each time that data is written
Send the slave address set (Data direction bit=0)
Yes
BBbit = 1?
No
Generate the start condition while sending the slave address
Wait for a certain amount of time
BB bit = 0 and AL bit = 1?
Yes
No
I2C operation disabled
LOOP
Set the number of bytes to be sent for each time that data is written
Receive the slave address set (Data direction bit=1)
Yes
BBbit = 1?
No
Generate the start condition while sending the slave address
Wait for a certain amount of time
BB bit = 0 and AL bit = 1?
Yes
No
I2C operation disabled
LOOP
Interrupt routine
Start
Bus error occurred?
Yes
②
No
AL occurred?
③
No
Master?
No
③
Yes
Was ACK returned?
No
①
Yes
Is the data direction bit (TRX) 1?
No
Yes
Is the number of remaining bytes to be sent 0?
Yes
①
No
Decrement the number of bytes to be sent
Set the send data
Clear the end interrupt factor
RETI
①
Stop condition generated
RETI
③
Acknowledge occurrence enabled
To the slave program interrupt routine
②
Clear the bus error interrupt source
I2C operation enabled
I2C initial setting
RETI
Is the number of remaining bytes to be received 0?
Yes
①
No
Is the number of remaining bytes to be received 1?
Yes
No
Acknowledge occurrence enabled
Acknowledge occurrence enabled
Is the first byte being received?
Yes
No
Decrement the number of bytes to be received
Store the receive data to the RAM
Clear the end interrupt factor
RETI

**Figure 16.4-6 Operation Flow of the Slave Program (with Interrupts) for the I$^2$C Interface**



# CHAPTER 17 8/10-BIT A/D CONVERTER

**This chapter describes the functions and operations of the MB90M405 series 8/10-bit A/D converter.**

# 17.1 Overview of the 8/10-Bit A/D Converter

**The 8/10-bit A/D converter has a function for converting the analog input voltage into a 10-bit or 8-bit value using the RC-type successive approximation conversion method.**

## ■ Functions of the 8/10-Bit A/D Converter

The following shows the functions of the 8/10-bit A/D converter:

- The minimum conversion time is 5.9 µs (for a machine clock of 16.8 MHz; includes the sampling time).
- The minimum sampling time is 2.1 µs (for a machine clock of 16.8 MHz).
- The converter uses the RC-type successive approximation conversion method with a sample hold circuit.
- A resolution of 10 bits or 8 bits can be selected.
- The input signal can be set using a program from the 8-channel analog input pins.
- At the end of A/D conversion, an interrupt request can be generated and $EI^2OS$ can be activated.
- If A/D conversion is performed while an interrupt is enabled, the conversion data protection function is activated.
- The conversion can be activated by software, 16-bit reload timer 1 output (rising edge), and 16-bit free-running timer zero detection edge.

Table 17.1-1 "8/10-bit A/D Converter Conversion Modes" lists four types of conversion modes.

**Table 17.1-1 8/10-bit A/D Converter Conversion Modes**

| Conversion mode | Single conversion | Scan conversion |
|---|---|---|
| Single conversion mode 1<br>Single conversion mode 2 | Converts the input of a specified channel (single channel) just once. | Converts the inputs of two or more consecutive channels (up to 16 channels) just once. |
| Continuous conversion mode | Converts the input of a specified channel (single channel) repeatedly. | Repeatedly converts the inputs of two or more consecutive channels (up to 16 channels). |
| Stop conversion mode | Converts the input of a specified channel (single channel), after which it is on standby for the next activation. | Converts the inputs of two or more consecutive channels (up to 16 channels) once, then enters standby mode and waits for the next start. |

## ■ 8/10-Bit A/D Converter Interrupts and $EI^2OS$

Table 17.1-2 8/10-Bit A/D Converter Interrupts and $EI^2OS$

| Interrupt No. | Interrupt control register | | Vector table address | | | $EI^2OS$ |
|---|---|---|---|---|---|---|
| | Register name | Address | Lower | Upper | Bank | |
| #37 ($25_H$) | ICR13 | $0000BD_H$ | $FFFF68_H$ | $FFFF69_H$ | $FFFF6A_H$ | O |

O: Available

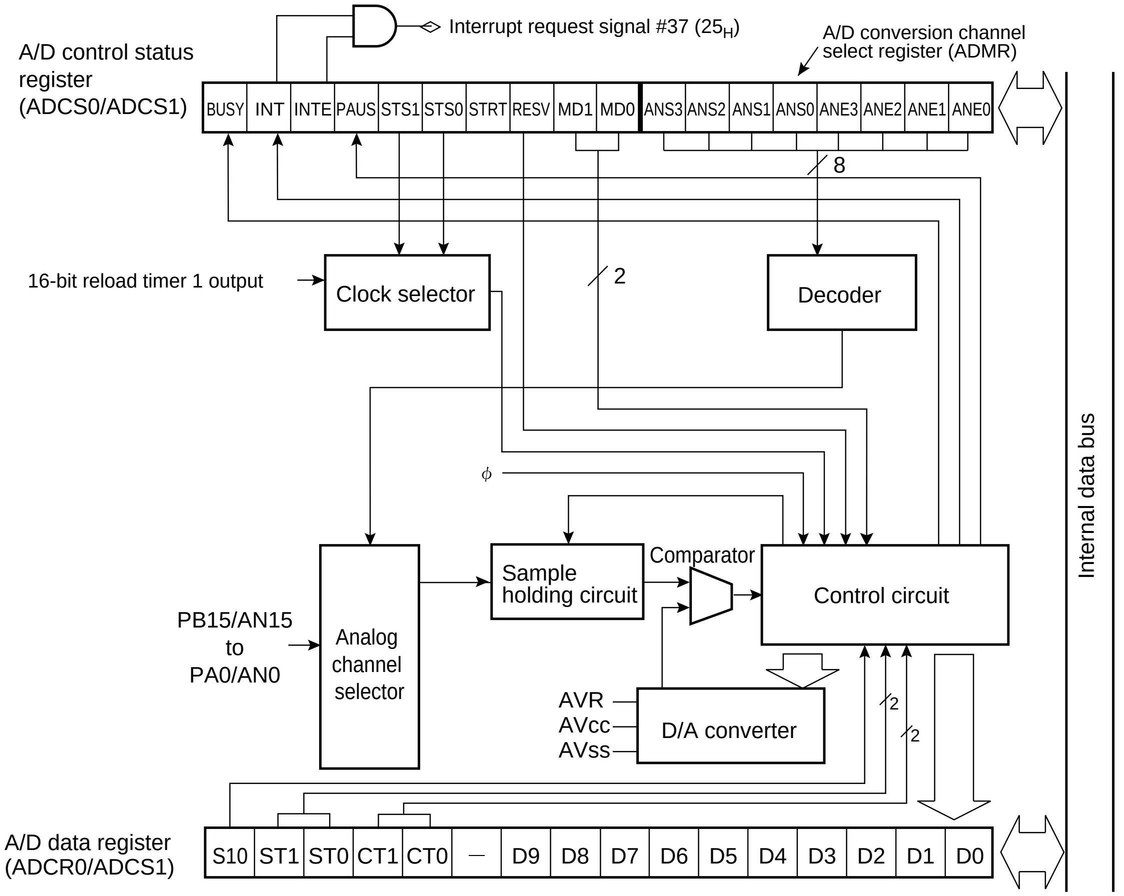
# 17.2 Configuration of the 8/10-Bit A/D Converter

**The 8/10-bit A/D converter consists of the following blocks:**

- **A/D control status register (ADCS0/ADCS1)**
- **A/D data register (ADCR0/ADCR1)**
- **A/D conversion channel select register (ADMR)**
- **Clock selector (Input clock selector for activating A/D conversion)**
- **Decoder**
- **Analog channel selector**
- **Sample hold circuit**
- **D/A converter**
- **Comparator**
- **Control circuit**

## ■ Block Diagram of the 8/10-Bit A/D Converter

**Figure 17.2-1 Block Diagram of the 8/10-Bit A/D Converter**



$\phi$ : Machine clock
* : Interrupt signal

### ❍ A/D control status register (ADCS0/ADCS1)

The A/D control status register (ADCS0) sets the A/D conversion mode.

The A/D control status register (ADCS1) sets the A/D conversion activation trigger and enable/ disable of interrupt requests and checks the interrupt request status and whether the A/D conversion has halted/is in progress.

### ❍ A/D data register (ADCR0/ADCR1)

This register stores the results of A/D conversion. It also sets the resolution for A/D conversion, sampling time during A/D conversion, and compare time during A/D conversion.

### ❍ A/D conversion channel select register (ADMR)

A/D conversion channel select register (ADMR) sets the A/D conversion start/end channel.

### ❍ Clock selector

The clock selector selects the clock for starting A/D conversion. The 16-bit reload timer 1 output can be used as the start clock.

### ❍ Decoder

This circuit sets the analog input pin to be used based on the A/D conversion end channel setting bits (ANE0 to ANE3) and A/D conversion start channel setting bits (ANS0 to ANS3) of the A/D control status register (ADCS0).

### ❍ Analog channel selector

This circuit selects the pin to be used from among 16 analog input pins.

### ❍ Sample hold circuit

This circuit maintains the input voltage from the pin set by the analog channel selector. By maintaining the input voltage just after starting A/D conversion, it is not affected by input voltage variations during A/D conversion (during comparison).

### ❍ D/A converter

This circuit generates a reference voltage for comparison with the input voltage maintained by the sample hold circuit.

### ❍ Comparator

This circuit compares the input voltage maintained by the sample hold circuit with the output voltage of the D/A converter to determine which is greater.

### ❍ Control circuit

This circuit determines the A/D conversion value based on the decision signal generated by the comparator. When the A/D conversion has been completed, the circuit sets the conversion result in the A/D data register (ADCR0/ADCR1) and generates an interrupt request.

## 17.3 8/10-Bit A/D Converter Pins

**This section describes the 8/10-bit A/D converter pins and provides pin block diagrams.**

### ■ 8/10-Bit A/D Converter Pins

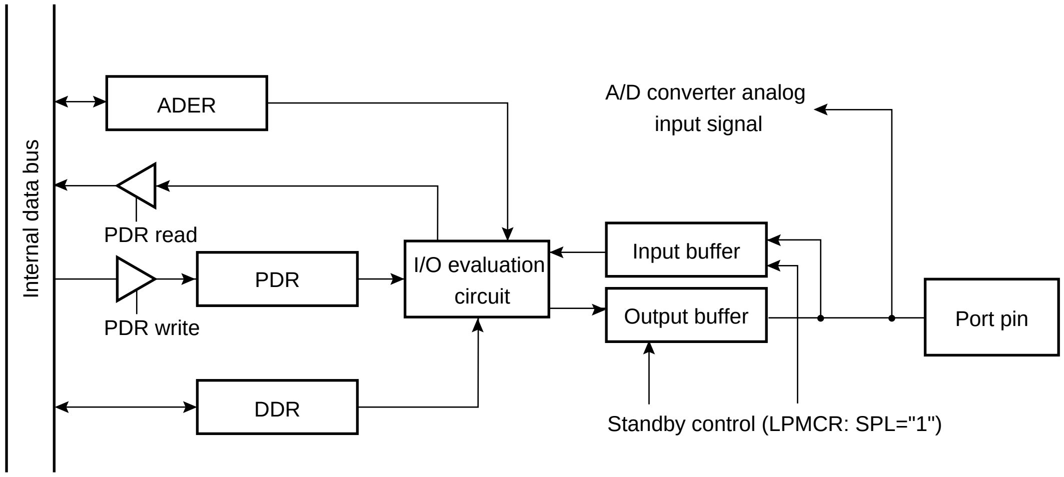
The A/D converter pins are also used as I/O ports.

**Table 17.3-1 8/10-bit A/D Converter Pins**

| Function | Pin name | Pin function | Input-output signal type | Pull-up option | Standby control | Setting for using the pin |
|---|---|---|---|---|---|---|
| Channel 0 | PA0/AN0 | Port A input-output or analog input | CMOS output/ CMOS hysteresis input or analog input | Not selectable | Not selectable | Set Port A as an input port (DDRA:bit0 to bit7="0").<br>Set as an analog port (ADER0:bit0 to bit7="1") |
| Channel 1 | PA1/AN1 | | | | | |
| Channel 2 | PA2/AN2 | | | | | |
| Channel 3 | PA3/AN3 | | | | | |
| Channel 4 | PA4/AN4 | | | | | |
| Channel 5 | PA5/AN5 | | | | | |
| Channel 6 | PA6/AN6 | | | | | |
| Channel 7 | PA7/AN7 | | | | | |
| Channel 8 | PB0/AN8 | Port B input-output or analog input | CMOS output/ CMOS hysteresis input or analog input | Not selectable | Not selectable | Set Port B as an input port (DDRB:bit0 to bit7="0").<br>Set as an analog port (ADER1:bit0 to bit7="1") |
| Channel 9 | PB1/AN9 | | | | | |
| Channel 10 | PB2/AN10 | | | | | |
| Channel 11 | PB3/AN11 | | | | | |
| Channel 12 | PB4/AN12 | | | | | |
| Channel 13 | PB5/AN13 | | | | | |
| Channel 14 | PB6/AN14 | | | | | |
| Channel 15 | PB7/AN15 | | | | | |

## ■ Block Diagrams of the 8/10-Bit A/D Converter Pins

**Figure 17.3-1 Block Diagram of the PA0/AN0 to PB7/AN15 Pins**



Standby control: Stop mode and LPMCR: SPL="1"

**Notes:**

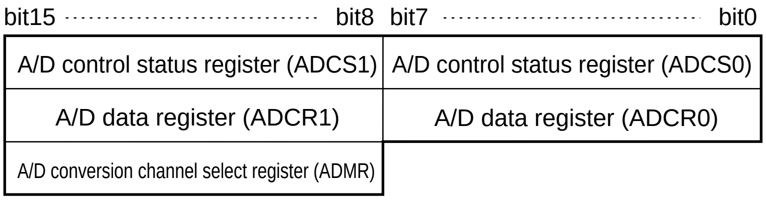
- To use a pin as an input port, set the corresponding bit (bit7 to bit0) of the port direction registers (DDRA and DDRB) to "0" and the corresponding bit (bit15 to bit0) of the analog input enable registers (ADER0 and ADER1) to "0".
- To use a pin as an analog input pin, set the corresponding bit (bit15 to bit0) of the analog input enable registers (ADER0 and ADER1) to "1". The value read from each of the port data registers (PDRA and PDRB) is $00_H$.

# 17.4 8/10-Bit A/D Converter Registers

**This section lists the 8/10-bit A/D converter registers.**

## ■ 8/10-Bit A/D Converter Registers

**Figure 17.4-1 List of Registers of the 8/10-Bit A/D Converter**



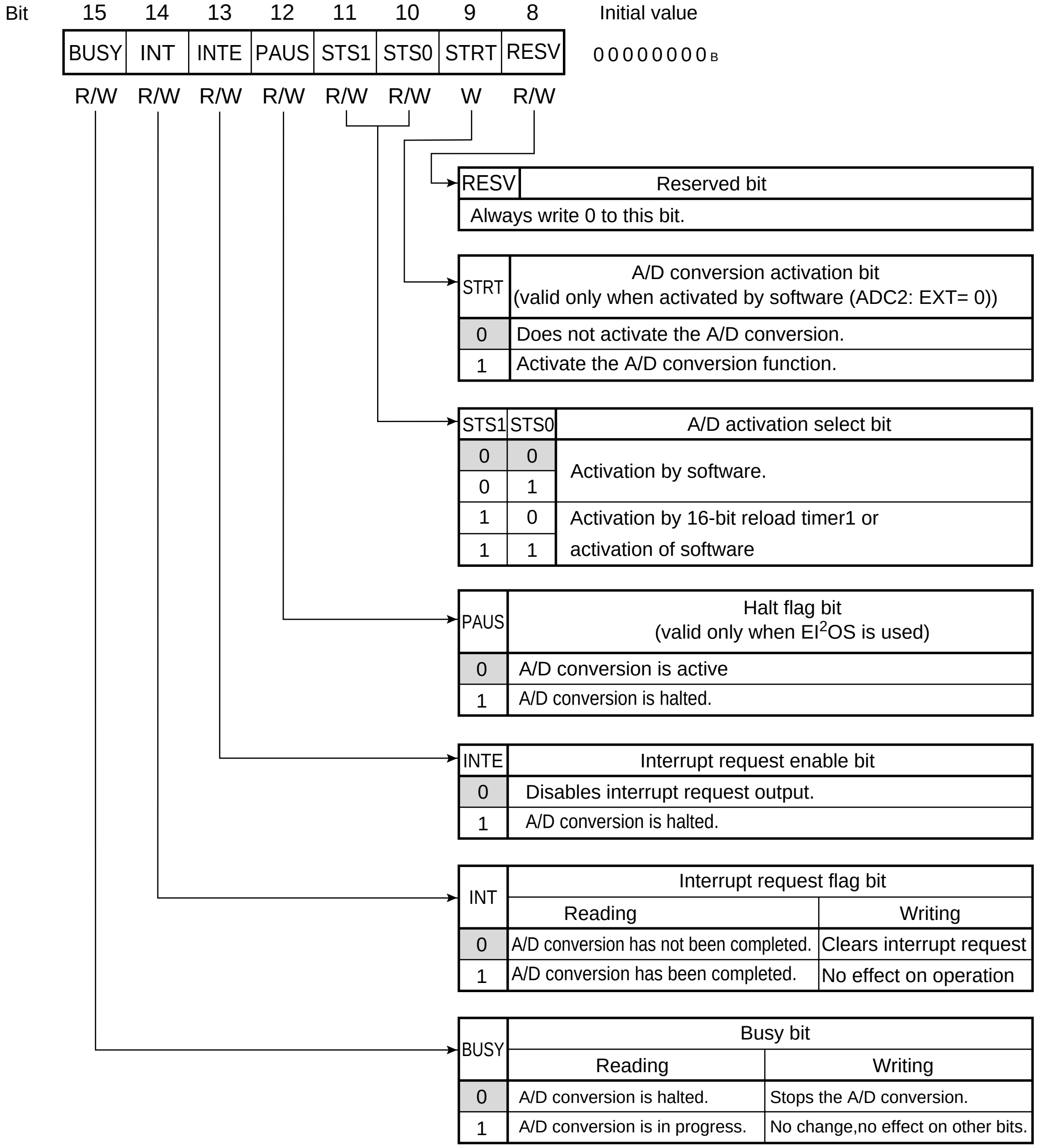
| bit15 ········ bit8 | bit7 ········ bit0 |
|---|---|
| A/D control status register (ADCS1) | A/D control status register (ADCS0) |
| A/D data register (ADCR1) | A/D data register (ADCR0) |
| A/D conversion channel select register (ADMR) | |

## 17.4.1 A/D Control Status Register 1 (ADCS1)

**The A/D control status register (ADCS1) sets the A/D conversion activation trigger and enable/disable of interrupt requests and checks the interrupt request status and whether the A/D conversion has halted/is in progress.**

### ■ A/D Control Status Register 1 (ADCS1)

**Figure 17.4-2 A/D Control Status Register 1 (ADCS1)**



| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | BUSY | INT | INTE | PAUS | STS1 | STS0 | STRT | RESV | 00000000B |
| | R/W | R/W | R/W | R/W | R/W | R/W | W | R/W | |

| RESV | Reserved bit |
|---|---|
| Always write 0 to this bit. | |

| STRT | A/D conversion activation bit<br>(valid only when activated by software (ADC2: EXT= 0)) |
|---|---|
| 0 | Does not activate the A/D conversion. |
| 1 | Activate the A/D conversion function. |

| STS1 | STS0 | A/D activation select bit |
|---|---|---|
| 0 | 0 | Activation by software. |
| 0 | 1 | |
| 1 | 0 | Activation by 16-bit reload timer1 or activation of software |
| 1 | 1 | |

| PAUS | Halt flag bit<br>(valid only when EI$^2$OS is used) |
|---|---|
| 0 | A/D conversion is active |
| 1 | A/D conversion is halted. |

| INTE | Interrupt request enable bit |
|---|---|
| 0 | Disables interrupt request output. |
| 1 | A/D conversion is halted. |

| INT | Interrupt request flag bit | |
|---|---|---|
| | Reading | Writing |
| 0 | A/D conversion has not been completed. | Clears interrupt request |
| 1 | A/D conversion has been completed. | No effect on operation |

| BUSY | Busy bit | |
|---|---|---|
| | Reading | Writing |
| 0 | A/D conversion is halted. | Stops the A/D conversion. |
| 1 | A/D conversion is in progress. | No change,no effect on other bits. |

R/W : Read/write
W : Write only
▨ : Initial value

**Table 17.4-1 Function Description of Each Bit of A/D Control Status Register 1 (ADCS1)**

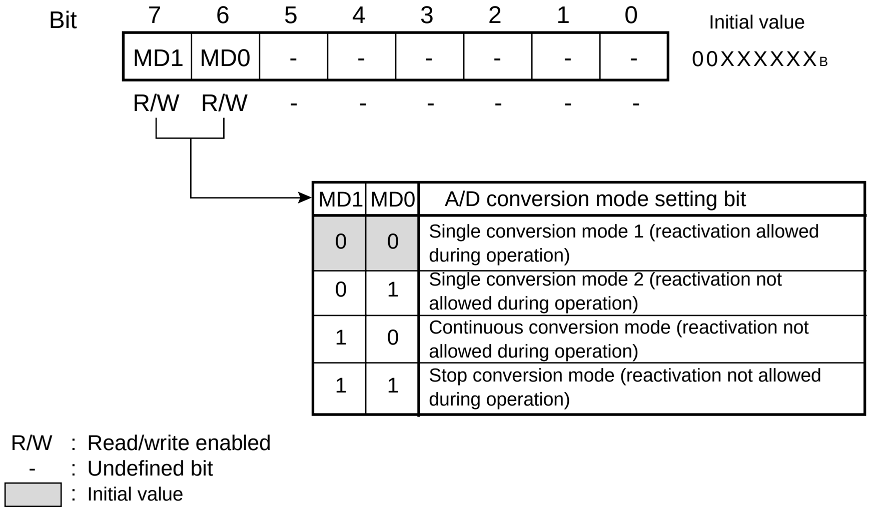
| Bit name | | Function |
|---|---|---|
| bit15 | BUSY:<br>Busy bit | • This bit indicates the operating status of the A/D converter<br>• When this bit is "0", the A/D conversion has halted.<br>• When this bit is "1", the A/D conversion is in progress.<br>• When this bit is set to "0", the A/D conversion is forced to halt.<br>• When this bit is set to "1", operation is not affected.<br>**Note:**<br>Do not set the forced A/D conversion stop and the activation (BUSY="0", STRT="1") simultaneously. |
| bit14 | INT:<br>Interrupt request flag bit | • This bit is a flag that requests an interrupt.<br>• This bit is set to "1" when A/D conversion results are stored in the A/D data register (ADCR0/ADCR1).<br>• When this bit is set to "1" while the interrupt request enable bit (INTE) is "1", an interrupt request is output.<br>• When this bit is set to "0", the interrupt request is cleared.<br>• When this bit is set to "1", operation is not affected.<br>• When $EI^2OS$ is used, this bit is cleared to "0".<br>**Note:**<br>To clear the interrupt requests, stop the A/D conversion. |
| bit13 | INTE:<br>Interrupt request enable bit | • This bit enables an interrupt request.<br>• When the interrupt request flag bit (INT) is set to "1" while this bit is set to "1", an interrupt request is output.<br>• To use $EI^2OS$, set this bit to "1". |
| bit12 | PAUS:<br>Halt flag bit | • This bit is set to "1" when the A/D conversion stops temporarily.<br>• When $EI^2OS$ is used in continuous conversion mode, this bit is set to "1" if transfer of the last piece of data to memory has not been completed even though A/D conversion is completed. Thus, the A/D conversion is stopped temporarily and conversion data is not stored in the A/D data register (ADCR0/ ADCR1).<br>• When data transfer of the last piece of data to memory is completed, this bit is cleared to "0" and the A/D conversion is then restarted.<br>**Note:**<br>This bit is valid when $EI^2OS$ is used. |
| bit11<br>bit10 | STS1, STS0:<br>A/D activation select bit | • These bits select how A/D conversion is to be activated.<br>• When two or more activation causes are shared, activation is the result of the cause that occurs first.<br>**Note:**<br>Change the setting during A/D conversion only while there is no corresponding activation cause, since the change becomes effective immediately. |
| bit9 | STRT:<br>A/D conversion activation bit | • This bit allows software to start A/D conversion.<br>• Writing 1 to this bit activates A/D conversion.<br>• In stop conversion mode, conversion cannot be reactivated with this bit.<br>**Note:**<br>Never perform the forced stop and activation (BUSY="0", STRT="1") of the A/D conversion simultaneously. |
| bit8 | RESV:<br>Reserved bit | **Note:**<br>Always write 0 to this bit. |

## 17.4.2 A/D Control Status Register 0 (ADCS0)

**The A/D control status register (ADCS0) sets the A/D conversion mode.**

### ■ A/D Control Status Register 0 (ADCS0)

**Figure 17.4-3 A/D Control Status Register 0 (ADCS0)**



| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | MD1 | MD0 | - | - | - | - | - | - | 00XXXXXX$_B$ |
| | R/W | R/W | - | - | - | - | - | - | |

| MD1 | MD0 | A/D conversion mode setting bit |
|---|---|---|
| 0 | 0 | Single conversion mode 1 (reactivation allowed during operation) |
| 0 | 1 | Single conversion mode 2 (reactivation not allowed during operation) |
| 1 | 0 | Continuous conversion mode (reactivation not allowed during operation) |
| 1 | 1 | Stop conversion mode (reactivation not allowed during operation) |

R/W : Read/write enabled
- : Undefined bit
░ : Initial value

Table 17.4-2 Function Description of Each Bit of A/D Control Status Register 0 (ADCS0)

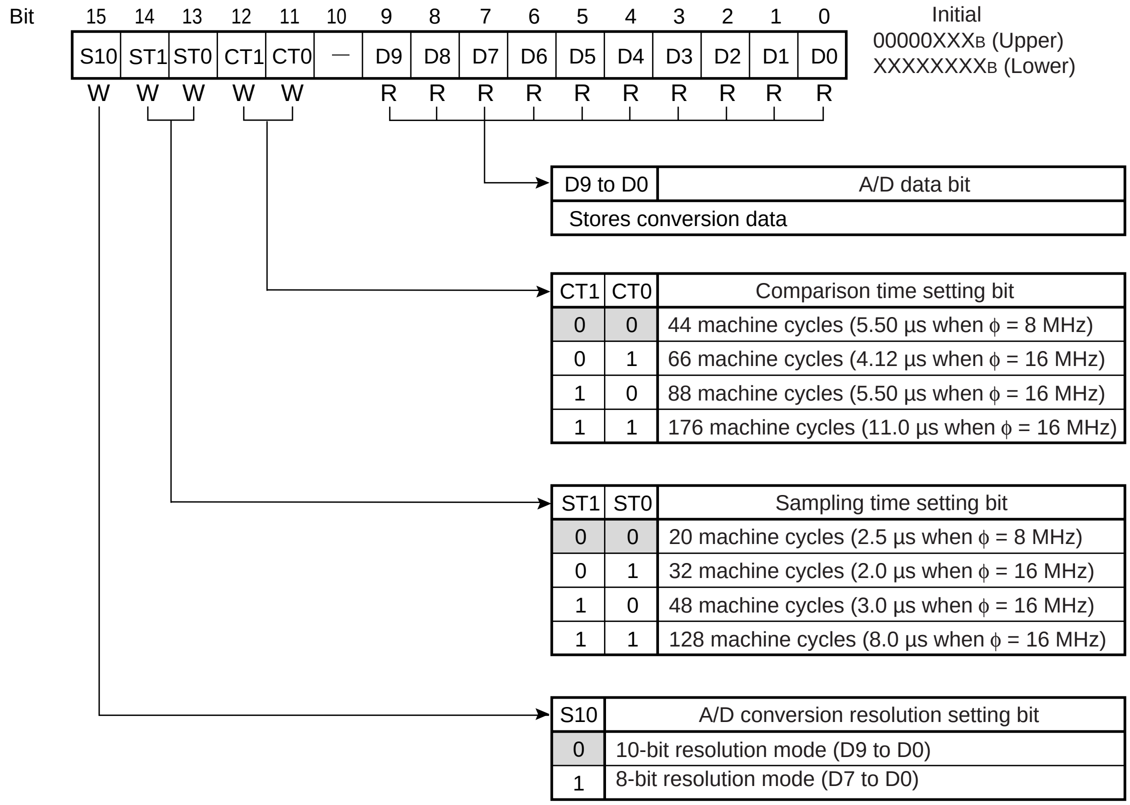
| Bit name | | Function |
|---|---|---|
| bit7<br>bit6 | MD1, MD0:<br>A/D conversion mode setting bit | • This bit is used to set the A/D conversion mode.<br>• Single conversion mode 1, single conversion mode 2, continuous conversion mode, and stop conversion mode can be set.<br>**Single conversion mode 1:**<br>A/D conversion is performed from the channel specified by the A/D conversion start channel setting bits (ANS3 to ANS0) to that specified by the A/D conversion end channel setting bits (ANE3 to ANE0) before terminating. Reactivation during operation is allowed.<br>**Single conversion mode 2:**<br>A/D conversion is performed from the channel specified by the A/D conversion start channel setting bits (ANS3 to ANS0) to that specified by the A/D conversion end channel setting bits (ANE3 to ANE0) before terminating. Reactivation during operation is not allowed.<br>**Continuous conversion mode:**<br>A/D conversion is performed from the channel specified by the A/D conversion start channel setting bits (ANS3 to ANS0) to that specified by the A/D conversion end channel setting bits (ANE3 to ANE0) is repeated until the conversion stop is forcibly implemented by the Converting bit (BUSY). Reactivation during operation is not allowed.<br>**Stop conversion mode:**<br>A/D conversion is performed from the channel specified by the A/D conversion start channel setting bits (ANS3 to ANS0) to that specified by the A/D conversion end channel setting bits (ANE3 to ANE0) is repeated by making a pause for each channel until the conversion stop is forcibly implemented by the Converting bit (BUSY). Reactivation during operation is not allowed. Reactivation during the pause depends on the activation trigger set in the A/D activation trigger setting bits (STS1, STS0).<br>**Note:**<br>The impossibility of reactivation of each conversion mode (simple, continuous, and stop) is applied to the 16-bit free-running timer 0 detection, 16-bit reload timer1, and activation of all software. |

## 17.4.3 A/D Data Register (ADCR0/ADCR1)

**The A/D data register (ADCR0/ADCR1) stores the results of A/D conversion. It also sets the resolution for A/D conversion, sampling time during A/D conversion, and compare time during A/D conversion.**

### ■ A/D Data Register (ADCR0/ADCR1)

**Figure 17.4-4 A/D Data Register (ADCR0/ADCR1)**



| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | S10 | ST1 | ST0 | CT1 | CT0 | — | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| | W | W | W | W | W | | R | R | R | R | R | R | R | R | R | R |

Initial
00000XXXB (Upper)
XXXXXXXXB (Lower)

| D9 to D0 | A/D data bit |
|---|---|
| | Stores conversion data |

| CT1 | CT0 | Comparison time setting bit |
|---|---|---|
| 0 | 0 | 44 machine cycles (5.50 µs when ϕ = 8 MHz) |
| 0 | 1 | 66 machine cycles (4.12 µs when ϕ = 16 MHz) |
| 1 | 0 | 88 machine cycles (5.50 µs when ϕ = 16 MHz) |
| 1 | 1 | 176 machine cycles (11.0 µs when ϕ = 16 MHz) |

| ST1 | ST0 | Sampling time setting bit |
|---|---|---|
| 0 | 0 | 20 machine cycles (2.5 µs when ϕ = 8 MHz) |
| 0 | 1 | 32 machine cycles (2.0 µs when ϕ = 16 MHz) |
| 1 | 0 | 48 machine cycles (3.0 µs when ϕ = 16 MHz) |
| 1 | 1 | 128 machine cycles (8.0 µs when ϕ = 16 MHz) |

| S10 | A/D conversion resolution setting bit |
|---|---|
| 0 | 10-bit resolution mode (D9 to D0) |
| 1 | 8-bit resolution mode (D7 to D0) |

| | | |
|---|---|---|
| R/W | : | Read-only |
| W | : | Write-only |
| X | : | Undefined |
| - | : | Undefined bit |
| ▨ | : | Initial value |
| ϕ | : | Machine clock frequency |

**Table 17.4-3 Function Description of Each Bit of A/D Control Status Register (ADCR0/ADCR1)**

| Bit name | | Function |
|---|---|---|
| bit15 | S10:<br>A/D conversion resolution setting bit | • This bit is used to set the resolution for A/D conversion.<br>• When this bit is "0", the 10-bit resolution is set.<br>• When this bit is "1", the 8-bit resolution is set.<br>**Note:**<br>A/D data bits to be used depend on the resolution. In 10-bit resolution mode, the D9 to D0 bits are used. In 8-bit resolution mode, the D7 to D0 bits are used. |
| bit14<br>bit13 | ST1, ST0: Sampling time setting bit | • This bit is used to set the sampling time for A/D conversion.<br>• When A/D conversion is activated, the analog input is captured for the time interval specified by the sampling time setting bits (ST1, ST0).<br>**Note:**<br>• When "$00_B$" is set, the machine clock frequency should be equal to or less than 8 MHz.<br>• If "$00_B$" is set when the machine clock frequency is 16 MHz, normal analog conversion values may not be obtained. |
| bit12<br>bit11 | CT1, CT0:<br>Comparison time setting bit | • This bit is used to set the compare time for A/D conversion.<br>• The A/D conversion result is determined after the analog input is captured (sampling time passed) and the compare time interval specified by the compare time setting bits (CT1, CT0). In 10-bit resolution mode, the result is stored in the A/D data bits (D9 to D0). In 8-bit resolution mode, the result is stored in the A/D data bits (D7 to D0).<br>**Note:**<br>• When "$00_B$" is set, the machine clock frequency should be equal to or less than 8 MHz.<br>• If "$00_B$" is set when the machine clock frequency is 16 MHz, normal analog conversion values may not be obtained. |
| vit10 | -:<br>Undefined bit | • The value read from this bit is undefined.<br>• The value set to this bit does not affect operation. |
| bit9 - bit0 | D9 - D0:<br>A/D data bit | • These bits are used to store A/D conversion results. They are rewritten after each A/D conversion is completed.<br>• Normally, the final conversion value is stored.<br>• The initial value is undefined.<br>**Note:**<br>The A/D conversion data protection function is available (For details, see Section 17.6 "Operation of the 8/10-Bit A/D Converter").<br>Do not write data to the A/D data bits during A/D conversion. |

**Notes:**

- Be sure to stop the A/D conversion before rewriting the A/D conversion resolution setting bit (S10). If the bit is rewritten after the A/D conversion started, the A/D data register (ADCR0/ADCR1) contents are undefined.
- To read the A/D data register (ADCR0/ADCR1), be sure to use the word transfer instructions (such as MOVW A and $002E_H$) when 10-bit resolution mode is specified.

## 17.4.4 A/D Conversion Channel Select Register (ADMR)

**The A/D Conversion Channel Select Register (ADMR) selects an A/D conversion channel.**

### ■ A/D Conversion Channel Select Register (ADMR)

**Figure 17.4-5 A/D Conversion Channel Select Register (ADMR)**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | ANS3 | ANS2 | ANS1 | ANS0 | ANE3 | ANE2 | ANE1 | ANE0 |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

Initial value
0000000$_B$

| ANE3 | ANE2 | ANE1 | ANE0 | A/D conversion end channel selection bit |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | AN0 |
| 0 | 0 | 0 | 1 | AN1 |
| 0 | 0 | 1 | 0 | AN2 |
| 0 | 0 | 1 | 1 | AN3 |
| 0 | 1 | 0 | 0 | AN4 |
| 0 | 1 | 0 | 1 | AN5 |
| 0 | 1 | 1 | 0 | AN6 |
| 0 | 1 | 1 | 1 | AN7 |
| 1 | 0 | 0 | 0 | AN8 |
| 1 | 0 | 0 | 1 | AN9 |
| 1 | 0 | 1 | 0 | AN10 |
| 1 | 0 | 1 | 1 | AN11 |
| 1 | 1 | 0 | 0 | AN12 |
| 1 | 1 | 0 | 1 | AN13 |
| 1 | 1 | 1 | 0 | AN14 |
| 1 | 1 | 1 | 1 | AN15 |

| ANS3 | ANS2 | ANS1 | ANS0 | A/D conversion start channel selection bit | | |
|---|---|---|---|---|---|---|
| | | | | Stopped | Reading during conversion | Reading during temporary stop in stop conversion mode |
| 0 | 0 | 0 | 0 | AN0 | Channel number being converted | Channel number converted just before halting |
| 0 | 0 | 0 | 1 | AN1 | | |
| 0 | 0 | 1 | 0 | AN2 | | |
| 0 | 0 | 1 | 1 | AN3 | | |
| 0 | 1 | 0 | 0 | AN4 | | |
| 0 | 1 | 0 | 1 | AN5 | | |
| 0 | 1 | 1 | 0 | AN6 | | |
| 0 | 1 | 1 | 1 | AN7 | | |
| 1 | 0 | 0 | 0 | AN8 | | |
| 1 | 0 | 0 | 1 | AN9 | | |
| 1 | 0 | 1 | 0 | AN10 | | |
| 1 | 0 | 1 | 1 | AN11 | | |
| 1 | 1 | 0 | 0 | AN12 | | |
| 1 | 1 | 0 | 1 | AN13 | | |
| 1 | 1 | 1 | 0 | AN14 | | |
| 1 | 1 | 1 | 1 | AN15 | | |

R/W : Read/write enabled
▨ : Initial value

**Table 17.4-4 Functions of the A/D Conversion Channel Selection Register (ADMR) Bits**

| Bit name | | Function |
|---|---|---|
| bit7<br>bit6<br>bit5<br>bit4 | ANS3, ANS2, ANS1, ANS0: A/D conversion start channel selection bits | • These bits are used to select an A/D conversion start channel and check a channel number being converted.<br>• A/D conversion starts from the channel selected via the A/D conversion start channel selection bits (ANS3 to ANS0).<br>• During A/D conversion, a channel number that is being converted is read. During temporary stop in stop conversion mode, a channel number converted just before the temporary stop is read. |
| bit3<br>bit2<br>bit1<br>bit0 | ANE3, ANE2, ANE1, ANE0: A/D conversion end channel selection bits | • These bits are used to select an A/D conversion end channel.<br>• A/D conversion ends with the channel selected by the A/D conversion end channel selection bits (ANE3 to ANE0).<br>• If the same channel is selected both by the A/D conversion start channel selection bits (ANS3 to ANS0) and by the A/D conversion end channel selection bits, A/D conversion is performed on the specified channel.<br>• If continuous conversion mode or stop conversion mode is specified, A/D conversion ends with the channel selected by the A/D conversion end channel selection bits (ANE3 to ANE0) and conversion is then repeated starting with the start channel selected by the A/D conversion start channel selection bits (ANS3 to ANS0). If the value specifying the start channel is greater than the value specifying the end channel, A/D conversion is performed on the start channel through AN15 and then on AN0 through the end channel, completing the first phase of conversion operation. |

# 17.5 8/10-Bit A/D Converter Interrupts

**The 8/10-bit A/D converter can generate an interrupt request when the data for the A/D conversion is set in the A/D data register. This function supports the extended intelligent I/O service (EI$^2$OS).**

## ■ 8/10-bit A/D Converter Interrupts

Table 17.5-1 Interrupt Control Bits of the 8/10-Bit A/D Converter and the Interrupt Cause

| | 8/10-bit A/D converter |
|---|---|
| Interrupt request flag bit | ADCS: INT="1" |
| Interrupt request enable bit | ADCS: INTE="1" |
| Interrupt cause | Writing the A/D conversion result to the A/D data register |

When A/D conversion is started and the A/D conversion result is stored in the A/D data register (ADCR0 and ADCR1), the interrupt request flag bit (INT) of the A/D control status register (ADCS1) is set to "1". If the interrupt request enable bit (INTE) has been set to "1", an interrupt request is output to the CPU.

## ■ 8/10-Bit A/D Converter Interrupts and EI$^2$OS

Table 17.5-2 8/10-Bit A/D Converter Interrupts and EI$^2$OS

| Interrupt No. | Interrupt control register | | Vector table address | | | EI$^2$OS |
|---|---|---|---|---|---|---|
| | Register name | Address | Lower | Upper | Bank | |
| #37 ($25_H$) | ICR13 | $0000BD_H$ | $FFFF68_H$ | $FFFF69_H$ | $FFFF6A_H$ | o |

o: Available

## ■ EI$^2$OS Function of the 8/10-Bit A/D Converter

The 10-bit A/D converter can transfer A/D conversion results to memory using the EI$^2$OS function. When the EI$^2$OS function is used, the conversion data protection function is activated to temporarily stop A/D conversion until A/D conversion data has been transferred to memory and the interrupt request flag bit (INT) of the A/D control status register (ADCS1) is cleared to "0". This function is useful for preventing the omission of data.

# 17.6 Operation of the 8/10-Bit A/D Converter

**The 8/10-bit A/D converter has four conversion modes: single conversion mode 1, single conversion mode 2, continuous conversion mode, and stop conversion mode. This section explains each of these modes.**

## ■ Operation in Single Conversion Mode

In single conversion mode, analog input specified by the ANS and ANE bits is converted in sequence. A/D conversion ends when it is performed on an end channel specified by the ANE bits. If the start channel and the end channel are the same (ANS=ANE), only one channel specified by the ANS bits is converted. To use single conversion mode, make settings as shown in Figure 17.6-1 "Settings for Single Conversion Mode".

**Figure 17.6-1 Settings for Single Conversion Mode**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADCS0/ADCS1 | BUSY | INT | INTE | PAUS | STS1 | STS0 | STRT | RESV | MD1 | MD0 | — | — | — | — | — | — |
|  | ◇ | ◇ | ◇ | ◇ | ◇ | ◇ | ◇ | 0 | ◇ | ◇ |  |  |  |  |  |  |
| ADCR0/ADCR1 | S10 | ST1 | ST0 | CT1 | CT0 | — | Stores conversion data |  |  |  |  |  |  |  |  |  |
|  | ◇ | ◇ | ◇ | ◇ | ◇ |  |  |  |  |  |  |  |  |  |  |  |
| ADMR | ANS3 | ANS2 | ANS1 | ANS0 | ANE3 | ANE2 | ANE1 | ANE0 |  |  |  |  |  |  |  |  |
|  | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ |  |  |  |  |  |  |  |  |

◇ : Used
◆ : Set to 1 the bit corresponding to the pin used.
0 : Set 0.

The following are sample conversion sequences in single conversion mode:

ANS = $0000_B$, ANE = $0011_B$:AN0 --> AN1 --> AN2 --> AN3 --> End

ANS = $1110_B$, ANE = $0010_B$:AN14 --> AN15 --> AN0 --> AN1 --> AN2 --> End

ANS = $0011_B$, ANE = $0011_B$:AN3 --> END

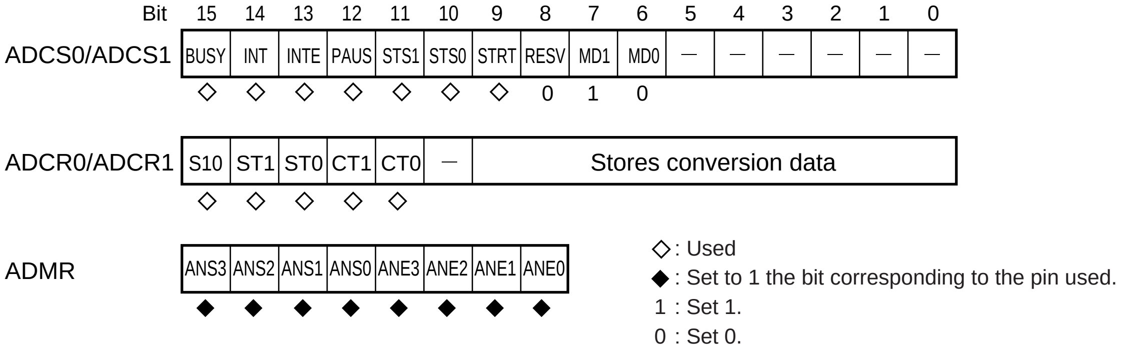
## ■ Operation in Continuous Conversion Mode

In continuous conversion mode, analog input from the start channel specified by the A/D conversion start channel setting bits (ANS3 to ANS0) of the A/D control status register (ADCS0) to the end channel specified by the A/D conversion end channel setting bits (ANE3 to ANE0) is A/D converted to return to the analog input specified by the A/D conversion start channel setting bits (ANS3 to ANS0) to repeat the A/D conversion.

If the start channel and the end channel are the same, the A/D conversion of the channel specified by the A/D conversion start channel setting bits (ANS3 to ANS0) is repeated.

The A/D conversion does not stop until the Converting bit (BUSY) of the A/D control status register (ADCS1) is set to "0". Reactivation during operation is not possible. For operation in continuous conversion mode, the settings shown in Figure 17.6-2 "Settings for Continuous Conversion Mode" are required.

**Figure 17.6-2 Settings for Continuous Conversion Mode**


Bit 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ADCS0/ADCS1
BUSY INT INTE PAUS STS1 STS0 STRT RESV MD1 MD0 — — — — — —
◇ ◇ ◇ ◇ ◇ ◇ ◇ 0 1 0
ADCR0/ADCR1
S10 ST1 ST0 CT1 CT0 —
Stores conversion data
◇ ◇ ◇ ◇ ◇
ADMR
ANS3 ANS2 ANS1 ANS0 ANE3 ANE2 ANE1 ANE0
◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆
◇ : Used
◆ : Set to 1 the bit corresponding to the pin used.
1 : Set 1.
0 : Set 0.

The following are sample conversion sequences in continuous conversion mode:

ANS = $0000_B$, ANE = $0011_B$:AN0 --> AN1 --> AN2--> AN3--> AN0--> Repeat

ANS = $1110_B$, ANE = $0010_B$:AN14 --> AN15 --> AN0--> AN1--> AN2 --> AN14 --> Repeat

ANS = $0011_B$, ANE = $0011_B$:AN3 --> AN3 --> Repeat

## ■ Operation in Stop Conversion Mode

In stop conversion mode, analog input from the start channel specified by the A/D conversion start channel setting bits (ANS3 to ANS0) of the A/D control status register (ADCS0) to the end channel specified by the A/D conversion end channel setting bits (ANE3 to ANE0) is A/D converted by making a pause for each channel before returning to the analog input specified by the A/D conversion start channel setting bits (ANS3 to ANS0) to repeat the A/D conversion and pause.

If the start channel and the end channel are the same, the A/D conversion of the channel specified by the A/D conversion start channel setting bits (ANS3 to ANS0) is repeated.

In the pause state, reactivation method of the A/D conversion depends on the activation cause specified in the activation cause set bit (STS1, STS0) in the A/D control status register (ADCS1).

The A/D conversion does not stop until the Converting bit (BUSY) of the A/D control status register (ADCS1) is set to "0". Reactivation during operation is not possible. For operation in stop conversion mode, the settings shown in Figure 17.6-3 "Settings for Stop Conversion Mode" are required.

**Figure 17.6-3 Settings for Stop Conversion Mode**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADCS | BUSY | INT | INTE | PAUS | STS1 | STS0 | STRT | RESV | MD1 | MD0 | — | — | — | — | — | — |
| | ◇ | ◇ | ◇ | ◇ | ◇ | ◇ | ◇ | 0 | 1 | 1 | | | | | | |
| ADCR | S10 | ST1 | ST0 | CT1 | CT0 | — | Stores conversion data | | | | | | | | | |
| | ◇ | ◇ | ◇ | ◇ | ◇ | | | | | | | | | | | |
| ADMR | ANS3 | ANS2 | ANS1 | ANS0 | ANE3 | ANE2 | ANE1 | ANE0 | | | | | | | | |
| | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | | | | | | | | |

◇ : Used
◆ : Set to 1 the bit corresponding to the pin used.
1 : Set 1.
0 : Set 0.

The following are sample conversion sequences in stop conversion mode:

ANS = $0000_B$, ANE = $0011_B$:

AN0 --> Pause --> AN1 --> Pause --> AN2 --> Pause --> AN3 --> Pause --> AN0 --> Repeat

ANS = $1100_B$, ANE = $0001_B$:

AN14 --> Pause --> AN15 --> Pause --> AN0 --> Pause --> AN1 --> Pause --> AN14 --> Repeat

ANS = $0011_B$, ANE = $0011_B$:

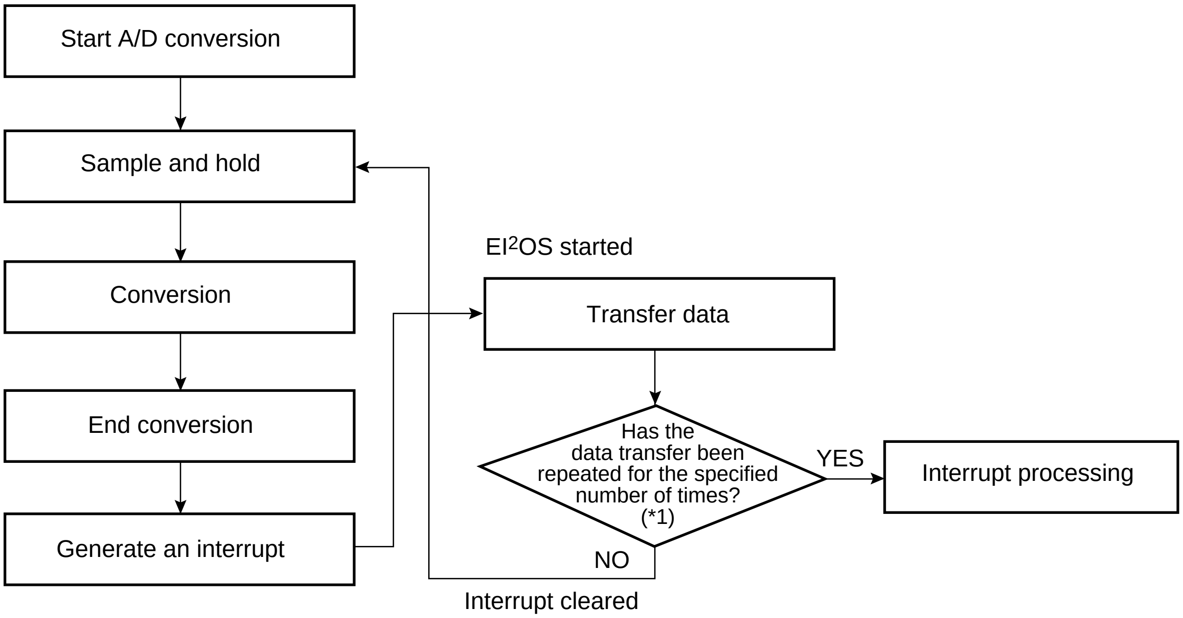
AN3 --> Pause --> AN3 --> Pause --> Repeat

## 17.6.1 Conversion Using EI$^2$OS

**The 8/10-bit A/D converter can use EI$^2$OS transfer the A/D conversion result to memory.**

### ■ Conversion Using EI$^2$OS

**Figure 17.6-4 Sample Operation Flowchart When EI$^2$OS is Used**



*1 The number of times is determined by an EI$^2$OS setting.

When EI$^2$OS is used, the conversion data protection function prevents any part of the data from being lost even in continuous conversion. Multiple data items can be safely transferred to memory.

## 17.6.2 A/D Conversion Data Protection Function

**When A/D conversion is performed in the interrupt enabled state, the conversion data protection function operates.**

### ■ A/D Conversion Data Protection Function

The 8/10-bit A/D converter has only one data register for conversion data storage. Thus, when A/D conversion is performed, the data stored in the data register is rewritten when the conversion is completed. In continuous conversion mode, if conversion data is transferred to memory too late, part of the stored data will be missing.

As measures against such data omission, the data protection function works as shown below if an interrupt request is enabled (ADCS1: INTE="1").

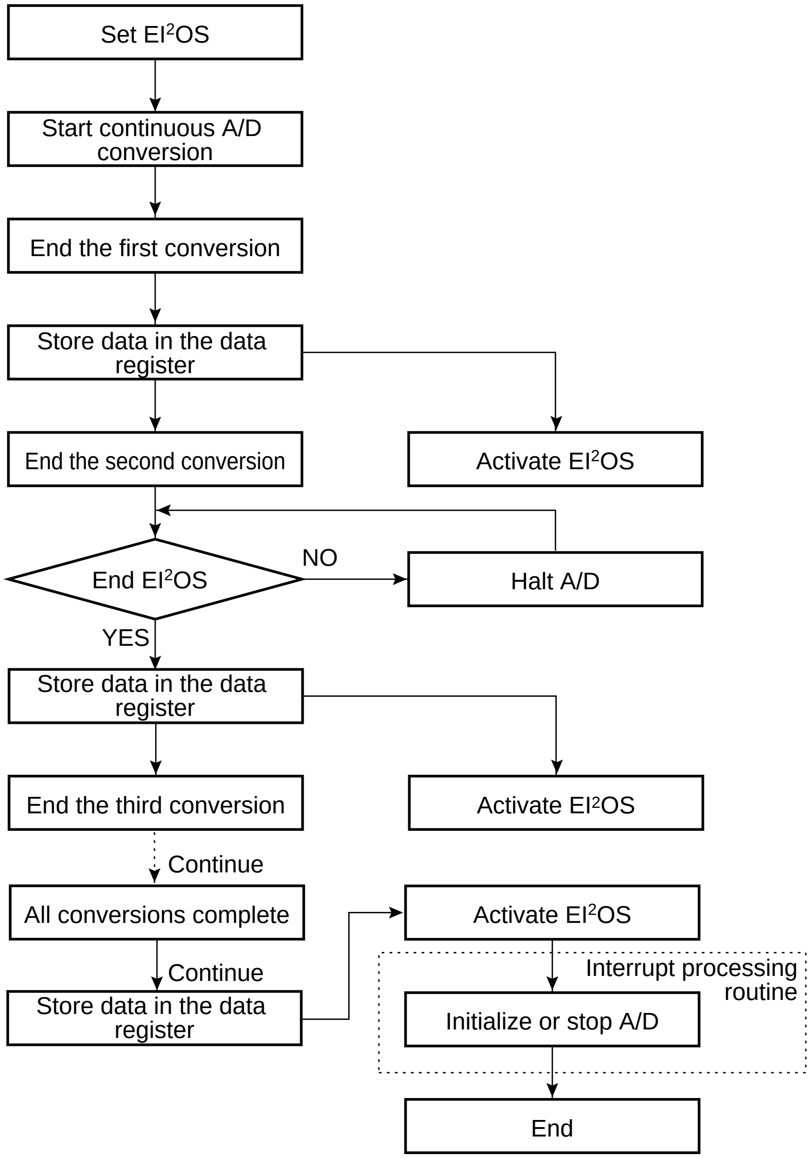
#### ❍ Data protection function when EI$^2$OS is not used

When conversion data is stored in the A/D data register (ADCR0/ADCR1), the interrupt request flag bit (INT) of the A/D control status register 1 (ADCS1) is set to "1" and the A/D conversion is halted. The A/D conversion is restarted after transferring the A/D data register (ADCR0/ ADCR1) values to memory in the interrupt routine and the interrupt request flag bit (INT) is cleared to "0".

#### ❍ Data protection function when EI$^2$OS is used

In continuous conversion mode, the pause flag bit (PAUS) of the A/D control status register 1 (ADCS1) is set to "1" if EI$^2$OS is used and A/D conversion is completed, but the transfer of the last piece of data to memory is not completed so that the A/D conversion is halted and conversion data is not stored in the A/D data register (ADCR0/ADCR1). When the transfer of the last piece of data to memory is completed, the pause flag bit (PAUS) is cleared to "0" and the A/D conversion is restarted.

**Figure 17.6-5 Flow of the Data Protection Function when EI$^2$OS is Used**



<Caution> The steps followed while the A/D converter is stopped are omitted.

**Notes:**

- The conversion data protection function operates only in the interrupt enabled state (ADCS1: INTE = 1).
- If interrupts are disabled during a pause of A/D conversion while EI$^2$OS is operating, the A/D conversion may be reactivated. This will cause new data to be written before the old data is transferred.
- Reactivation attempted during a pause will destroy the standby data.

# 17.7 Usage Notes on the 8/10-Bit A/D Converter

**Notes on using the 8/10-bit A/D converter.**

## ■ Usage Notes on the 8/10-Bit A/D Converter

### ❍ Analog input pin

The analog input pins of the A/D converter are used also as the I/O pins of ports A and B. Use these pins by switching the port direction registers (DDRA and DDRB) and the analog input enable registers (ADER0 and ADER1). To use a pin as an analog input pin of the A/D converter, set the corresponding bit (bit 7 to bit 0) of the port direction registers (DDRA and DDRB) to "0" (set it as an input port) and then set the corresponding bit (bit 7 to bit 0 and bit 15 to bit 8) of the analog input enable registers 0 and 1 (ADER0 and ADER1) to "1" to permanently select the input gate on the port side. In port I/O mode (bit 7 to bit 0 of ADER0 and ADER1 ="0" and bit7 to bit0="0"), the input of an intermediate-level signal causes an input leakage current to flow through the gate.

### ❍ Note on using an internal timer

To activate the A/D converter with the internal timer, set the A/D activation trigger setting bits (STS1, STS0) of the A/D control status register (ADCS1). Set the internal timer to the inactive level ("L" for the internal clock). If the internal clock remains at the active level and data is written to the A/D control status register (ADCS0/ADCS1), the A/D converter may be activated.

### ❍ Sequence of turning on the A/D converter and analog input

Be sure to apply the voltage to the power supply pins ($AV_{CC}$, AVR, and $AV_{SS}$) of the A/D converter and the analog input pins (AN0 to AN15) after turning on the digital power supply ($V_{CC}$). Turn off the digital power supply ($V_{CC}$) after turning off the A/D converter and the analog input power supply. Turn on and turn off the voltage so that AVR does not exceed $AV_{CC}$.

### ❍ Supply voltage to the A/D converter

The supply voltage to the A/D converter ($AV_{CC}$) must not exceed the digital power supply ($V_{CC}$); otherwise, latchup may occur.

# CHAPTER 18 FL CONTROL CIRCUIT

**This chapter explains the functions and operation of the MB90M405 series FL control circuit.**

## 18.1 Overview of FL Control Circuit

**The FL control circuit provides functions for automatic display on fluorescent tube and LED displays.**
**The automatic fluorescent display function provides up to 32 points for displaying digits and up to 60 points for displaying digits and segments.**
**The automatic LED display function can output data at a 1/2-duty cycle from the LED01 to LED16 pins while using the LED00 pin as common output.**

### ■ High Dielectric Output Pins

- 60 high dielectric output pins (FIP0 to FIP59) are provided.
- 34 high-current output pins (FIP0 to FIP33) and 26 medium-current output pins (FIP34 to FIP59) are provided.
- Use of pull-down resistors can be set for all high dielectric outputs as well as for combinations of high dielectric outputs.

### ■ Automatic Fluorescent Tube Display Function

- A display RAM area of 32 x 60 bits is provided.
- The display timing can be specified with a value from 1 to 32.
- For each point of the timing, 60 bits can be used for specifying a combination of digits and segments.
- The digit pins from FIP0 to FIP31 can be consecutively set according to the numbers stored in the digit count register, starting with the pin for which start of display is specified.
- Output control of up to 59 segments is supported.
- Four types of display scan cycles (segment widths) are supported.
- In segment output, digit dimmer control is performed during the two T periods that apply to each digit output, as shown in Figure 18.1-1 "Digit Dimmer Control". Seven steps of adjustment are supported. (Dimming is effective for all digits.)
- The output level of all digits and segments can be inverted from "H" to "L" or "L" to "H".
- Gradation display (use of a segment dimmer) can be applied to segment output with an arbitrary timing. As shown in Figure 18.1-2 "Segment Dimmer Control", control is performed for the two T periods that apply to each segment output.

**Figure 18.1-1 Digit Dimmer Control**

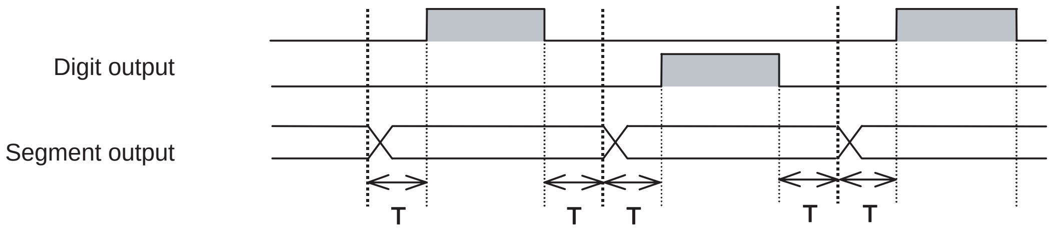
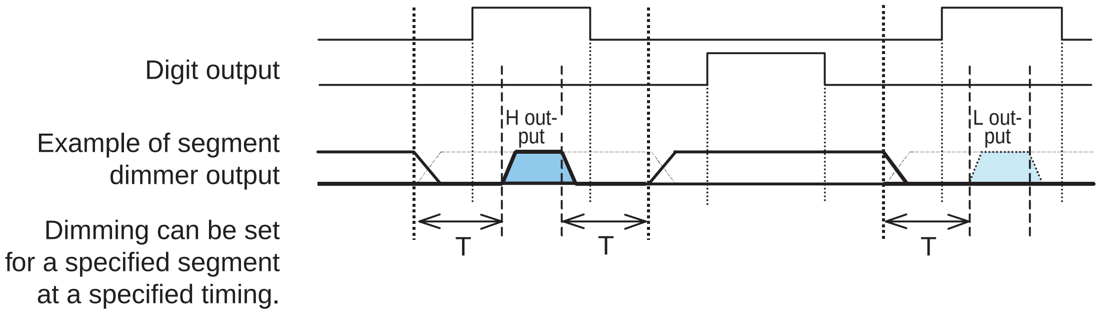


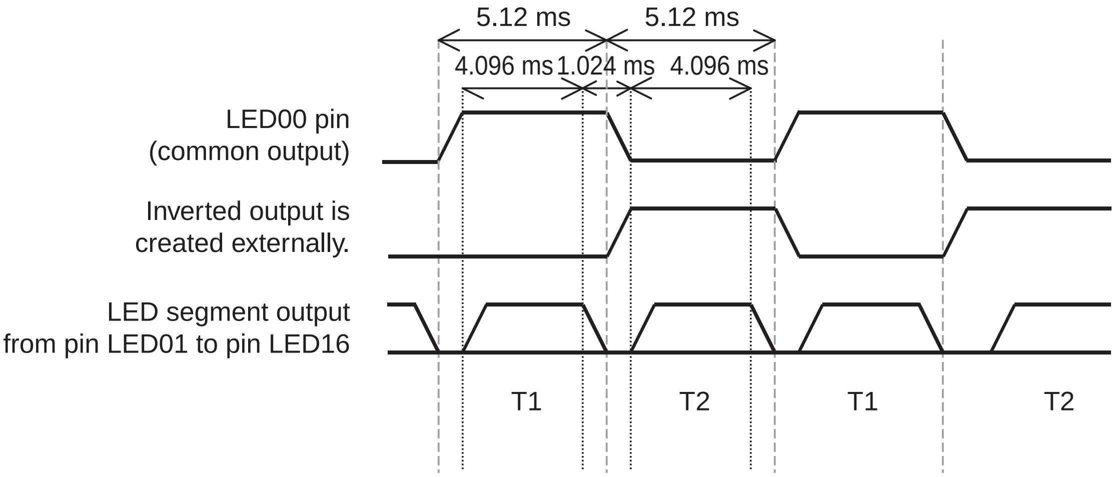
**Figure 18.1-2 Segment Dimmer Control**


Digit output
Example of segment dimmer output
H output
L output
T
T
T
Dimming can be set for a specified segment at a specified timing.

## ■ Automatic LED Display Function

- Any LED pin from LED00 to LED16 can be set, provided it has not already been set.
- As shown in Figure 18.1-3 "Timing Chart of Automatic LED Display", pin LED00 becomes a common pin, while the 16 pins from LED01 to LED16 become LED segment outputs.
- When pin LED00 is at the "H" level, the corresponding value is output to pins LED01 to LED16 at the point "T1" of the timing stored in the display RAM. When pin LED00 is at the "L" level, the corresponding value is output to pins LED01 to LED16 at the point "T2".
- By inverting the output level of common pin LED00 externally, LED output with 1/2-duty cycle can be obtained.
- Figure 18.1-3 "Timing Chart of Automatic LED Display" shows the output time for pins LED01 to LED16 as determined by pin LED00 and its inverting signal. Pin LED00 has an output time of 5.12 ms, and pins LED01 to LED16 have an output time of 4.096 ms each (for a machine clock [peripheral operation clock] frequency of 16 MHz).

**Figure 18.1-3 Timing Chart of Automatic LED Display**


5.12 ms
5.12 ms
4.096 ms
1.024 ms
4.096 ms
LED00 pin (common output)
Inverted output is created externally.
LED segment output from pin LED01 to pin LED16
T1
T2
T1
T2

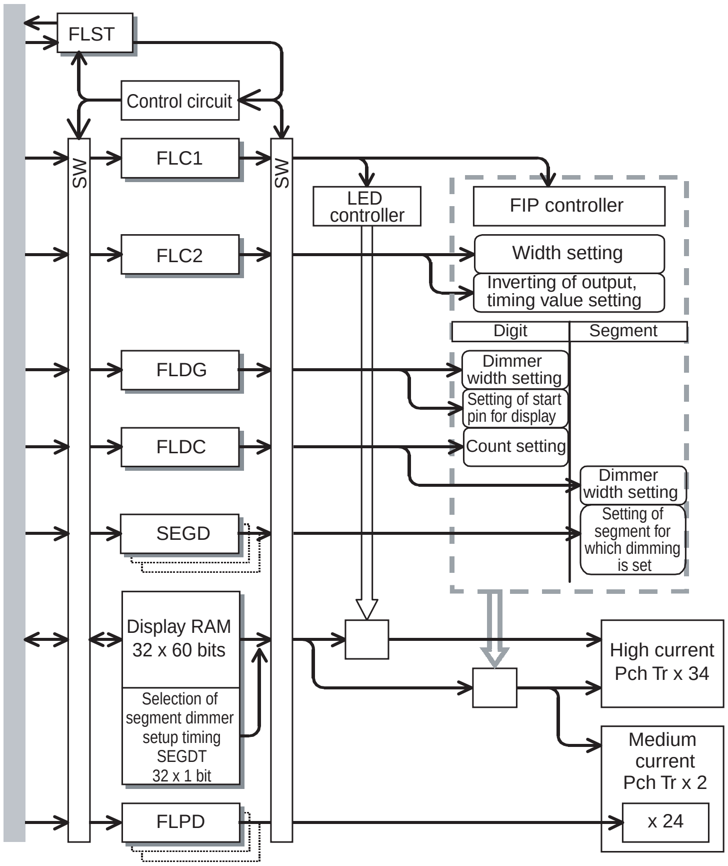
## 18.2 Configuration of FL Control Circuit

**The FL control circuit consists of the following blocks:**

- **Control circuit for automatic fluorescent tube display**
- **Control circuit for automatic LED display**
- **Display RAM**
- **Display control register (FLC1)**
- **Display control register (FLC2)**
- **Digit setting register (FLDG)**
- **Digit count register (FLDC)**
- **Segment dimmer setting register (SEGD0 to SEGD7)**
- **Port register (FLPD0 to FLPD2)**
- **Status/authorization register (FLST)**

### ■ Block Diagram of the FL Control Circuit

**Figure 18.2-1 Block Diagram 1 of the FL Control Circuit**



## 18.3 FL Control Circuit Pins

**This section describes the FL control circuit pins and provides a block diagram.**

### ■ Block Diagram of FL Control Circuit Pins

**Figure 18.3-1 Block Diagram of the FL Control Circuit Pins (FIP00 to FIP16)**

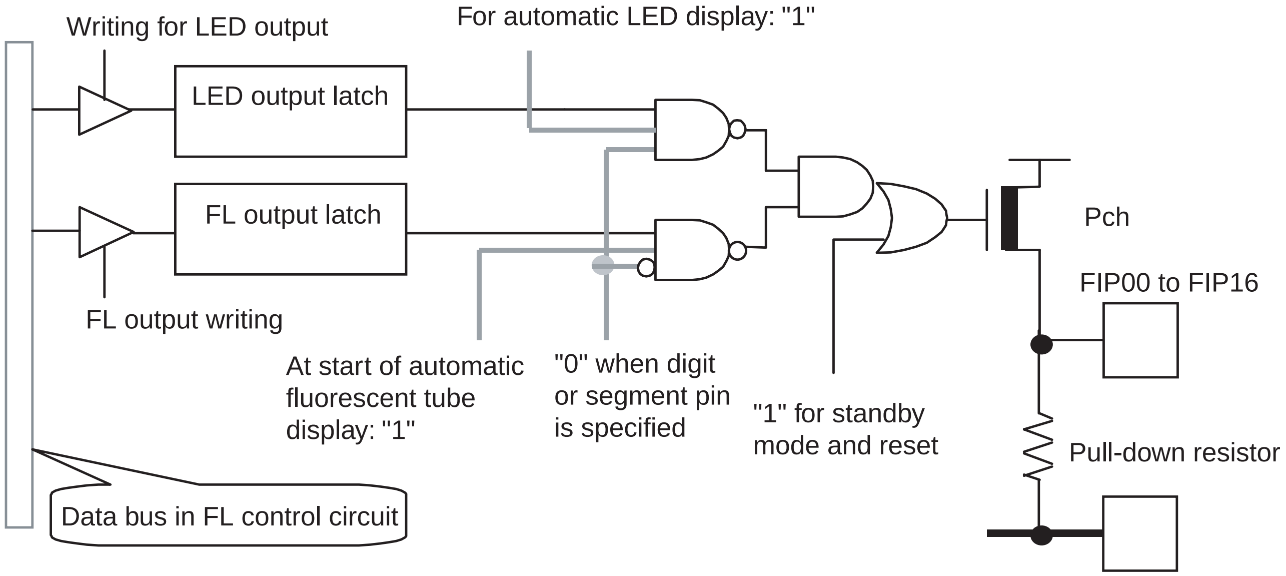

Writing for LED output
For automatic LED display: "1"
LED output latch
FL output latch
Pch
FIP00 to FIP16
FL output writing
At start of automatic fluorescent tube display: "1"
"0" when digit or segment pin is specified
"1" for standby mode and reset
Pull-down resistor
Data bus in FL control circuit

**Figure 18.3-2 Block Diagram of the FL Control Circuit Pins (FIP17 to FIP35)**

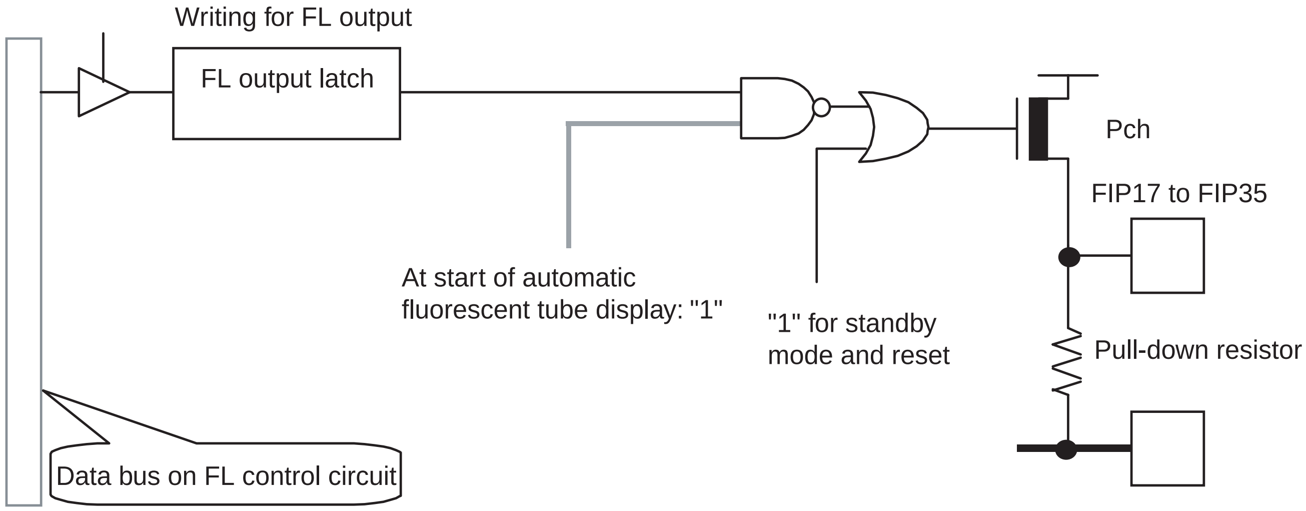
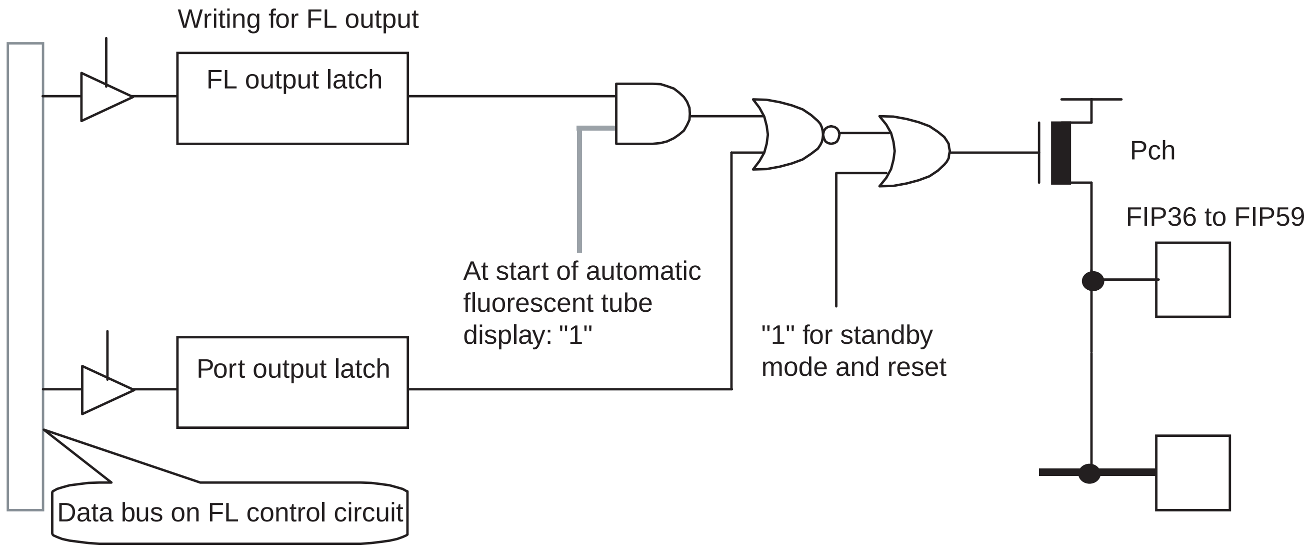

Writing for FL output
FL output latch
Pch
FIP17 to FIP35
At start of automatic fluorescent tube display: "1"
"1" for standby mode and reset
Pull-down resistor
Data bus on FL control circuit

**Figure 18.3-3  Block Diagram of the FL Control Circuit Pins (FIP36 to FIP59)**

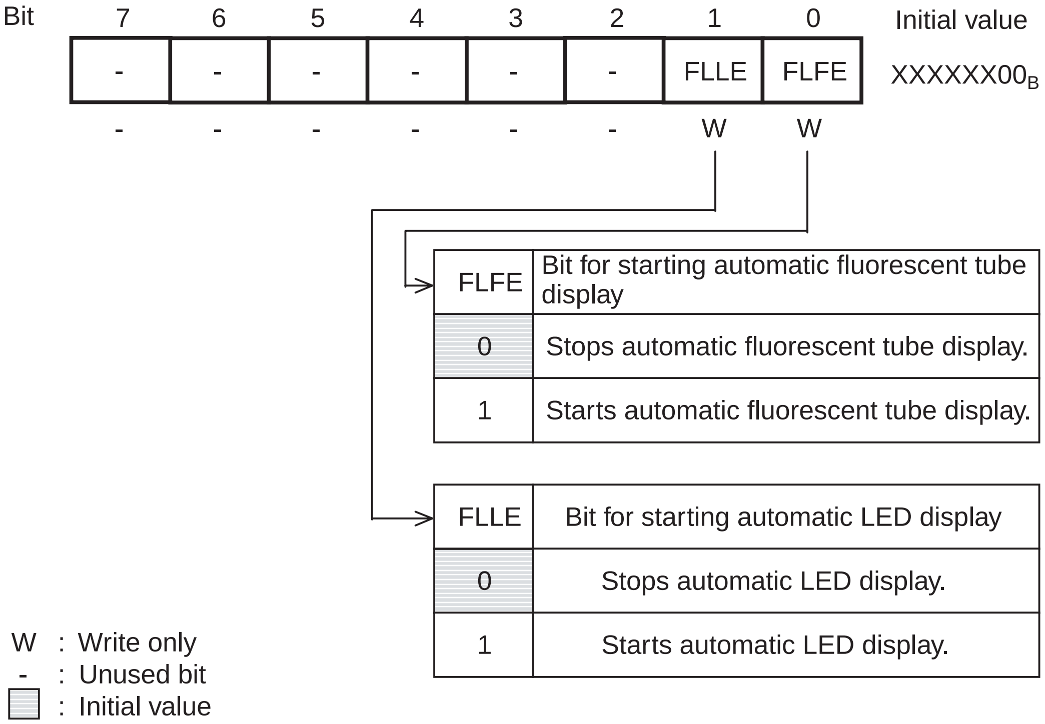

Writing for FL output
FL output latch
Pch
FIP36 to FIP59
At start of automatic fluorescent tube display: "1"
"1" for standby mode and reset
Port output latch
Data bus on FL control circuit

## 18.3.1 Display Control Register 1 (FLC1)

**This register is used to start, stop, and set up the automatic fluorescent tube and LED display.**
**This register is a write-only register and does not support bit operations. Byte access to this register is supported.**

### ■ Display Control Register 1 (FLC1)

**Figure 18.3-4 Display Control Register 1 (FLC1)**



| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | - | - | - | - | - | - | FLLE | FLFE | XXXXXX00$_B$ |
| | - | - | - | - | - | - | W | W | |

| FLFE | Bit for starting automatic fluorescent tube display |
|---|---|
| 0 | Stops automatic fluorescent tube display. |
| 1 | Starts automatic fluorescent tube display. |

| FLLE | Bit for starting automatic LED display |
|---|---|
| 0 | Stops automatic LED display. |
| 1 | Starts automatic LED display. |

W : Write only
- : Unused bit
▨ : Initial value

### Table 18.3-1 Functions of the Display Control Register 1 (FLC1) Bits

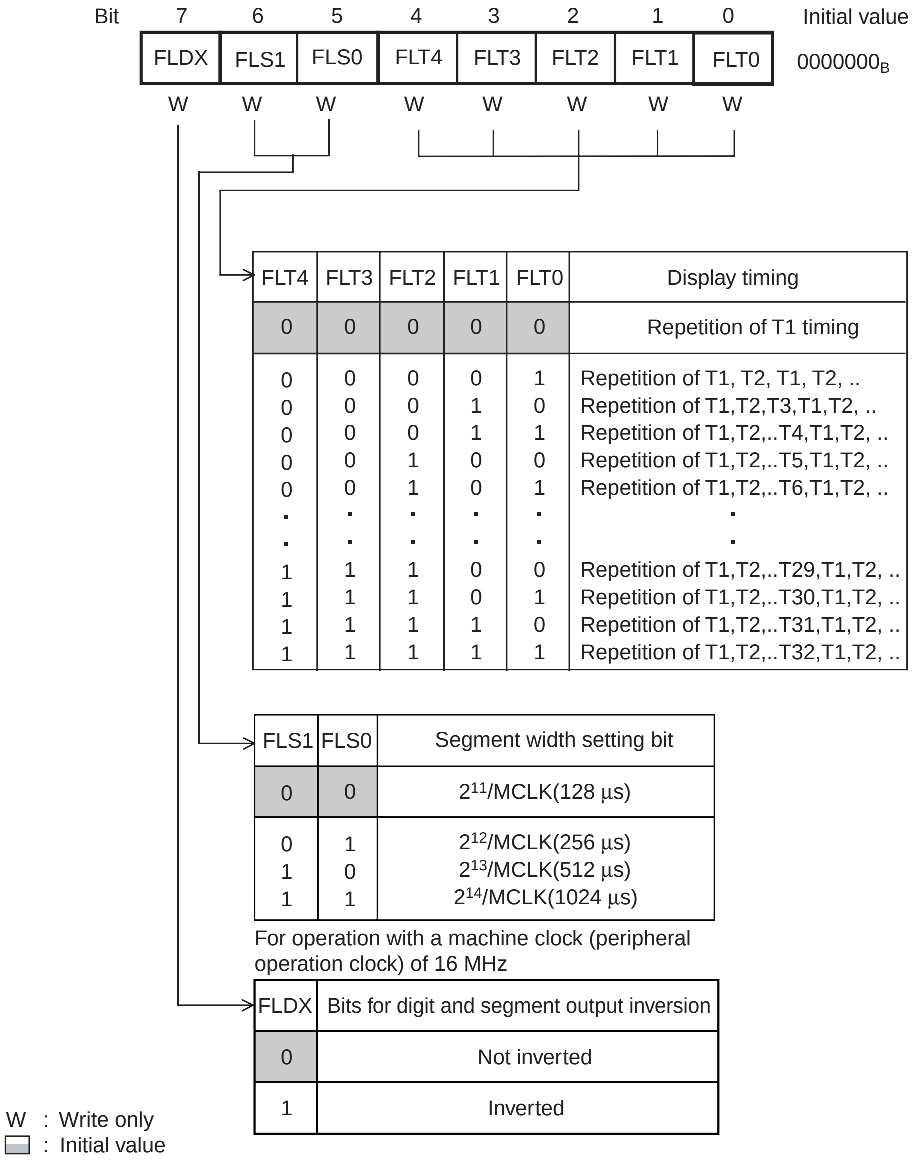
| Bit name | | Function |
|---|---|---|
| Bit 7<br>to<br>Bit 2 | -:<br>Unused bits | • Read value is undefined.<br>• Setting a value has no effect on operation. |
| Bit 1 | FLLE:<br>Bit for starting automatic LED display | • This bit specifies the start of automatic LED display.<br>• When this bit is set to 1, automatic LED display starts.<br>• When this bit is set to 0, automatic LED display stops.<br>• When automatic LED display starts, LED00 outputs the "H" level and the display RAM value for T1 is output.<br>• When automatic LED display stops, all LED outputs are switched off even during display.<br>**Note:**<br>After the required display settings have been made, set display control register 2, the digit setting register, and the digit count register to "1". |
| Bit 0 | FLFE:<br>Bit for starting automatic fluorescent tube display | • This bit specifies the start of automatic fluorescent tube display.<br>• When this bit is set to "1", automatic fluorescent tube display starts.<br>• When this bit is set to "0", automatic fluorescent tube display stops.<br>• When automatic fluorescent tube display starts, output starts with the display RAM value for T1.<br>• When automatic fluorescent tube display stops, the output is set to off after output for the current timing value has ended.<br>**Note:**<br>After the required display settings have been made, set display control register 2, the digit setting register, the digit count register, the display RAM digits, and the segment dimmer (SEGD) to "1". |

## 18.3.2 Display Control Register 2 (FLC2)

**This register is used to set inverted output and to specify a segment width and timing value. This register is a write-only register and does not support bit operations. Byte access to this register is supported.**

### ■ Display Control Register 2 (FLC2)

**Figure 18.3-5 Display Control Register 2 (FLC2)**



| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | FLDX | FLS1 | FLS0 | FLT4 | FLT3 | FLT2 | FLT1 | FLT0 | $0000000_B$ |
| | W | W | W | W | W | W | W | W | |

| FLT4 | FLT3 | FLT2 | FLT1 | FLT0 | Display timing |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | Repetition of T1 timing |
| 0 | 0 | 0 | 0 | 1 | Repetition of T1, T2, T1, T2, .. |
| 0 | 0 | 0 | 1 | 0 | Repetition of T1,T2,T3,T1,T2, .. |
| 0 | 0 | 0 | 1 | 1 | Repetition of T1,T2,..T4,T1,T2, .. |
| 0 | 0 | 1 | 0 | 0 | Repetition of T1,T2,..T5,T1,T2, .. |
| 0 | 0 | 1 | 0 | 1 | Repetition of T1,T2,..T6,T1,T2, .. |
| · | · | · | · | · | · |
| · | · | · | · | · | · |
| 1 | 1 | 1 | 0 | 0 | Repetition of T1,T2,..T29,T1,T2, .. |
| 1 | 1 | 1 | 0 | 1 | Repetition of T1,T2,..T30,T1,T2, .. |
| 1 | 1 | 1 | 1 | 0 | Repetition of T1,T2,..T31,T1,T2, .. |
| 1 | 1 | 1 | 1 | 1 | Repetition of T1,T2,..T32,T1,T2, .. |

| FLS1 | FLS0 | Segment width setting bit |
|---|---|---|
| 0 | 0 | $2^{11}$/MCLK(128 µs) |
| 0 | 1 | $2^{12}$/MCLK(256 µs) |
| 1 | 0 | $2^{13}$/MCLK(512 µs) |
| 1 | 1 | $2^{14}$/MCLK(1024 µs) |

For operation with a machine clock (peripheral operation clock) of 16 MHz

| FLDX | Bits for digit and segment output inversion |
|---|---|
| 0 | Not inverted |
| 1 | Inverted |

W : Write only
▨ : Initial value

**Table 18.3-2 Functions of the Display Control Circuit 2 (FLC2) Bits**

| Bit name | | Function |
|---|---|---|
| Bit 7 | FLDX:<br>Bit for digit and segment output inversion | • When this bit is set to "1", the digit and segment output levels for automatic fluorescent tube display are inverted. (*1)<br>• The output level for automatic LED display is not inverted. |
| Bit 6<br>Bit 5 | FLS1, FLS0:<br>Segment width setting bit | • Sets a segment width. (*1) |
| Bit 4<br>to<br>Bit 0 | FLT4 to FLT0:<br>Display timing setting bit | • Sets a display timing value. (*1) |

*1: Make the settings while the automatic fluorescent tube and LED display are stopped.

## 18.3.3 Digit Setting Register (FLDG)

**This register is used to set the digit dimmer and digit display starting pin.**
**This register is a write-only register and does not support bit operations. Byte access to this register is supported.**

### ■ Digit Setting Register (FLDG)

Figure 18.3-6 Digit Setting Register (FLDG)

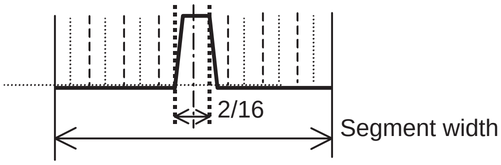
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | FLDM2 | FLDM1 | FLDM0 | FLDS4 | FLDS3 | FLDS2 | FLDS1 | FLDS0 | $00000000_B$ |
| | W | W | W | W | W | W | W | W | |

| FLDS4 | FLDS3 | FLDS2 | FLDS1 | FLDS0 | Bit for specifying the start pin of digit display |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | Starts digit display from FIP00. |
| 0 | 0 | 0 | 0 | 1 | Starts digit display from FIP01. |
| 0 | 0 | 0 | 1 | 0 | Starts digit display from FIP02. |
| 0 | 0 | 0 | 1 | 1 | Starts digit display from FIP03. |
| 0 | 0 | 1 | 0 | 0 | Starts digit display from FIP04. |
| 0 | 0 | 1 | 0 | 1 | Starts digit display from FIP05. |
| · | · | · | · | · | Starts digit display from FIP06. |
| · | · | · | · | · | · |
| 1 | 1 | 1 | 0 | 0 | · |
| 1 | 1 | 1 | 0 | 1 | Starts digit display from FIP29. |
| 1 | 1 | 1 | 1 | 0 | Starts digit display from FIP30. |
| 1 | 1 | 1 | 1 | 1 | Starts digit display from FIP31. |

| FLDM2 | FLDM1 | FLDM0 | Digit dimmer setting bit |
|---|---|---|---|
| 0 | 0 | 0 | 2/16 |
| 0 | 0 | 1 | 4/16 |
| 0 | 1 | 0 | 6/16 |
| 0 | 1 | 1 | 8/16 |
| 1 | 0 | 0 | 10/16 |
| 1 | 0 | 1 | 12/16 |
| 1 | 1 | 0 | 14/16 |

W : Write only
▨ : Initial value

**Table 18.3-3 Functions of the Digit Setting Register (FLDG) Bits**

| Bit name | | Function |
|---|---|---|
| Bit 7<br>to<br>Bit 5 | FLDM2, FLDM1, FLDM0:<br>Digit dimmer setting bits | • These bits specify the digit dimmer width. (*1)<br>• When set to $000_B$ (2/16), the following waveform is generated.<br>• Digit output<br> |
| Bit 4<br>to<br>Bit 0 | FLDS4 to FLDS0:<br>Setting bits for digit display start pin | These bits specify the digit display start pin. (*1) |

*1: Make the settings while the automatic fluorescent tube and LED display is stopped.

# 18.3.4 Digit Count Register (FLDC)

**The digit count register is used to set the segment dimmer and digit pin count. This register is a write-only register and does not support bit operations. Byte access to this register is supported.**

## ■ Digit Count Register (FLDC)

**Figure 18.3-7 Digit Count Register (FLDC)**

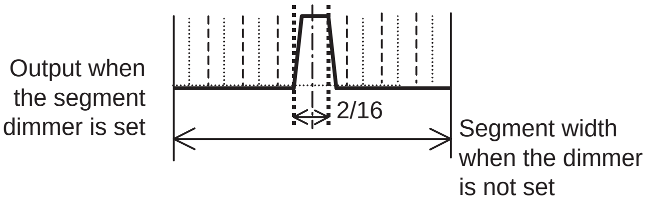
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | FLSD2 | FLSD1 | FLSD0 | FLDC4 | FLDC3 | FLDC2 | FLDC1 | FLDC0 | $00000000_B$ |
| | W | W | W | W | W | W | W | W | |

| FLDC4 | FLDC3 | FLDC2 | FLDC1 | FLDC0 | Digit count setting bit |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | Setting for digit count 1 |
| 0 | 0 | 0 | 0 | 1 | Setting for digit count 2 |
| 0 | 0 | 0 | 1 | 0 | Setting for digit count 3 |
| 0 | 0 | 0 | 1 | 1 | Setting for digit count 4 |
| 0 | 0 | 1 | 0 | 0 | Setting for digit count 5 |
| 0 | 0 | 1 | 0 | 1 | Setting for digit count 6 |
| · | · | · | · | · | · |
| · | · | · | · | · | · |
| 1 | 1 | 1 | 0 | 0 | Setting for digit count 29 |
| 1 | 1 | 1 | 0 | 1 | Setting for digit count 30 |
| 1 | 1 | 1 | 1 | 0 | Setting for digit count 31 |
| 1 | 1 | 1 | 1 | 1 | Setting for digit count 32 |

| FLSD2 | FLSD1 | FLSD0 | Segment dimmer setting bits |
|---|---|---|---|
| 0 | 0 | 0 | 2/16 |
| 0 | 0 | 1 | 4/16 |
| 0 | 1 | 0 | 6/16 |
| 0 | 1 | 1 | 8/16 |
| 1 | 0 | 0 | 10/16 |
| 1 | 0 | 1 | 12/16 |
| 1 | 1 | 0 | 14/16 |

W : Write only
▨ : Initial value

**Table 18.3-4 Functions of the Digit Count Register (FLDC) Bits**

| Bit name | | Function |
|---|---|---|
| Bit 7<br>to<br>Bit 5 | FLSD2 to FLSD0:<br>Segment dimmer setting bits | Holds the settings for the segment dimmer waveform. (*1)<br>SEGDT (bit 7 of address $0E0_H$ to $0FF_H$) in the display RAM specifies the display timing that is to be applied to the segment dimmer. The segment dimmer register (SEGD) specifies segments.<br>When the segment dimmer settings have not been made, set the segment dimmer register (SEGD) or SEGDT to all zeros.<br>When value $000_B$ (2/16) is set, the following waveform is generated.<br> Output when the segment dimmer is set<br>2/16<br>Segment width when the dimmer is not set |
| Bit 4<br>to<br>Bit 0 | FLDC4 to FLDC0:<br>Digit count setting bit | • Specify a digit count. (*1)<br>• The settings of the digit display start bits (bit 0 to bit 4 of FLDG) take priority over the digit count settings. 32 pins (FIP00 to FIP31) can be specified by the digit count value. If the digit display start bit is set for FIP30, the maximum digit count becomes 2, and 2 will be assumed if a digit count of 3 or more is set. |

*1: Make the settings while the automatic fluorescent tube and LED display is stopped.

**Reference:**

The segment dimmer is set only for the segment output at specific times (T01 to T32). This makes it possible to set the dimmer for the segment in a specific digit; in other words, control of dimmer operation for a specific character is supported.

## 18.3.5 Port Register (FLPD)

**The 24 pins from FIP36 to FIP59 can be used as output ports by setting a value in the port register. When a pin is used as an output port, "0" must be set for that pin in the display RAM for all times before automatic fluorescent tube display is started.**
**This register is a write-only register and does not support bit operations. Byte access to this register is supported.**

### ■ Port Register (FLPD)

**Figure 18.3-8 Port Register (FLPD)**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| FLPD0 | FIP43 | FIP42 | FIP41 | FIP40 | FIP39 | FIP38 | FIP37 | FIP36 | $00000000_B$ |
| FLPD1 | FIP51 | FIP50 | FIP49 | FIP48 | FIP47 | FIP46 | FIP45 | FIP44 | $00000000_B$ |
| FLPD2 | FIP59 | FIP58 | FIP57 | FIP56 | FIP55 | FIP54 | FIP53 | FIP52 | $00000000_B$ |
| | W | W | W | W | W | W | W | W | |

W: Write only

### ■ Port Register

❍ **When a pin is used as a port**

Before automatic fluorescent tube display is started, "0" must be set for that pin in the display RAM for all times.

When FLDX (bit for inverted digit/segment output) is set to specify display in reverse video, "1" must be set for that pin in the display RAM for all times before automatic fluorescent tube display is started.

When the port register (FLPD) is set to "0", the Pch high dielectric output is switched off, and the $V_{KK}$ pin voltage is connected via the pull-down resistor.

We recommend the addition of a diode clamping circuit.

When the port register (FLPD) is set to "1", the Pch high dielectric output is switched on.

❍ **When a pin is used for automatic fluorescent tube display**

A "0" must be set in the port register (FLPD) for the pin.

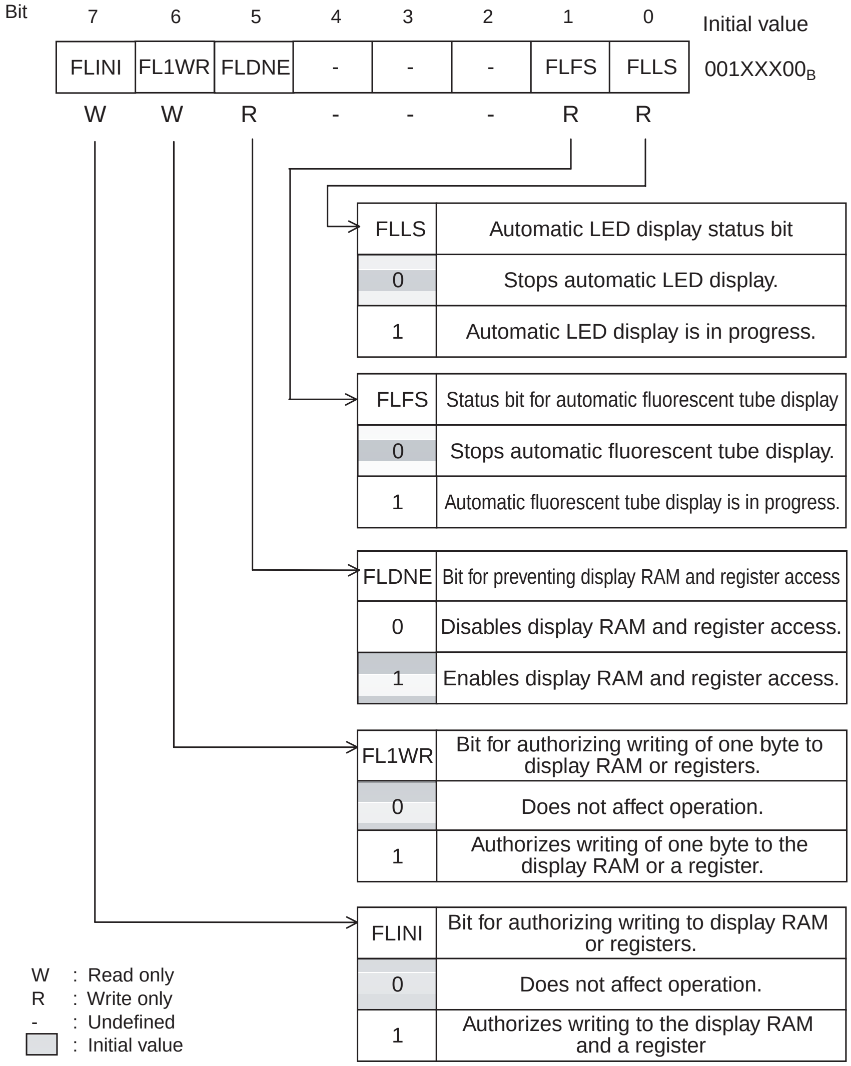
Because the initial value of the port register (FLPD) is 0 at power-on and reset, automatic fluorescent tube display is assumed unless the register is set to another value.

## 18.3.6 Status/Authorization Register (FLST)

**This register includes the following types of bits: Bits for confirming fluorescent tube and automatic LED display, write authorization bits for display RAM and registers (display control register 1, display control register 2, digit setting register, digit count register, and segment dimmer setting register), and bits for preventing access to the display RAM and registers.**
**Bit 7 and bit 6 are write-only, while bit 5, bit 1, and bit 0 are read-only bits. Bit operations are not supported for this register, but byte access is.**

### ■ Status/Authorization Register (FLST)

**Figure 18.3-9 Status/Authorization Register (FLST)**



| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | FLINI | FL1WR | FLDNE | - | - | - | FLFS | FLLS | $001XXX00_B$ |
| | W | W | R | - | - | - | R | R | |

| FLLS | Automatic LED display status bit |
|---|---|
| 0 | Stops automatic LED display. |
| 1 | Automatic LED display is in progress. |

| FLFS | Status bit for automatic fluorescent tube display |
|---|---|
| 0 | Stops automatic fluorescent tube display. |
| 1 | Automatic fluorescent tube display is in progress. |

| FLDNE | Bit for preventing display RAM and register access |
|---|---|
| 0 | Disables display RAM and register access. |
| 1 | Enables display RAM and register access. |

| FL1WR | Bit for authorizing writing of one byte to display RAM or registers. |
|---|---|
| 0 | Does not affect operation. |
| 1 | Authorizes writing of one byte to the display RAM or a register. |

| FLINI | Bit for authorizing writing to display RAM or registers. |
|---|---|
| 0 | Does not affect operation. |
| 1 | Authorizes writing to the display RAM and a register |

| | | |
|---|---|---|
| W | : | Read only |
| R | : | Write only |
| - | : | Undefined |
| ▨ | : | Initial value |

**Table 18.3-5 Functions of the Status/Authorization Register Bits**

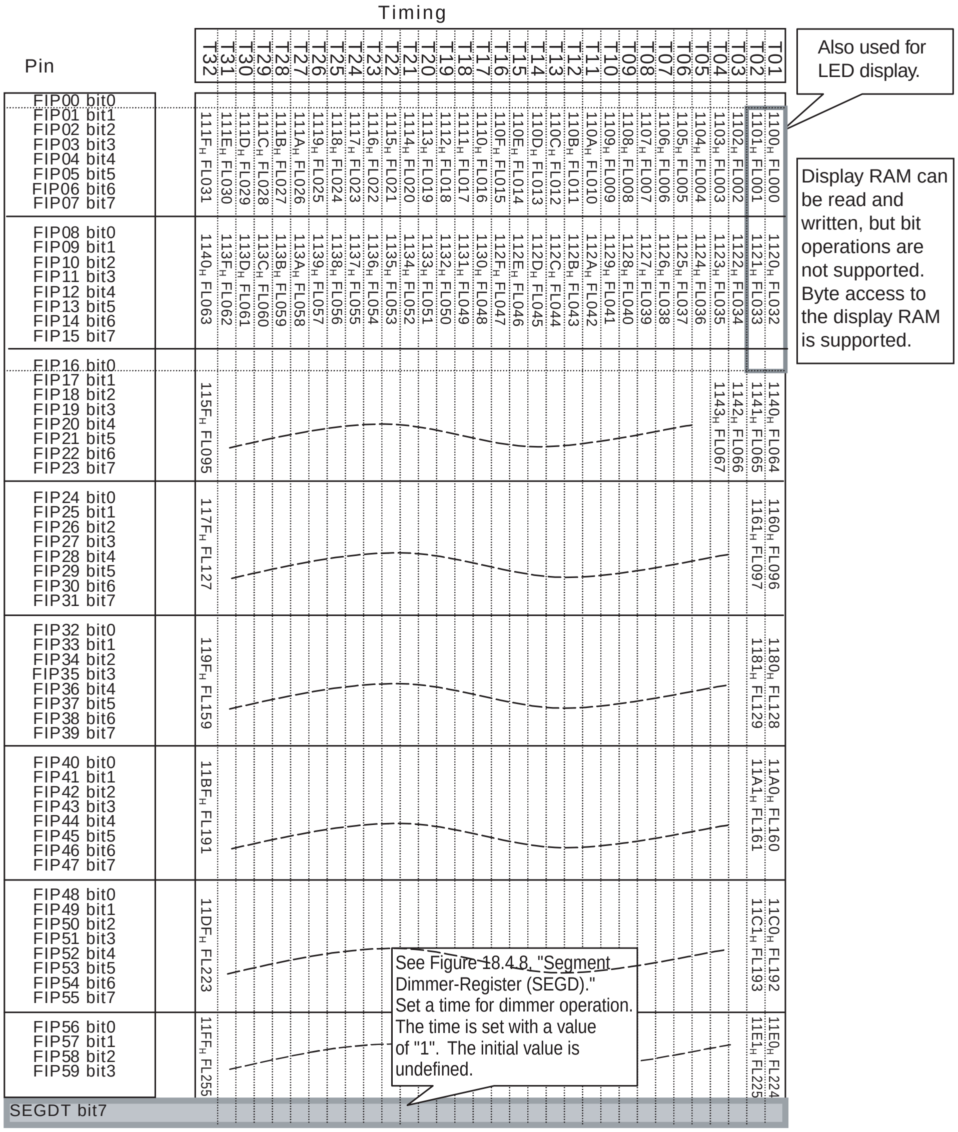
| Bit name | | Function |
|---|---|---|
| Bit 7 | FLINI:<br>Bit for authorizing writing to display RAM or registers | • This bit authorizes writing to the display RAM or a register.<br>• When setting a value in the display RAM or a registers (display control register 1, display control register 2, digit setting register, digit count register, or segment dimmer setting register), the respective value changes after this bit is set to "1".<br>• Setting this bit to "0" has no effect on operation. |
| Bit 6 | FL1WR:<br>Bit for authorizing writing of one byte to display RAM or registers | • This bit authorizes writing of one byte to the display RAM or a register.<br>• When setting a one-byte value in the display RAM or a register (display control register 1, display control register 2, digit setting register, digit count register, or segment dimmer setting register), the respective value changes after this bit is set to "1".<br>• Setting this bit to "0" has no effect on operation.<br>• When FLFS=1 or FLLS=1, writing of any new value to display control register 2, the digit setting register, the digit count register, or the segment dimmer setting register is not authorized. To authorize a write operation for these registers, select FLFE=0 and FLLE=0. |
| Bit 5 | FLDNE:<br>Bit for preventing display RAM and register access | • This bit prevents access to the display RAM and registers.<br>• When this bit is set to "1", reading from or writing to the display RAM and reading from or writing to any of the registers is allowed.<br>• When this bit is set to "0", access to the display RAM and all registers is prohibited (with the exception of reading from the status/authorization register). |
| Bit 4<br>to<br>Bit 2 | -:<br>Undefined | • The value read from this bit is undefined.<br>• Setting this bit has no effect on operation. |
| Bit 1 | FLFS:<br>Status bit for automatic fluorescent tube display | • This bit indicates the status of automatic fluorescent tube display.<br>• When this bit is "1", automatic fluorescent tube display is active.<br>• When this bit is "0", automatic fluorescent tube display is inactive. |
| Bit 0 | FLLS:<br>Automatic LED display status bit | • This bit indicates the status of automatic LED display.<br>• When this bit is "1", automatic LED display is active.<br>• When this bit is "0", automatic LED display is inactive. |

## 18.3.7 Display RAM

**The display RAM contains the settings for digit and segment data based on the timing for automatic fluorescent tube display.**
**Make settings for automatic LED display using the bits for the T01 and TP2 timing LED pins.**
**Specify the time at which the segment dimmer operates in bit 7 of the addresses 1E0$_H$ to 11FF$_{H.}$**

### ■ Display RAM

**Figure 18.3-10 Display RAM**



Also used for LED display.

Display RAM can be read and written, but bit operations are not supported. Byte access to the display RAM is supported.

See Figure 18.4.8, "Segment Dimmer-Register (SEGD)." Set a time for dimmer operation. The time is set with a value of "1". The initial value is undefined.

## 18.3.8 Segment Dimmer Setting Register (SEGD)

**Dimming can be set for any time and any segment of the fluorescent tube.**
**Using the segment dimmer setting register, set a timing for the segment with bit 7 of 11E0$_H$ to 11FF$_H$ in the display RAM.**
**This register is a write-only register and does not support bit operations.**
**Write data to this register while automatic fluorescent tube and LED display are stopped.**
**Byte access is supported for this register.**

### ■ Segment Dimmer Setting Register (SEGD)

**Figure 18.3-11 Segment Dimmer Setting Register (SEGD)**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| SEGD0 | FIP07 | FIP06 | FIP05 | FIP04 | FIP03 | FIP02 | FIP01 | FIP00 | XXXXXXXX$_B$ |
| SEGD1 | FIP15 | FIP14 | FIP13 | FIP12 | FIP11 | FIP10 | FIP09 | FIP08 | XXXXXXXX$_B$ |
| SEGD2 | FIP23 | FIP22 | FIP21 | FIP20 | FIP19 | FIP18 | FIP17 | FIP16 | XXXXXXXX$_B$ |
| SEGD3 | FIP31 | FIP30 | FIP29 | FIP28 | FIP27 | FIP26 | FIP25 | FIP24 | XXXXXXXX$_B$ |
| SEGD4 | FIP39 | FIP38 | FIP37 | FIP36 | FIP35 | FIP34 | FIP33 | FIP32 | XXXXXXXX$_B$ |
| SEGD5 | FIP47 | FIP46 | FIP45 | FIP44 | FIP43 | FIP42 | FIP41 | FIP40 | XXXXXXXX$_B$ |
| SEGD6 | FIP55 | FIP54 | FIP53 | FIP52 | FIP51 | FIP50 | FIP49 | FIP48 | XXXXXXXX$_B$ |
| SEGD7 | - | - | - | - | FIP59 | FIP58 | FIP57 | FIP56 | XXXXXXXX$_B$ |
|  | W | W | W | W | W | W | W | W |  |

W: Write only

### ■ Segment Gradation Display (Segment Dimmer)

Specify a segment using registers SEGD0 to SEGD7. Specify the segment by setting a value of "1".

Set a timing with bit 7 of 11E0$_H$ to 11FF$_H$ in the display RAM.

Specify the timing by writing a value of "1".

The SEGD0 to SEGD7 bits for digit pins are ignored.

Dimmer operation can be specified more than once.

**Note:**

Because the initial values of SEGD0 to SEGD7 and 11E0$_H$ to 11FF$_H$ of the display RAM are undefined, set these values before starting with display.

# 18.4 FL Control Circuit Operation

**This section explains the operation of automatic fluorescent tube and LED display.**

## ■ Assignment of Automatic Fluorescent Tube Display Digits and Segments and Automatic LED Display Pins

Automatic fluorescent tube display digits can be specified using pins FIP0 to FIP31. These digits are consecutively set according to the count specified in the digit count setting bits of the digit count register, starting with the pin that is specified in the digit display starting pin setting bits of the digit setting register.

Figure 18.4-1 "Example of assignment for LED, Digit, and Segment Display" shows an example of the resulting assignment.

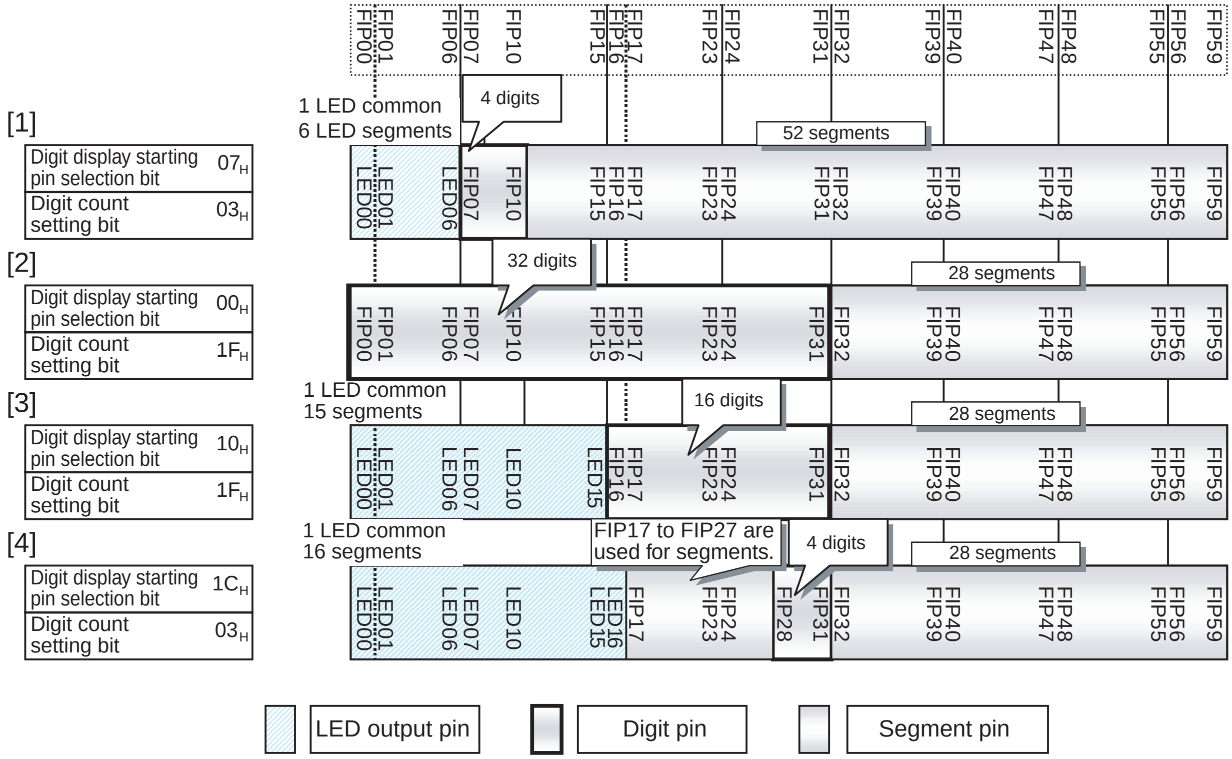
When both the digit display start setting bit and digit count setting bit are set, the digit display start setting bit has priority. (In item [3] of Figure 18.4-1 "Example of assignment for LED, Digit, and Segment Display", the digit count setting bits are set to 1FH (32 digits), but 16 digit pins are used in the actual operation.)

All pins after pins that are used for setting digits become segment pins.

When the value expressed in the digit display pin setting bits is larger than 17H, all pins from FIP17 to the one before the pin specified in the digit display starting pin selection bit are used for segments. (See item [4] in Figure 18.4-1 "Example of assignment for LED, Digit, and Segment Display.")

The automatic LED display pins are set in the LED00 to LED16 registers that are not used for automatic fluorescent tube display.

**Figure 18.4-1 Example of assignment for LED, Digit, and Segment Display**


FIP00 FIP01 FIP06 FIP07 FIP10 FIP15 FIP16 FIP17 FIP23 FIP24 FIP31 FIP32 FIP39 FIP40 FIP47 FIP48 FIP55 FIP56 FIP59
[1]
Digit display starting pin selection bit 07H
Digit count setting bit 03H
1 LED common
6 LED segments
4 digits
52 segments
LED00 LED01 LED06 FIP07 FIP10 FIP15 FIP16 FIP17 FIP23 FIP24 FIP31 FIP32 FIP39 FIP40 FIP47 FIP48 FIP55 FIP56 FIP59
[2]
Digit display starting pin selection bit 00H
Digit count setting bit 1FH
32 digits
28 segments
FIP00 FIP01 FIP06 FIP07 FIP10 FIP15 FIP16 FIP17 FIP23 FIP24 FIP31 FIP32 FIP39 FIP40 FIP47 FIP48 FIP55 FIP56 FIP59
[3]
Digit display starting pin selection bit 10H
Digit count setting bit 1FH
1 LED common
15 segments
16 digits
28 segments
LED00 LED01 LED06 LED07 LED10 LED15 FIP16 FIP17 FIP23 FIP24 FIP31 FIP32 FIP39 FIP40 FIP47 FIP48 FIP55 FIP56 FIP59
[4]
Digit display starting pin selection bit 1CH
Digit count setting bit 03H
1 LED common
16 segments
FIP17 to FIP27 are used for segments.
4 digits
28 segments
LED00 LED01 LED06 LED07 LED10 LED15 LED16 FIP17 FIP23 FIP24 FIP28 FIP31 FIP32 FIP39 FIP40 FIP47 FIP48 FIP55 FIP56 FIP59
LED output pin
Digit pin
Segment pin

## ■ Reading from/Writing to Registers or Display RAM in the FL Control Circuit

Display RAM and the following registers support byte access, but not bit operations.

- Registers in the FL control circuit
- Display control register 1 (FLC1)
- Display control register 2 (FLC2)
- Digit setting register (FLDG)
- Digit count register (FLDC)
- Segment dimmer setting registers (SEGD0 to SEGD7)
- Port registers (FLPD0 to FLPD2)
- Status/authorization register (FLST)

Writing to the display RAM and the registers in the FL control circuit can only be performed when a "1" is set in the status/authorization register (FLST) display RAM and the bit for preventing register access (FLDNE).

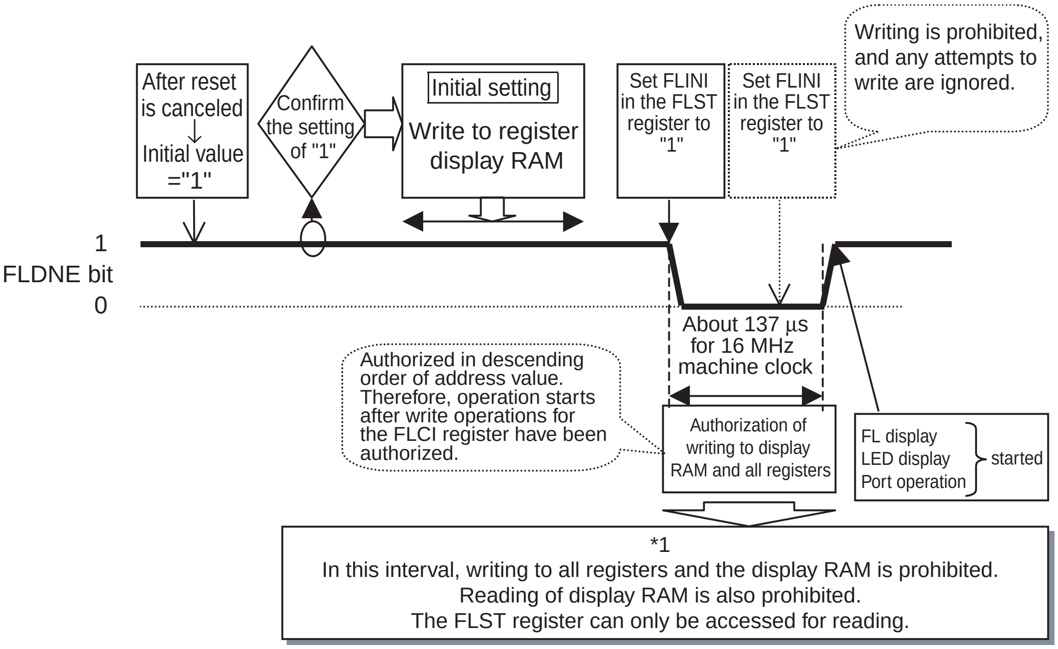
Reading from the display RAM can be performed even when the bit for preventing register access (FLDNE) is set to "1".

After writing to the display RAM and any register other than the status/authorization register (FLST), the display RAM/register write authorization bit (FLINI) in the status/authorization register (FLST) must be set to "1".

The following diagram explains the operation.

### ❍ At initialization

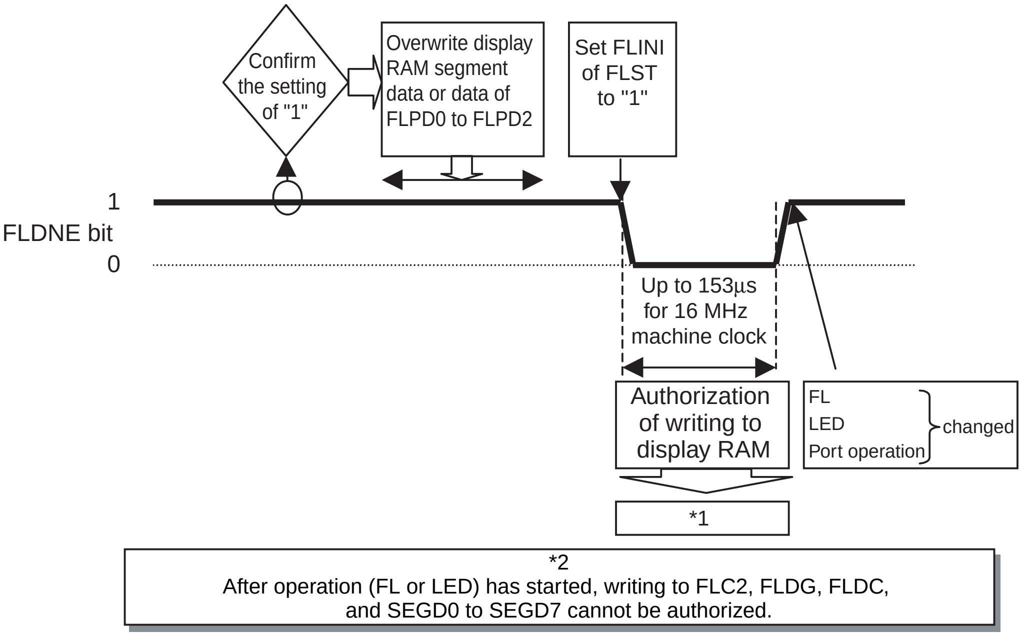
**Figure 18.4-2 At Initialization**



After reset is canceled ↓ Initial value ="1"
Confirm the setting of "1"
Initial setting
Write to register display RAM
Set FLINI in the FLST register to "1"
Set FLINI in the FLST register to "1"
Writing is prohibited, and any attempts to write are ignored.
1
FLDNE bit
0
About 137 µs for 16 MHz machine clock
Authorized in descending order of address value. Therefore, operation starts after write operations for the FLCI register have been authorized.
Authorization of writing to display RAM and all registers
FL display
LED display
Port operation
started
*1
In this interval, writing to all registers and the display RAM is prohibited.
Reading of display RAM is also prohibited.
The FLST register can only be accessed for reading.

- **When segment data (multiple bytes) is overwritten during automatic fluorescent tube display**

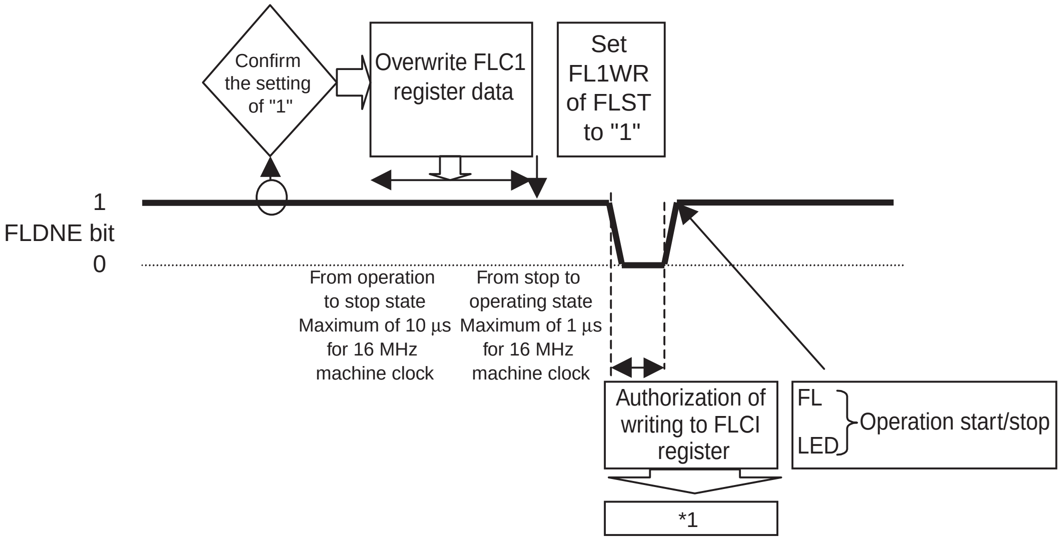
**Figure 18.4-3  When Segment Data Is Overwritten During Automatic Fluorescent Tube Display (for Multiple Bytes)**


Confirm the setting of "1"
Overwrite display RAM segment data or data of FLPD0 to FLPD2
Set FLINI of FLST to "1"
1
FLDNE bit
0
Up to 153µs for 16 MHz machine clock
Authorization of writing to display RAM
FL
LED
Port operation
changed
*1
*2
After operation (FL or LED) has started, writing to FLC2, FLDG, FLDC, and SEGD0 to SEGD7 cannot be authorized.

The last one-byte writing operation to the FLST register FL1WR is authorized.  The following diagram explains this operation.

- **Stop while operation is in progress or start from stop state (for 1 byte)**

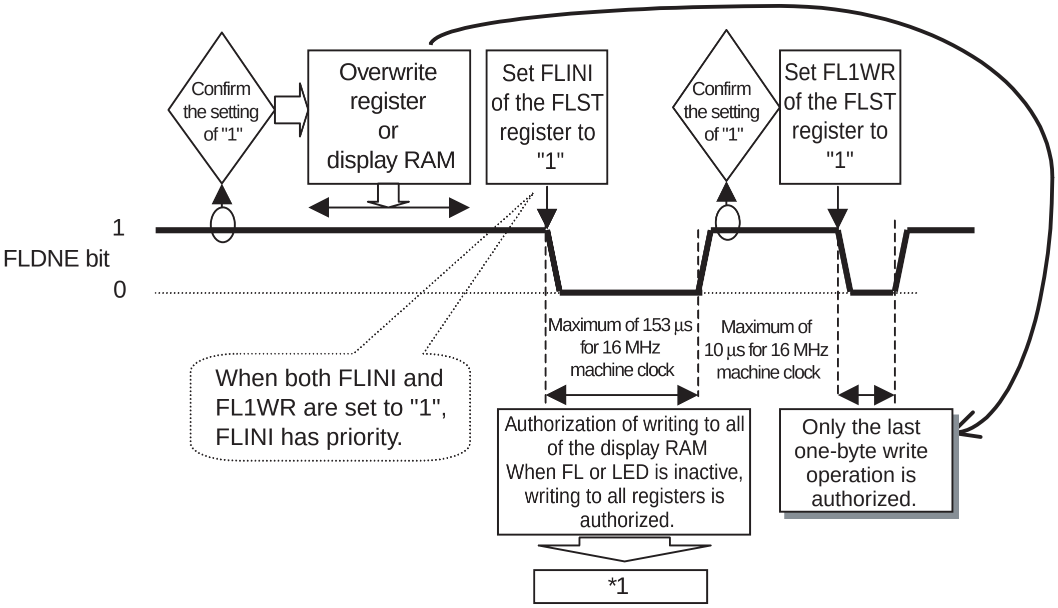
**Figure 18.4-4  Stop While Operation Is in Progress or Start from Stop State (for 1 Byte)**


Confirm the setting of "1"
Overwrite FLC1 register data
Set FL1WR of FLST to "1"
1
FLDNE bit
0
From operation to stop state Maximum of 10 µs for 16 MHz machine clock
From stop to operating state Maximum of 1 µs for 16 MHz machine clock
Authorization of writing to FLCI register
FL
LED
Operation start/stop
*1

The following diagram provides an example for combined setting of FL1WR of the FLST register and FLINI of the FLST register.  Note that the value of FLINI can only be changed after a reset.

❍ **Setting FL1WR after setting FLINI**

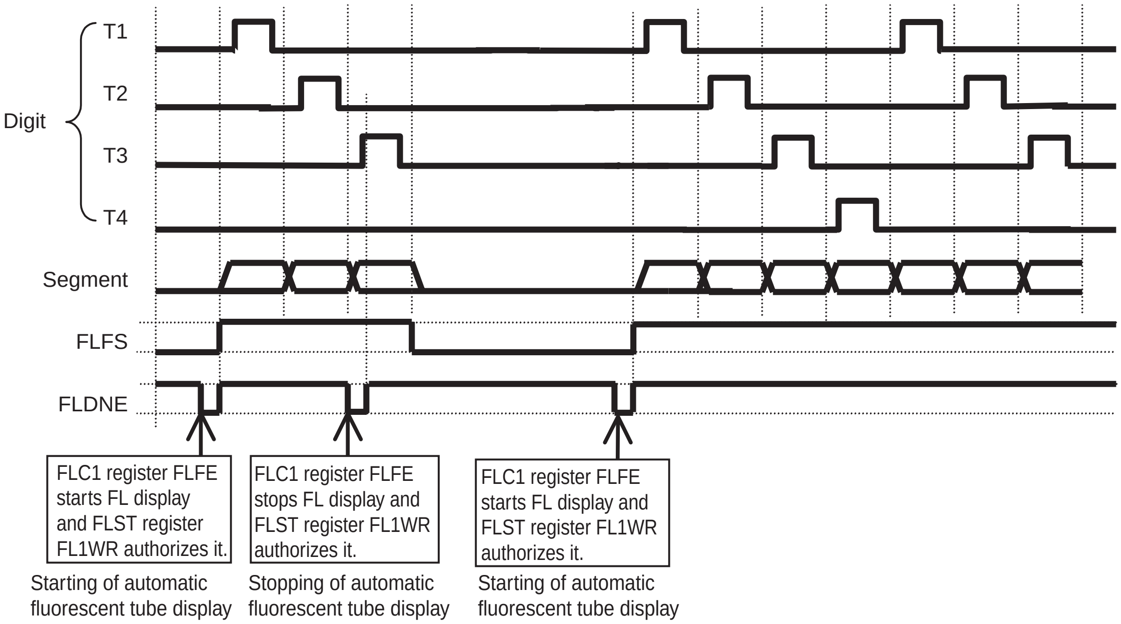
**Figure 18.4-5 Setting FL1WR after Setting FLINI**


Confirm the setting of "1"
Overwrite register or display RAM
Set FLINI of the FLST register to "1"
Confirm the setting of "1"
Set FL1WR of the FLST register to "1"
1
FLDNE bit
0
Maximum of 153 µs for 16 MHz machine clock
Maximum of 10 µs for 16 MHz machine clock
When both FLINI and FL1WR are set to "1", FLINI has priority.
Authorization of writing to all of the display RAM When FL or LED is inactive, writing to all registers is authorized.
Only the last one-byte write operation is authorized.
*1

## ■ Start and Stop Timings of Automatic Fluorescent Tube Display

Automatic fluorescent tube display starts with timing point T1. When automatic fluorescent tube display is stopped, output stops after the output of the current segment output has been completed.

**Figure 18.4-6 Example of Start and Stop Timing of Automatic Fluorescent Tube Display**

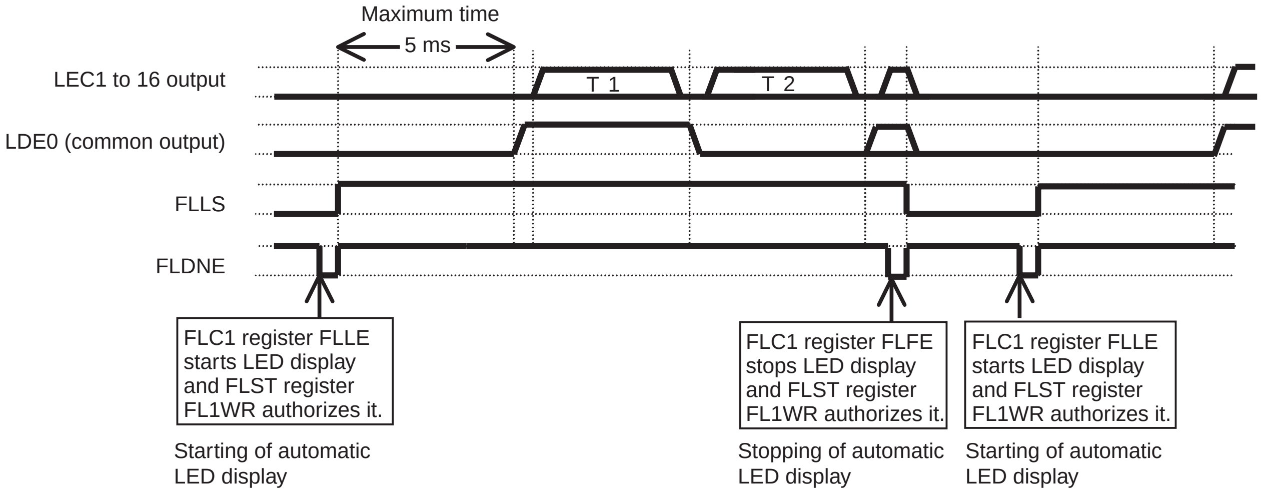

T1
T2
Digit
T3
T4
Segment
FLFS
FLDNE
FLC1 register FLFE starts FL display and FLST register FL1WR authorizes it.
FLC1 register FLFE stops FL display and FLST register FL1WR authorizes it.
FLC1 register FLFE starts FL display and FLST register FL1WR authorizes it.
Starting of automatic fluorescent tube display
Stopping of automatic fluorescent tube display
Starting of automatic fluorescent tube display

## ■ Start and Stop Timing of Automatic LED Display

Up to the first 5 milliseconds after the start of automatic LED display, "L" level (the Pch open drain output is off) is output from common and segment output.

Next, the data for the timing point of T1 is output as segment output when the common output (LED0) is at the "H" level. When automatic LED display stops, the common and segment output switch to "L" level after authorization of writing to the registers.

**Figure 18.4-7 Start and Stop Timings of Automatic LED Display**


Maximum time
5 ms
LEC1 to 16 output
T 1
T 2
LDE0 (common output)
FLLS
FLDNE
FLC1 register FLLE starts LED display and FLST register FL1WR authorizes it.
Starting of automatic LED display
FLC1 register FLFE stops LED display and FLST register FL1WR authorizes it.
Stopping of automatic LED display
FLC1 register FLLE starts LED display and FLST register FL1WR authorizes it.
Starting of automatic LED display

## ■ Operation in Stop Mode and Sleep Mode

### ❍ In stop mode

The FL control circuit is reset in stop mode. (All Pch high dielectric outputs are switched off.) After stop mode is released, initialization must be performed.

### ❍ In sleep mode

The FL control circuit is operational in sleep mode. To enter low-power mode, such operations as stopping automatic display must be performed before sleep mode is set.

# CHAPTER 19 WATCH CLOCK OUTPUT

**This chapter describes the functions and operations of MB90M405 series watch clock output.**

## 19.1 Overview of the Watch Clock Output Circuit

**The watch clock output circuit divides the oscillation clock by the timebase timer and outputs the specified divided clock externally.**
**One of the divisions 32, 64, 128, and 256 of the oscillation clock can be selected.**

### ■ Watch Clock Output Circuit

The watch clock output circuit is disabled during a reset or in stop mode and in enabled in normal run mode, sleep mode, and pseudo watch mode.

**Table 19.1-1 Watch Clock Output Modes**

| | PLL_Run | Main_Run | Sleep | Pseudo watch | STOP | Reset |
|---|---|---|---|---|---|---|
| Operating status | o | o | o | o | x | x |

**Notes:**

The clock cannot be output correctly if the timebase timer is cleared while the watch clock output circuit is in use.

For information about the conditions for clearing the timebase timer, see Chapter 10, "Timebase Timer".

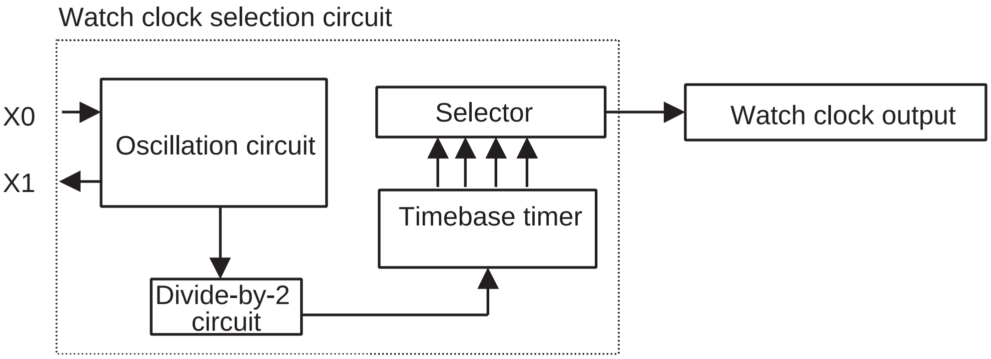
# 19.2 Configuration of the Watch Clock Output Circuit

**The watch clock output circuit consists of the following blocks:**

- **Watch clock selection circuit**
- **Watch clock output control register (TMCS)**

## ■ Block Diagram of the Watch Clock Selection Circuit

**Figure 19.2-1 Block Diagram of the Watch Clock Selection Circuit**

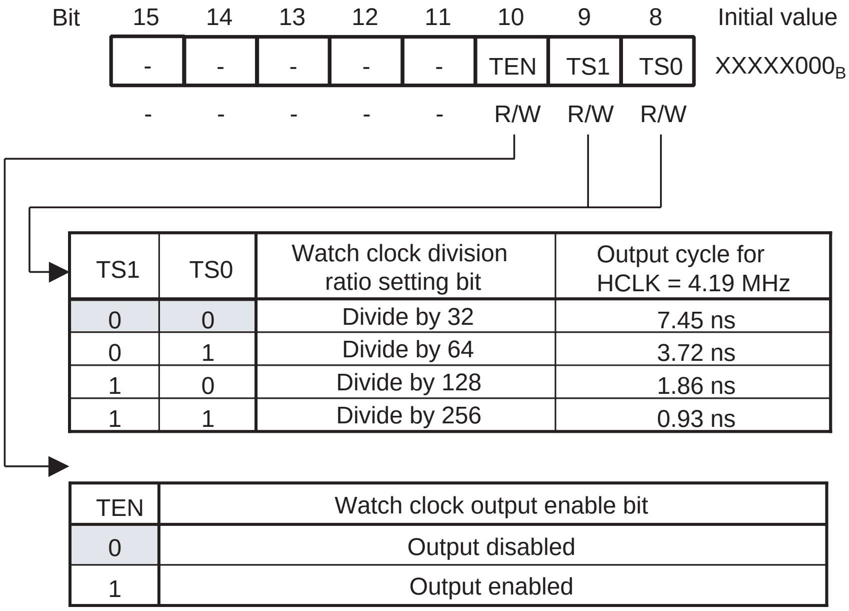


# 19.3 Watch Clock Output Control Register (TMCS)

**The watch clock output control register (TMCS) sets the watch clock division ratio.**

## ■ Watch Clock Output Control Register (TMCS)

**Figure 19.3-1 Watch Clock Output Control Register (TMCS)**



| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | - | - | - | - | - | TEN | TS1 | TS0 | XXXXX000$_B$ |
| | - | - | - | - | - | R/W | R/W | R/W | |

| TS1 | TS0 | Watch clock division ratio setting bit | Output cycle for HCLK = 4.19 MHz |
|---|---|---|---|
| 0 | 0 | Divide by 32 | 7.45 ns |
| 0 | 1 | Divide by 64 | 3.72 ns |
| 1 | 0 | Divide by 128 | 1.86 ns |
| 1 | 1 | Divide by 256 | 0.93 ns |

| TEN | Watch clock output enable bit |
|---|---|
| 0 | Output disabled |
| 1 | Output enabled |

R/W : Read/write enabled
- : Undefined bit
▨ : Initial value
HCLK : Oscillation clock frequency

**Table 19.3-1 Functions of the I$^2$C Status Register (IBSR) Bits**

| Bit name | | Function |
|---|---|---|
| bit15<br>to<br>bit11 | -:<br>Undefined bit | • The reading value read from this bit is undefined.<br>• The value set for this bit does not affect operation. |
| bit10 | TEN:<br>Watch clock output enable bit | • This bit enables watch clock output. To use this function, be sure to specify a port in ADER0 and to specify output from the port in DDRA. |
| bit9<br>bit8 | TS1, TS0:<br>Time clock division ratio setting bit | • Set TS1 and TS2, and specify output from the port to output the clock for the watch clock. |

**Note:**

The first waveform that is output when the TEN bit is set to Enabled may be different from the actually specified output waveform because the watch clock is started asynchronously with the timebase timer.

# CHAPTER 20 DELAYED INTERRUPT GENERATOR MODULE

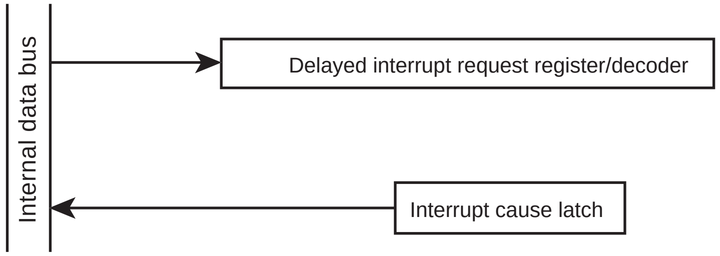
**This chapter describes the functions and operation of the MB90M405 series delayed interrupt generator module.**

## 20.1 Overview of the Delayed Interrupt Generator Module

**The delayed interrupt generator module outputs interrupt requests for task switching. By using this module, interrupt requests for task switching can be output and canceled for the MB90M405 series CPU via software.**

### ■ Block Diagram of the Delayed Interrupt Generator Module

**Figure 20.1-1  Block Diagram of the Delayed Interrupt Generator Module**



# 20.2 Delayed Interrupt Cause/Cancel Register (DIRR)

**This section explains the delayed interrupt cause/cancel register (DIRR).**

## ■ Delayed Interrupt Cause/Cancel Register (DIRR)

**Figure 20.2-1 Delayed Interrupt Cause/Cancel Register (DIRR)**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | — | — | — | — | — | — | — | R0 | $XXXXXXX0_B$ |
| | — | — | — | — | — | — | — | R/W | |

R/W : Read/write
X : Undefined
- : Undefined bit

**Table 20.2-1 Functional Explanation of Each Bit of the Delayed Interrupt Cause/Cancel Register (DIRR)**

| Bit name | | Function |
|---|---|---|
| bit15 to bit9 | -:<br>Undefined bit | • When these bits are read, the values are undefined.<br>• Writing to these bits does not affect operation. |
| bit8 | R0:<br>Delayed interrupt request output bit | • This bit sets the generation/cancel of a delayed interrupt request.<br>• When this bit is "1", a delayed interrupt request is output.<br>• When this bit is "0", the delayed interrupt request is cleared.<br>• When a reset is specified, interrupt causes are canceled (cleared to "0"). |

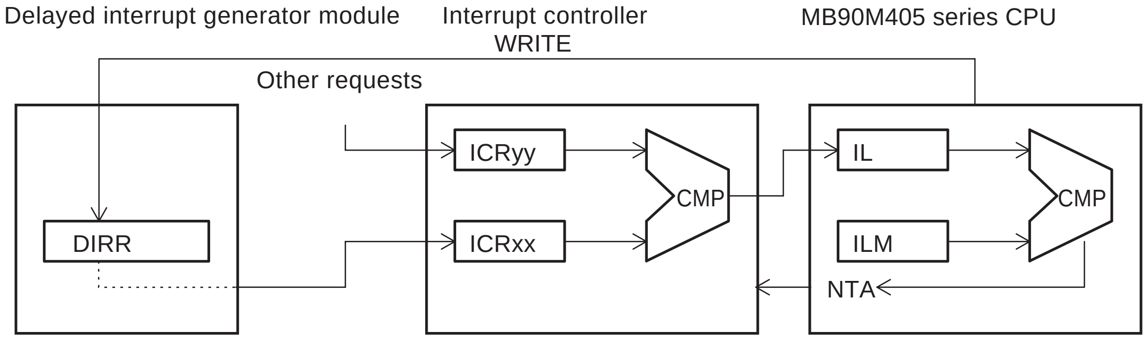
# 20.3 Operation of the Delayed Interrupt Generator Module

**When the delayed interrupt request output bit (R0) of the delayed interrupt cause/ cancel register (DIRR) is set to "1" using software, a delayed interrupt request is output to the interrupt controller.**

## ■ Operation of the Delayed Interrupt Generator Module

When the delayed interrupt request output bit (R0) of the delayed interrupt cause/cancel register (DIRR) is set to "1" using software, an interrupt request is output to the interrupt controller. If interrupt requests other than the delayed interrupt have lower priorities or there is no interrupt request other than the delayed interrupt request, the interrupt controller outputs an interrupt request to the CPU. The CPU compares the interrupt request level with the interrupt level mask register (ILM) in the processor status register (PS). If the interrupt request level is higher than the interrupt level mask register (ILM), the hardware imbedded processing microprogram is activated after the instruction currently being executed is completed, to execute the delayed interrupt processing routine. If the delayed interrupt request output bit (R0) of the delayed interrupt cause/cancel register (DIRR) is set to "0" in the interrupt processing routine, the delayed interrupt cause is cleared and the task is switched.

**Figure 20.3-1 Operation of the Delayed Interrupt Generator Module**



| | | |
|---|---|---|
| DIRR | : | Delayed interrupt cause/cancel register |
| IL | : | Interrupt level setting bit in the interrupt control register (ICR) |
| ILM | : | Interrupt level mask register in PS |
| CMP | : | Comparator |
| ICR | : | Interrupt control register |

# 20.4 Precautions to Follow when Using the Delayed Interrupt Generator Module

**This section explains the precautions to follow when using the delayed interrupt generator module.**

## ■ Precautions to Follow when Using the Delayed Interrupt Generator Module

### ❍ Delayed interrupt request

If the delayed interrupt request output bit (R0) of the delayed interrupt cause/cancel register (DIRR) is not set to "0" after interrupt processing is completed using an interrupt processing routine or while an interrupt processing routine is being executed, it is not possible to return from the interrupt processing.

# CHAPTER 21 ADDRESS MATCH DETECTION FUNCTION

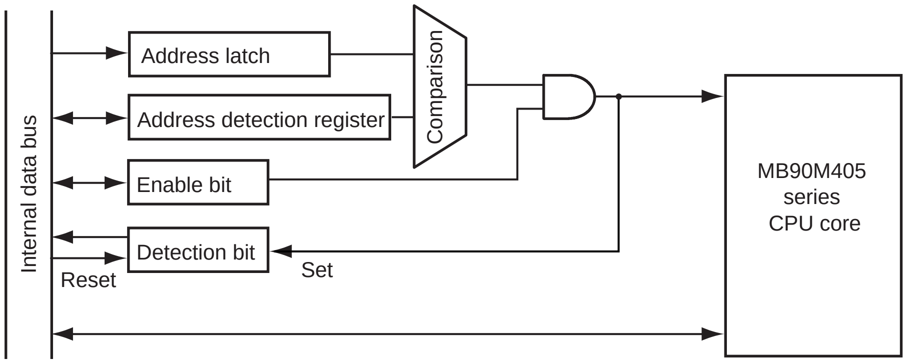
**This chapter describes the address match detection function of the MB90M405 series and its operations.**

# 21.1 Overview of the Address Match Detection Function

**If the program address matches the value set in the address match detection register, the instruction code to be read by the CPU is replaced with an INT9 instruction code. By performing processing using an INT #9 interrupt routine, a program patch application function can be implemented.**

## ■ Block Diagram of the Address Match Detection Function

**Figure 21.1-1 Block Diagram of the Address Match Detection Function**

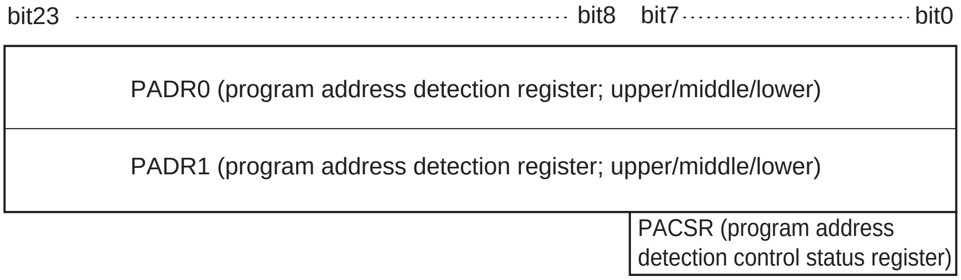


# 21.2 Registers of the Address Match Detection Function

**This section shows a list of registers of the address match detection function.**

## ■ List of Registers of the Address Match Detection Function

**Figure 21.2-1 List of Registers of the Address Match Detection Function**

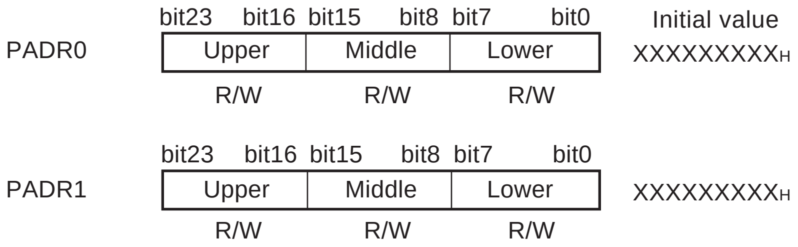

bit23
bit8
bit7
bit0
PADR0 (program address detection register; upper/middle/lower)
PADR1 (program address detection register; upper/middle/lower)
PACSR (program address detection control status register)

## 21.2.1 Program Address Detection Register for Upper, Middle, and Lower Parts of Address (PADR0/PADR1)

**The program address detection register for upper, middle, and lower parts of an address (PADR0/PADR1) is used to set an address for comparison.**

### ■ Program Address Detection Register for Upper, Middle and Lower Parts of an Address (PADR0/PADR1)

**Figure 21.2-2 Program Address Detection Register for Upper, Middle, and Lower Parts of an Address (PADR0/PADR1)**



R/W : Read/write enabled
X : Undefined

If the corresponding interrupt enable bit of the program address detection control status register (ACSR) is set to "1", the program address is compared with the value stored in the program address detection register for the upper, middle, and lower parts of an address (PADR0/PADR1). If the program address value (PC value) matches the value stored in the program address detection register for upper, middle, and lower parts of an address (PADR0/PADR1), the corresponding interrupt flag bit is set to "1" and an INT9 instruction is output. If the interrupt enable bit is set to "0", no INT9 instruction is output.

The following table lists the correspondence between the program address detection control status register (PASCR) and the interrupt request enable bits and the interrupt request flag bits.

| Address detection register | Interrupt enable bit | Interrupt request flag bit |
|---|---|---|
| PADR0 | AD0E | AD0D |
| PADR1 | AD1E | AD1D |

## 21.2.2 Program Address Detection Control Status Register (PACSR)

**The program address detection control status register (PACSR) is used to perform the interrupt control of the address match detection function.**

### ■ Program Address Detection Control Status Register (PACSR)

**Figure 21.2-3 Program Address Detection Control Status Register (PACSR)**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | RESV | RESV | RESV | RESV | AD1E | AD1D | AD0E | AD0D | 00000000B |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

R/W : Read/write
X : Undefined

**Table 21.2-1 Functional Explanation of Each Bit of the Program Address Detection Control Status Register (PACSR)**

| Bit name | | Function |
|---|---|---|
| bit7 to bit4 | RESV:<br>Reserved bit | • Always set "0". |
| bit3 | AD1E:<br>PADR1 interrupt request enable bit | • This bit enables an interrupt of PADR1.<br>• When this bit is "1", the program address detection register (PADR1) and the program address are compared. If the program address detection register (PADR1) matches the program address, the PADR1 interrupt flag bit (AD1D) is set to "1" and an INT9 instruction is output. |
| bit2 | AD1D:<br>PADR1 interrupt request flag bit | • This bit enables an interrupt request of PADR1.<br>• This bit is set to "1" when the program address detection register (PADR1) and the program address are compared and match.<br>• While the PAD1 request enable bit (AD1E) is "1", setting this bit to "1" outputs an INT9 instruction.<br>• Setting this bit to "0"clears it to "0".<br>• Setting this bit to "1" has no effect on operation. |
| bit1 | AD0E:<br>PADR0 interrupt request enable bit | • This bit enables an interrupt of PADR0.<br>• When this bit is "1", the program address detection register (PADR0) value and the program address are compared. If both match, the interrupt flag bit (AD0D) of the PADR0 is set to "1" and an INT9 instruction is output. |
| bit0 | AD0D:<br>PADR0 interrupt request flag bit | • This bit enables an interrupt request of PADR0.<br>• This bit is set to "1" when the program address detection register (PADR1) and the program address are compared and match.<br>• While the PAD0 request enable bit (AD0E) is "1", setting this bit to "1" outputs an INT9 instruction.<br>• Setting this bit to "0" clears it to "0".<br>• Setting this bit to "1" does not affect operation. |

# 21.3 Operation of the Address Match Detection Function

**This section explains the operation of the address match detection function.**

## ■ Operation of the Address Match Detection Function

If the program address matches the value set in the address match detection register, the instruction code to be read by the CPU is replaced with an INT9 instruction code ($01_H$). When the CPU executes the instruction at the specified program address, the INT9 instruction is executed. By performing processing using an INT #9 interrupt routine, a program patch application function can be implemented.

There are two program address detection registers (PADR0/PADR1), each of which has an interrupt enable bit (AD1E, AD0E) and an interrupt flag bit (AD1D, AD0D). If the interrupt enable bit (AD1E, AD0E) is set to "1", the value stored in the address detection register and the program address are compared. If they match, the interrupt flag bit (AD1D, AD0D) is set to "1", and the instruction code to be read by the CPU is replaced with an INT9 instruction code. Setting the interrupt flag bit (AD1D, AD0D) to "0"clears it to "0".

**Note:**

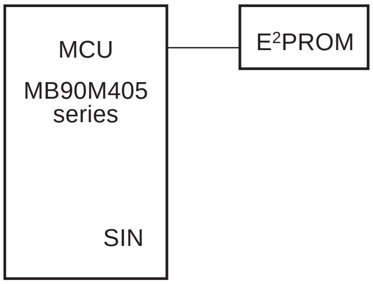
The address detection function does not work correctly if a program address after the 1st byte of the instruction is set in the address detection register. Change the address detection register after setting the interrupt enable bit to "0". If the setting of the address detection register is changed while the interrupt enable bit is set to "1", an address detection may be mistakenly performed while making settings.

# 21.4 Example of Using the Address Match Detection Function

**This section contains example of Using the Address Match Detection Function.**

## ■ System Configuration

**Figure 21.4-1 System Configuration Example**



## ■ $E^2$PROM Memory Map

**Table 21.4-1 $E^2$PROM Memory Map**

| Address | Meaning |
|---|---|
| $0000_H$ | Number of bytes of patch program No. 0<br>(0 for no program error) |
| $0001_H$ | Bit 7 to bit 0 of program address No. 0 |
| $0002_H$ | Bit 15 to bit 8 of program address No. 0 |
| $0003_H$ | Bit 24 to bit 16 of program address No. 0 |
| $0004_H$ | Number of bytes of patch program No. 1<br>(0 for no program error) |
| $0005_H$ | Bit 7 to bit 0 of program address No. 1 |
| $0006_H$ | Bit 15 to bit 8 of program address No. 1 |
| $0007_H$ | Bit 24 to bit 16 of program address No. 1 |
| $0010_H$+<br>Number of bytes of patch program No. 0 | Original of patch program No. 0 |

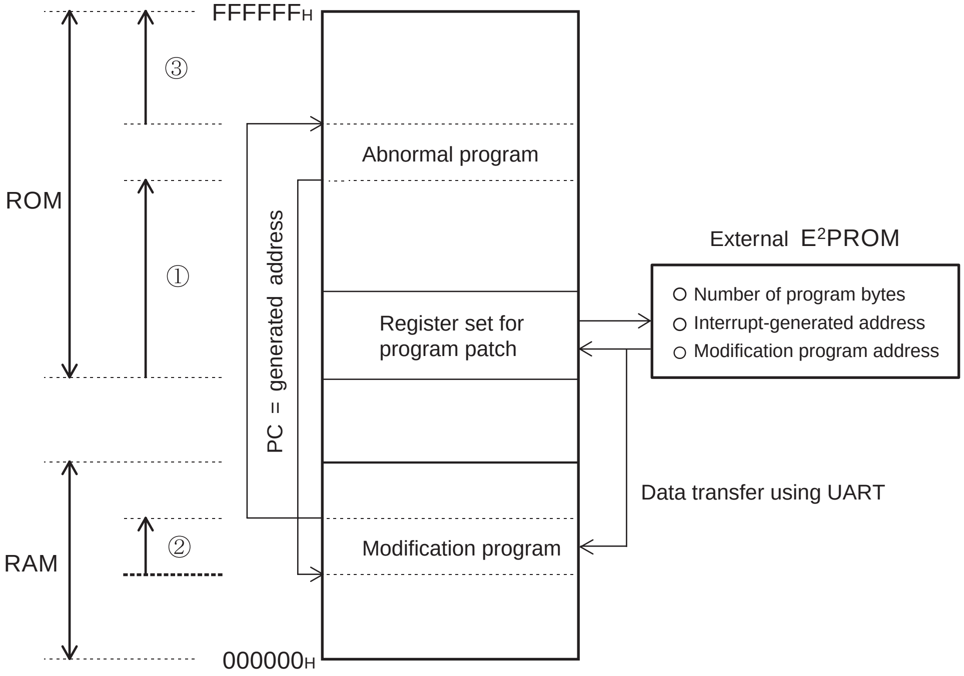
## ■ Initial State

The contents of $E^2$PROM are all 0s.

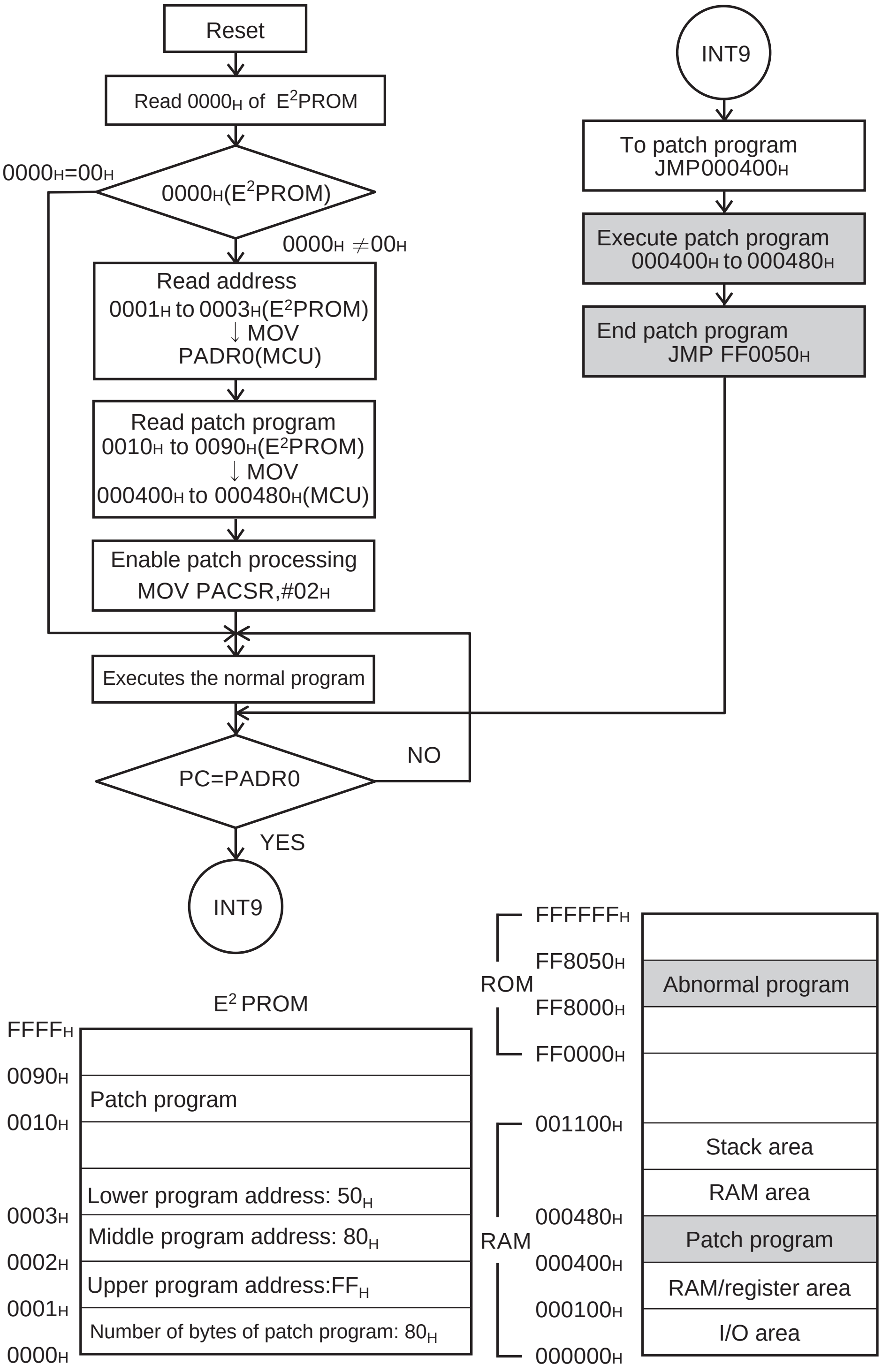
## ■ INT9 Interrupt

An interrupt routine finds the address detection cause for which an interrupt request was output by referencing the interrupt flag bits (AD1D, AD0D) of the detection control status register (PACSR). The routine then causes a branch to the program from which the interrupt request was output. If a branch to the program is caused, information saved on the stack due to the interrupt becomes invalid, and the interrupt flag bits (AD1D, AD0D) are cleared to "0".

**Figure 21.4-2 System Configuration Example**



### Figure 21.4-3 Flowchart of Program Patch Processing



# CHAPTER 22 ROM MIRRORING FUNCTION SELECTION MODULE

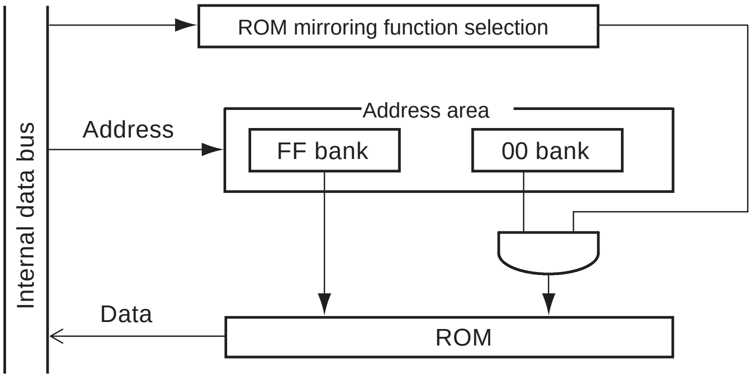
**This chapter describes the function and operation of the MB90M405 series ROM mirroring function selection module.**

## 22.1 Overview of the ROM Mirroring Function Selection Module

**The ROM mirroring function selection module can access ROM data in bank FF from bank 00 by setting its register.**
**The ROM mirroring function enables access from the target area (FF4000$_H$ to FFFFFF$_H$) to the I/O area or the RAM area without bank accesses.**

### ■ Block diagram of the ROM mirroring function selection module

**Figure 22.1-1 Block Diagram**



# 22.2 ROM Mirroring Function Selection Register (ROMM)

**This section explains the register in the ROM mirroring function selection module.**

## ■ ROM Mirroring Function Selection Register (ROMM)

**Figure 22.2-1 ROM Mirroring Function Selection Register (ROMM)**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | - | - | - | - | - | - | - | MI | $XXXXXXX1_B$ |
| | - | - | - | - | - | - | - | W | |

W : Write only
X : Undefined
- : Undefined bit

**Note:**

Do not set the ROM mirroring function selection register while accessing the addresses "$004000_H$ to $00FFFF_H$".

**Table 22.2-1 Functional Explanation of the ROM Mirroring Function Selection Register (ROMM)**

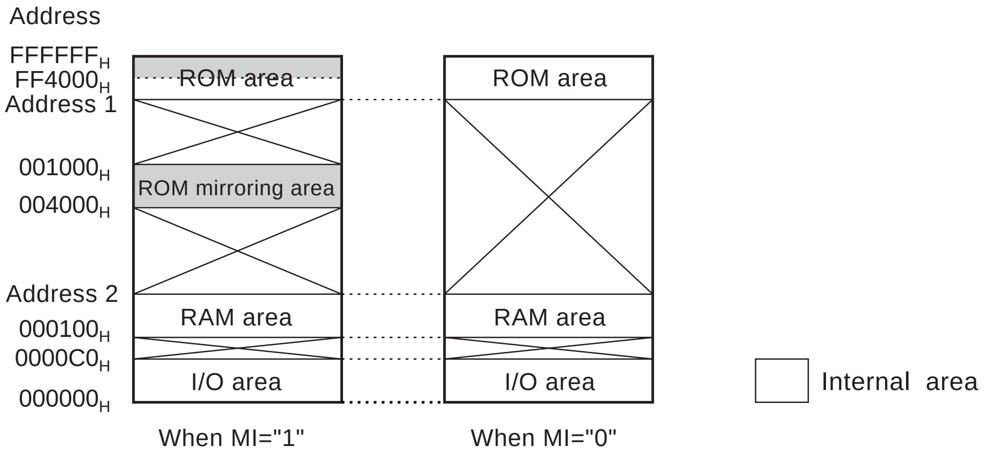
| Bit name | | Function |
|---|---|---|
| bit15 to bit9 | -:<br>Undefined bit | • The values read from these bits are undefined.<br>• Setting these bits to a new value does not affect operation. |
| bit8 | MI:<br>ROM mirroring function setting bit | • This bit is used to set the ROM mirroring function.<br>• When this bit is "1", ROM data in bank FF can be read from bank 00.<br>• When this bit is "0", ROM data in bank FF cannot be read from bank 00. |

**Note:**

The ROM mirroring function accesses addresses $004000_H$ to $00FFFF_H$ from addresses $FF4000_H$ to $FFFFF_H$. Therefore, addresses $FF0000_H$ to $FF3FFF_H$ cannot be accessed when use of the ROM mirroring function is set.

| | MB90M407/M407A | MB90M408/M408A | MB90MF408/MF408A | MB90MV405 |
|---|---|---|---|---|
| Address 1 | $FE8000_H$ | $FE0000_H$ | $FE0000_H$ | $FE0000_H$ |
| Address 2 | $001100_H$ | $001100_H$ | $001100_H$ | $001100_H$ |

**Figure 22.2-2  Memory Space**



# CHAPTER 23 1M-BIT FLASH MEMORY

**This chapter describes the functions and operations of the MB90M405 series 1M-bit flash memory.**

# 23.1 Overview of the 1M-Bit Flash Memory

**The 1M-bit flash memory is allocated in banks $FE_H$ and $FF_H$ on the CPU memory map. Like mask ROM, it does support read access and program access from the CPU because of the flash memory interface circuit function. Data can be written to or deleted from the flash memory via the flash memory interface circuit using CPU instructions. Because this approach enables rewriting the flash memory when it is installed under control of the built-in CPU, programs and data can be changed more efficiently.**

## ■ Writing Data to or Deleting Data from the Flash Memory

You can write data to or delete data from the flash memory in one of the following two ways:

1. Dedicated serial programmer (AF220 manufactured by YDC)
2. Writing and deletion using a program

This section describes the second option of executing a program.

## ■ Features of the 1M-bit Flash Memory

- Configuration of 128K words x 8K or 64K words x 16 bits (16K + 8K + 8K + 32K + 64K) sector
- Automatic program algorithm (equivalent to Embedded Algorithm: MBM29F400TA)
- Function for temporarily stopping and restarting deletion
- Detection of the completion of writing and deletion using CPU interrupts
- Compatibility with JEDEC standard commands
- Sector-by-sector deletion (Any combination of sectors)
- 10,000 writes and deletes guaranteed

Embedded Algorithm is a trademark of Advanced Micro Devices Corporation.

## ■ Writing to and Deletion from the Flash Memory

Writing to and deletion from the flash memory must not be performed at the same time. To write data to or delete data from the flash memory, first copy the program to RAM and then execute the copied program in RAM.

For more information, see Section 23.5.2 "Writing Data to the Flash Memory".

# 23.2 Registers and Sector Configuration of the Flash Memory

**Figure 23.2-1 "Sector Configuration of the 1M-bit Flash Memory" shows the registers and sector configuration of the flash memory.**

## ■ Registers of the Flash Memory

### ❍ Flash memory control status register (FMCS)

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
|  | INTE | RDYINT | WE | RDY | Reserved | LPM1 | Reserved | LPM0 | $00000000_B$ |
|  | R/W | R/W | R/W | R | W | R/W | W | R/W |  |

R/W : Read/write enabled
R : Read only
W : Write only

## ■ Sector Configuration

Figure 23.2-1 "Sector Configuration of the 1M-bit Flash Memory" shows the sector configuration of the 1M-bit flash memory. In this figure, high-order and low-order addresses are shown for each sector.

For access from the CPU, SA0 is stored in the FE bank register and SA1 to SA4 are stored in the FF bank register.

**Figure 23.2-1 Sector Configuration of the 1M-bit Flash Memory**

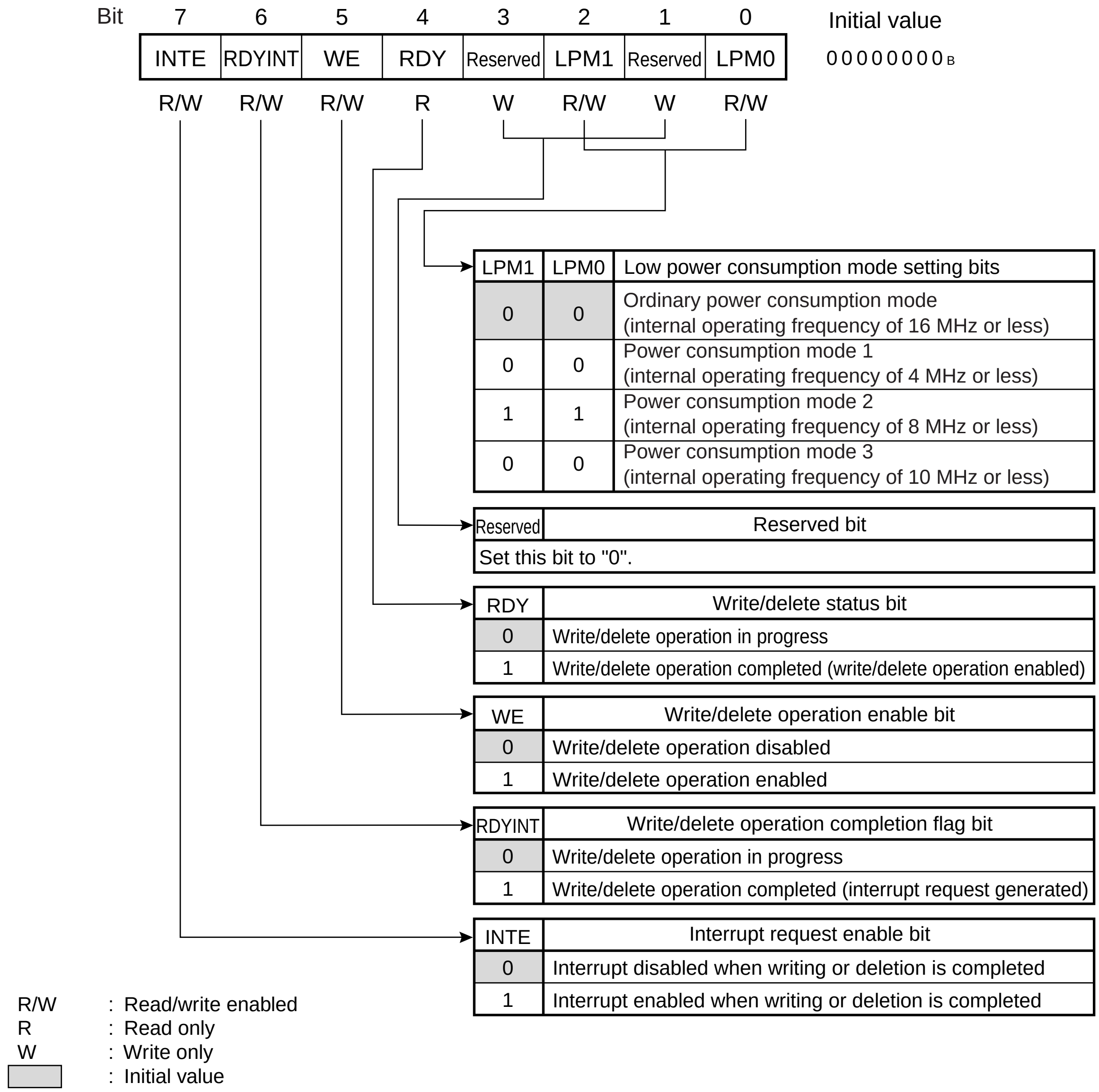
| Flash memory |  | CPU address |
|---|---|---|
| SA4(16 KB) | High order | $FFFFFF_H$ |
|  | Low order | $FFC000_H$ |
| SA3(8 KB) | High order | $FFBFFF_H$ |
|  | Low order | $FFA000_H$ |
| SA2(8 KB) | High order | $FF9FFF_H$ |
|  | Low order | $FF8000_H$ |
| SA1(32 KB) | High order | $FF7FFF_H$ |
|  | Low order | $FF0000_H$ |
| SA0(64 KB) | High order | $FEFFFF_H$ |
|  | Low order | $FE0000_H$ |

# 23.3 Flash Memory Control Status Register (FMCS)

**This section describes the functions of the flash memory control status register (FMCS).**

## ■ Flash Memory Control Status Register (FMCS)

**Figure 23.3-1 Flash Memory Control Status Register (FMCS)**



| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | INTE | RDYINT | WE | RDY | Reserved | LPM1 | Reserved | LPM0 | 00000000B |
| | R/W | R/W | R/W | R | W | R/W | W | R/W | |

| LPM1 | LPM0 | Low power consumption mode setting bits |
|---|---|---|
| 0 | 0 | Ordinary power consumption mode<br>(internal operating frequency of 16 MHz or less) |
| 0 | 0 | Power consumption mode 1<br>(internal operating frequency of 4 MHz or less) |
| 1 | 1 | Power consumption mode 2<br>(internal operating frequency of 8 MHz or less) |
| 0 | 0 | Power consumption mode 3<br>(internal operating frequency of 10 MHz or less) |

| Reserved | Reserved bit |
|---|---|
| Set this bit to "0". | |

| RDY | Write/delete status bit |
|---|---|
| 0 | Write/delete operation in progress |
| 1 | Write/delete operation completed (write/delete operation enabled) |

| WE | Write/delete operation enable bit |
|---|---|
| 0 | Write/delete operation disabled |
| 1 | Write/delete operation enabled |

| RDYINT | Write/delete operation completion flag bit |
|---|---|
| 0 | Write/delete operation in progress |
| 1 | Write/delete operation completed (interrupt request generated) |

| INTE | Interrupt request enable bit |
|---|---|
| 0 | Interrupt disabled when writing or deletion is completed |
| 1 | Interrupt enabled when writing or deletion is completed |

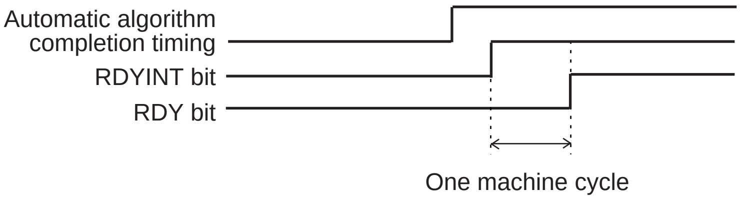
| | | |
|---|---|---|
| R/W | : | Read/write enabled |
| R | : | Read only |
| W | : | Write only |
| ▨ | : | Initial value |

Table 23.3-1 Functions of the Flash Memory Control Status Register (FMCS) Bits

| Bit name | | Function |
|---|---|---|
| bit7 | INTE:<br>Interrupt request enable bit | • This bit enables the output of an interrupt request to the CPU when a flash memory write or delete operation is completed.<br>• An interrupt request is output if this bit is set to "1" and the RDYINT bit is set to "1".<br>• No interrupt request is output if this bit is set to "0" and the RDYINT bit is set to "1". |
| bit6 | RDYINT:<br>Write/delete operation completion flag bit | • When a flash memory write or delete operation is completed, this bit is set to "1", enabling a flash memory write or delete operation.<br>• If this bit is set to "0", this bit is cleared to "0", disabling flash memory write or delete operations.<br>• Setting this bit to "1" does not affect operation.<br>• This bit is also set to "1" when the automatic algorithm (see Section 23.4 "Starting the Automatic Algorithm of the Flash Memory") of the flash memory has been completed.<br>• Read-modify-write (RMW) instructions always return "1" for this bit. |
| bit5 | WE:<br>Write/delete operation enable bit | • If this bit is set to "1", writing to or deletion from the flash memory is enabled after the write/delete command sequence to the FF bank is completed (see Section 23.4 "Startting the Automatic Algorithm of the Flash Memory").<br>• If this bit is set to "0", the execution of the write/delete command sequence to the FF bank does not generate a write or a delete signal.<br>•<br>**Note:**<br>• Set this bit to "0" if neither a write nor a delete operation will be used. |
| bit4 | RDY:<br>Write/delete operation enable bit | • When this bit is "0", writing to or deletion from the flash memory is disabled.<br>• When this bit is "0", read, reset and suspend commands, such as during temporary stop of sector deletion, can be accepted.<br>• This bit is set to "1" when a write or delete operation has been completed. |
| bit3<br>bit1 | Reserved:<br>Reserved bit | • Be sure to set this bit to "0". |
| bit2<br>bit0 | LPM1, LPM0:<br>Low power consumption mode setting bits | • These bits allow power consumption of the flash memory to be controlled from the CPU.<br>• The values that can be specified in these bits vary depending on the internal operating frequency.<br>• The lower the internal operating frequency, the lower the power consumption of the flash memory. |

**Note:**

The operation completion flag bit (RDYINT) and the write/delete status bit (RDY) do not change at the same time. Create a program so that the completion of writing or deletion is determined using the RDYINT or the RDY bit.


Automatic algorithm completion timing
RDYINT bit
RDY bit
One machine cycle

# 23.4 Starting the Automatic Algorithm of the Flash Memory

**Four types of commands are supported to start the automatic algorithm of the flash memory: read/reset, write, chip deletion, and sector deletion. Temporary stop and restart can be controlled for sector deletion.**

## ■ Command Sequence Table

Table 23.4-1 "Command Sequence Table" lists the commands used to write data to and delete data from the flash memory. Although all the data items to be written to the command register have a length of bytes, use word access to write data. The data in the upper bytes written during word access will be ignored.

**Table 23.4-1 Command Sequence Table**

| Command sequence | Bus write access | 1st bus write cycle | | 2nd bus write cycle | | 3rd bus write cycle | | 4th bus write cycle | | 5th bus write cycle | | 6th bus write cycle | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Address | Data | Address | Data | Address | Data | Address | Data | Address | Data | Address | Data |
| Read/ reset (*1) | 1 | FxXXXX$_H$ | XXF0$_H$ | - | - | - | - | - | - | - | - | - | - |
| Read/ reset (*1) | 4 | FxAAAA$_H$ | XXAA$_H$ | Fx5554$_H$ | XX55$_H$ | FxAAAA$_H$ | XXF0$_H$ | RA | RD | - | - | - | - |
| Write program | 4 | FxAAAA$_H$ | XXAA$_H$ | Fx5554$_H$ | XX55$_H$ | FxAAAA$_H$ | XXA0$_H$ | PA(even) | PD (word) | - | - | - | - |
| Chip deletion | 6 | FxAAAA$_H$ | XXAA$_H$ | Fx5554$_H$ | XX55$_H$ | FxAAAA$_H$ | XX80$_H$ | FxAAAA$_H$ | XXAA$_H$ | Fx5554$_H$ | XX55$_H$ | FxAAAA$_H$ | XX10$_H$ |
| Sector deletion | 6 | FxAAAA$_H$ | XXAA$_H$ | Fx5554$_H$ | XX55$_H$ | FxAAAA$_H$ | XX80$_H$ | FxAAAA$_H$ | XXAA$_H$ | Fx5554$_H$ | XX55$_H$ | SA(even) | XX30$_H$ |
| Sector deletion temporary stop | | Enter the data (XXB0$_H$) at address FXXXXX$_H$ to temporarily stop sector deletion. | | | | | | | | | | | |
| Sector deletion restart | | Enter the data (XX30$_H$) at address FXXXXX$_H$ to resume sector deletion after temporarily stop. | | | | | | | | | | | |

*1: These two types of read/reset commands can reset the flash memory to read mode.

**Notes:**

- Address Fx in the table means FF and FE. Use the address for the bank to be accessed during address-related operations.
- The address in the table is a value in the CPU memory map. "X" stands for an arbitrary value.
- RA: Read address
- PA: Write address. An even-numbered address can be specified.
- SA: Sector address. See Section 23.2 "Registers and Sector Configuration of the Flash Memory". An even-numbered address can be specified.
- RD: Read data
- PD: Write data. Word data can be specified.

# 23.5 Detailed Description of Flash Memory Writing and Deletion

**This section describes the procedures for issuing commands that start the automatic algorithm to perform read/reset, write, chip deletion, sector deletion, temporary stop of sector deletion, or restart of sector deletion for the flash memory.**

## ■ Detailed Description of Flash Memory Writing and Deletion

The automatic algorithm can start read/reset, write, chip deletion, sector deletion, sector deletion temporary stop, or sector deletion restart operations by writing a command sequence (see Section 23.4 "Starting the Automatic Algorithm of the Flash Memory") from the CPU to the flash memory.  Writing from the CPU to the flash memory must be performed continuously. When writing ends normally, the status returns to the read or reset status.

The subsequent sections describe the details of the deletion restart operation in the following order:

- Placing the flash memory in read/reset status
- Writing data
- Deleting all data items (all chip deletion)
- Deleting a data item (sector deletion)
- Temporarily stopping sector deletion
- Restarting sector deletion

## 23.5.1 Placing the Flash Memory in Read/Reset Status

**This section describes the procedure for executing the read/reset command to place the flash memory in read/reset status.**

### ■ Placing Flash Memory in Read/Reset Status

To place the flash memory in read/reset status, continuously send the read/reset command listed in the command sequence table (Table 23.4-1 "Command Sequence Table") from the CPU to the flash memory.

There are two command sequences for the read/reset command, the results of which are the same.

Read/reset status is the initial status of the flash memory. The flash memory is always in read/reset status immediately after power is turned on or a command ends normally. In read/reset status, the system waits for input of a new command.

In read/reset status, perform read access to the flash memory to read flash memory data. Like the mask ROM, flash memory does support program access from the CPU. No read/reset command is required to perform read access to the flash memory.

If a command does not end normally, use the read/reset command to initialize the automatic algorithm.

## 23.5.2 Writing Data to the Flash Memory

**This section describes the procedure for executing the write command to write data to the flash memory. Figure 23.5-1 "Example of the Procedure for Flash Memory Writing" shows an example of the procedure for flash memory writing.**

### ■ Writing Data to the Flash Memory

To start the automatic algorithm for writing data to the flash memory, continuously send the write command listed in the command sequence table (Table 23.4-1 "Command Sequence Table"), from the CPU to the flash memory. When data writing to the target address ends after the fourth cycle, the automatic algorithm is started to start automatic writing.

### ■ Addressing

Only an even-numbered address can be specified as the write address in the write data cycle. If an odd-numbered address is specified, data cannot be written correctly. Use word access to write to an even-numbered address in word units.

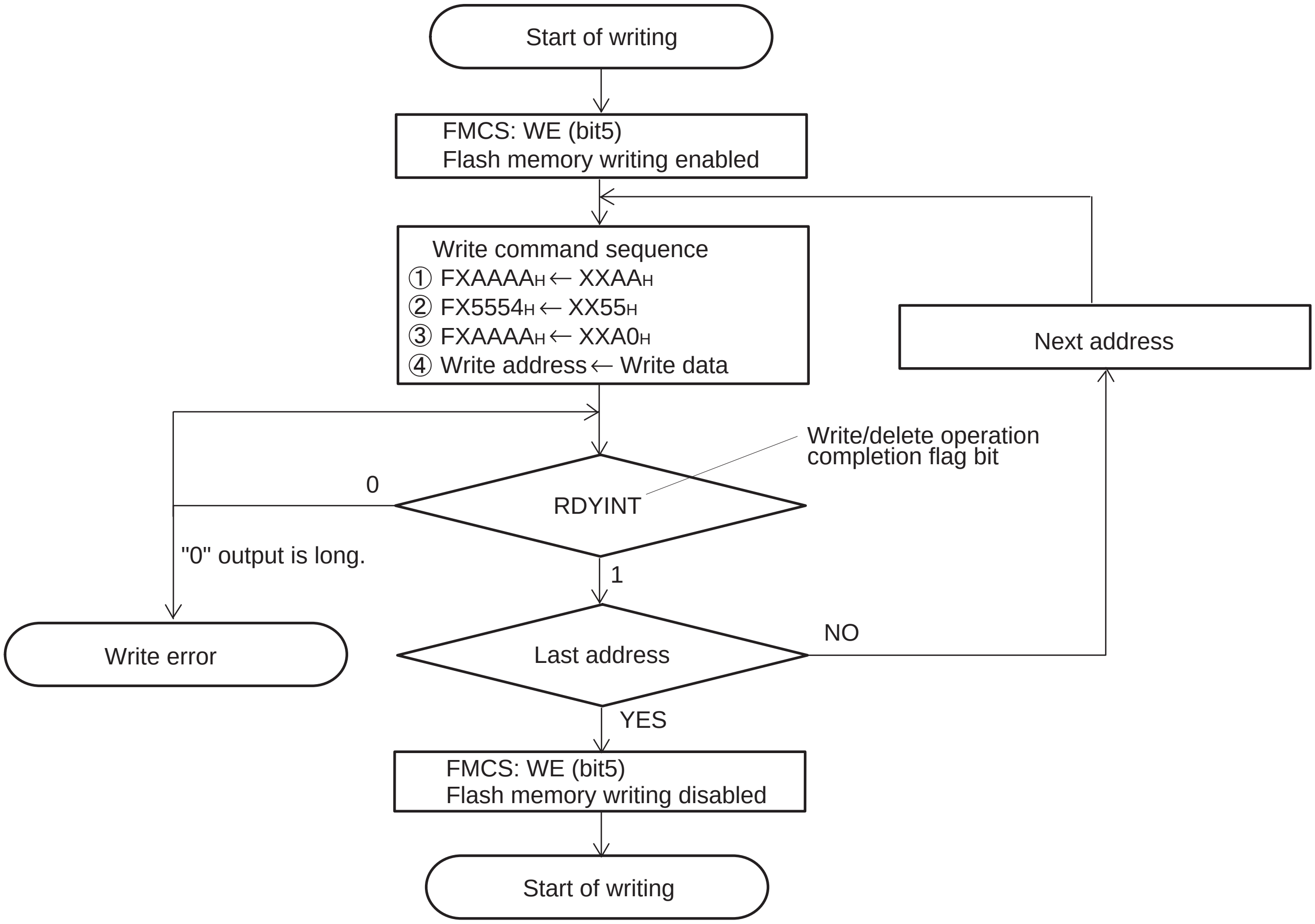
Writing is enabled in any order of addresses and across sector boundaries. However, only one word of data is written each time the write command is executed.

### ■ Notes on Writing Data

The data of a bit cannot be changed from "0" to "1" by write operations. When there is an attempt to overwrite "0" by "1", the data polling algorithm or the toggle operation does not end, the flash memory component will be assumed to be faulty. IN this case, it will only appear as if "1" had been written, but "0" will be returned when the data is read in read/reset status. To change a bit from "0" to "1", perform a delete operation.

All other commands are ignored while automatic writing is in progress. Data at a write address cannot be assured if a hardware reset occurs during writing.

**Figure 23.5-1 Example of the Procedure for Flash Memory Writing**



## 23.5.3 Deleting All Data Items from the Flash Memory (Chip Deletion)

**This section describes the procedure for executing the chip deletion command to delete all data items from the flash memory.**

### ■ Deleting All Data Items from the Flash Memory (Chip Deletion)

To delete all data items from the flash memory, continuously send the chip deletion command listed in the command sequence table (Table 23.4-1 "Command Sequence Table") from the CPU to the flash memory.

The chip deletion command starts the chip deletion operation when writing completes after the sixth cycle. Writing to the flash memory before chip deletion is not required. During the processing for the automatic deletion function, the flash memory performs verification by writing "0" before deleting all bits of the data.

## 23.5.4 Deleting a Data Item from the Flash Memory (Sector Deletion)

**This section describes the procedure for executing the sector deletion command to delete a data item from the flash memory (sector deletion). Data in any sector can be deleted. More than one sector can be specified for deletion. Figure 23.5-2 "Example of the Sector Deletion Procedure for the Flash Memory" shows an example of the procedure for flash memory sector deletion.**

### ■ Deleting a Data Item from the Flash Memory (Sector Deletion)

To delete data in any sector of the flash memory, continuously send the sector deletion command listed in the command sequence table (Table 23.4-1 "Command Sequence Table") from the CPU to the flash memory.

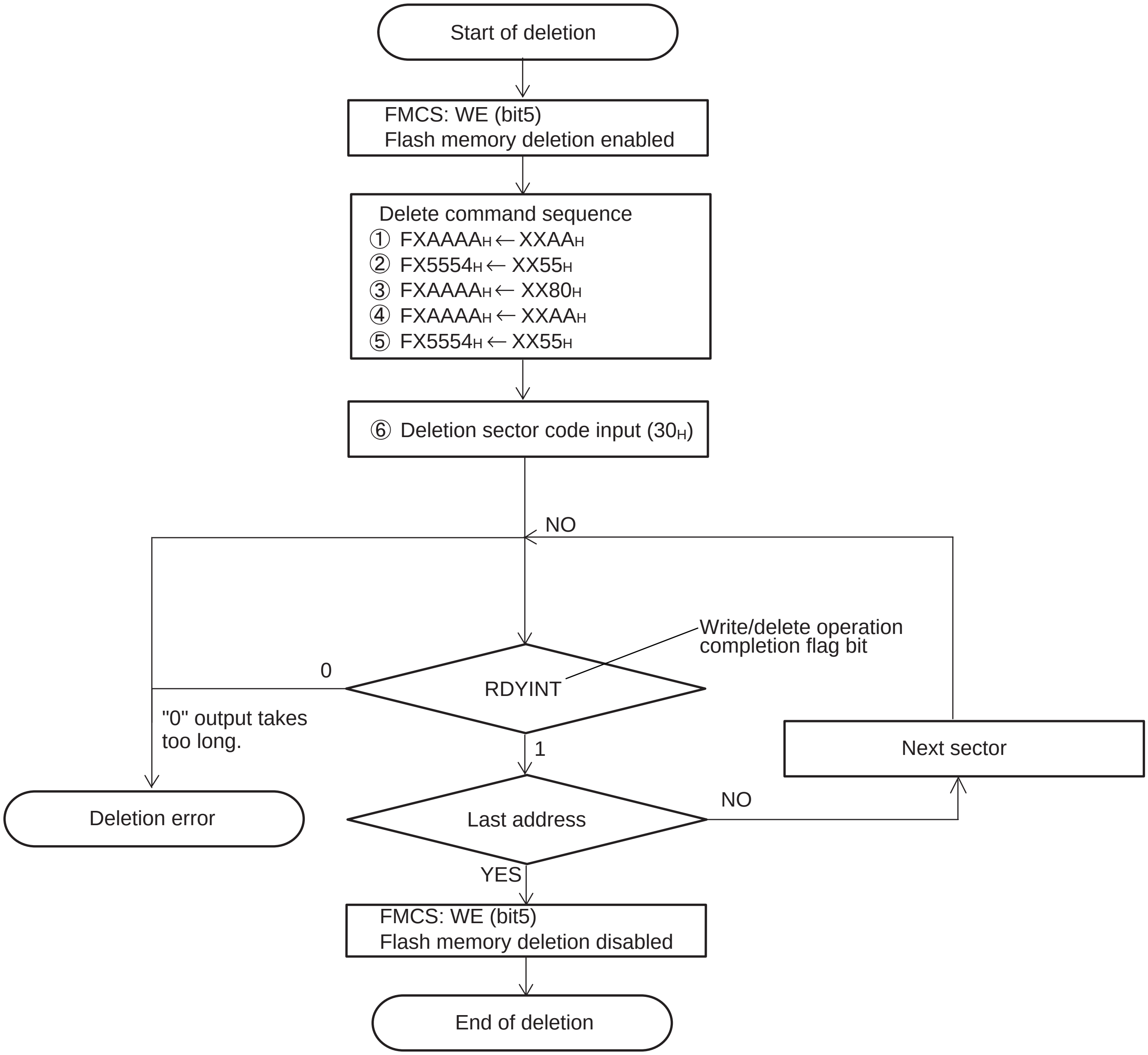
### ■ Specifying a Sector

The sector deletion command starts a sector deletion wait of 50 µs by writing in the sixth cycle the sector deletion code ($30_H$) to any even-numbered address in the target sector that can be accessed. To delete more than one sector, write the deletion code ($30_H$) to the addresses of additional sectors to be deleted.

### ■ Notes on Specifying More Than One Sector

Deletion starts upon completion of the 50 µs sector deletion wait period since the last sector deletion code was written. To delete more than one sector at the same time, input the deletion sector address and the deletion code (in the sixth cycle of the command sequence) within 50 µs. The address and the code are accepted only if they are input within 50 µs.

Figure 23.5-2 Example of the Sector Deletion Procedure for the Flash Memory



## 23.5.5 Temporarily Stopping Deletion of Sectors from the Flash Memory

**This section describes the procedure for executing the sector deletion temporary stop command to temporarily stop the deletion of sectors from the flash memory. Data can be read from sectors that are not being deleted.**

### ■ Temporarily Stopping Sector Deletion

To temporarily stop the  deletion of sectors from the flash memory, continuously send the sector deletion temporary stop command listed in the command sequence table (Table 23.4-1 "Command Sequence Table") from the CPU to the flash memory.

The sector deletion temporary stop command temporarily stops the deletion of sectors and the reading of data from sectors that are not being deleted.  In sector deletion temporary stop status, reading is enabled and writing is disabled.  The sector deletion temporary stop command is valid only during sector deletion (including the deletion wait time) and is ignored during chip deletion or writing.

To execute the sector deletion temporary stop command, write the deletion temporary stop code ($B0_H$).  Specify any address in the flash memory.  During deletion temporary stop, the deletion temporary stop command is ignored if it is executed.

If the sector deletion temporary stop command is input during the sector deletion wait period, the sector deletion wait ends and deletion is suspended, causing the system to enter deletion stop status.  If the deletion temporary stop command is input during sector deletion after the sector deletion wait period, the system enters deletion temporary stop status after waiting for a maximum of 15 µs.

## 23.5.6 Resuming Flash Memory Sector Deletion

**This section describes the procedure for issuing the sector deletion restart command to resume a flash memory sector deletion that has been temporarily stopped.**

### ■ Resuming the Flash Memory Sector Deletion

To resume a sector deletion that has been temporarily stopped, continuously send the sector deletion restart command listed in the command sequence table (Table 23.4-1 "Command Sequence Table") from the CPU to the flash memory.

The sector deletion restart command is used to restart sector deletion that has been temporarily stopped by execution of the sector deletion temporary stop command. To execute the sector deletion restart command, write the sector deletion restart code ($30_H$). Specify any address in the flash memory.

During sector deletion, the sector deletion restart command is ignored if it is executed.

# CHAPTER 24 EXAMPLE OF MB90MF408/MF408A SERIAL PROGRAMMING CONNECTION

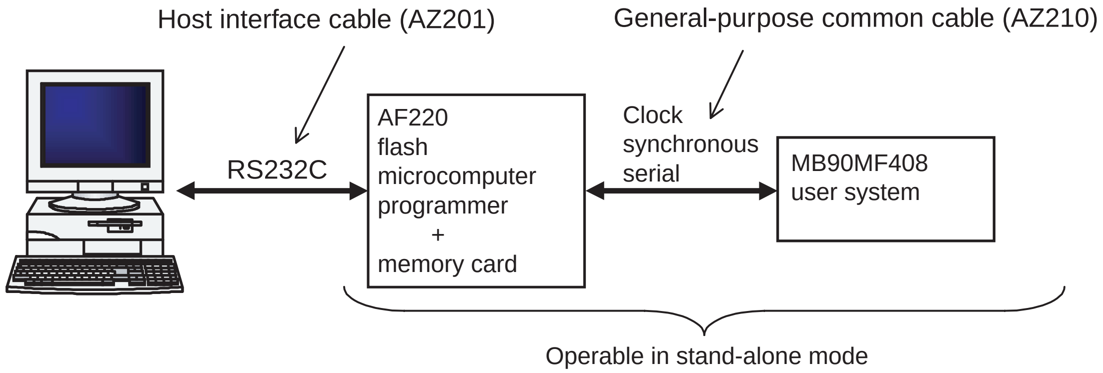
**This chapter provides examples of connection for serial programming using the AF220 flash microcomputer programmer manufactured by YDC Corporation.**

# 24.1 Standard Configuration for Serial Programming Connection to MB90MF408/MF408A

**MB90MF408/MF408A supports serial on-board writing (Fujitsu standard) to flash ROM. This describes the specifications for serial on-board writing.**

## ■ Standard Configuration for Serial Programming Connection to MB90MF408/MF408A

The AF220 flash microcomputer programmer, manufactured by YDC, is used for Fujitsu standard serial on-board writing.



**Note:**

Contact YDC for information about the functions and operations of the AF220 flash microcomputer programmer and information about the general-purpose common connection cable (AZ210) and connectors.

**Table 24.1-1 Pins Used for Fujitsu Standard Serial on-board Programming**

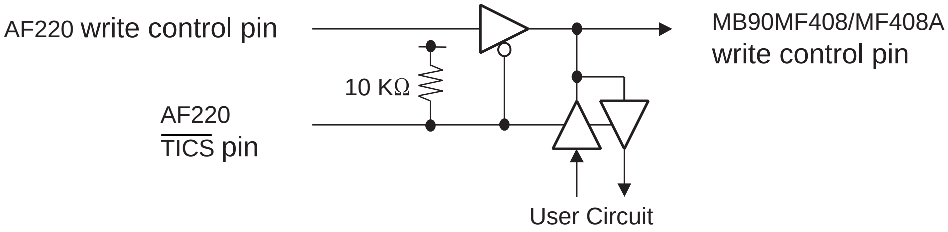
| Pin | Function | Description |
|---|---|---|
| MD2,MD1, MD0 | Mode pin | Switches to programming mode from the flash microcomputer programmer. |
| X0, X1 | Oscillator pin | Since the internal operating clock of the CPU is set to the same clock cycle as the PLL clock, the oscillation clock frequency is used as the internal operation clock. An oscillation clock from 1 MHz to 16 MHz is therefore used for serial on-board programming. |
| P80, P81 | Programming start pin | - |
| $\overline{\text{RST}}$ | Reset pin | - |
| SI1 | Serial data input pin | The UART is used in clock synchronous mode. |
| SO1 | Serial data output pin | |
| SC0 | Serial clock input pin | |
| C | C terminal | Capacitor terminal for stabilizing the power supply. Connect an external ceramic capacitor of about 0.1 μF. |

**Table 24.1-1 Pins Used for Fujitsu Standard Serial on-board Programming (Continued)**

| Pin | Function | Description |
|---|---|---|
| $V_{CC}$ | Power supply pin | The flash microcomputer programmer does not need to be connected to supply the programming voltage (5 V ± 10%) from the user system. When you connect this pin, make sure that no short-circuit occurs between it and the user-side power supply. |
| $V_{SS}$ | Ground pin | Also use this pin as the ground pin for the flash microcomputer programmer. |

When the P80, SI0, SO0, and SC0 pins are also used by the user system, the control circuit shown in Figure 24.1-1 "Control Circuit" is required. (The $\overline{\text{TCIS}}$ signal of the flash microcomputer programmer can separate the user circuit during serial writing. See the connection example shown later.)

**Figure 24.1-1 Control Circuit**



Refer to the following four serial writing examples in Sections 24.2 to 24.5.

- Example of serial programming connection
  - When power supplied by user
- Example of serial programming connection
  - When power is supplied from the programmer
- Example of minimum connection to flash microcomputer programmer
  - When power supplied from user
- Example of minimum connection to flash microcomputer programmer
  - When power is supplied from the programmer

**Table 24.1-2 System Configuration of AF220 Flash Microcomputer Programmer (Manufactured by YDC)**

| Model | Function |
|---|---|
| AF220 | Advanced flash microcomputer programmer |
| AF200 ACP | AC adapter (center +) (optional) |
| AZ201 | Host interface cable (RS232C cable for PCAT) |
| AZ210 | Standard target probe (Type A) Length: 1 m |
| FF001 | Fujitsu $F^2$MC-16LX flash microcomputer control module |
| FF001 P2 | 2 MB PC card (optional) |
| FF001 P4 | 4 MB PC card (optional) |

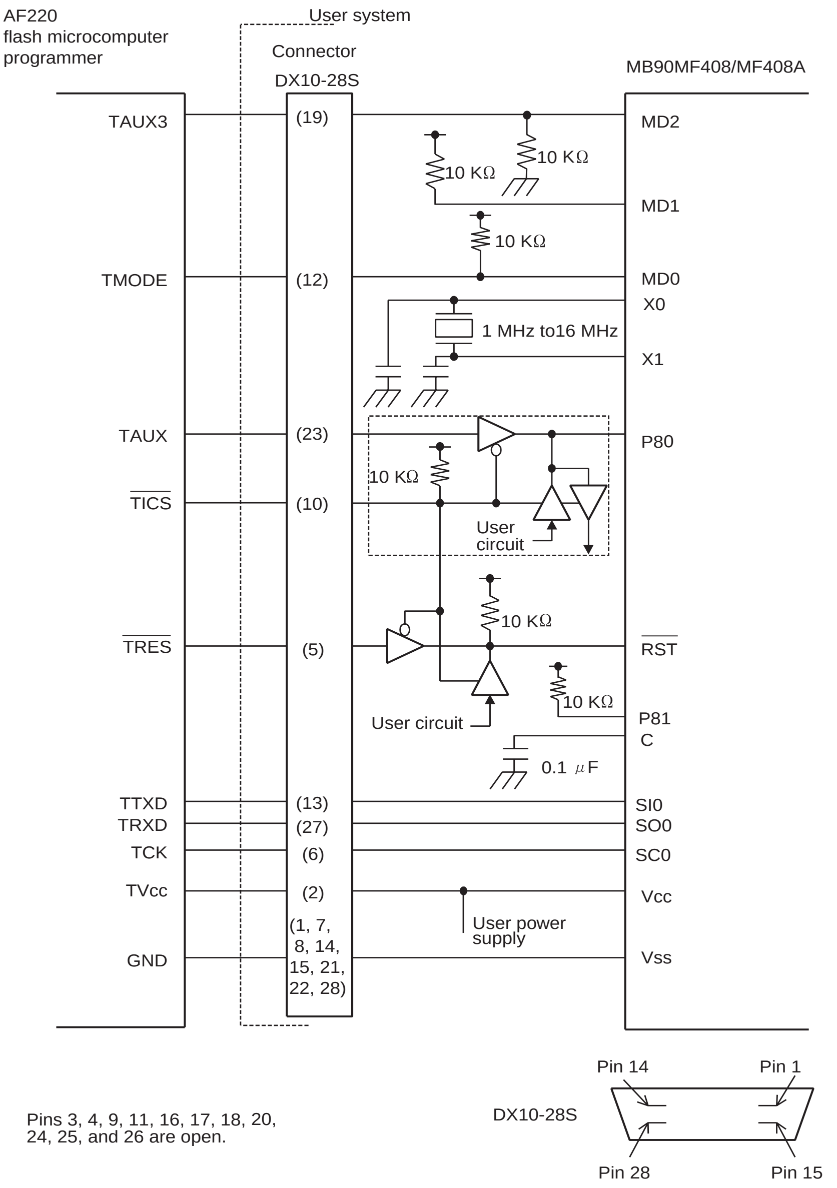
For additional information, contact: YDC Corporation
Telephone number: (81)-42-333-6224

## 24.2 Example of Connection for Serial Programming (Power Supplied by User)

**Figure 24.2-1 "Example of Connection for Serial Programming by MB90MF408/MF408A (Power Supplied by User)" shows an example of connection when microcomputer power is supplied by the user. The mode pins are set to single-chip mode (MD0=1, MD1=1, and MD2=0).**

### ■ Example of Connection for Serial Programming (When Power Supplied by User)

**Figure 24.2-1 Example of Connection for Serial Programming by MB90MF408/MF408A (Power Supplied by User)**

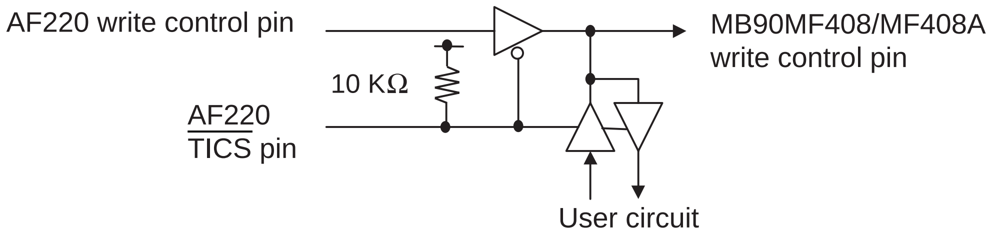


Pins 3, 4, 9, 11, 16, 17, 18, 20, 24, 25, and 26 are open.

DX10-28, right-angle type

Connector (Hirose Electronics) pin layout

- When the user system also uses the P80, SI0, SO0, and SC0 pins, the following control circuit is necessary. During serial programming, disconnect the user circuit using the $\overline{\text{TCIS}}$ signal of the flash microcomputer programmer.



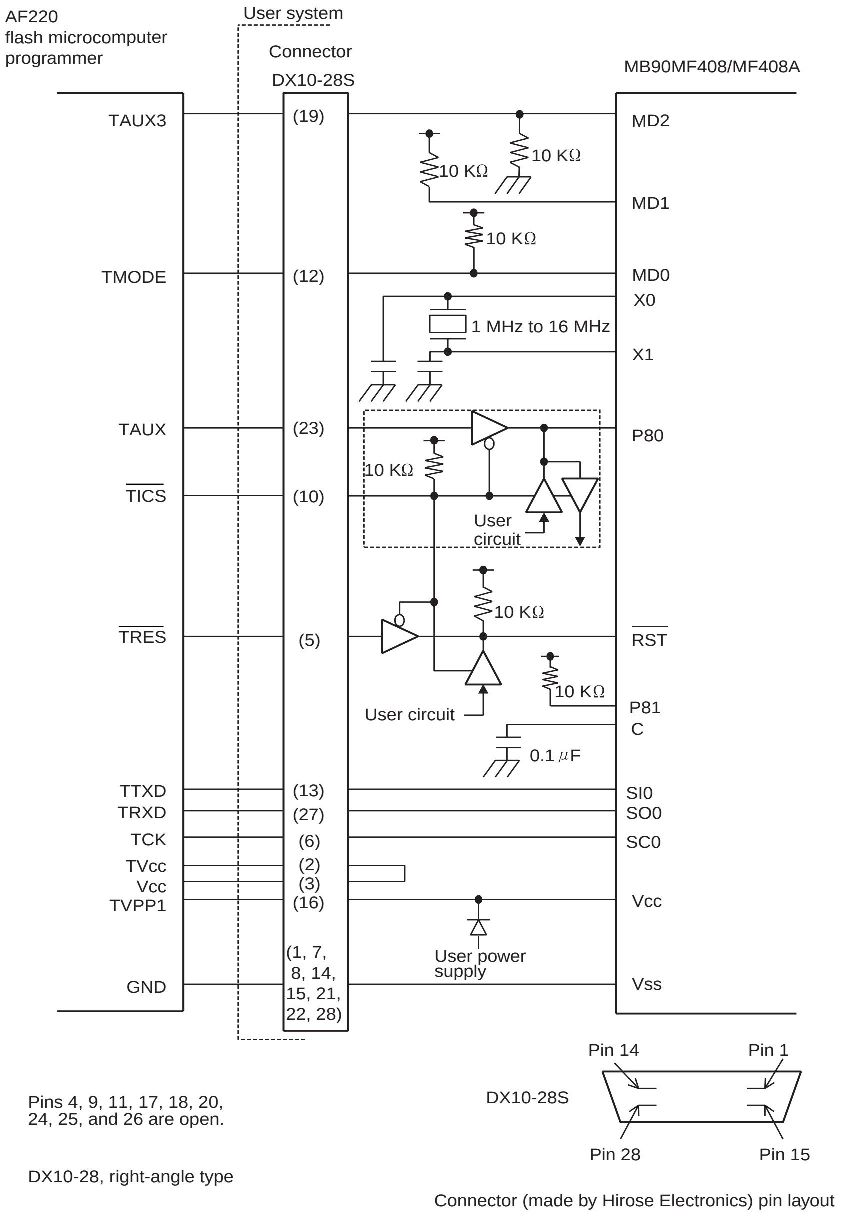
- Before connecting to AF220, turn off the user power.

## 24.3 Example of Connection for Serial Programming (When Power Supplied from Programmer)

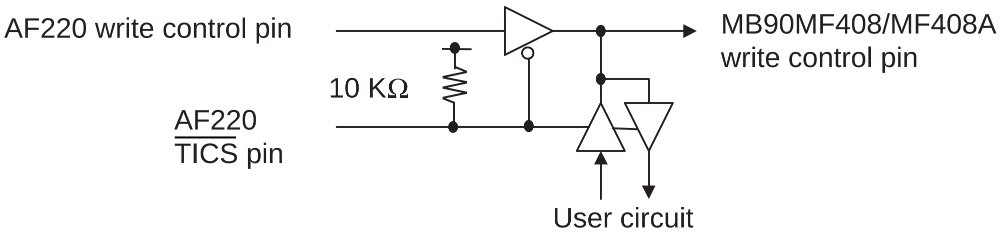
**Figure 24.3-1 "Example of Connection for Serial Programming by MB90MF408/MF408A (Power Supplied by Programmer)" shows an example of connection when microcomputer power is supplied by the programmer. The mode pins are set to single-chip mode (MD0=1, MD1=1, and MD2=0).**

### ■ Example of Connection for Serial Programming (When Power Supplied from Programmer)

**Figure 24.3-1 Example of Connection for Serial Programming by MB90MF408/MF408A (Power Supplied by Programmer)**



- When the user system also uses the P80, SI0, SO0, and SC0 pins, the following control circuit is necessary. During serial programming, disconnect the user circuit using the $\overline{\text{TCIS}}$ signal of the flash microcomputer programmer.



- Before connecting the AF220, turn off the power supplied by the user.
- When supplying power from the AF220, do not create a short circuit to the power supplied by the user.

# 24.4 Example of Minimum Connection with Flash Microcomputer Programmer (When Power Supplied from User)

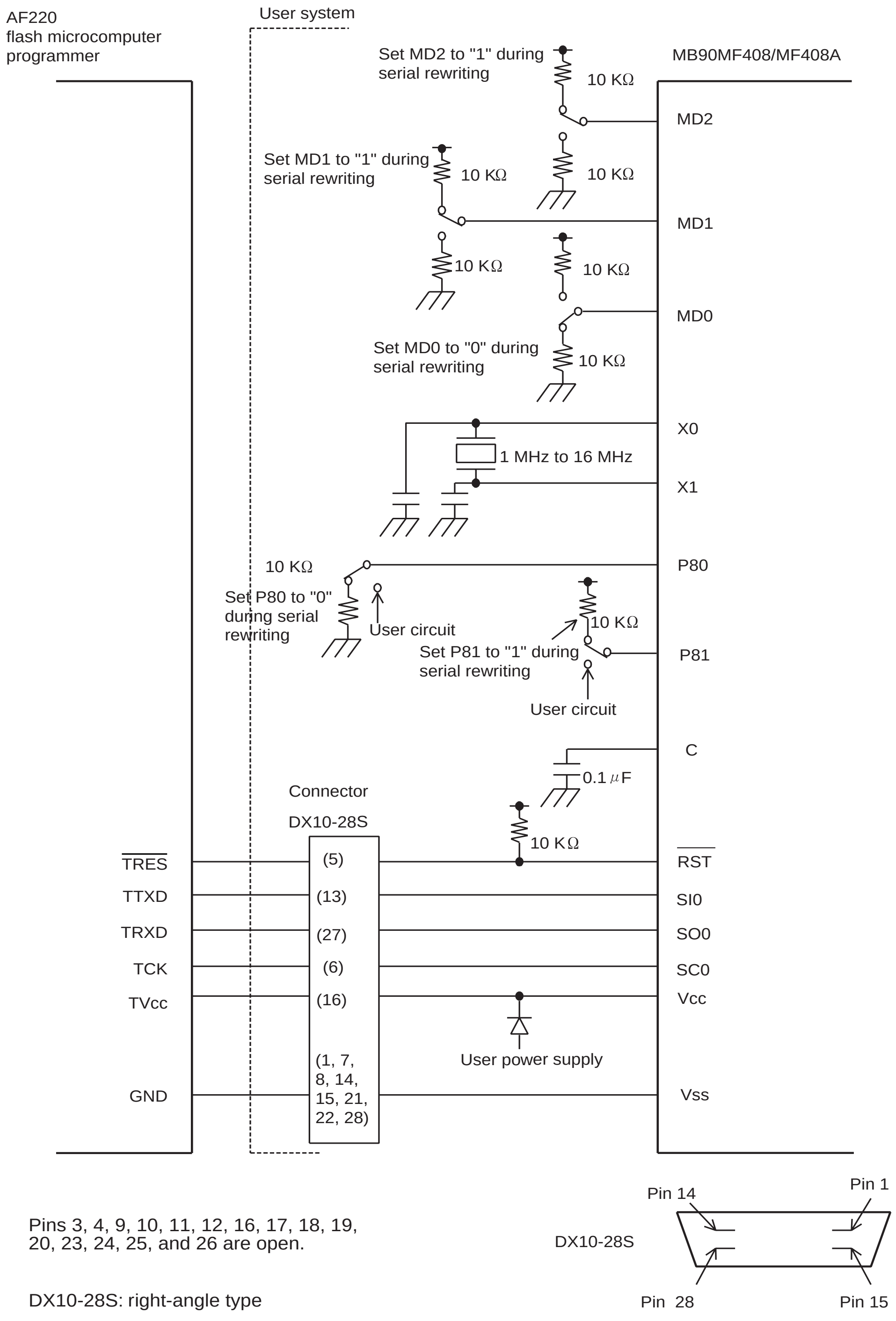
**Figure 24.4-1 "Example of Minimum Connection with Flash Microcomputer Programmer (When Power Supplied by User)" shows an example of minimum connection with the flash microcomputer programmer.**

## ■ Example of Minimum Connection with Flash Microcomputer Programmer (When Power Supplied by User)

If the pins are set as shown in Figure 24.4-1 "Example of Minimum Connection with Flash Microcomputer Programmer (When Power Supplied by User)" for writing data to the flash memory, then MD2, MD1, MD0, and P80 do not need to be connected to the flash microcomputer programmer.

## 24.4 Example of Minimum Connection with Flash Microcomputer Programmer (When Power Supplied from User)

**Figure 24.4-1 Example of Minimum Connection with Flash Microcomputer Programmer (When Power Supplied by User)**



Pins 3, 4, 9, 10, 11, 12, 16, 17, 18, 19, 20, 23, 24, 25, and 26 are open.

DX10-28S: right-angle type

Connector (made by Hirose Electronics) pin layout

- Turn off the user circuit power before connecting to the AF220.

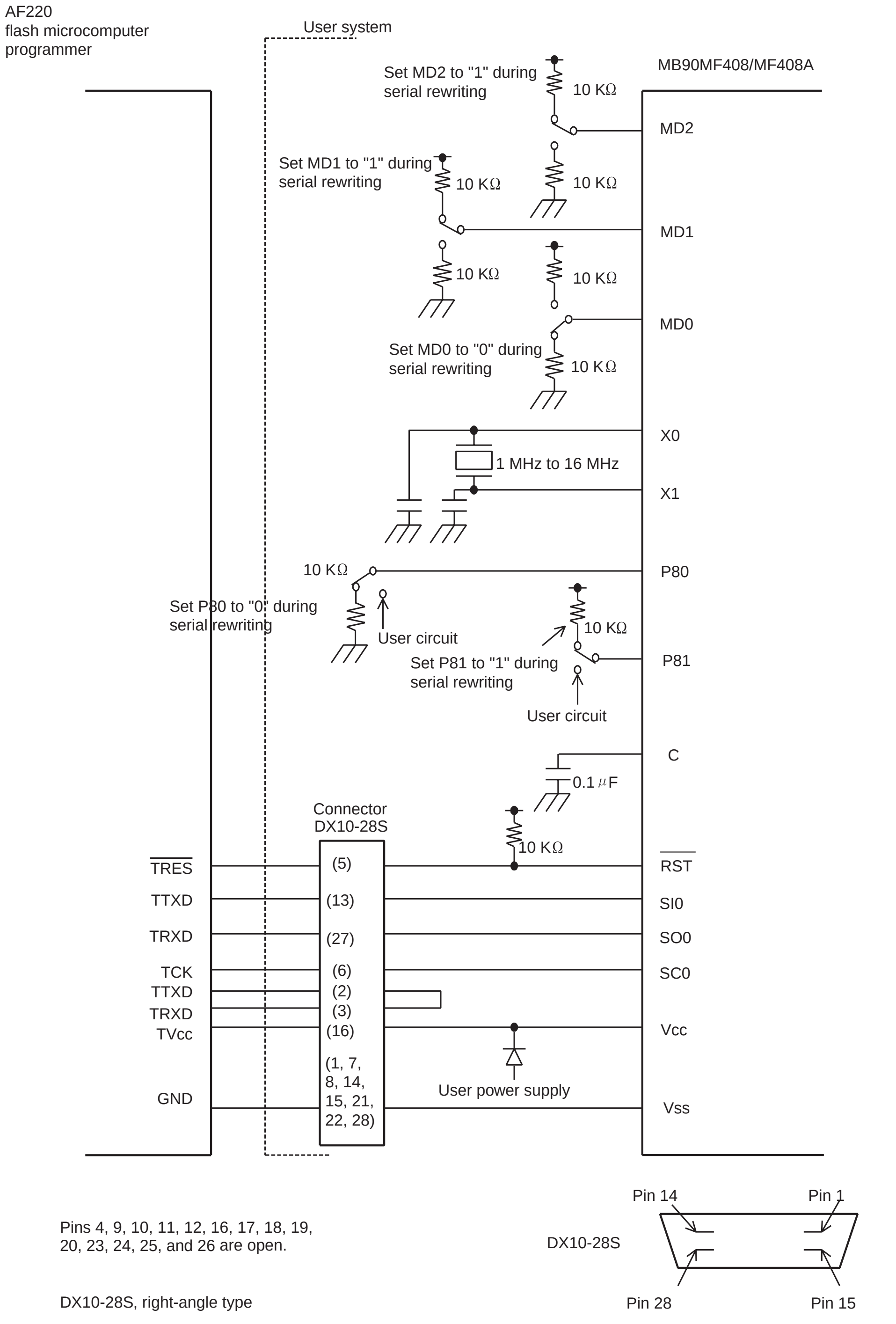
# 24.5 Example of Minimum Connection with Flash Microcomputer Programmer (When Power Supplied from Programmer)

**Figure 24.5-1 "Example of Minimum Connection with Flash Microcomputer Programmer (When Power Supplied from programmer)" shows an example of minimum connection with the flash microcomputer programmer.**

## ■ Example of Minimum Connection with Flash Microcomputer Programmer (When Power Supplied from programmer)

If data is written to the flash memory with the pin settings as shown in Figure 24.5-1 "Example of Minimum Connection with Flash Microcomputer Programmer (When Power Supplied from programmer)", MD2, MD1, MD0, and P80 do not need to be connected to the flash microcomputer programmer.

**Figure 24.5-1 Example of Minimum Connection with Flash Microcomputer Programmer (When Power Supplied from programmer)**



Pins 4, 9, 10, 11, 12, 16, 17, 18, 19, 20, 23, 24, 25, and 26 are open.

DX10-28S, right-angle type

Connector (made by Hirose Electronics) pin layout

- Before connecting the AF220, turn off the power supplied by the user.
- When power is supplied from the AF220, do not create a short circuit to the power supplied by the user.

# APPENDIX

**This appendix includes I/O maps, instruction lists, and other information.**

# APPENDIX A   I/O Map

**Table A-1 lists the addresses assigned to the registers for peripheral functions in the MB90M405 series.**

## ■ I/O Map

**Table A-1  I/O Map**

| Address | Abbreviation | Register | Read/write | Resource name | Initial value |
|---|---|---|---|---|---|
| $000000_H$ to $000007_H$ | Prohibited area | | | | |
| $000008_H$ | PDR8 | Port 8 data register | R/W | Port 8 | $XXXXXXXX_B$ |
| $000009_H$ | PDR9 | Port 9 data register | R/W | Port 9 | $XXXXXXXX_B$ |
| $00000A_H$ | PDRA | Port A data register | R/W | Port A | $XXXXXXXX_B$ |
| $00000B_H$ | PDRB | Port B data register | R/W | Port B | $XXXXXXXX_B$ |
| $00000C_H$ to $000017_H$ | Prohibited area | | | | |
| $000018_H$ | DDR8 | Port 8 direction register | R/W | Port 8 | $00000000_B$ |
| $000019_H$ | DDR9 | Port 9 direction register | R/W | Port 9 | $XXXXXX00_B$ |
| $00001A_H$ | DDRA | Port A direction register | R/W | Port A | $00000000_B$ |
| $00001B_H$ | DDRB | Port B direction register | R/W | Port B | $00000000_B$ |
| $00001C_H$ to $00001D_H$ | Prohibited area | | | | |
| $00001E_H$ | ADER0 | Analog input enable register 0 | R/W | Port A, A/D | $11111111_B$ |
| $00001F_H$ | ADER1 | Analog input enable register 1 | R/W | Port B, A/D | $11111111_B$ |
| $000020_H$ | SMR0 | Mode register ch0 | R/W | UART ch0 | $00000X00_B$ |
| $000021_H$ | SCR0 | Control register ch0 | R/W | | $00000100_B$ |
| $000022_H$ | SIDR0 | Input data register ch0 | R | | $XXXXXXXX_B$ |
| | SODR0 | Output data register ch0 | W | | |
| $000023_H$ | SSR0 | Status register ch0 | R/W | | $00001000_B$ |

Table A-1  I/O Map (Continued)

| Address | Abbreviation | Register | Read/write | Resource name | Initial value |
|---|---|---|---|---|---|
| $000024_H$ | SMR1 | Mode register ch1 | R/W | UART ch1 | $00000X00_B$ |
| $000025_H$ | SCR1 | Control register ch1 | R/W | | $00000100_B$ |
| $000026_H$ | SIDR1 | Input data register ch1 | R | | $XXXXXXXX_B$ |
| | SODR1 | Output data register ch1 | W | | |
| $000027_H$ | SSR1 | Status register ch1 | R/W | | $00001000_B$ |
| $000028_H$ | CDCR0 | Communication prescaler control register ch0 | R/W | Communication prescaler 0 | $0XXX0000_B$ |
| $000029_H$ | CDCR1 | Communication prescaler control register ch1 | R/W | Communication prescaler 1 | $0XXX0000_B$ |
| $00002A_H$ | IBSR | $I^2C$ status register | R | $I^2C$ interface | $00000000_B$ |
| $00002B_H$ | IBCR | $I^2C$ control register | R/W | | $00000000_B$ |
| $00002C_H$ | ICCR | $I^2C$ clock control register | R/W | | $XX0XXXXX_B$ |
| $00002D_H$ | IADR | $I^2C$ address register | R/W | | $XXXXXXXX_B$ |
| $00002E_H$ | IDAR | $I^2C$ data register | R/W | | $XXXXXXXX_B$ |
| $00002F_H$ | ISEL | $I^2C$ port selection register | R/W | | $XXXXXXX0_B$ |
| $000030_H$ | ENIR | DTP/interrupt enable register | R/W | DTP/external interrupt | $XXXX0000_B$ |
| $000031_H$ | EIRR | DTP/interrupt cause register | R/W | | $XXXXXXXX_B$ |
| $000032_H$ | ELVR | Request level setting register | R/W | | $00000000_B$ |
| $000033_H$ | Prohibited area | | | | |
| $000034_H$ | ADCS0 | A/D control status register 0 (lower) | R/W | A/D converter | $00XXXXXX_B$ |
| $000035_H$ | ADCS1 | A/D control status register 1 (upper) | R/W | | $00000000_B$ |
| $000036_H$ | ADCR0 | A/D data register 0 (lower) | R/W | | $XXXXXXXX_B$ |
| $000037_H$ | ADCR1 | A/D data register 1 (upper) | R/W | | $00000XXX_B$ |
| $000038_H$ | Prohibited area | | | | |
| $000039_H$ | ADMR | A/D conversion channel select register | R/W | A/D converter | $00000000_B$ |
| $00003A_H$ to $00003F_H$ | Prohibited area | | | | |
| $000040_H$ | TCCS | Timer counter control status register | R/W | 16-bit free-running timer | $00000000_B$ |
| $000041_H$ | Prohibited area | | | | |

Table A-1 I/O Map (Continued)

| Address | Abbreviation | Register | Read/write | Resource name | Initial value |
|---|---|---|---|---|---|
| $000042_H$ | TCDT | Timer counter data register | R/W | 16-bit free-running timer | $00000000_B$ |
| $000043_H$ | | | | | $00000000_B$ |
| $000044_H$ | IPC0 | Input capture data register ch0 | R | Input capture | $XXXXXXXX_B$ |
| $000045_H$ | | | | | $XXXXXXXX_B$ |
| $000046_H$ | IPC1 | Input capture data register ch1 | R | | $XXXXXXXX_B$ |
| $000047_H$ | | | | | $XXXXXXXX_B$ |
| $000048_H$ | ICSO1 | Input capture control status register | R/W | | $00000000_B$ |
| $000049_H$ | Prohibited area | | | | |
| $00004A_H$ | OCCP0 | Output compare register | R/W | Output compare | $XXXXXXXX_B$ |
| $00004B_H$ | | | | | $XXXXXXXX_B$ |
| $00004C_H$ | OCS0 | Output compare control status register | R/W | | $XX00XXX0_B$ |
| $00004D_H$ | Reserved area | | | | |
| $00004E_H$ to $00004F_H$ | Prohibited area | | | | |
| $000050_H$ | TMCSR0 | Timer control status register ch0 | R/W | 16-bit reload timer ch0 | $00000000_B$ |
| $000051_H$ | | | | | $XXXX0000_B$ |
| $000052_H$ | TMR0/ TMRLR0 | 16-bit timer register ch0 (R)<br>16-bit reload register ch0 (W) | TMR0: R<br>TMRLR0: W | | $XXXXXXXX_B$ |
| $000053_H$ | | | | | $XXXXXXXX_B$ |
| $000054_H$ | TMCSR1 | Timer control status register ch1 | R/W | 16-bit reload timer ch1 | $00000000_B$ |
| $000055_H$ | | | | | $XXXX0000_B$ |
| $000056_H$ | TMR1/ TMRLR1 | 16-bit timer register ch1 (R)<br>16-bit reload register ch1 (W) | TMR1: R<br>TMRLR1: W | | $XXXXXXXX_B$ |
| $000057_H$ | | | | | $XXXXXXXX_B$ |
| $000058_H$ | TMCSR2 | Timer control status register ch2 | R/W | 16-bit reload timer ch2 | $00000000_B$ |
| $000059_H$ | | | | | $XXXX0000_B$ |
| $00005A_H$ | TMR2/ TMRLR2 | 16-bit timer register ch2 (R)<br>16-bit reload register ch2 (W) | TMR2: R<br>TMRLR2: W | | $XXXXXXXX_B$ |
| $00005B_H$ | | | | | $XXXXXXXX_B$ |
| $00005C_H$ to $00005F_H$ | Prohibited area | | | | |

Table A-1  I/O Map (Continued)

| Address | Abbreviation | Register | | Read/write | Resource name | Initial value |
|---|---|---|---|---|---|---|
| 000060$_H$ | SMCS2 | Serial mode control status register ch2 | | R/W | Serial I/O ch2 | XXXX0000$_B$ |
| 000061$_H$ | | | | | | 00000010$_B$ |
| 000062$_H$ | SDR2 | Serial shift data register ch2 | | R/W | | XXXXXXXX$_B$ |
| 000063$_H$ | Prohibited area | | | | | |
| 000064$_H$ | SMCS3 | Serial mode control status register ch3 | | R/W | Serial I/O ch3 | XXXX0000$_B$ |
| 000065$_H$ | | | | | | 00000010$_B$ |
| 000066$_H$ | SDR3 | Serial shift data register ch3 | | R/W | | XXXXXXXX$_B$ |
| 000067$_H$ | Prohibited area | | | | | |
| 000068$_H$ | FLC1 | Display control register 1 | | W | FL control circuit | XXXXXX00$_B$ |
| 000069$_H$ | FLC2 | Display control register 2 | | W | | 00000000$_B$ |
| 00006A$_H$ | FLDG | Digit setting register | | W | | 00000000$_B$ |
| 00006B$_H$ | FLDC | Digit count register | | W | | 00000000$_B$ |
| 00006C$_H$ | Prohibited area | | | | | |
| 00006D$_H$ | FLST | Status register/establishment register | | R | FL control circuit | XX1XXX00$_B$ |
| | | | | W | | 00XXXXXX$_B$ |
| 00006E$_H$ | Prohibited area | | | | | |
| 00006F$_H$ | ROMM | ROM mirroring function selection register | | W | ROM mirroring function selection module | XXXXXXX1$_B$ |
| 000070$_H$ to 000077$_H$ | SEGD0 to 7 | Segment dimmer setting register | | W | FL control circuit | XXXXXXXX$_B$ |
| 000078$_H$ | FLPD0 | Port register | FIP36 to 43 | W | | 00000000$_B$ |
| 000079$_H$ | FLPD1 | | FIP44 to 51 | W | | 00000000$_B$ |
| 00007A$_H$ | FLPD2 | | FIP52 to 59 | W | | 00000000$_B$ |
| 00007B$_H$ to 00009D$_H$ | Prohibited area | | | | | |
| 00009E$_H$ | PACSR | Program address detection control status register | | R/W | Address match detection circuit | 00000000$_B$ |
| 00009F$_H$ | DIRR | Delayed interrupt cause generation/release register | | R/W | Delayed interrupt | XXXXXXX0$_B$ |
| 0000A0$_H$ | LPMCR | Low power consumption mode control register | | R/W | Low power consumption control circuit | 00011000$_B$ |
| 0000A1$_H$ | CKSCR | Clock selection register | | R/W | | 11111100$_B$ |

**Table A-1  I/O Map (Continued)**

| Address | Abbreviation | Register | Read/write | Resource name | Initial value |
|---|---|---|---|---|---|
| $0000A2_H$<br>to<br>$0000A7_H$ | Prohibited area | | | | |
| $0000A8_H$ | WDTC | Watchdog timer control register | R/W | Watchdog timer | $XXXXX111_B$ |
| $0000A9_H$ | TBTC | Timebase timer control register | R/W | Timebase timer | $1XX00100_B$ |
| $0000AA_H$<br>to<br>$0000AD_H$ | Prohibited area | | | | |
| $0000AE_H$ | FMCS | Flash memory control status register | R/W | 1M-bit flash memory | $00000000_B$ |
| $0000AF_H$ | TMCS | Watch clock output control register | R/W | Watch clock division | $XXXXX000_B$ |
| $0000B0_H$ | ICR00 | Interrupt control register 00 (for writing) | W, R/W | Interrupt | $00000111_B$ |
| | | Interrupt control register 00 (for reading) | R, R/W | | $XX000111_B$ |
| $0000B1_H$ | ICR01 | Interrupt control register 01 (for writing) | W, R/W | | $00000111_B$ |
| | | Interrupt control register 01 (for reading) | R, R/W | | $XX000111_B$ |
| $0000B2_H$ | ICR02 | Interrupt control register 02 (for writing) | W, R/W | | $00000111_B$ |
| | | Interrupt control register 02 (for reading) | R, R/W | | $XX000111_B$ |
| $0000B3_H$ | ICR03 | Interrupt control register 03 (for writing) | W, R/W | | $00000111_B$ |
| | | Interrupt control register 03 (for reading) | R, R/W | | $XX000111_B$ |
| $0000B4_H$ | ICR04 | Interrupt control register 04 (for writing) | W, R/W | | $00000111_B$ |
| | | Interrupt control register 04 (for reading) | R, R/W | | $XX000111_B$ |
| $0000B5_H$ | ICR05 | Interrupt control register 05 (for writing) | W, R/W | | $00000111_B$ |
| | | Interrupt control register 05 (for reading) | R, R/W | | $XX000111_B$ |
| $0000B6_H$ | ICR06 | Interrupt control register 06 (for writing) | W, R/W | | $00000111_B$ |
| | | Interrupt control register 06 (for reading) | R, R/W | | $XX000111_B$ |

Table A-1 I/O Map (Continued)

| Address | Abbreviation | Register | Read/write | Resource name | Initial value |
|---|---|---|---|---|---|
| $0000B7_H$ | ICR07 | Interrupt control register 07 (for writing) | W, R/W | Interrupt | $00000111_B$ |
| | | Interrupt control register 07 (for reading) | R, R/W | | $XX000111_B$ |
| $0000B8_H$ | ICR08 | Interrupt control register 08 (for writing) | W, R/W | | $00000111_B$ |
| | | Interrupt control register 08 (for reading) | R, R/W | | $XX000111_B$ |
| $0000B9_H$ | ICR09 | Interrupt control register 09 (for writing) | W, R/W | | $00000111_B$ |
| | | Interrupt control register 09 (for reading) | R, R/W | | $XX000111_B$ |
| $0000BA_H$ | ICR10 | Interrupt control register 10 (for writing) | W, R/W | | $00000111_B$ |
| | | Interrupt control register 10 (for reading) | R, R/W | | $XX000111_B$ |
| $0000BB_H$ | ICR11 | Interrupt control register 11 (for writing) | W, R/W | | $00000111_B$ |
| | | Interrupt control register 11 (for reading) | R, R/W | | $XX000111_B$ |
| $0000BC_H$ | ICR12 | Interrupt control register 12 (for writing) | W, R/W | | $00000111_B$ |
| | | Interrupt control register 12 (for reading) | R, R/W | | $XX000111_B$ |
| $0000BD_H$ | ICR13 | Interrupt control register 13 (for writing) | W, R/W | | $00000111_B$ |
| | | Interrupt control register 13 (for reading) | R, R/W | | $XX000111_B$ |
| $0000BE_H$ | ICR14 | Interrupt control register 14 (for writing) | W, R/W | | $00000111_B$ |
| | | Interrupt control register 14 (for reading) | R, R/W | | $XX000111_B$ |
| $0000BF_H$ | ICR15 | Interrupt control register 15 (for writing) | W, R/W | | $00000111_B$ |
| | | Interrupt control register 15 (for reading) | R, R/W | | $XX000111_B$ |
| $0000C0_H$ to $0000FF_H$ | Unused area | | | | |

**Table A-1  I/O Map (Continued)**

| Address | Abbreviation | Register | Read/write | Resource name | Initial value |
|---|---|---|---|---|---|
| 000100$_H$ to #$_H$ | RAM area | | | | |
| 001100$_H$ to 0011FF$_H$ | FL000 to 255 | Display data RAM | R/W | FL control circuit | XXXXXXXX$_B$ |
| 001200$_H$ to 001FEF$_H$ | Reserved area | | | | |
| 001FF0$_H$ | PADR0 | Program address detection register (lower) | R/W | Address match detection function | XXXXXXXX$_B$ |
| 001FF1$_H$ | | Program address detection register (middle) | R/W | | XXXXXXXX$_B$ |
| 001FF2$_H$ | | Program address detection register (upper) | R/W | | XXXXXXXX$_B$ |
| 001FF3$_H$ | PADR1 | Program address detection register (lower) | R/W | | XXXXXXXX$_B$ |
| 001FF4$_H$ | | Program address detection register (middle) | R/W | | XXXXXXXX$_B$ |
| 001FF5$_H$ | | Program address detection register (upper) | R/W | | XXXXXXXX$_B$ |
| 001FF6$_H$ to 001FFF$_H$ | Unused area | | | | |

❍ **Meaning of abbreviations used for reading and writing**

R/W:  Read/write enabled

R:  Read only

W:  Write only

❍ **Explanation of initial values**

0:  The initial value is 0.

1:  The initial value is 1.

X:  The initial value is undefined.

# APPENDIX B  Instructions

**Appendix B describes the instructions used by the F$^2$MC-16LX.**

# B.1 Instruction Types

**The F$^2$MC-16LX supports 351 types of instructions. Addressing is enabled by using an effective address field of each instruction or using the instruction code itself.**

## ■ Instruction Types

The F$^2$MC-16LX supports the following 351 types of instructions:

- 41 transfer instructions (byte)
- 38 transfer instructions (word or long word)
- 42 addition/subtraction instructions (byte, word, or long word)
- 12 increment/decrement instructions (byte, word, or long word)
- 11 comparison instructions (byte, word, or long word)
- 11 unsigned multiplication/division instructions (word or long word)
- 11 signed multiplication/division instructions (word or long word)
- 39 logic instructions (byte or word)
- 6 logic instructions (long word)
- 6 sign inversion instructions (byte or word)
- 1 normalization instruction (long word)
- 18 shift instructions (byte, word, or long word)
- 50 branch instructions
- 6 accumulator operation instructions (byte or word)
- 28 other control instructions (byte, word, or long word)
- 21 bit operation instructions
- 10 string instructions

# B.2 Addressing

**With the F$^2$MC-16LX, the address format is determined by the instruction effective address field or the instruction code itself (implied). When the address format is determined by the instruction code itself, specify an address in accordance with the instruction code used. Some instructions permit the user to select several types of addressing.**

## ■ Addressing

The F$^2$MC-16LX supports the following 23 types of addressing:

- Immediate (#imm)
- Register direct
- Direct branch address (addr16)
- Physical direct branch address (addr24)
- I/O direct (io)
- Abbreviated direct address (dir)
- Direct address (addr16)
- I/O direct bit address (io:bp)
- Abbreviated direct bit address (dir:bp)
- Direct bit address (addr16:bp)
- Vector address (#vct)
- Register indirect (@RWj j = 0 to 3)
- Register indirect with post increment (@RWj+ j = 0 to 3)
- Register indirect with displacement (@RWi + disp8 i = 0 to 7, @RWj+ disp16 j = 0 to 3)
- Long register indirect with displacement (@RLi + disp8 i = 0 to 3)
- Program counter indirect with displacement (@PC + disp16)
- Register indirect with base index (@RW0 + RW7, @RW1 + RW7)
- Program counter relative branch address (rel)
- Register list (rlst)
- Accumulator indirect (@A)
- Accumulator indirect branch address (@A)
- Indirectly-specified branch address (@ear)
- Indirectly-specified branch address (@eam)

## ■ Effective Address Field

Table B.2-1 "Effective Address Field" lists the address formats specified by the effective address field.

**Table B.2-1 Effective Address Field**

| Code | Representation | | | Address format | Default bank |
|---|---|---|---|---|---|
| 00 | R0 | RW0 | RL0 | Register direct: Individual parts correspond to the byte, word, and long word types in order from the left. | None |
| 01 | R1 | RW1 | (RL0) | | |
| 02 | R2 | RW2 | RL1 | | |
| 03 | R3 | RW3 | (RL1) | | |
| 04 | R4 | RW4 | RL2 | | |
| 05 | R5 | RW5 | (RL2) | | |
| 06 | R6 | RW6 | RL3 | | |
| 07 | R7 | RW7 | (RL3) | | |
| 08 | @RW0 | | | Register indirect | DTB |
| 09 | @RW1 | | | | DTB |
| 0A | @RW2 | | | | ADB |
| 0B | @RW3 | | | | SPB |
| 0C | @RW0+ | | | Register indirect with post increment | DTB |
| 0D | @RW1+ | | | | DTB |
| 0E | @RW2+ | | | | ADB |
| 0F | @RW3+ | | | | SPB |
| 10 | @RW0+disp8 | | | Register indirect with 8-bit displacement | DTB |
| 11 | @RW1+disp8 | | | | DTB |
| 12 | @RW2+disp8 | | | | ADB |
| 13 | @RW3+disp8 | | | | SPB |
| 14 | @RW4+disp8 | | | Register indirect with 8-bit displacement | DTB |
| 15 | @RW5+disp8 | | | | DTB |
| 16 | @RW6+disp8 | | | | ADB |
| 17 | @RW7+disp8 | | | | SPB |
| 18 | @RW0+disp16 | | | Register indirect with 16-bit displacement | DTB |
| 19 | @RW1+disp16 | | | | DTB |
| 1A | @RW2+disp16 | | | | ADB |
| 1B | @RW3+disp16 | | | | SPB |
| 1C | @RW0+RW7 | | | Register indirect with index | DTB |
| 1D | @RW1+RW7 | | | Register indirect with index | DTB |
| 1E | @PC+disp16 | | | PC indirect with 16-bit displacement | PCB |
| 1F | addr16 | | | Direct address | DTB |

# B.3 Direct Addressing

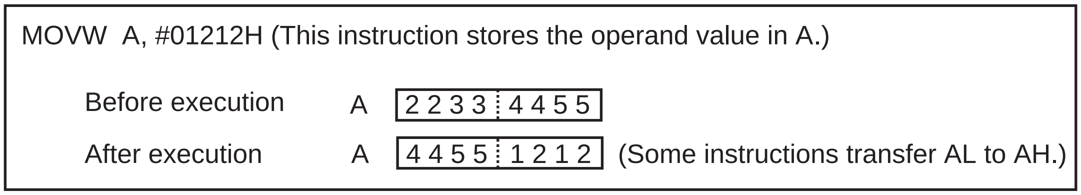
**An operand value, register, or address is specified explicitly in direct addressing mode.**

## ■ Direct Addressing

### ❍ Immediate addressing (#imm)

Specify an operand value explicitly (#imm4/ #imm8/ #imm16/ #imm32).

**Figure B.3-1 Example of immediate addressing (#imm)**


MOVW A, #01212H (This instruction stores the operand value in A.)
Before execution
A
2 2 3 3
4 4 5 5
After execution
A
4 4 5 5
1 2 1 2
(Some instructions transfer AL to AH.)

### ❍ Register direct addressing

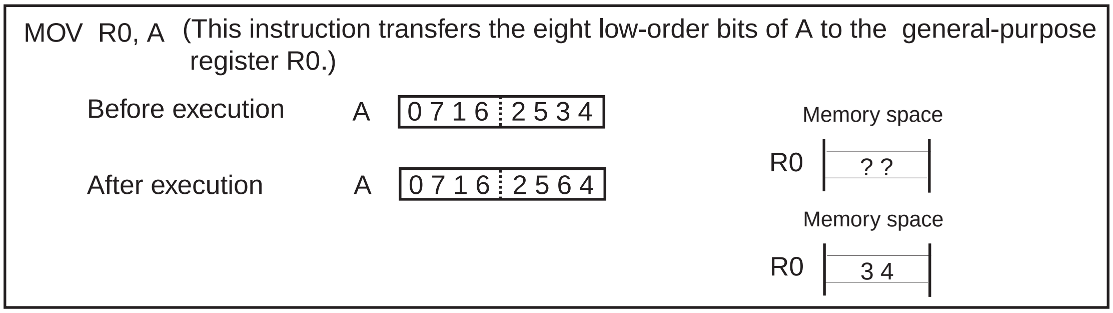
Specify a register explicitly as an operand. Table B.3-1 "Direct Addressing Registers" lists the registers that can be specified. Figure B.3-2 "Example of Register Direct Addressing" shows an example of register direct addressing.

**Table B.3-1 Direct Addressing Registers**

| | | |
|---|---|---|
| General-purpose register | Byte | R0, R1, R2, R3, R4, R5, R6, R7 |
| | Word | RW0, RW1, RW2, RW3, RW4, R5W, RW6, RW7 |
| | Long word | RL0, RL1, RL2, RL3 |
| Special-purpose register | Accumulator | A, AL |
| | Pointer | SP [*1] |
| | Bank | PCB, DTB, USB, SSB, ADB |
| | Page | DPR |
| | Control | PS, CCR, RP, ILM |

*1: One of the user stack pointer (USP) and system stack pointer (SSP) is selected and used depending on the value of the S flag bit in the condition code register (CCR). For branch instructions, the program counter (PC) is not specified in an instruction operand but is specified implicitly.

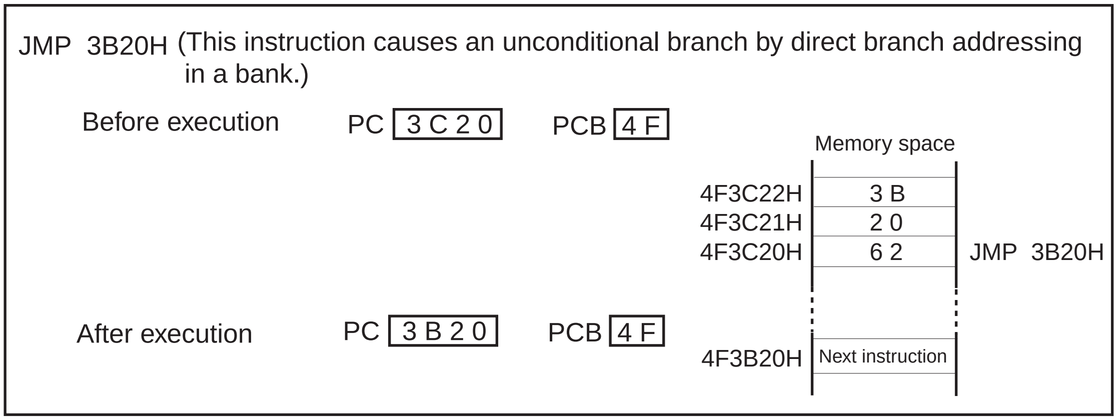
**Figure B.3-2  Example of Register Direct Addressing**


MOV  R0, A  (This instruction transfers the eight low-order bits of A to the general-purpose register R0.)
Before execution
A
0 7 1 6 2 5 3 4
After execution
A
0 7 1 6 2 5 6 4
Memory space
R0
? ?
Memory space
R0
3 4

### ❍ Direct branch addressing (addr16)

Specify an offset explicitly for the branch destination address.  The size of the offset is 16 bits, which indicates the branch destination in the logical address space.  Direct branch addressing is used for an unconditional branch, subroutine call, or software interrupt instruction.  Bits 23 to 16 of the address are specified by the program bank register (PCB).

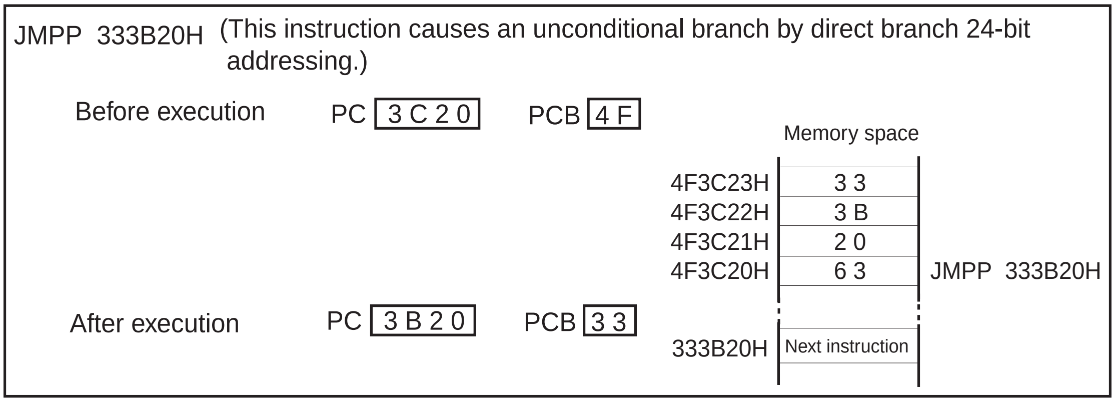
**Figure B.3-3  Example of Direct Branch Addressing (addr16)**



### ❍ Physical direct branch addressing (addr24)

Specify an offset explicitly for the branch destination address.  The size of the offset is 24 bits. Physical direct branch addressing is used for unconditional branch, subroutine call, or software interrupt instruction.

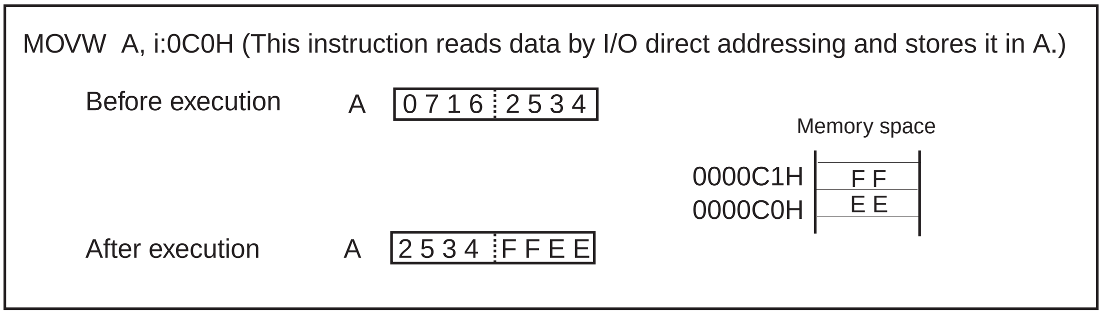
**Figure B.3-4  Example of Direct Branch Addressing (addr24)**



### ❍ I/O direct addressing (io)

Specify an 8-bit offset explicitly for the memory address in an operand. The I/O address space in the physical address space from 000000H to 0000FFH is accessed regardless of the data bank register (DTB) and direct page register (DPR). A bank select prefix for bank addressing is invalid if specified before an instruction using I/O direct addressing.

**Figure B.3-5 Example of I/O Direct Addressing (io)**

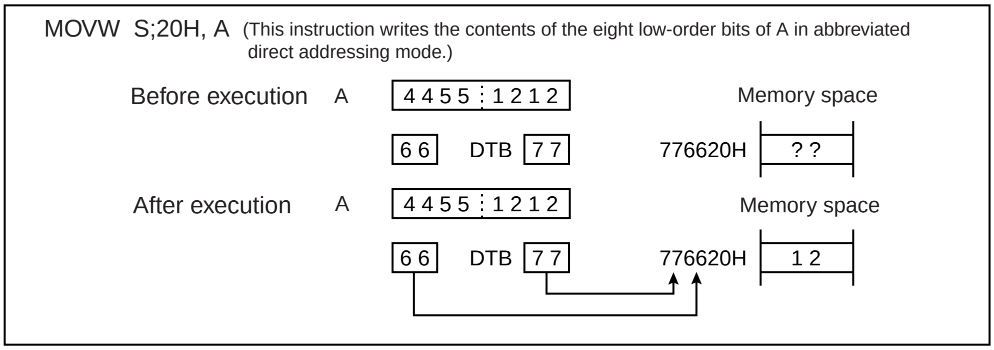


MOVW A, i:0C0H (This instruction reads data by I/O direct addressing and stores it in A.)

### ❍ Abbreviated direct addressing (dir)

Specify the eight low-order bits of a memory address explicitly in an operand. Address bits 8 to 15 are specified by the direct page register (DPR). Address bits 16 to 23 are specified by the data bank register (DTB).

**Figure B.3-6 Example of Abbreviated Direct Addressing (dir)**

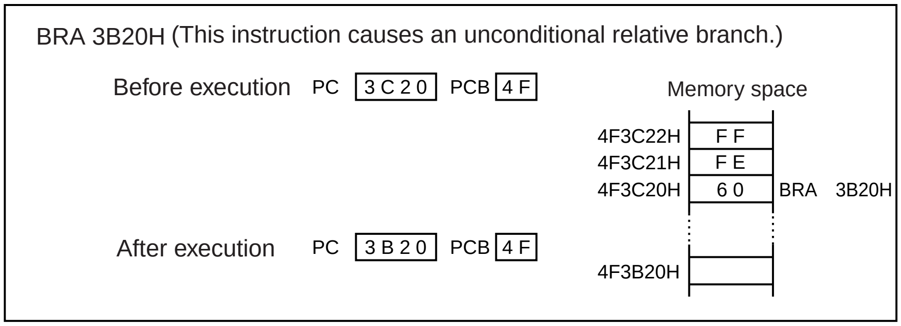


MOVW S;20H, A (This instruction writes the contents of the eight low-order bits of A in abbreviated direct addressing mode.)

### ❍ Direct addressing (addr16)

Specify the 16 low-order bits of a memory address explicitly in an operand. Address bits 16 to 23 are specified by the data bank register (DTB). A prefix instruction for access space addressing is invalid for this mode of addressing.

**Figure B.3-7 Example of Direct Addressing (addr16)**

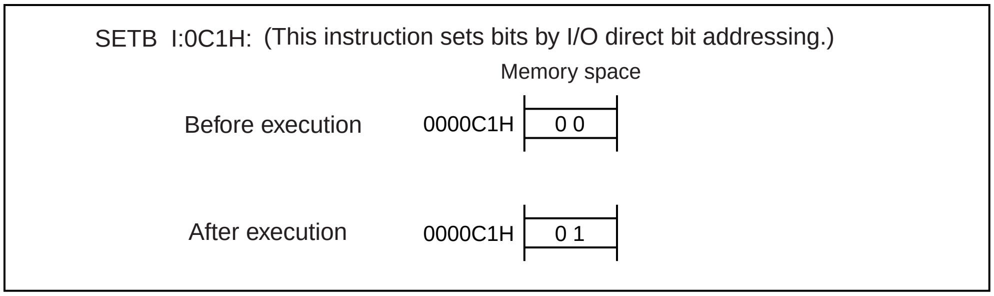


BRA 3B20H (This instruction causes an unconditional relative branch.)

### ❍ I/O direct bit addressing (io:bp)

Specify bits in physical addresses 000000H to 0000FFH explicitly.  Bit positions are indicated by ":bp", where the larger number indicates the most significant bit (MSB) and the lower number indicates the least significant bit (LSB).

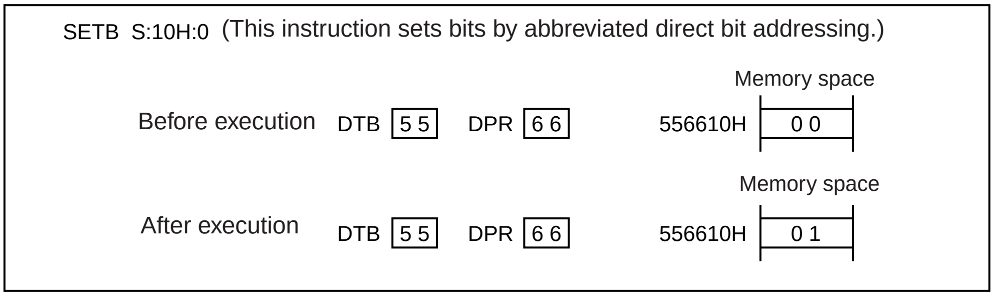
**Figure B.3-8  Example of I/O Direct Bit Addressing (io:bp)**


SETB  I:0C1H:  (This instruction sets bits by I/O direct bit addressing.)
Memory space
Before execution
0000C1H
0 0
After execution
0000C1H
0 1

### ❍ Abbreviated direct bit addressing (dir:bp)

Specify the eight low-order bits of a memory address explicitly in an operand.  Address bits 8 to 15 are specified by the direct page register (DPR).  Address bits 16 to 23 are specified by the data bank register (DTB).  Bit positions are indicated by ":bp", where the larger number indicates the most significant bit (MSB) and the lower number indicates the least significant bit (LSB).

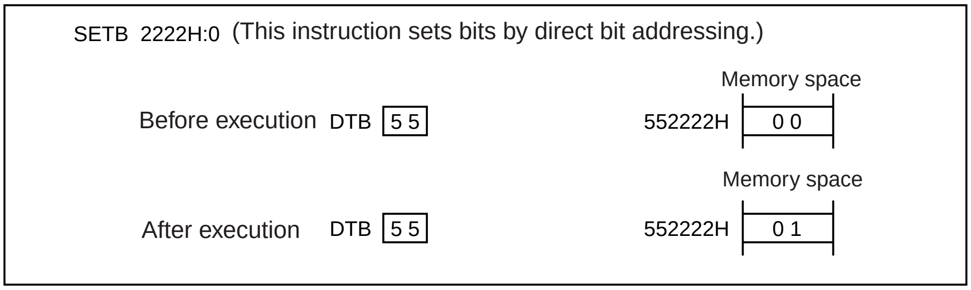
**Figure B.3-9  Example of Abbreviated Direct Bit Addressing (dir:bp)**



### ❍ Direct bit addressing (addr16:bp)

Specify arbitrary bits in 64 kilobytes explicitly.  Address bits 16 to 23 are specified by the data bank register (DTB).  Bit positions are indicated by ":bp", where the larger number indicates the most significant bit (MSB) and the lower number indicates the least significant bit (LSB).

**Figure B.3-10  Example of Direct Bit addressing (addr16:bp)**



### ❍ Vector Addressing (#vct)

Specify vector data in an operand to indicate the branch destination address. There are two sizes for vector numbers: 4 bits and 8 bits. Vector addressing is used for a subroutine call or software interrupt instruction.

**Figure B.3-11 Example of Vector Addressing (#vct)**

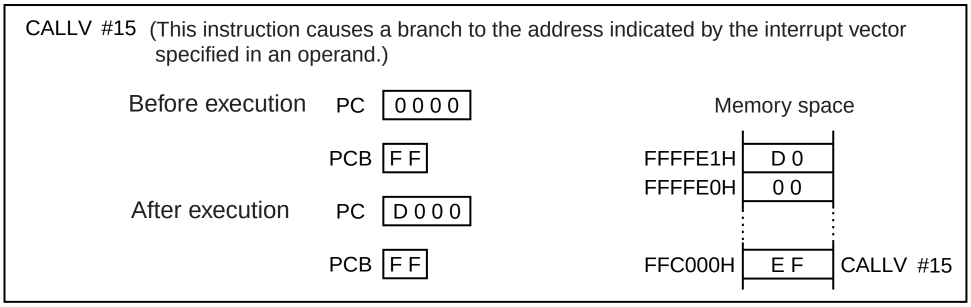

CALLV #15 (This instruction causes a branch to the address indicated by the interrupt vector specified in an operand.)
Before execution
PC
0 0 0 0
PCB
F F
After execution
PC
D 0 0 0
PCB
F F
Memory space
FFFFE1H
D 0
FFFFE0H
0 0
FFC000H
E F
CALLV #15

**Table B.3-2 CALLV Vector List**

| Instruction | Vector address L | Vector address H |
|---|---|---|
| CALLV #0 | XXFFFE$_H$ | XXFFFF$_H$ |
| CALLV #1 | XXFFFC$_H$ | XXFFFD$_H$ |
| CALLV #2 | XXFFFA$_H$ | XXFFFB$_H$ |
| CALLV #3 | XXFFF8$_H$ | XXFFF9$_H$ |
| CALLV #4 | XXFFF6$_H$ | XXFFF7$_H$ |
| CALLV #5 | XXFFF4$_H$ | XXFFF5$_H$ |
| CALLV #6 | XXFFF2$_H$ | XXFFF3$_H$ |
| CALLV #7 | XXFFF0$_H$ | XXFFF1$_H$ |
| CALLV #8 | XXFFEE$_H$ | XXFFEF$_H$ |
| CALLV #9 | XXFFEC$_H$ | XXFFED$_H$ |
| CALLV #10 | XXFFEA$_H$ | XXFFEB$_H$ |
| CALLV #11 | XXFFE8$_H$ | XXFFE9$_H$ |
| CALLV #12 | XXFFE6$_H$ | XXFFE7$_H$ |
| CALLV #13 | XXFFE4$_H$ | XXFFE5$_H$ |
| CALLV #14 | XXFFE2$_H$ | XXFFE3$_H$ |
| CALLV #15 | XXFFE0$_H$ | XXFFE1$_H$ |

Note: A PCB register value is set in XX.

**Note:**

When the program bank register (PCB) is FF$_H$, the vector area overlaps the vector area of INT #vct8 (#0 to #7). Use vector addressing carefully (see Table B.3-2 "CALLV Vector List").

# B.4 Indirect Addressing

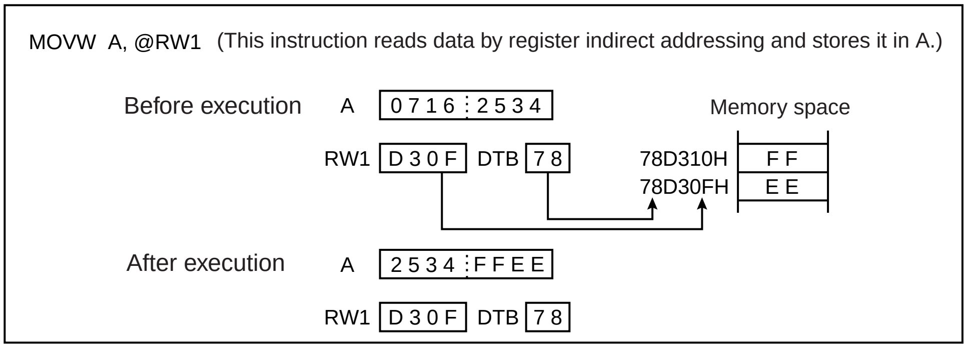
**In indirect addressing mode, an address is specified indirectly by the address data of an operand.**

## ■ Indirect Addressing

### ❍ Register indirect addressing (@RWj  j = 0 to 3)

Memory is accessed using the contents of general-purpose register RWj as an address. Address bits 16 to 23 are indicated by the data bank register (DTB) when RW0 or RW1 is used, system stack bank register (SSB) or user stack bank register (USB) when RW3 is used, or additional data bank register (ADB) when RW2 is used.

**Figure B.4-1  Example of Register Indirect Addressing (@RWj  j = 0 to 3)**



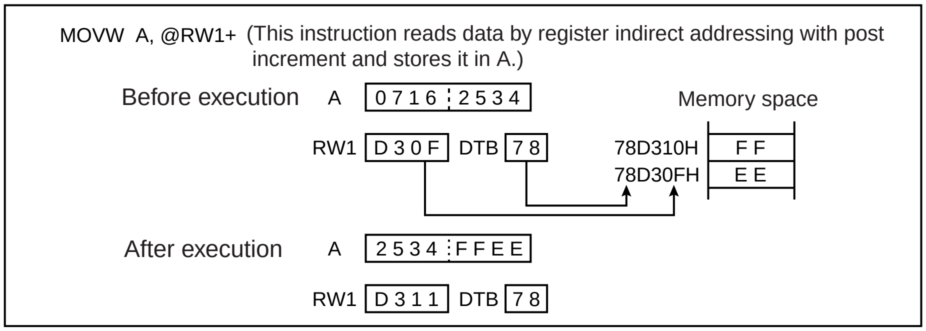
MOVW  A, @RW1  (This instruction reads data by register indirect addressing and stores it in A.)

### ❍ Register indirect addressing with post increment (@RWj+  j = 0 to 3)

Memory is accessed using the contents of general-purpose register RWj as an address. After operand operation, RWj is incremented by the operand size (1 for a byte, 2 for a word, or 4 for a long word). Address bits 16 to 23 are indicated by the data bank register (DTB) when RW0 or RW1 is used, system stack bank register (SSB) or user stack bank register (USB) when RW3 is used, or additional data bank register (ADB) when RW2 is used.

If the post increment results in the address of the register that specifies the increment, the incremented value is referenced after that. In this case, if the next instruction is a write instruction, priority is given to writing by an instruction and, therefore, the register that would be incremented becomes write data.

**Figure B.4-2 Example of Register Indirect Addressing with Post Increment (@RWj + j = 0 to 3)**

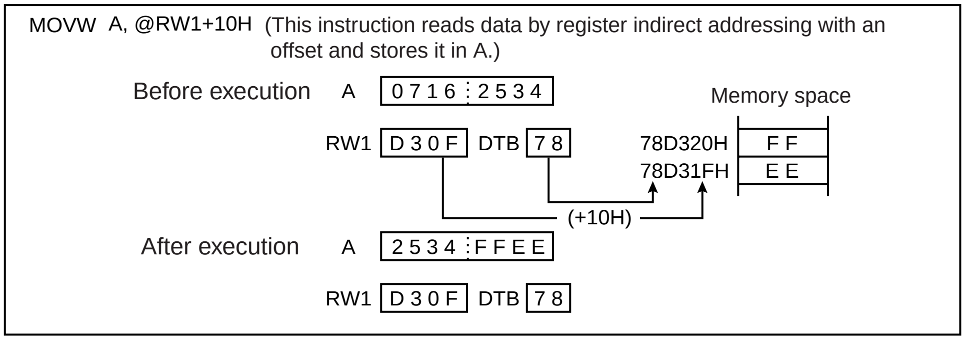


MOVW A, @RW1+ (This instruction reads data by register indirect addressing with post increment and stores it in A.)

## ❍ Register indirect addressing with offset (@RWi + disp8 i = 0 to 7, @RWj + disp16 j = 0 to 3)

Memory is accessed using the address obtained by adding an offset to the contents of general-purpose register RWj. Two types of offset, byte and word offsets, are used. They are added as signed numeric values. Address bits 16 to 23 are indicated by the data bank register (DTB) when RW0, RW1, RW4, or RW5 is used, system stack bank register (SSB) or user stack bank register (USB) when RW3 or RW7 is used, or additional data bank register (ADB) when RW2 or RW6 is used.

**Figure B.4-3 Example of Register Indirect Addressing with Offset (@RWi + disp8 i = 0 to 7, @RWj + disp16 j = 0 to 3)**

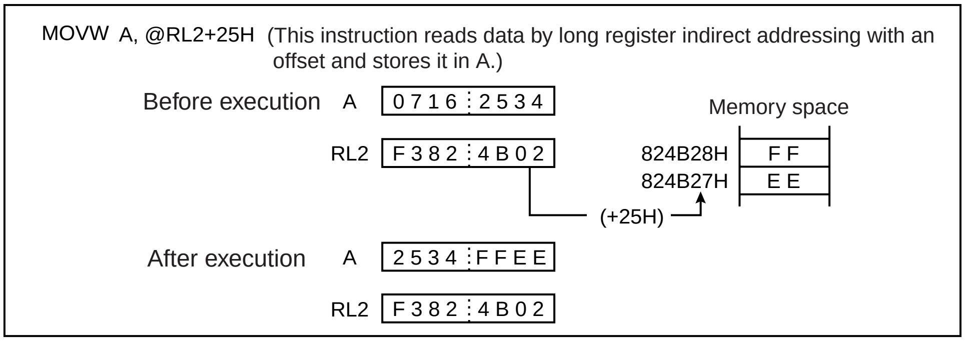


MOVW A, @RW1+10H (This instruction reads data by register indirect addressing with an offset and stores it in A.)

## ❍ Long register indirect addressing with offset (@RLi + disp8 i = 0 to 3)

Memory is accessed using the address that is the 24 low-order bits obtained by adding an offset to the contents of general-purpose register RLi. The offset is 8-bits long and is added as a signed numeric value.

**Figure B.4-4 Example of Long Register Indirect Addressing with Offset (@RLi + disp8 i = 0 to 3)**



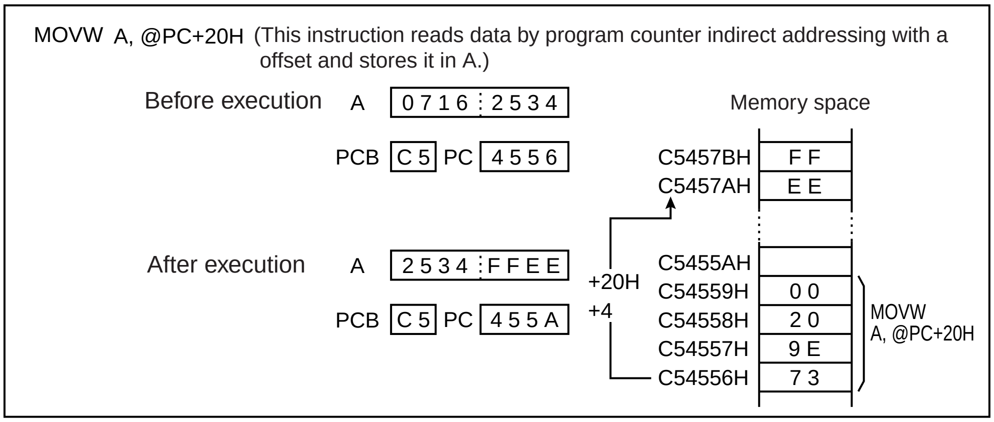
MOVW A, @RL2+25H (This instruction reads data by long register indirect addressing with an offset and stores it in A.)

### ❍ Program counter indirect addressing with offset (@PC + disp16)

Memory is accessed using the address indicated by (instruction address + 4 + disp16). The offset is one word long. Address bits 16 to 23 are specified by the program bank register (PCB). Note that the operand address of each of the following instructions is not deemed to be (next instruction address + disp16):

- DBNZ eam, rel
- DWBNZ eam, rel
- CBNE eam, #imm8, rel
- CWBNE eam, #imm16, rel
- MOV eam, #imm8
- MOVW eam, #imm16

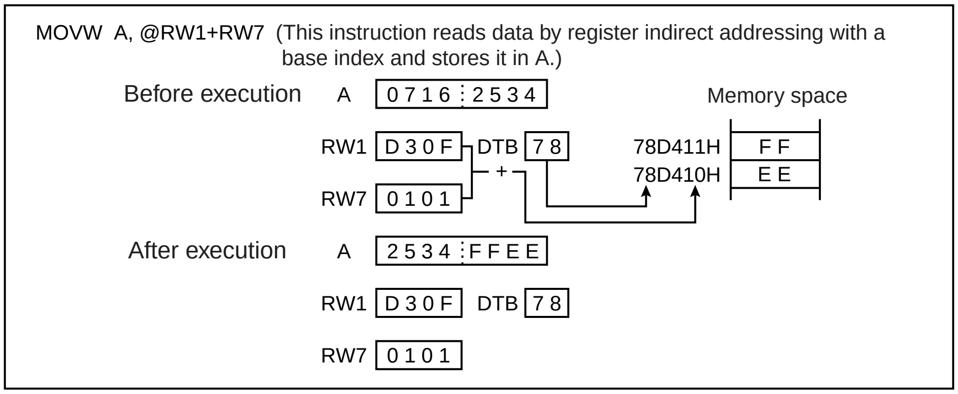
**Figure B.4-5 Example of Program Counter Indirect Addressing with Offset (@PC + disp16)**



### ❍ Register indirect addressing with base index (@RW0 + RW7, @RW1 + RW7)

Memory is accessed using the address determined by adding RW0 or RW1 to the contents of general-purpose register RW7. Address bits 16 to 23 are indicated by the data bank register (DTB).

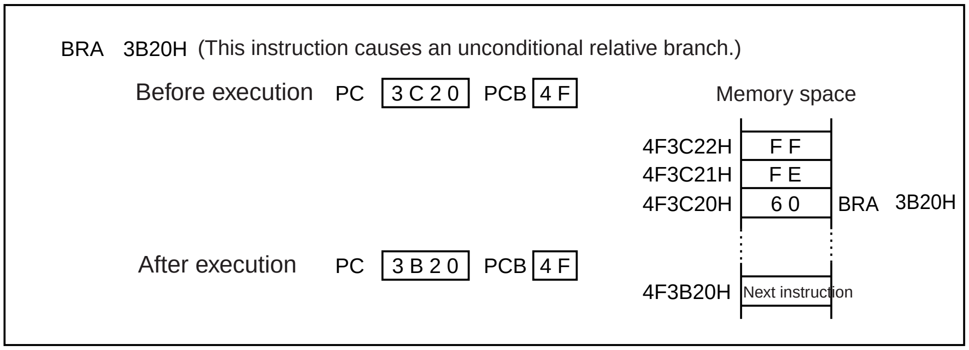
**Figure B.4-6 Example of Register Indirect Addressing with Base Index (@RW0 + RW7, @RW1 + RW7)**



### ❍ Program counter relative branch addressing (rel)

The address of the branch destination is a value determined by adding an 8-bit offset to the program counter (PC) value. If the result of addition exceeds 16 bits, bank register incrementing or decrementing is not performed and the excess part is ignored, and therefore the address is contained within a 64-kilobyte bank. This addressing is used for both conditional and unconditional branch instructions. Address bits 16 to 23 are indicated by the program bank register (PCB).

**Figure B.4-7 Example of Program Counter Relative Branch Addressing (rel)**



### ❍ Register list (rlst)

Specify a register to be pushed onto or popped from a stack.

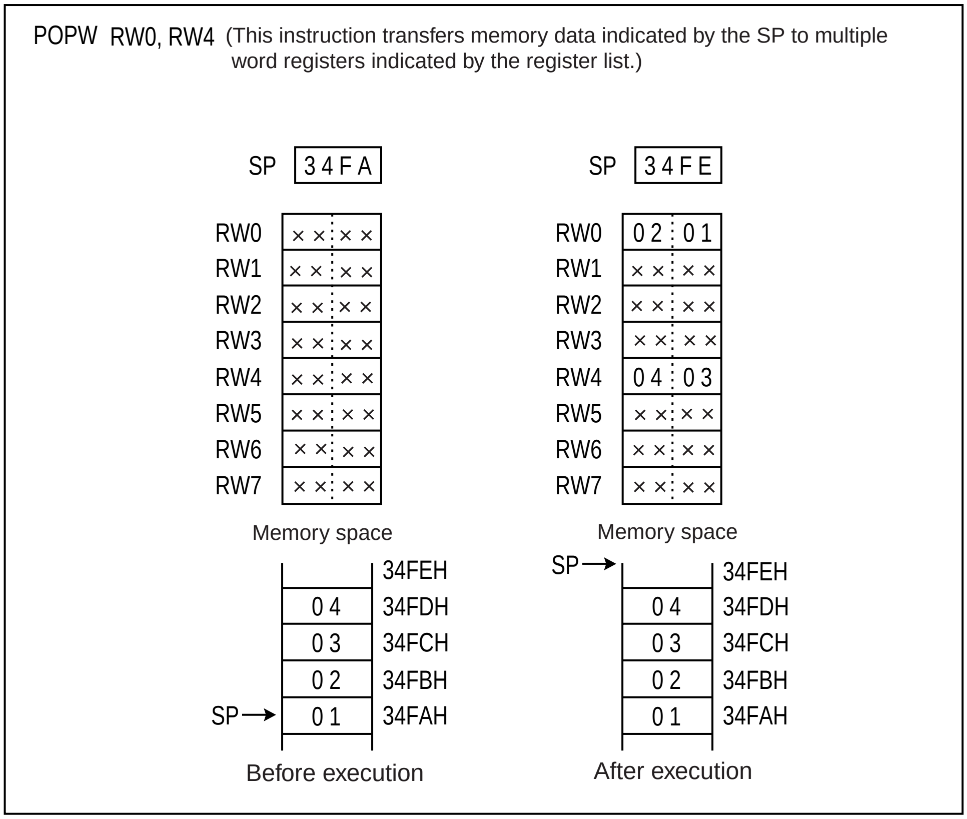
**Figure B.4-8 Configuration of the Register List**

MSB LSB

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| RW7 | RW6 | RW5 | RW4 | RW3 | RW2 | RW1 | RW0 |

A register is selected when the corresponding bit is 1 and deselected when the bit is 0.

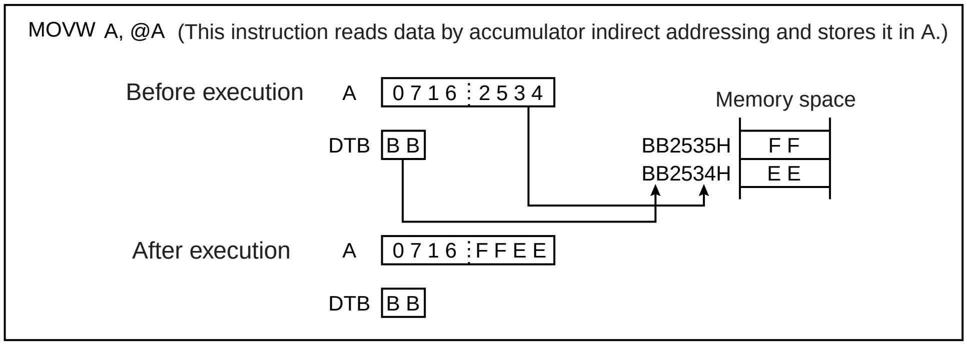
**Figure B.4-9 Example of Register List (rlst)**



## ❍ Accumulator indirect addressing (@A)

Memory is accessed using the address indicated by the contents of the low-order bytes (16 bits) of the accumulator (AL). Address bits 16 to 23 are specified by a mnemonic in the data bank register (DTB).

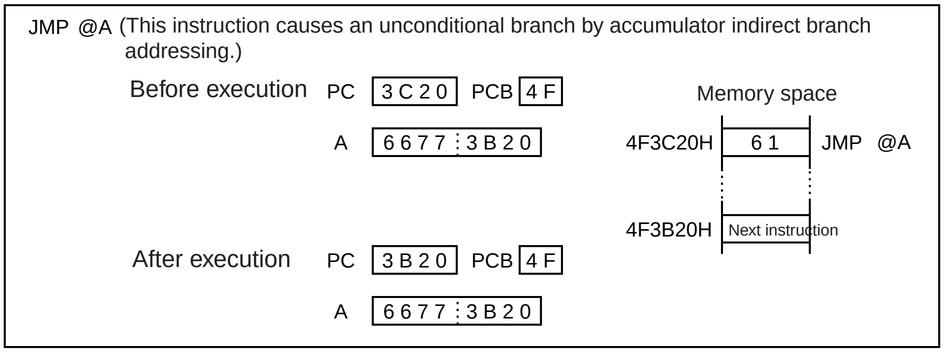
**Figure B.4-10 Example of Accumulator Indirect Addressing (@A)**



### ❍ Accumulator indirect branch addressing (@A)

The address of the branch destination is the content (16 bits) of the low-order bytes (AL) of the accumulator. It indicates the branch destination in the bank address space. Address bits 16 to 23 are specified by the program bank register (PCB). For the Jump Context (JCTX) instruction, however, address bits 16 to 23 are specified by the data bank register (DTB). This addressing is used for unconditional branch instructions.

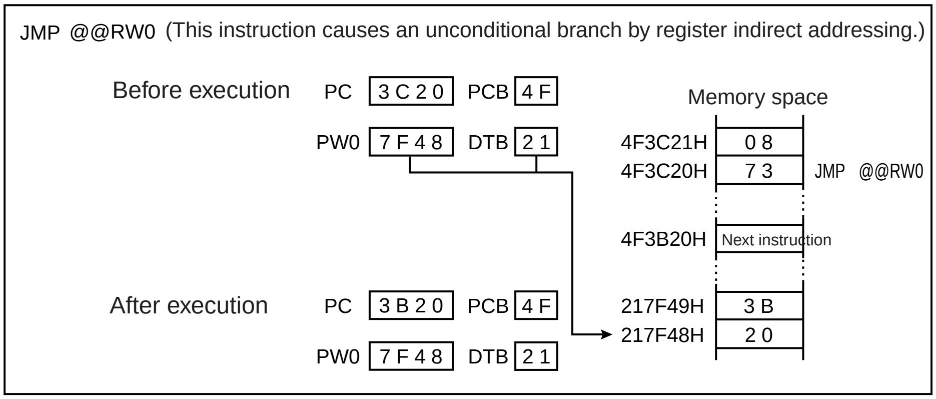
**Figure B.4-11 Example of Accumulator Indirect Branch Addressing (@A)**



### ❍ Indirect specification branch addressing (@ear)

The address of the branch destination is the word data at the address indicated by ear.

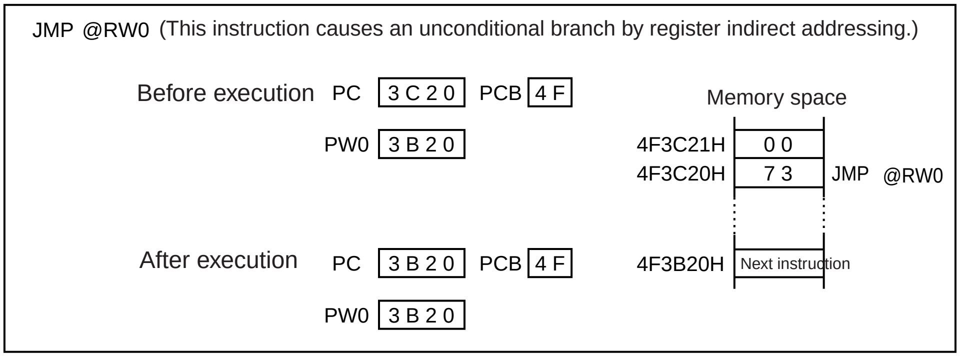
**Figure B.4-12 Example of Indirect Specification Branch Addressing (@ear)**



### ❍ Indirect specification branch addressing (@eam)

The address of the branch destination is the word data at the address indicated by eam.

**Figure B.4-13 Example of Indirect Specification Branch Addressing (@eam)**



# B.5 Execution Cycle Count

**The number of cycles required for instruction execution (execution cycle count) is obtained by adding the number of cycles required for each instruction, "correction value" determined by the condition, and the number of cycles for instruction fetch.**

## ■ Execution Cycle Count

The number of cycles required for instruction execution (execution cycle count) is obtained by adding the number of cycles required for each instruction, "correction value" determined by the condition, and the number of cycles for instruction fetch. In the mode of fetching an instruction from memory such as internal ROM connected to a 16-bit bus, the program fetches the instruction being executed in word increments. Therefore, intervening in data access increases the execution cycle count.

Similarly, in the mode of fetching an instruction from memory connected to an 8-bit external bus, the program fetches every byte of an instruction being executed. Therefore, intervening in data access increases the execution cycle count. In CPU intermittent operation mode, access to a general-purpose register, internal ROM, internal RAM, internal I/O, or external data bus causes the clock to the CPU to halt for the cycle count specified by the CG0 and CG1 bits of the low power consumption mode control register. Therefore, for the cycle count required for instruction execution in CPU intermittent operation mode, add the "access count x cycle count for the halt" as a correction value to the normal execution count.

## ■ Calculating the Execution Cycle Count

Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" lists execution cycle counts and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" and Table B.5-3 "Cycle Count Correction Values for Counting Instruction Fetch Cycles" summarize correction value data.

**Table B.5-1 Execution Cycle Counts in Each Addressing Mode**

| Code | Operand | (a) (*1) | Register access count in each addressing mode |
|---|---|---|---|
| | | Execution cycle count in each addressing mode | |
| 00<br>\|<br>07 | Ri<br>Rwi<br>RLi | See the instruction list. | See the instruction list. |
| 08<br>\|<br>0B | @RWj | 2 | 1 |
| 0C<br>\|<br>0F | @RWj+ | 4 | 2 |
| 10<br>\|<br>17 | @RWi+disp8 | 2 | 1 |
| 18<br>\|<br>1B | @RWi+disp16 | 2 | 1 |
| 1C<br>1D<br>1E<br>1F | @RW0+RW7<br>@RW1+RW7<br>@PC+disp16<br>addr16 | 4<br>4<br>2<br>1 | 2<br>2<br>0<br>0 |

*1: (a) is used for ～ (cycle count) and B (correction value) in B.8 "F$^2$MC-16LX Instruction List".

**Table B.5-2 Cycle Count Correction Values for Counting Execution Cycles**

| Operand | (b) byte (*1) | | (c) word (*1) | | (d) long (*1) | |
|---|---|---|---|---|---|---|
| | Cycle count | Access count | Cycle count | Access count | Cycle count | Access count |
| Internal register | +0 | 1 | +0 | 1 | +0 | 2 |
| Internal memory<br>Even address | +0 | 1 | +0 | 1 | +0 | 2 |
| Internal memory<br>Odd address | +0 | 1 | +2 | 2 | +4 | 4 |
| External data bus<br>16-bit even address | +1 | 1 | +1 | 1 | +2 | 2 |
| External data bus<br>16-bit odd address | +1 | 1 | +4 | 2 | +8 | 4 |
| External data bus<br>8-bits | +1 | 1 | +4 | 2 | +8 | 4 |

*1: (b), (c), and (d) are used for ∼ (cycle count) and B (correction value) in B.8 "$F^2$MC-16LX Instruction List".

**Note:**

When an external data bus is used, the cycle counts during which an instruction is made to wait by ready input or automatic ready must also be added.

**Table B.5-3 Cycle Count Correction Values for Counting Instruction Fetch Cycles**

| Instruction | Byte boundary | Word boundary |
|---|---|---|
| Internal memory | - | +2 |
| External data bus 16-bits | - | +3 |
| External data bus 8-bits | +3 | - |

**Note:**

- When an external data bus is used, the cycle counts during which an instruction is made to wait by ready input or automatic ready must also be added.
- Actually, instruction execution is not delayed by every instruction fetch. Therefore, use the correction values to calculate the worst case.

# B.6 Effective Address Field

**Table B.6-1 "Effective Address Field" shows the effective address field.**

## ■ Effective Address Field

Table B.6-1 Effective Address Field

| Code | Representation | | | Address format | Byte count of extended address part (*1) |
|---|---|---|---|---|---|
| 00<br>01<br>02<br>03<br>04<br>05<br>06<br>07 | R0<br>R1<br>R2<br>R3<br>R4<br>R5<br>R6<br>R7 | RW0<br>RW1<br>RW2<br>RW3<br>RW4<br>RW5<br>RW6<br>RW7 | RL0<br>(RL0)<br>RL1<br>(RL1)<br>RL2<br>(RL2)<br>RL3<br>(RL3) | Register direct: Individual parts correspond to the byte, word, and long word types in order from the left. | - |
| 08<br>09<br>0A<br>0B | @RW0<br>@RW1<br>@RW2<br>@RW3 | | | Register indirect | 0 |
| 0C<br>0D<br>0E<br>0F | @RW0+<br>@RW1+<br>@RW2+<br>@RW3+ | | | Register indirect with post increment | 0 |
| 10<br>11<br>12<br>13<br>14<br>15<br>16<br>17 | @RW0+disp8<br>@RW1+disp8<br>@RW2+disp8<br>@RW3+disp8<br>@RW4+disp8<br>@RW5+disp8<br>@RW6+disp8<br>@RW7+disp8 | | | Register indirect with 8-bit displacement | 1 |
| 18<br>19<br>1A<br>1B | @RW0+disp16<br>@RW1+disp16<br>@RW2+disp16<br>@RW3+disp16 | | | Register indirect with 16-bit displacement | 2 |
| 1C<br>1D<br>1E<br>1F | @RW0+RW7<br>@RW1+RW7<br>@PC+disp16<br>addr16 | | | Register indirect with index<br>Register indirect with index<br>PC indirect with 16-bit displacement<br>Direct address | 0<br>0<br>2<br>2 |

*1: Each byte count of the extended address part applies to + in the # (byte count) column in B.8 "F$^2$MC-16LX Instruction List".

# B.7 How to Read the Instruction List

**Table B.7-1 "Description of Items in the Instruction List" describes the items used in the F2MC-16LX Instruction List, and Table B.7-2 "Explanation on Symbols in the Instruction List" describes the symbols used in the same list.**

## ■ Description of instruction presentation items and symbols

Table B.7-1 Description of Items in the Instruction List

| Item | Description |
|---|---|
| Mnemonic | Uppercase, symbol: Represented as is in the assembler.<br>Lowercase: Rewritten in the assembler.<br>Number of following lowercase: Indicates bit length in the instruction. |
| # | Indicates the number of bytes. |
| ~ | Indicates the number of cycles.<br>See Table B.2-1 "Effective Address Field" for the alphabetical letters in items. |
| RG | Indicates the number of times a register access is performed during instruction execution.<br>The number is used to calculate the correction value for CPU intermittent operation. |
| B | Indicates the correction value used to calculate the actual number of cycles during instruction execution.<br>The actual number of cycles during instruction execution can be determined by adding the value in the ~ column to this value. |
| Operation | Indicates the instruction operation. |
| LH | Indicates the special operation for bits 15 to 08 of the accumulator.<br>Z: Transfers 0.<br>X: Transfers after sign extension.<br>-: No transfer |
| AH | Indicates the special operation for the 16 high-order bits of the accumulator.<br>*: Transfers from AL to AH.<br>-: No transfer<br>Z: Transfers 00 to AH.<br>X: Transfers 00H or FFH to AH after AL sign extension. |

**Table B.7-1  Description of Items in the Instruction List (Continued)**

| Item | Description |
|---|---|
| I | Each indicates the state of each flag:  I (interrupt enable), S (stack), T (sticky bit), N (negative), Z (zero), V (overflow), C (carry).<br>*:  Changes upon instruction execution.<br>-:  No change<br>Z:  Set upon instruction execution.<br>X:  Reset upon instruction execution. |
| S | |
| T | |
| N | |
| Z | |
| V | |
| C | |
| RMW | Indicates whether the instruction is a Read Modify Write instruction (reading data from memory by the I instruction and writing the result to memory).<br>*:  Read Modify Write instruction<br>-:  Not Read Modify Write instruction<br>Note:  Cannot be used for an address that has different meanings between read and write operations. |

**Table B.7-2  Explanation on Symbols in the Instruction List**

| Symbol | Explanation |
|---|---|
| A | The bit length used varies depending on the 32-bit accumulator instruction.<br>Byte:  Low-order 8 bits of byte AL<br>Word:  16 bits of word AL<br>Long word:   32 bits of AL and AH |
| AH<br>AL | 16 high-order bits of A<br>16 low-order bits of A |
| SP | Stack pointer (USP or SSP) |
| PC | Program counter |
| PCB | Program bank register |
| DTB | Data bank register |
| ADB | Additional data bank register |
| SSB | System stack bank register |
| USB | User stack bank register |
| SPB | Current stack bank register (SSB or USB) |
| DPR | Direct page register |
| brg1 | DTB, ADB, SSB, USB, DPR, PCB, SPB |
| brg2 | DTB, ADB, SSB, USB, DPR, SPB |
| Ri | R0, R1, R2, R3, R4, R5, R6, R7 |

Table B.7-2 Explanation on Symbols in the Instruction List (Continued)

| Symbol | Explanation |
|---|---|
| RWi | RW0, RW1, RW2, RW3, RW4, RW5, RW6, RW7 |
| RWj | RW0, RW1, RW2, RW3 |
| RLi | RL0, RL1, RL2, RL3 |
| dir | Abbreviated direct addressing |
| addr16<br>addr24<br>ad24 0-15<br>ad24 16-23 | Direct addressing<br>Physical direct addressing<br>Bits 0 to 15 of addr24<br>Bits 16 to 23 of addr24 |
| io | I/O area (000000H to 0000FFH) |
| #imm4<br>#imm8<br>#imm16<br>#imm32<br>ext (imm8) | 4-bit immediate data<br>8-bit immediate data<br>16-bit immediate data<br>32-bit immediate data<br>16-bit data obtained by sign extension of 8-bit immediate data |
| disp8<br>disp16 | 8-bit displacement<br>16-bit displacement |
| bp | Bit offset |
| vct4<br>vct8 | Vector number (0 to 15)<br>Vector number (0 to 255) |
| ( ) b | Bit address |
| rel | PC relative branch |
| ear<br>eam | Effective addressing (code 00 to 07)<br>Effective addressing (code 08 to 1F) |
| rlst | Register list |

# B.8 F$^2$MC-16LX Instruction List

## Table B.8-1 "41 Transfer Instructions (byte)" to Table B.9-19 "MOVW ea, Rwi Instruction (first byte = 7DH)" list the instructions used by the F$^2$MC-16LX.

### ■ F$^2$MC-16LX Instruction List

Table B.8-1 41 Transfer Instructions (byte)

| Mnemonic | | # | ~ | RG | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MOV | A,dir | 2 | 3 | 0 | (b) | byte (A) <-- (dir) | Z | * | - | - | * | * | - | - | - | - |
| MOV | A,addr16 | 3 | 4 | 0 | (b) | byte (A) <-- (addr16) | Z | * | - | - | * | * | - | - | - | - |
| MOV | A,Ri | 1 | 2 | 1 | 0 | byte (A) <-- (Ri) | Z | * | - | - | * | * | - | - | - | - |
| MOV | A,ear | 2 | 2 | 1 | 0 | byte (A) <-- (ear) | Z | * | - | - | * | * | - | - | - | - |
| MOV | A,eam | 2+ | 3+(a) | 0 | (b) | byte (A) <-- (eam) | Z | * | - | - | * | * | - | - | - | - |
| MOV | A,io | 2 | 3 | 0 | (b) | byte (A) <-- (io) | Z | * | - | - | * | * | - | - | - | - |
| MOV | A,#imm8 | 2 | 2 | 0 | 0 | byte (A) <-- imm8 | Z | * | - | - | * | * | - | - | - | - |
| MOV | A,@A | 2 | 3 | 0 | (b) | byte (A) <-- ((A)) | Z | - | - | - | * | * | - | - | - | - |
| MOV | A,@RLi+disp8 | 3 | 10 | 2 | (b) | byte (A) <-- ((RLi)+disp8) | Z | * | - | - | * | * | - | - | - | - |
| MOVN | A,#imm4 | 1 | 1 | 0 | 0 | byte (A) <-- imm4 | Z | * | - | - | R | * | - | - | - | - |
| MOVX | A,dir | 2 | 3 | 0 | (b) | byte (A) <-- (dir) | X | * | - | - | - | * | * | - | - | - |
| MOVX | A,addr16 | 3 | 4 | 0 | (b) | byte (A) <-- (addr16) | X | * | - | - | - | * | * | - | - | - |
| MOVX | A,Ri | 2 | 2 | 1 | 0 | byte (A) <-- (Ri) | X | * | - | - | - | * | * | - | - | - |
| MOVX | A,ear | 2 | 2 | 1 | 0 | byte (A) <-- (ear) | X | * | - | - | - | * | * | - | - | - |
| MOVX | A,eam | 2+ | 3+(a) | 0 | (b) | byte (A) <-- (eam) | X | * | - | - | - | * | * | - | - | - |
| MOVX | A,io | 2 | 3 | 0 | (b) | byte (A) <-- (io) | X | * | - | - | - | * | * | - | - | - |
| MOVX | A,#imm8 | 2 | 2 | 0 | 0 | byte (A) <-- imm8 | X | * | - | - | - | * | * | - | - | - |
| MOVX | A,@A | 2 | 3 | 0 | (b) | byte (A) <-- ((A)) | X | - | - | - | - | * | * | - | - | - |
| MOVX | A,@RWi+disp8 | 2 | 5 | 1 | (b) | byte (A) <-- ((RWi)+disp8) | X | * | - | - | - | * | * | - | - | - |
| MOVX | A,@RLi+disp8 | 3 | 10 | 2 | (b) | byte (A) <-- ((RLi)+disp8 | X | * | - | - | - | * | * | - | - | - |
| MOV | dir,A | 2 | 3 | 0 | (b) | byte (dir) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOV | addr16,A | 3 | 4 | 0 | (b) | byte (addr16) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOV | Ri,A | 1 | 2 | 1 | 0 | byte (Ri) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOV | ear,A | 2 | 2 | 1 | 0 | byte (ear) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOV | eam,A | 2+ | 3+(a) | 0 | (b) | byte (eam) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOV | io,A | 2 | 3 | 0 | (b) | byte (io) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOV | @RLi+disp8,A | 3 | 10 | 2 | (b) | byte ((RLi)+disp8) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOV | Ri,ear | 2 | 3 | 2 | 0 | byte (Ri) <-- (ear) | - | - | - | - | - | * | * | - | - | - |
| MOV | Ri,eam | 2+ | 4+(a) | 1 | (b) | byte (Ri) <-- (eam) | - | - | - | - | - | * | * | - | - | - |
| MOV | ear,Ri | 2 | 4 | 2 | 0 | byte (ear) <-- (Ri) | - | - | - | - | - | * | * | - | - | - |
| MOV | eam,Ri | 2+ | 5+(a) | 1 | (b) | byte (eam) <-- (Ri) | - | - | - | - | - | * | * | - | - | - |
| MOV | Ri,#imm8 | 2 | 2 | 1 | 0 | byte (Ri) <-- imm8 | - | - | - | - | - | * | * | - | - | - |
| MOV | io,#imm8 | 3 | 5 | 0 | (b) | byte (io) <-- imm8 | - | - | - | - | - | - | - | - | - | - |
| MOV | dir,#imm8 | 3 | 5 | 0 | (b) | byte (dir) <-- imm8 | - | - | - | - | - | - | - | - | - | - |
| MOV | ear,#imm8 | 3 | 2 | 1 | 0 | byte (ear) <-- imm8 | - | - | - | - | - | * | * | - | - | - |
| MOV | eam,#imm8 | 3+ | 4+(a) | 0 | (b) | byte (eam) <-- imm8 | - | - | - | - | - | - | - | - | - | - |
| MOV | @AL,AH/ MOV @A,T | 2 | 3 | 0 | (b) | byte ((A)) <-- (AH) | - | - | - | - | - | * | * | - | - | - |
| XCH | A,ear | 2 | 4 | 2 | 0 | byte (A) <--> (ear) | Z | - | - | - | - | - | - | - | - | - |
| XCH | A,eam | 2+ | 5+(a) | 0 | 2×(b) | byte (A) <--> (eam) | Z | - | - | - | - | - | - | - | - | - |
| XCH | Ri,ear | 2 | 7 | 4 | 0 | byte (Ri) <--> (ear) | - | - | - | - | - | - | - | - | - | - |
| XCH | Ri,eam | 2+ | 9+(a) | 2 | 2×(b) | byte (Ri) <--> (eam) | - | - | - | - | - | - | - | - | - | - |

**Note:**

See Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" for information on (a) to (d) in the table.

## Table B.8-2 38 Transfer Instructions (byte)

| Mnemonic | | # | ~ | RG | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MOVW | A,dir | 2 | 3 | 0 | (c) | word (A) <-- (dir) | - | * | - | - | - | * | * | - | - | - |
| MOVW | A,addr16 | 3 | 4 | 0 | (c) | word (A) <-- (addr16) | - | * | - | - | - | * | * | - | - | - |
| MOVW | A,SP | 1 | 1 | 0 | 0 | word (A) <-- (SP) | - | * | - | - | - | * | * | - | - | - |
| MOVW | A,RWi | 1 | 2 | 1 | 0 | word (A) <-- (RWi) | - | * | - | - | - | * | * | - | - | - |
| MOVW | A,ear | 2 | 2 | 1 | 0 | word (A) <-- (ear) | - | * | - | - | - | * | * | - | - | - |
| MOVW | A,eam | 2+ | 3+(a) | 0 | (c) | word (A) <-- (eam) | - | * | - | - | - | * | * | - | - | - |
| MOVW | A,io | 2 | 3 | 0 | (c) | word (A) <-- (io) | - | * | - | - | - | * | * | - | - | - |
| MOVW | A,@A | 2 | 3 | 0 | (c) | word (A) <-- ((A)) | - | - | - | - | - | * | * | - | - | - |
| MOVW | A,#imm16 | 3 | 2 | 0 | 0 | word (A) <-- imm16 | - | * | - | - | - | * | * | - | - | - |
| MOVW | A,@RWi+disp8 | 2 | 5 | 1 | (c) | word (A) <-- ((RWi)+disp8) | - | * | - | - | - | * | * | - | - | - |
| MOVW | A,@RLi+disp8 | 3 | 10 | 2 | (c) | word (A) <-- ((RLi)+disp8) | - | * | - | - | - | * | * | - | - | - |
| MOVW | dir,A | 2 | 3 | 0 | (c) | word (dir) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW | addr16,A | 3 | 4 | 0 | (c) | word (addr16) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW | SP,A | 1 | 1 | 0 | 0 | word (SP) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW | RWi,A | 1 | 2 | 1 | 0 | word (RWi) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW | ear,A | 2 | 2 | 1 | 0 | word (ear) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW | eam,A | 2+ | 3+(a) | 0 | (c) | word (eam) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW | io,A | 2 | 3 | 0 | (c) | word (io) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW | @RWi+disp8,A | 2 | 5 | 1 | (c) | word ((RWi)+disp8) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW | @RLi+disp8,A | 3 | 10 | 2 | (c) | word ((RLi)+disp8) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW | RWi,ear | 2 | 3 | 2 | 0 | word (RWi) <-- (ear) | - | - | - | - | - | * | * | - | - | - |
| MOVW | RWi,eam | 2+ | 4+(a) | 1 | (c) | word (RWi) <-- (eam) | - | - | - | - | - | * | * | - | - | - |
| MOVW | ear,Rwi | 2 | 4 | 2 | 0 | word (ear) <-- (RWi) | - | - | - | - | - | * | * | - | - | - |
| MOVW | eam,Rwi | 2+ | 5+(a) | 1 | (c) | word (eam) <-- (RWi) | - | - | - | - | - | * | * | - | - | - |
| MOVW | RWi,#imm16 | 3 | 2 | 1 | 0 | word (RWi) <-- imm16 | - | - | - | - | - | * | * | - | - | - |
| MOVW | io,#imm16 | 4 | 5 | 0 | (c) | word (io) <-- imm16 | - | - | - | - | - | - | - | - | - | - |
| MOVW | ear,#imm16 | 4 | 2 | 1 | 0 | word (ear) <-- imm16 | - | - | - | - | - | * | * | - | - | - |
| MOVW | eam,#imm16 | 4+ | 4+(a) | 0 | (c) | word (eam) <-- imm16 | - | - | - | - | - | - | - | - | - | - |
| MOVW | @AL,AH/MOVW<br>@A,T | 2 | 3 | 0 | (c) | word ((A)) <-- (AH) | - | - | - | - | - | * | * | - | - | - |
| XCHW | A,ear | 2 | 4 | 2 | 0 | word (A) <--> (ear) | - | - | - | - | - | - | - | - | - | - |
| XCHW | A,eam | 2+ | 5+(a) | 0 | 2 x (c) | word (A) <-- >(eam) | - | - | - | - | - | - | - | - | - | - |
| XCHW | RWi, ear | 2 | 7 | 4 | 0 | word (RWi) <--> (ear) | - | - | - | - | - | - | - | - | - | - |
| XCHW | RWi, eam | 2+ | 9+(a) | 2 | 2 x (c) | word (RWi) <--> (eam) | - | - | - | - | - | - | - | - | - | - |
| MOVL | A,ear | 2 | 4 | 2 | 0 | long (A) <-- (ear) | - | - | - | - | - | * | * | - | - | - |
| MOVL | A,eam | 2+ | 5+(a) | 0 | (d) | long (A) <-- (eam) | - | - | - | - | - | * | * | - | - | - |
| MOVL | A,#imm32 | 5 | 3 | 0 | 0 | long (A) <-- imm32 | - | - | - | - | - | * | * | - | - | - |
| MOVL | ear,A | 2 | 4 | 2 | 0 | long (ear1) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVL | eam,A | 2+ | 5+(a) | 0 | (d) | long(eam1) <-- (A) | - | - | - | - | - | * | * | - | - | - |

**Note:**

See Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" for information on (a) to (d) in the table.

### Table B.8-3  42 Addition/subtraction Instructions (byte, word, long word)

| Mnemonic | | # | ~ | RG | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADD | A,#imm8 | 2 | 2 | 0 | 0 | byte (A) <-- (A) + imm8 | Z | - | - | - | - | * | * | * | * | - |
| ADD | A,dir | 2 | 5 | 0 | (b) | byte (A) <-- (A) + (dir) | Z | - | - | - | - | * | * | * | * | - |
| ADD | A,ear | 2 | 3 | 1 | 0 | byte (A) <-- (A) + (ear) | Z | - | - | - | - | * | * | * | * | - |
| ADD | A,eam | 2+ | 4+(a) | 0 | (b) | byte (A) <-- (A) + (eam) | Z | - | - | - | - | * | * | * | * | - |
| ADD | ear,A | 2 | 3 | 2 | 0 | byte (ear) <-- (ear) + (A) | - | - | - | - | - | * | * | * | * | - |
| ADD | eam,A | 2+ | 5+(a) | 0 | 2×(b) | byte (eam) <-- (eam) + (A) | Z | - | - | - | - | * | * | * | * | * |
| ADDC | A | 1 | 2 | 0 | 0 | byte (A) <-- (AH) + (AL) + (C) | Z | - | - | - | - | * | * | * | * | - |
| ADDC | A,ear | 2 | 3 | 1 | 0 | byte (A) <-- (A) + (ear)+ (C) | Z | - | - | - | - | * | * | * | * | - |
| ADDC | A,eam | 2+ | 4+(a) | 0 | (b) | byte (A) <-- (A) + (eam)+ (C) | Z | - | - | - | - | * | * | * | * | - |
| ADDDC | A | 1 | 3 | 0 | 0 | byte (A) <-- (AH) + (AL) + (C) (decimal) | Z | - | - | - | - | * | * | * | * | - |
| SUB | A,#imm8 | 2 | 2 | 0 | 0 | byte (A) <-- (A) - imm8 | Z | - | - | - | - | * | * | * | * | - |
| SUB | A,dir | 2 | 5 | 0 | (b) | byte (A) <-- (A) - (dir) | Z | - | - | - | - | * | * | * | * | - |
| SUB | A,ear | 2 | 3 | 1 | 0 | byte (A) <-- (A) - (ear) | Z | - | - | - | - | * | * | * | * | - |
| SUB | A,eam | 2+ | 4+(a) | 0 | (b) | byte (A) <-- (A) - (eam) | Z | - | - | - | - | * | * | * | * | - |
| SUB | ear,A | 2 | 3 | 2 | 0 | byte (ear) <-- (ear) - (A) | - | - | - | - | - | * | * | * | * | - |
| SUB | eam,A | 2+ | 5+(a) | 0 | 2×(b) | byte (eam) <-- (eam) - (A) | - | - | - | - | - | * | * | * | * | * |
| SUBC | A | 1 | 2 | 0 | 0 | byte (A) <-- (AH) - (AL) - (C) | Z | - | - | - | - | * | * | * | * | - |
| SUBC | A,ear | 2 | 3 | 1 | 0 | byte (A) <-- (A) - (ear) - (C) | Z | - | - | - | - | * | * | * | * | - |
| SUBC | A,eam | 2+ | 4+(a) | 0 | (b) | byte (A) <-- (A) - (eam) - (C) | Z | - | - | - | - | * | * | * | * | - |
| SUBDC | A | 1 | 3 | 0 | 0 | byte (A) <-- (AH) - (AL) - (C) (decimal) | Z | - | - | - | - | * | * | * | * | - |
| ADDW | A | 1 | 2 | 0 | 0 | word (A) <-- (AH) + (AL) | - | - | - | - | - | * | * | * | * | - |
| ADDW | A,ear | 2 | 3 | 1 | 0 | word (A) <-- (A) + (ear) | - | - | - | - | - | * | * | * | * | - |
| ADDW | A,eam | 2+ | 4+(a) | 0 | (c) | word (A) <-- (A) + (eam) | - | - | - | - | - | * | * | * | * | - |
| ADDW | A,#imm16 | 3 | 2 | 0 | 0 | word (A) <-- (A) + imm16 | - | - | - | - | - | * | * | * | * | - |
| ADDW | ear,A | 2 | 3 | 2 | 0 | word (ear) <-- (ear) + (A) | - | - | - | - | - | * | * | * | * | - |
| ADDW | eam,A | 2+ | 5+(a) | 0 | 2×(c) | word (eam) <-- (eam) + (A) | - | - | - | - | - | * | * | * | * | * |
| ADDCW | A,ear | 2 | 3 | 1 | 0 | word (A) <-- (A) + (ear) + (C) | - | - | - | - | - | * | * | * | * | - |
| ADDCW | A,eam | 2+ | 4+(a) | 0 | (c) | word (A) <-- (A) + (eam) + (C) | - | - | - | - | - | * | * | * | * | - |
| SUBW | A | 1 | 2 | 0 | 0 | word (A) <-- (AH) - (AL) | - | - | - | - | - | * | * | * | * | - |
| SUBW | A,ear | 2 | 3 | 1 | 0 | word (A) <-- (A) - (ear) | - | - | - | - | - | * | * | * | * | - |
| SUBW | A,eam | 2+ | 4+(a) | 0 | (c) | word (A) <-- (A) - (eam) | - | - | - | - | - | * | * | * | * | - |
| SUBW | A,#imm16 | 3 | 2 | 0 | 0 | word (A) <-- (A) - imm16 | - | - | - | - | - | * | * | * | * | - |
| SUBW | ear,A | 2 | 3 | 2 | 0 | word (ear) <-- (ear) - (A) | - | - | - | - | - | * | * | * | * | - |
| SUBW | eam,A | 2+ | 5+(a) | 0 | 2×(c) | word (eam) <-- (eam) - (A) | - | - | - | - | - | * | * | * | * | * |
| SUBCW | A,ear | 2 | 3 | 1 | 0 | word (A) <-- (A) - (ear) - (C) | - | - | - | - | - | * | * | * | * | - |
| SUBCW | A,eam | 2+ | 4+(a) | 0 | (c) | word (A) <-- (A) - (eam) - (C) | - | - | - | - | - | * | * | * | * | - |
| ADDL | A,ear | 2 | 6 | 2 | 0 | long (A) <-- (A) + (ear) | - | - | - | - | - | * | * | * | * | - |
| ADDL | A,eam | 2+ | 7+(a) | 0 | (d) | long (A) <-- (A) + (eam) | - | - | - | - | - | * | * | * | * | - |
| ADDL | A,#imm32 | 5 | 4 | 0 | 0 | long (A) <-- (A) + imm32 | - | - | - | - | - | * | * | * | * | - |
| SUBL | A,ear | 2 | 6 | 2 | 0 | long (A) <-- (A) - (ear) | - | - | - | - | - | * | * | * | * | - |
| SUBL | A,eam | 2+ | 7+(a) | 0 | (d) | long (A) <-- (A) - (eam) | - | - | - | - | - | * | * | * | * | - |
| SUBL | A,#imm32 | 5 | 4 | 0 | 0 | long (A) <-- (A) - imm32 | - | - | - | - | - | * | * | * | * | - |

**Note:**

See Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" for information on (a) to (d) in the table.

### Table B.8-4 12 Increment/decrement Instructions (byte, word, long word)

| Mnemonic | | # | ～ | RG | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| INC | ear | 2 | 3 | 2 | 0 | byte (ear) <-- (ear) + 1 | - | - | - | - | - | * | * | * | - | - |
| INC | eam | 2+ | 5+(a) | 0 | 2×(b)<br>0 | byte (eam) <-- (eam) + 1 | - | - | - | - | - | * | * | * | - | * |
| DEC | ear | 2 | 3 | 2 | 2×(b) | byte (ear) <-- (ear) - 1 | - | - | - | - | - | * | * | * | - | - |
| DEC | eam | 2+ | 5+(a) | 0 | | byte (eam) <-- (eam) - 1 | - | - | - | - | - | * | * | * | - | * |
| INCW | ear | 2 | 3 | 2 | 0 | word (ear) <-- (ear) + 1 | - | - | - | - | - | * | * | * | - | - |
| INCW | eam | 2+ | 5+(a) | 0 | 2×(c) | word (eam) <-- (eam) + 1 | - | - | - | - | - | * | * | * | - | * |
| DECW | ear | 2 | 3 | 2 | 0 | word (ear) <-- (ear) - 1 | - | - | - | - | - | * | * | * | - | - |
| DECW | eam | 2+ | 5+(a) | 0 | 2×(c) | word (eam) <-- (eam) - 1 | - | - | - | - | - | * | * | * | - | * |
| INCL | ear | 2 | 7 | 4 | 0 | long (ear) <-- (ear) + 1 | - | - | - | - | - | * | * | * | - | - |
| INCL | eam | 2+ | 9+(a) | 0 | 2×(d) | long (eam) <-- (eam) + 1 | - | - | - | - | - | * | * | * | - | * |
| DECL | ear | 2 | 7 | 4 | 0 | long (ear) <-- (ear) - 1 | - | - | - | - | - | * | * | * | - | - |
| DECL | eam | 2+ | 9+(a) | 0 | 2×(d) | long (eam) <-- (eam) - 1 | - | - | - | - | - | * | * | * | - | * |

**Note:**

See Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" for information on (a) to (d) in the table.

### Table B.8-5 11 Compare Instructions (byte, word, long word)

| Mnemonic | | # | ～ | RG | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CMP | A | 1 | 1 | 0 | 0 | byte (AH) - (AL) | - | - | - | - | - | * | * | * | * | - |
| CMP | A,ear | 2 | 2 | 1 | 0 | byte (A) - (ear) | - | - | - | - | - | * | * | * | * | - |
| CMP | A,eam | 2+ | 3+(a) | 0 | (b) | byte (A) - (eam) | - | - | - | - | - | * | * | * | * | - |
| CMP | A,#imm8 | 2 | 2 | 0 | 0 | byte (A) - imm8 | - | - | - | - | - | * | * | * | * | - |
| CMPW | A | 1 | 1 | 0 | 0 | word (AH) - (AL) | - | - | - | - | - | * | * | * | * | - |
| CMPW | A,ear | 2 | 2 | 1 | 0 | word (A) - (ear) | - | - | - | - | - | * | * | * | * | - |
| CMPW | A,eam | 2+ | 3+(a) | 0 | (c) | word (A) - (eam) | - | - | - | - | - | * | * | * | * | - |
| CMPW | A,#imm16 | 3 | 2 | 0 | 0 | word (A) - imm16 | - | - | - | - | - | * | * | * | * | - |
| CMPL | A,ear | 2 | 6 | 2 | 0 | long (A) - (ear) | - | - | - | - | - | * | * | * | * | - |
| CMPL | A,eam | 2+ | 7+(a) | 0 | (d) | long (A) - (eam) | - | - | - | - | - | * | * | * | * | - |
| CMPL | A,#imm32 | 5 | 3 | 0 | 0 | long (A) - imm32 | - | - | - | - | - | * | * | * | * | - |

**Note:**

See Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" for information on (a) to (d) in the table.

### Table B.8-6  11 unsigned multiplication/division instructions (word, long word)

| Mnemonic | | # | ~ | RG | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DIVU | A | 1 | *1 | 0 | 0 | word (AH) / byte (AL)<br>quotient --> byte (AL) remainder --> byte (AH) | - | - | - | - | - | - | - | * | * | - |
| DIVU | A,ear | 2 | *2 | 1 | 0 | word (A) / byte (ear)<br>quotient --> byte (A) remainder --> byte (ear) | - | - | - | - | - | - | - | * | * | - |
| DIVU | A,eam | 2+ | *3 | 0 | *6 | word (A) / byte (eam)<br>quotient --> byte (A) remainder --> byte (eam) | - | - | - | - | - | - | - | * | * | - |
| DIVUW | A,ear | 2 | *4 | 1 | 0 | long (A) / word (ear)<br>quotient --> word(A) remainder --> word(ear) | - | - | - | - | - | - | - | * | * | - |
| DIVUW | A,eam | 2+ | *5 | 0 | *7 | long (A) / word (eam)<br>quotient --> word(A) remainder --> word(eam) | - | - | - | - | - | - | - | * | * | - |
| MULU | A | 1 | *8 | 0 | 0 | byte (AH) * byte (AL) --> word (A) | - | - | - | - | - | - | - | - | - | - |
| MULU | A,ear | 2 | *9 | 1 | 0 | byte (A) * byte (ear) --> word (A) | - | - | - | - | - | - | - | - | - | - |
| MULU | A,eam | 2+ | *10 | 0 | (b) | byte (A) * byte (eam) --> word (A) | - | - | - | - | - | - | - | - | - | - |
| MULUW | A | 1 | *11 | 0 | 0 | word (AH) * word (AL) --> Long (A) | - | - | - | - | - | - | - | - | - | - |
| MULUW | A,ear | 2 | *12 | 1 | 0 | word (A) * word (ear) --> Long (A) | - | - | - | - | - | - | - | - | - | - |
| MULUW | A,eam | 2+ | *13 | 0 | (c) | word (A) * word (eam) --> Long (A) | | | | | | | | | | |

*1:  3: Division by 0  7: Overflow  15: Normal
*2:  4: Division by 0  8: Overflow  16: Normal
*3:  6+(a): Division by 0  9+(a): Overflow  19+(a): Normal
*4:  4: Division by 0  7: Overflow  22: Normal
*5:  6+(a): Division by 0  8+(a): Overflow  26+(a): Normal
*6:  (b): Division by 0 or overflow  2 x (b): Normal
*7:  (c): Division by 0 or overflow  2 x (c): Normal
*8:  3: Byte (AH) is 0.  7: Byte (AH) is not 0.
*9:  4: Byte (ear) is 0.  8: Byte (ear) is not 0.
*10:  5+(a): Byte (eam) is 0, 9+(a): Byte (eam) is not 0.
*11:  3: Word(AH) is 0. 11: Word (AH) is not 0.
*12:  4: Word(ear) is 0. 12: Word (ear) is not 0.
*13:  5+(a): Word (eam) is 0. 13+(a): Word (eam) is not 0.

**Note:**

See Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" for information on (a) to (d) in the table.

### Table B.8-7  11 Signed Multiplication/Division Instructions (word, long word)

| Mnemonic | | # | ~ | RG | B | Operation | L<br>H | A<br>H | I | S | T | N | Z | V | C | R<br>M<br>W |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DIVU | A | 2 | *1 | 0 | 0 | word (AH) / byte (AL)<br>quotient --> byte (AL) remainder --> byte (AH) | Z | - | - | - | - | - | - | * | * | - |
| DIVU | A,ear | 2 | *2 | 1 | 0 | word (A) / byte (ear)<br>quotient --> byte (A) remainder --> byte (ear) | Z | - | - | - | - | - | - | * | * | - |
| DIVU | A,eam | 2+ | *3 | 0 | *6 | word (A) / byte (eam)<br>quotient --> byte (A) remainder --> byte (eam) | Z | - | - | - | - | - | - | * | * | - |
| DIVUW | A,ear | 2 | *4 | 1 | 0 | long (A) / word (ear)<br>quotient --> word(A) remainder -->  word(ear) | - | - | - | - | - | - | - | * | * | - |
| DIVUW | A,eam | 2+ | *5 | 0 | *7 | long (A) / word (eam)<br>quotient --> word(A) remainder -->  word(eam) | - | - | - | - | - | - | - | * | * | - |
| MUL | A | 2 | *8 | 0 | 0 | byte (AH) * byte (AL) --> word (A) | - | - | - | - | - | - | - | - | - | - |
| MUL | A,ear | 2 | *9 | 1 | 0 | byte (A) * byte (ear) --> word (A) | | - | - | - | - | - | - | - | - | - |
| MUL | A,eam | 2+ | *10 | 0 | (b) | byte (A) * byte (eam) --> word (A) | - | - | - | - | - | - | - | - | - | - |
| MULW | A | 2 | *11 | 0 | 0 | word (AH) * word (AL) --> Long (A) | | - | - | - | - | - | - | - | - | - |
| MULW | A,ear | 2 | *12 | 1 | 0 | word (A) * word (ear) --> Long (A) | - | - | - | - | - | - | - | - | - | - |
| MULW | A,eam | 2+ | *13 | 0 | (c) | word (A) * word (eam) --> Long (A) | - | | | | | | | | | |
| | | | | | | | - | | | | | | | | | |

*1:  3: Division by 0, 8 or 18: Overflow, 18: Normal
*2:  4: Division by 0, 11 or 22: Overflow, 23: Normal
*3:  5+(a): Division by 0, 12+(a) or 23+(a): Overflow, 24+(a): Normal
*4:  When dividend is positive; 4: Division by 0, 12 or 30: Overflow, 31: Normal
When dividend is negative; 4: Division by 0, 12 or 31: Overflow, 32: Normal
*5:  When dividend is positive; 5+(a): Division by 0, 12+(a) or 31+(a): Overflow, 32+(a): Normal
When dividend is negative; 5+(a): Division by 0, 12+(a) or 32+(a): Overflow, 33+(a): Normal
*6:  (b): Division by 0 or overflow, 2 x (b): Normal
*7:  (c): Division by 0 or overflow, 2 x (c): Normal
*8:  3: Byte (AH) is 0, 12: result is positive, 13: result is negative
*9:  4: Byte (ear) is 0, 13: result is positive, 14: result is negative
*10:  5+(a): Byte (eam) is 0, 14+(a): result is positive, 15+(a): result is negative
*11:  3: Word(AH) is 0, 16: result is positive, 19: result is negative
*12:  4: Word(ear) is 0, 17: result is positive, 20: result is negative
*13:  5+(a): Word(eam) is 0, 18+(a): result is positive, 21+(a): result is negative

**Note:**

- The execution cycle count found when an overflow occurs in a DIV or DIVW instruction may be a pre-operation count or a post-operation count depending on the detection timing.
- When an overflow occurs with DIV or DIVW instruction, the contents of the AL are destroyed.
- See Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" for information on (a) to (d) in the table.

### Table B.8-8 39 Logic 1 Instructions (byte, word)

| Mnemonic | | # | ~ | RG | B | Operation | L<br>H | A<br>H | I | S | T | N | Z | V | C | R<br>M<br>W |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AND | A,#imm8 | 2 | 2 | 0 | 0 | byte (A) <-- (A) and imm8 | - | - | - | - | - | * | * | R | - | - |
| AND | A,ear | 2 | 3 | 1 | 0 | byte (A) <-- (A) and (ear) | - | - | - | - | - | * | * | R | - | - |
| AND | A,eam | 2+ | 4+(a) | 0 | (b) | byte (A) <-- (A) and (eam) | - | - | - | - | - | * | * | R | - | - |
| AND | ear,A | 2 | 3 | 2 | 0 | byte (ear) <-- (ear)and (A) | - | - | - | - | - | * | * | R | - | - |
| AND | eam,A | 2+ | 5+(a) | 0 | 2×(b) | byte (eam) <-- (eam)and (A) | - | - | - | - | - | * | * | R | - | * |
| OR | A,#imm8 | 2 | 2 | 0 | 0 | byte (A) <-- (A) or imm8 | - | - | - | - | - | * | * | R | - | - |
| OR | A,ear | 2 | 3 | 1 | 0 | byte (A) <-- (A) or (ear) | - | - | - | - | - | * | * | R | - | - |
| OR | A,eam | 2+ | 4+(a) | 0 | (b) | byte (A) <-- (A) or (eam) | - | - | - | - | - | * | * | R | - | - |
| OR | ear,A | 2 | 3 | 2 | 0 | byte (ear) <-- (ear)or (A) | - | - | - | - | - | * | * | R | - | - |
| OR | eam,A | 2+ | 5+(a) | 0 | 2×(b) | byte (eam) <-- (eam)or (A) | - | - | - | - | - | * | * | R | - | * |
| XOR | A,#imm8 | 2 | 2 | 0 | 0 | byte (A) <-- (A) xor imm8 | - | - | - | - | - | * | * | R | - | - |
| XOR | A,ear | 2 | 3 | 1 | 0 | byte (A) <-- (A) xor (ear) | - | - | - | - | - | * | * | R | - | - |
| XOR | A,eam | 2+ | 4+(a) | 0 | (b) | byte (A) <-- (A) xor (eam) | - | - | - | - | - | * | * | R | - | - |
| XOR | ear,A | 2 | 3 | 2 | 0 | byte (ear) <-- (ear) xor (A) | - | - | - | - | - | * | * | R | - | - |
| XOR | eam,A | 2+ | 5+(a) | 0 | 2×(b) | byte (eam) <-- (eam) xor (A) | - | - | - | - | - | * | * | R | - | * |
| NOT | A | 1 | 2 | 0 | 0 | byte (A) <-- not (A) | - | - | - | - | - | * | * | R | - | - |
| NOT | ear | 2 | 3 | 2 | 0 | byte (ear) <-- not (ear) | - | - | - | - | - | * | * | R | - | - |
| NOT | eam | 2+ | 5+(a) | 0 | 2×(b) | byte (eam) <-- not (eam) | - | - | - | - | - | * | * | R | - | * |
| ANDW | A | 1 | 2 | 0 | 0 | word (A) <-- (AH) and (A) | - | - | - | - | - | * | * | R | - | - |
| ANDW | A,#imm16 | 3 | 2 | 0 | 0 | word (A) <-- (A) and imm16 | - | - | - | - | - | * | * | R | - | - |
| ANDW | A,ear | 2 | 3 | 1 | 0 | word (A) <-- (A) and (ear) | - | - | - | - | - | * | * | R | - | - |
| ANDW | A,eam | 2+ | 4+(a) | 0 | (c) | word (A) <-- (A) and (eam) | - | - | - | - | - | * | * | R | - | - |
| ANDW | ear,A | 2 | 3 | 2 | 0 | word (ear) <-- (ear) and (A) | - | - | - | - | - | * | * | R | - | - |
| ANDW | eam,A | 2+ | 5+(a) | 0 | 2×(c) | word (eam) <-- (eam) and (A) | - | - | - | - | - | * | * | R | - | * |
| ORW | A | 1 | 2 | 0 | 0 | word (A) <-- (AH) or (A) | - | - | - | - | - | * | * | R | - | - |
| ORW | A,#imm16 | 3 | 2 | 0 | 0 | word (A) <-- (A) or imm16 | - | - | - | - | - | * | * | R | - | - |
| ORW | A,ear | 2 | 3 | 1 | 0 | word (A) <-- (A) or (ear) | - | - | - | - | - | * | * | R | - | - |
| ORW | A,eam | 2+ | 4+(a) | 0 | (c) | word (A) <-- (A) or (eam) | - | - | - | - | - | * | * | R | - | - |
| ORW | ear,A | 2 | 3 | 2 | 0 | word (ear) <-- (ear) or (A) | - | - | - | - | - | * | * | R | - | - |
| ORW | eam,A | 2+ | 5+(a) | 0 | 2×(c) | word (eam) <-- (eam) or (A) | - | - | - | - | - | * | * | | - | * |
| | | | | | | | | | | | | | | R | | |
| XORW | A | 1 | 2 | 0 | 0 | word (A) <-- (AH) xor (A) | - | - | - | - | - | * | * | R | - | - |
| XORW | A,#imm16 | 3 | 2 | 0 | 0 | word (A) <-- (A) xor imm16 | - | - | - | - | - | * | * | R | - | - |
| XORW | A,ear | 2 | 3 | 1 | 0 | word (A) <-- (A) xor (ear) | - | - | - | - | - | * | * | R | - | - |
| XORW | A,eam | 2+ | 4+(a) | 0 | (c) | word (A) <-- (A) xor (eam) | - | - | - | - | - | * | * | R | - | - |
| XORW | ear,A | 2 | 3 | 2 | 0 | word (ear) <-- (ear) xor (A) | - | - | - | - | - | * | * | R | - | * |
| XORW | eam,A | 2+ | 5+(a) | 0 | 2×(c) | word (eam) <-- (eam) xor (A) | - | - | - | - | - | * | * | R | - | - |
| NOTW | A | 1 | 2 | 0 | 0 | word (A) <-- not (A) | - | - | - | - | - | * | * | R | - | - |
| NOTW | ear | 2 | 3 | 2 | 0 | word (ear) <-- not (ear) | - | - | - | - | - | * | * | R | - | * |
| NOTW | eam | 2+ | 5+(a) | 0 | 2×(c) | word (eam) <-- not (eam) | | | | | | | | | | |

**Note:**

See Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" for information on (a) to (d) in the table.

### Table B.8-9 Six Logic 2 Instructions (long word)

| Mnemonic | | # | ~ | RG | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ANDL | A,ear | 2 | 6 | 2 | 0 | long (A) <-- (A) and (ear) | - | - | - | - | - | * | * | R | - | - |
| ANDL | A,eam | 2+ | 7+(a) | 0 | (d) | long (A) <-- (A) and (eam) | - | - | - | - | - | * | * | R | - | - |
| ORL | A,ear | 2 | 6 | 2 | 0 | long (A) <-- (A) or (ear) | - | - | - | - | - | * | * | R | - | - |
| ORL | A,eam | 2+ | 7+(a) | 0 | (d) | long (A) <-- (A) or (eam) | - | - | - | - | - | * | * | R | - | - |
| XORL | A,ear | 2 | 6 | 2 | 0 | long (A) <-- (A) xor (ear) | - | - | - | - | - | * | * | R | - | - |
| XORL | A,eam | 2+ | 7+(a) | 0 | (d) | long (A) <-- (A) xor (eam) | - | - | - | - | - | * | * | R | - | - |

**Note:**

See Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" for information on (a) to (d) in the table.

### Table B.8-10 Six Sign Inversion Instructions (byte, word)

| Mnemonic | | # | ~ | RG | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NEG | A | 1 | 2 | 0 | 0 | byte (A) <-- 0 - (A) | X | - | - | - | - | * | * | * | * | - |
| NEG | ear | 2 | 3 | 2 | 0 | byte (ear) <-- 0 - (ear) | - | - | - | - | - | * | * | * | * | - |
| NEG | eam | 2+ | 5+(a) | 0 | 2×(b) | byte (eam) <-- 0 - (eam) | - | - | - | - | - | * | * | * | * | * |
| NEGW | A | 1 | 2 | 0 | 0 | word (A) <-- 0 - (A) | - | - | - | - | - | * | * | * | * | - |
| NEGW | ear | 2 | 3 | 2 | 0 | word (ear) <-- 0 - (ear) | - | - | - | - | - | * | * | * | * | - |
| NEGW | eam | 2+ | 5+(a) | 0 | 2×(c) | word (eam) <-- 0 - (eam) | - | - | - | - | - | * | * | * | * | * |

**Note:**

See Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" for information on (a) to (d) in the table.

### Table B.8-11 One Normalization Instruction (long word)

| Mnemonic | | # | ~ | RG | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NRML | A,R0 | 2 | *1 | 1 | 0 | long (A) <-- Shifts to the position where '1' is set for the first time.<br>byte (RD) <-- Shift count at that time | - | - | - | - | - | - | * | - | - | - |

*1: 4 when all accumulators have a value of 0; otherwise, 6+(R0)

### Table B.8-12 18 Shift Instructions (byte, word, long word)

| Mnemonic | | # | ~ | RG | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RORC | A | 2 | 2 | 0 | 0 | byte (A) <-- With right rotation carry | - | - | - | - | - | * | * | - | * | - |
| ROLC | A | 2 | 2 | 0 | 0 | byte (A) <-- With left rotation carry | - | - | - | - | - | * | * | - | * | - |
| RORC | ear | 2 | 3 | 2 | 0 | byte (ear) <-- With right rotation carry | - | - | - | - | - | * | * | - | * | - |
| RORC | eam | 2+ | 5+(a) | 0 | 2×(b) | byte (eam) <-- With right rotation carry | - | - | - | - | - | * | * | - | * | * |
| ROLC | ear | 2 | 3 | 2 | 0 | byte (ear) <-- With left rotation carry | - | - | - | - | - | * | * | - | * | - |
| ROLC | eam | 2+ | 5+(a) | 0 | 2×(b) | byte (eam) <-- With left rotation carry | - | - | - | - | - | * | * | - | * | * |
| ASR | A,R0 | 2 | *1 | 1 | 0 | byte (A) <-- Arithmetic right shift (A, 1 bit) | - | - | - | - | * | * | * | - | * | - |
| LSR | A,R0 | 2 | *1 | 1 | 0 | byte (A) <-- Logical right barrel shift (A, R0) | - | - | - | - | - | * | * | - | * | - |
| LSL | A,R0 | 2 | *1 | 1 | 0 | byte (A) <-- Logical left barrel shift (A, R0) | - | - | - | - | - | * | * | - | * | - |
| ASRW | A | 1 | 2 | 0 | 0 | word (A) <-- Arithmetic right shift (A, 1 bit) | - | - | - | - | * | * | * | - | * | - |
| LSRW | A/SHRW A | 1 | 2 | 0 | 0 | word (A) <-- Logical right shift (A, 1 bit) | - | - | - | - | * | R | * | - | * | - |
| LSLW | A/SHLW A | 1 | 2 | 0 | 0 | word (A) <-- Logical left shift (A, 1 bit) | - | - | - | - | - | * | * | - | * | - |
| ASRW | A,R0 | 2 | *1 | 1 | 0 | word (A) <-- Arithmetic right barrel shift (A, R0) | - | - | - | - | * | * | * | - | * | - |
| LSRW | A,R0 | 2 | *1 | 1 | 0 | word (A) <-- Logical right barrel shift (A, R0) | - | - | - | - | * | * | * | - | * | - |
| LSLW | A,R0 | 2 | *1 | 1 | 0 | word (A) <-- Logical left barrel shift (A, R0) | - | - | - | - | - | * | * | - | * | - |
| ASRL | A,R0 | 2 | *2 | 1 | 0 | long (A) <-- Arithmetic right barrel shift (A, R0) | - | - | - | - | * | * | * | - | * | - |
| LSRL | A,R0 | 2 | *2 | 1 | 0 | long (A) <-- Logical right barrel shift (A, R0) | - | - | - | - | * | * | * | - | * | - |
| LSLL | A,R0 | 2 | *2 | 1 | 0 | long (A) <-- Logical left barrel shift (A, R0) | - | - | - | - | - | * | * | - | * | - |

*1: 6 when R0 is 0; otherwise, 5 + (R0)

*2: 6 when R0 is 0; otherwise, 6 + (R0)

**Note:**

See Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" for information on (a) to (d) in the table.

### Table B.8-13 31 Branch 1 Instructions

| Mnemonic | | # | ~ | RG | B | Operation | L<br>H | A<br>H | I | S | T | N | Z | V | C | R<br>M<br>W |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BZ/BEQ | rel | 2 | *1 | 0 | 0 | Branch on (Z) = 1 | - | - | - | - | - | - | - | - | - | - |
| BNZ/BNE | rel | 2 | *1 | 0 | 0 | Branch on (Z) = 0 | - | - | - | - | - | - | - | - | - | - |
| BC/BLO | rel | 2 | *1 | 0 | 0 | Branch on (C) = 1 | - | - | - | - | - | - | - | - | - | - |
| BNC/BHS | rel | 2 | *1 | 0 | 0 | Branch on (C) = 0 | - | - | - | - | - | - | - | - | - | - |
| BN | rel | 2 | *1 | 0 | 0 | Branch on (N) = 1 | - | - | - | - | - | - | - | - | - | - |
| BP | rel | 2 | *1 | 0 | 0 | Branch on (N) = 0 | - | - | - | - | - | - | - | - | - | - |
| BV | rel | 2 | *1 | 0 | 0 | Branch on (V) = 1 | - | - | - | - | - | - | - | - | - | - |
| BNV | rel | 2 | *1 | 0 | 0 | Branch on (V) = 0 | - | - | - | - | - | - | - | - | - | - |
| BT | rel | 2 | *1 | 0 | 0 | Branch on (T) = 1 | - | - | - | - | - | - | - | - | - | - |
| BNT | rel | 2 | *1 | 0 | 0 | Branch on (T) = 0 | - | - | - | - | - | - | - | - | - | - |
| BLT | rel | 2 | *1 | 0 | 0 | Branch on (V) nor (N) = 1 | - | - | - | - | - | - | - | - | - | - |
| BGE | rel | 2 | *1 | 0 | 0 | Branch on (V) nor (N) = 0 | - | - | - | - | - | - | - | - | - | - |
| BLE | rel | 2 | *1 | 0 | 0 | Branch on ((V) xor (N)) or (Z) = 1 | - | - | - | - | - | - | - | - | - | - |
| BGT | rel | 2 | *1 | 0 | 0 | Branch on ((V) xor (N)) or (Z) = 0 | - | - | - | - | - | - | - | - | - | - |
| BLS | rel | 2 | *1 | 0 | 0 | Branch on (C) or (Z) = 1 | - | - | - | - | - | - | - | - | - | - |
| BHI | rel | 2 | *1 | 0 | 0 | Branch on (C) or (Z) = 0 | - | - | - | - | - | - | - | - | - | - |
| BRA | rel | 2 | *1 | 0 | 0 | Unconditional branch | - | - | - | - | - | - | - | - | - | - |
| JMP | @A | 1 | 2 | 0 | 0 | word (PC) <-- (A) | - | - | - | - | - | - | - | - | - | - |
| JMP | addr16 | 3 | 3 | 0 | 0 | word (PC) <-- addr16 | - | - | - | - | - | - | - | - | - | - |
| JMP | @ear | 2 | 3 | 1 | 0 | word (PC) <-- (ear) | - | - | - | - | - | - | - | - | - | - |
| JMP | @eam | 2+ | 4+(a) | 0 | (c) | word (PC) <-- (eam) | - | - | - | - | - | - | - | - | - | - |
| JMPP | @ear *3 | 2 | 5 | 2 | 0 | word (PC) <-- (ear), (PCB) <-- (ear+2) | - | - | - | - | - | - | - | - | - | - |
| JMPP | @eam *3 | 2+ | 6+(a) | 0 | (d) | word (PC) <-- (eam), (PCB) <-- (eam+2) | - | - | - | - | - | - | - | - | - | - |
| JMPP | addr24 | 4 | 4 | 0 | 0 | word(PC) <-- ad24 0-15,(PCB) <-- ad24 16-23 | - | - | - | - | - | - | - | - | - | - |
| CALL | @ear *4 | 2 | 6 | 1 | (c) | word (PC) <-- (ear) | - | - | - | - | - | - | - | - | - | - |
| CALL | @eam *4 | 2+ | 7+(a) | 0 | 2×(c) | word (PC) <-- (eam) | - | - | - | - | - | - | - | - | - | - |
| CALL | addr16 *5 | 3 | 6 | 0 | (c) | word (PC) <-- addr16 | - | - | - | - | - | - | - | - | - | - |
| CALLV | #vct4 *5 | 1 | 7 | 0 | 2×(c) | Vector call instruction | - | - | - | - | - | - | - | - | - | - |
| CALLP | @ear *6 | 2 | 10 | 2 | 2×(c) | word(PC) <-- (ear)0-15,(PCB) <-- (ear)16-23 | - | - | - | - | - | - | - | - | - | - |
| CALLP | @eam *6 | 2+ | 11+(a) | 0 | *2 | word(PC) <-- (eam)0-15,(PCB) <-- (eam)16-23 | - | - | - | - | - | - | - | - | - | - |
| CALLP | addr24 *7 | 4 | 10 | 0 | 2×(c) | word(PC) <-- addr0-15, (PCB) <-- addr16-23 | - | - | - | - | - | - | - | - | - | - |

*1: 4 when a branch is made; otherwise, 3

*2: 3 x (c) + (b)

*3: Read (word) of branch destination address

*4: W: Save to stack (word) R: Read (word) of branch destination address

*5: Save to stack (word)

*6: W: Save to stack (long word), R: Read (long word) of branch destination address

*7: Save to stack (long word)

**Note:**

See Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" for information on (a) to (d) in the table.

**Table B.8-14   19 Branch 2 Instructions**

| Mnemonic | | # | ~ | RG | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CBNE | A,#imm8,rel | 3 | *1 | 0 | 0 | Branch on byte (A) not equal to imm8 | - | - | - | - | - | * | * | * | * | - |
| CWBNE | A,#imm16,rel | 4 | *1 | 0 | 0 | Branch on word (A) not equal to imm16 | - | - | - | - | - | * | * | * | * | - |
| CBNE | ear,#imm8,rel | 4 | *2 | 1 | 0 | Branch on byte (ear) not equal to imm8 | - | - | - | - | - | * | * | * | * | - |
| CBNE | eam,#imm8,rel *9 | 4+ | *3 | 0 | (b) | Branch on byte (eam) not equal to imm8 | - | - | - | - | - | * | * | * | * | - |
| CWBNE | ear,#imm16,rel | 5 | *4 | 1 | 0 | Branch on word (ear) not equal to imm16 | - | - | - | - | - | * | * | * | * | - |
| CWBNE | eam,#imm16,rel*9 | 5+ | *3 | 0 | (c) | Branch on word (eam) not equal to imm16 | - | - | - | - | - | * | * | * | * | - |
| DBNZ | ear,rel | 3 | *5 | 2 | 0 | Branch on byte (ear) = (ear) - 1, (ear)not equal to 0 | - | - | - | - | - | * | * | * | - | - |
| DBNZ | eam,rel | 3+ | *6 | 2 | 2×(b) | Branch on byte (eam) = (eam) - 1, (eam) not equal to 0 | - | - | - | - | - | * | * | * | - | * |
| DWBNZ | ear,rel | 3 | *5 | 2 | 0 | Branch on word (ear) = (ear) - 1, (ear) not equal to 0 | - | - | - | - | - | * | * | * | - | - |
| DWBNZ | eam,rel | 3+ | *6 | 2 | 2×(c) | Branch on word (eam) = (eam) - 1, (eam) not equal to 0 | - | - | - | - | - | * | * | * | - | * |
| INT | #vct8 | 2 | 20 | 0 | 8×(c) | Software interrupt | - | - | R | S | - | - | - | - | - | - |
| INT | addr16 | 3 | 16 | 0 | 6×(c) | Software interrupt | - | - | R | S | - | - | - | - | - | - |
| INTP | addr24 | 4 | 17 | 0 | 6×(c) | Software interrupt | - | - | R | S | - | - | - | - | - | - |
| INT9 | | 1 | 20 | 0 | 8×(c) | Software interrupt | - | - | R | S | - | - | - | - | - | - |
| RETI | | 1 | *8 | 0 | *7 | Return from interrupt | - | - | * | * | * | * | * | * | * | - |
| LINK | #imm8 | 2 | 6 | 0 | (c) | Saves the old frame pointer in the stack upon entering the function, then sets the new frame pointer and reserves the local pointer area. | - | - | - | - | - | - | - | - | - | - |
| UNLINK | | 1 | 5 | 0 | (c) | Recovers the old frame pointer from the stack upon exiting the function. | - | - | - | - | - | - | - | - | - | - |
| RET | *10 | 1 | 4 | 0 | (c) | Return from subroutine | - | - | - | - | - | - | - | - | - | - |
| RETP | *11 | 1 | 6 | 0 | (d) | Return from subroutine | - | - | - | - | - | - | - | - | - | - |

*1: 5 when a branch is made; otherwise, 4

*2: 13 when a branch is made; otherwise, 12

*3: 7+(a) when a branch is made; otherwise, 6+(a)

*4: 8 when a branch is made; otherwise, 7

*5: 7 when a branch is made; otherwise, 6

*6: 8+(a) when a branch is made; otherwise, 7+(a)

*7: 3 x (b) + 2 x (c) when jumping to the next interruption request; 6 x (c) when returning from the current interruption

*8: 15 when jumping to the next interruption request; 17 when returning from the current interruption

*9: Do not use RWj+ addressing mode with a CBNE or CWBNE instruction.

*10: Return from stack (word)

*11: Return from stack (long word)

**Note:**

See Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" for information on (a) to (d) in the table.

### Table B.8-15   28 Other Control Instructions (byte, word, long word)

| Mnemonic | | # | ~ | RG | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PUSHW | A | 1 | 4 | 0 | (c) | word (SP) <-- (SP) - 2 , ((SP)) <-- (A) | - | - | - | - | - | - | - | - | - | - |
| PUSHW | AH | 1 | 4 | 0 | (c) | word (SP) <-- (SP) - 2 , ((SP)) <-- (AH) | - | - | - | - | - | - | - | - | - | - |
| PUSHW | PS | 1 | 4 | 0 | (c) | word (SP) <-- (SP) - 2 , ((SP)) <-- (PS) | - | - | - | - | - | - | - | - | - | - |
| PUSHW | rlst | 2 | *3 | *5 | *4 | (SP) <-- (SP) - 2n , ((SP)) <-- (rlst) | - | - | - | - | - | - | - | - | - | - |
| POPW | A | 1 | 3 | 0 | (c) | word (A) <-- ((SP)) , (SP) <-- (SP) + 2 | - | * | - | - | - | - | - | - | - | - |
| POPW | AH | 1 | 3 | 0 | (c) | word (AH) <-- ((SP)) , (SP) <-- (SP) + 2 | - | - | - | - | - | - | - | - | - | - |
| POPW | PS | 1 | 4 | 0 | (c) | word (PS) <-- ((SP)) , (SP) <-- (SP) + 2 | - | - | * | * | * | * | * | * | * | - |
| POPW | rlst | 2 | *2 | *5 | *4 | (rlst) <-- ((SP)) , (SP) <-- (SP) | - | - | - | - | - | - | - | - | - | - |
| JCTX | @A | 1 | 14 | 0 | 6×(c) | Context switch instruction | - | - | * | * | * | * | * | * | * | - |
| AND | CCR,#imm8 | 2 | 3 | 0 | 0 | byte (CCR) <-- (CCR) and imm8 | - | - | * | * | * | * | * | * | * | - |
| OR | CCR,#imm8 | 2 | 3 | 0 | 0 | byte(CCR) <-- (CCR) or imm8 | - | - | * | * | * | * | * | * | * | - |
| MOV | RP,#imm8 | 2 | 2 | 0 | 0 | byte (RP) <-- imm8 | - | - | - | - | - | - | - | - | - | - |
| MOV | ILM,#imm8 | 2 | 2 | 0 | 0 | byte (ILM) <-- imm8 | - | - | - | - | - | - | - | - | - | - |
| MOVEA | RWi,ear | 2 | 3 | 1 | 0 | word (RWi) <-- ear | - | - | - | - | - | - | - | - | - | - |
| MOVEA | RWi,eam | 2+ | 2+(a) | 1 | 0 | word (RWi) <-- eam | - | - | - | - | - | - | - | - | - | - |
| MOVEA | A,ear | 2 | 1 | 0 | 0 | word (A) <-- ear | - | * | - | - | - | - | - | - | - | - |
| MOVEA | A,eam | 2+ | 1+(a) | 0 | 0 | word (A) <-- eam | - | * | - | - | - | - | - | - | - | - |
| ADDSP | #imm8 | 2 | 3 | 0 | 0 | word (SP) <-- ext(imm8) | - | - | - | - | - | - | - | - | - | - |
| ADDSP | #imm16 | 3 | 3 | 0 | 0 | word (SP) <-- imm16 | - | - | - | - | - | - | - | - | - | - |
| MOV | A,brg1 | 2 | *1 | 0 | 0 | byte (A) <-- (brg1) | Z | * | - | - | - | * | * | - | - | - |
| MOV | brg2,A | 2 | 1 | 0 | 0 | byte (brg2) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| NOP | | 1 | 1 | 0 | 0 | No operation | - | - | - | - | - | - | - | - | - | - |
| ADB | | 1 | 1 | 0 | 0 | Prefix code for AD space access | - | - | - | - | - | - | - | - | - | - |
| DTB | | 1 | 1 | 0 | 0 | Prefix code for DT space access | - | - | - | - | - | - | - | - | - | - |
| PCB | | 1 | 1 | 0 | 0 | Prefix code for PC space access | - | - | - | - | - | - | - | - | - | - |
| SPB | | 1 | 1 | 0 | 0 | Prefix code for SP space access | - | - | - | - | - | - | - | - | - | - |
| NCC | | 1 | 1 | 0 | 0 | Prefix code for flag no-change | - | - | - | - | - | - | - | - | - | - |
| CMR | | 1 | 1 | 0 | 0 | Prefix code for common register bank | - | - | - | - | - | - | - | - | - | - |

*1: PCB, ADB, SSB, USB, SPB:  1, DTB, DPR:  2

*2: 7 + 3×(POP count) + 2×(POP last register number), 7 when RLST = 0 (no transfer register)

*3: 29 + 3×(PUSH count) - 3×(PUSH last register number), 8 when RLST = 0 (no transfer register)

*4: (POP count)×(c) or (PUSH count)×(c)

*5: (POP count) or (PUSH count)

**Note:**

See Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" for information on (a) to (d) in the table.

### Table B.8-16 21 Bit Operand Instructions

| Mnemonic | | # | ~ | RG | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MOVB | A,dir:bp | 3 | 5 | 0 | (b) | byte (A) <-- ( dir:bp )b | Z | * | - | - | - | * | * | - | - | - |
| MOVB | A,addr16:bp | 4 | 5 | 0 | (b) | byte (A) <-- ( addr16:bp )b | Z | * | - | - | - | * | * | - | - | - |
| MOVB | A,io:bp | 3 | 4 | 0 | (b) | byte (A) <-- ( io:bp )b | Z | * | - | - | - | * | * | - | - | - |
| MOVB | dir:bp,A | 3 | 7 | 0 | 2×(b) | bit ( dir:bp )b <-- (A) | - | - | - | - | - | * | * | - | - | * |
| MOVB | addr16:bp,A | 4 | 7 | 0 | 2×(b) | bit ( addr16:bp )b <-- (A) | - | - | - | - | - | * | * | - | - | * |
| MOVB | io:bp,A | 3 | 6 | 0 | 2×(b) | bit ( io:bp )b <-- (A) | - | - | - | - | - | * | * | - | - | * |
| SETB | dir:bp | 3 | 7 | 0 | 2×(b) | bit ( dir:bp )b <-- 1 | - | - | - | - | - | - | - | - | - | * |
| SETB | addr16:bp | 4 | 7 | 0 | 2×(b) | bit ( addr16:bp )b <-- 1 | - | - | - | - | - | - | - | - | - | * |
| SETB | io:bp | 3 | 7 | 0 | 2×(b) | bit ( io:bp )b <-- 1 | - | - | - | - | - | - | - | - | - | * |
| CLRB | dir:bp | 3 | 7 | 0 | 2×(b) | bit ( dir:bp )b <-- 0 | - | - | - | - | - | - | - | - | - | * |
| CLRB | addr16:bp | 4 | 7 | 0 | 2×(b) | bit ( addr16:bp )b <-- 0 | - | - | - | - | - | - | - | - | - | * |
| CLRB | io:bp | 3 | 7 | 0 | 2×(b) | bit ( io:bp )b <-- 0 | - | - | - | - | - | - | - | - | - | * |
| BBC | dir:bp,rel | 4 | *1 | 0 | (b) | Branch on (dir:bp) b = 0 | - | - | - | - | - | - | * | - | - | - |
| BBC | addr16:bp,rel | 5 | *1 | 0 | (b) | Branch on (addr16:bp) b = 0 | - | - | - | - | - | - | * | - | - | - |
| BBC | io:bp,rel | 4 | *2 | 0 | (b) | Branch on (io:bp) b = 0 | - | - | - | - | - | - | * | - | - | - |
| BBS | dir:bp,rel | 4 | *1 | 0 | (b) | Branch on (dir:bp) b = 1 | - | - | - | - | - | - | * | - | - | - |
| BBS | addr16:bp,rel | 5 | *1 | 0 | (b) | Branch on (addr16:bp) b = 1 | - | - | - | - | - | - | * | - | - | - |
| BBS | io:bp,rel | 4 | *1 | 0 | (b) | Branch on (io:bp) b = 1 | - | - | - | - | - | - | * | - | - | - |
| SBBS | addr16:bp,rel | 5 | *3 | 0 | 2×(b) | Branch on (addr16:bp) b = 1, bit = 1 | - | - | - | - | - | - | * | - | - | * |
| WBTS | io:bp | 3 | *4 | 0 | *5 | Waits until (io:bp) b = 1 | - | - | - | - | - | - | - | - | - | - |
| WBTC | io:bp | 3 | *4 | 0 | *5 | Waits until (io:bp) b = 0 | - | - | - | - | - | - | - | - | - | - |

*1: 8 when a branch is made; otherwise, 7
*2: 7 when a branch is made; otherwise, 6
*3: 10 when the condition is met; otherwise 9
*4: Undefined count
*5: Until the condition is met( dir:bp )b

**Note:**

See Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" for information on (a) to (d) in the table.

### Table B.8-17 Six Accumulator Operation Instructions (byte, word)

| Mnemonic | # | ~ | RG | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SWAP | 1 | 3 | 0 | 0 | byte (A)0-7 <--> (A)8-15 | - | - | - | - | - | - | - | - | - | - |
| SWAPW/XCHW A,T | 1 | 2 | 0 | 0 | word (AH) <--> (AL) | - | * | - | - | - | - | - | - | - | - |
| EXT | 1 | 1 | 0 | 0 | Byte sign extension | X | - | - | - | - | * | * | - | - | - |
| EXTW | 1 | 2 | 0 | 0 | Word sign extension | - | X | - | - | - | * | * | - | - | - |
| ZEXT | 1 | 1 | 0 | 0 | Byte zero extension | Z | - | - | - | - | R | * | - | - | - |
| ZEXTW | 1 | 1 | 0 | 0 | Word zero extensionbyte | - | z | - | - | - | R | * | - | - | - |

### Table B.8-18   Ten String Instructions

| Mnemonic | # | ~ | RG | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MOVS / MOVSI | 2 | *2 | *5 | *3 | byte transfer @AH+ <-- @AL+, counter = RW0 | - | - | - | - | - | - | - | - | - | - |
| MOVSD | 2 | *2 | *5 | *3 | byte transfer @AH- <-- @AL-, counter = RW0 | - | - | - | - | - | - | - | - | - | - |
| SCEQ / SCEQI | 2 | *1 | *5 | *4 | byte search @AH+ <-- AL, counter RW0 | - | - | - | - | - | * | * | * | * | - |
| SCEQD | 2 | *1 | *5 | *4 | byte search @AH- <-- AL, counter RW0 | - | - | - | - | - | * | * | * | * | - |
| FILS / FILSI | 2 | 6m+6 | *5 | *3 | byte fill @AH+ <-- AL, counter RW0 | - | - | - | - | - | * | * | - | - | - |
| MOVSW / MOVSWI | 2 | *2 | *8 | *6 | word transfer @AH+ <-- @AL+, counter = RW0 | - | - | - | - | - | - | - | - | - | - |
| MOVSWD | 2 | *2 | *8 | *6 | word transfer @AH- <-- @AL-, counter = RW0 | - | - | - | - | - | - | - | - | - | - |
| SCWEQ / SCWEQI | 2 | *1 | *8 | *7 | word search @AH+ - AL, counter = RW0 | - | - | - | - | - | * | * | * | * | - |
| SCWEQD | 2 | *1 | *8 | *7 | word search @AH- - AL, counter = RW0 | - | - | - | - | - | * | * | * | * | - |
| FILSW / FILSWI | 2 | 6m+6 | *8 | *6 | word fill @AH+ <-- AL, counter = RW0 | - | - | - | - | - | * | * | - | - | - |

*1: 5 when RW0 is 0, 4 + 7 × (RW0) when the counter expires, or 7n + 5 when a match occurs

*2: 5 when RW0 is 0; otherwise, 4 + 8 × (RW0)

*3: (b) × (RW0) + (b) × (RW0)  When the source and destination access different areas, calculate the (b) item individually.

*4: (b) × n

*5: 2 × (R × W0)

*6: (c) × (RW0) + (c) × (RW0)  When the source and destination access different areas, calculate the (c) item individually.

*7: (c) × n

*8: 2 × 0(RW0)

**Note:**

m:  RW0 value (counter value), n:  Loop count

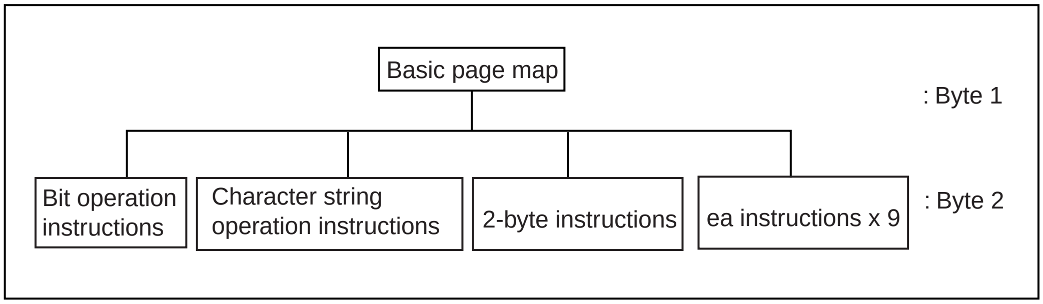
See Table B.5-1 "Execution Cycle Counts in Each Addressing Mode" and Table B.5-2 "Cycle Count Correction Values for Counting Execution Cycles" for information on (a) to (d) in the table.

# B.9 Instruction Map

**Each $F^2$MC-16LX instruction code consists of 1 or 2 bytes. Therefore, the instruction map consists of multiple pages. Table B.9-2 "Basic Page Map" to Table B.9-21 "XCHW RWi, ea Instruction (first byte = $7F_H$)" summarize the $F^2$MC-16LX instruction map.**

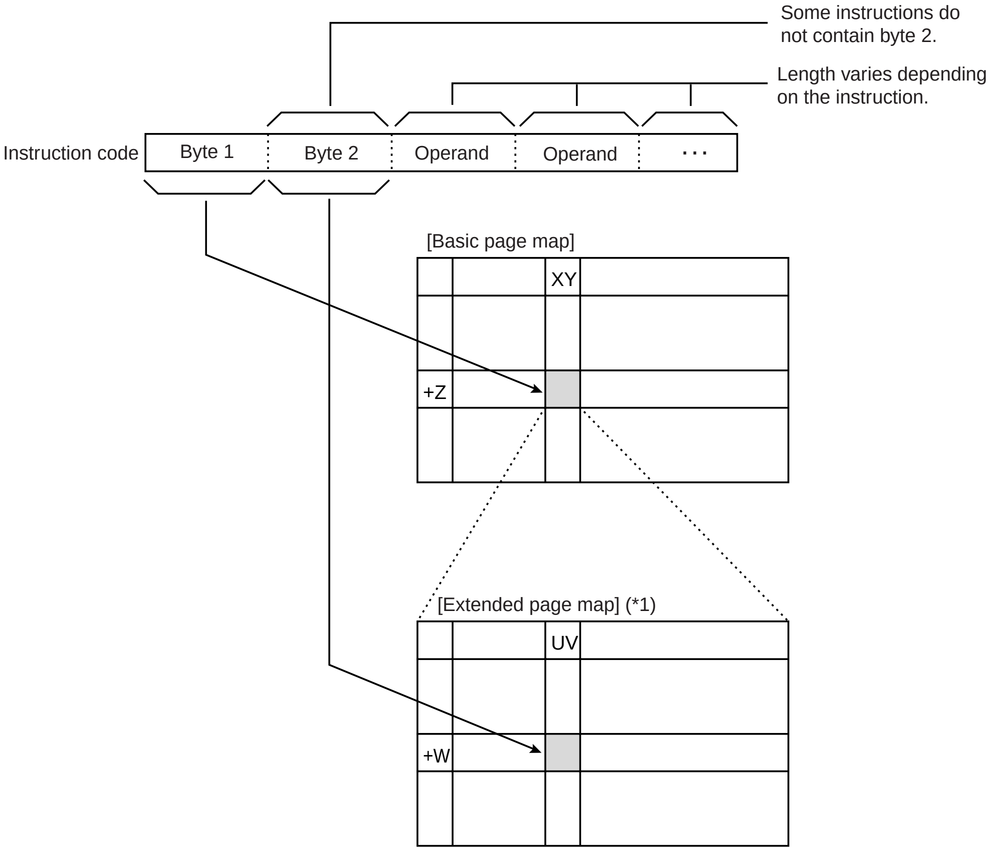
## ■ Structure of Instruction Map

**Figure B.9-1  Structure of Instruction Map**



An instruction such as the NOP instruction that ends in one byte is completed within the basic page. An instruction such as the MOVS instruction that requires two bytes recognizes the existence of byte 2 when it references byte 1, and can check the following one byte by referencing the map for byte 2. Figure B.9-2 "Correspondence between Actual Instruction Code and Instruction Map" shows the correspondence between an actual instruction code and instruction map.

**Figure B.9-2  Correspondence between Actual Instruction Code and Instruction Map**



*1  The extended page map is a generic name of maps for bit operation instructions, character string operation instructions, 2-byte instructions, and ea instructions.  Actually, there are multiple extended page maps for each type of instructions.

An example of an instruction code is shown in Table B.9-1 "Example of an Instruction Code".

**Table B.9-1  Example of an Instruction Code**

| Instruction | Byte 1<br>(from basic page map) | Byte 2<br>(from extended page map) |
|---|---|---|
| NOP | 00 +0=00 | - |
| AND A, #8 | 30 +4=34 | - |
| MOV A, ADB | 60 +F=6F | 00 +0=00 |
| @RW2+d8, #8rel | 70 +0=70 | F0 +2=F2 |

### Table B.9-2 Basic Page Map

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + 0 | NOP | CMR | ADD A,dir | ADD A,#8 | MOV A,dir | MOV A,io | BRA rel | ea instruction 1 | MOV A,Ri | MOV Ri,A | MOV Ri,#8 | MOVX A,Ri | MOVX A, @RWi+d8 | MOVN A,#4 | CALLV #4 | BZ /BEQ rel |
| + 1 | INT9 | NCC | SUB A,dir | SUB A,#8 | MOV dir,A | MOV io,A | JMP @A | ea instruction 2 | | | | | | | | BNZ/BNE rel |
| + 2 | ADDDC A | SUBDC A | ADDC A | SUBC A | MOV A,#8 | MOV A,addr16 | JMP addr16 | ea instruction 3 | | | | | | | | BC /BLO rel |
| + 3 | NEG A | JCTX @A | CMP A | CMP A,#8 | MOVX A,#8 | MOV addr16,A | JMPP addr24 | ea instruction 4 | | | | | | | | BNC /BHS rel |
| + 4 | PCB | EXT | AND CCR,#8 | AND A,#8 | MOV dir,#8 | MOV io,#8 | CALL addr16 | ea instruction 5 | | | | | | | | BN rel |
| + 5 | DTB | ZEXT | OR CCR,#8 | OR A,#8 | MOVX A,dir | MOVX A,io | CALLP addr24 | ea instruction 6 | | | | | | | | BP rel |
| + 6 | ADB | SWAP | DIVU A | XOR A,#8 | MOVW A,SP | MOVW io,#16 | RETP | ea instruction 7 | | | | | | | | BV rel |
| + 7 | SPB | ADDSP #8 | MULU A | NOT A | MOVW SP,A | MOVX A,addr16 | RET | ea instruction 8 | | | | | | | | BNV rel |
| + 8 | LINK #imm8 | ADDL A,#32 | ADDW A | ADDW A,#16 | MOVW A,dir | MOVW A,io | INT #vct8 | ea instruction 9 | MOVW A,RWi | MOVW RWi,A | MOVW RWi,#16 | MOVW A, @RWi+d8 | MOVW @RWi+d8,A | | | BT rel |
| + 9 | UNLINK | SUBL A,#32 | SUBW A | SUBW A,#16 | MOVW dir,A | MOVW io,A | INT addr16 | MOVEA RWi,ea | | | | | | | | BNT rel |
| + A | MOV RP,#8 | MOV ILM,#8 | CBNE A, #8,rel | CWBNE A, #16,rel | MOVW A,#16 | MOVW A,addr16 | INTP addr24 | MOV Ri,ea | | | | | | | | BLT rel |
| + B | NEGW | CMPL A,#32 | CMPW A | CMPW A,#16 | MOVL A,#32 | MOVW addr16,A | RETI | MOVW RWi,ea | | | | | | | | BGE rel |
| + C | LSLW A | EXTW | ANDW A | ANDW A,#16 | PUSHW A | POPW A | Bit operation instruction | MOV ea,Ri | | | | | | | | BLE rel |
| + D | | ZEXTW | ORW A | ORW A,#16 | PUSHW AH | POPW AH | | MOVW ea,RWi | | | | | | | | BGT rel |
| + E | ASRW A | SWAPW | XORW A | XORW A,#16 | PUSHW PS | POPW PS | Character string operation instruction | XCH Ri,ea | | | | | | | | BLS rel |
| + F | LSRW A | ADDSP #16 | MULUW A | NOTW A | PUSHW rlst | POPW rlst | 2-byte instruction | XCHW RWi,ea | | | | | | | | BHI rel |

## Table B.9-3 Bit Operation Instruction Map (first byte = 6C$_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + 0 | MOVB<br>A,io:bp | | MOVB<br>io:bp,A | | CLRB<br>io:bp | | SETB<br>io:bp | | BBC io<br>:bp,rel | | BBS io<br>:bp,rel | | WBTS<br>io:bp | | WBTC<br>io:bp | |
| + 1 | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | |
| + 2 | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | |
| + 3 | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | |
| + 4 | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | |
| + 5 | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | |
| + 6 | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | |
| + 7 | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | |
| + 8 | MOVB<br>A,dir:bp | MOVB A,a<br>ddr16:bp | MOVB<br>dir:bp,A | MOVB<br>addr16:bp,A | CLRB<br>dir:bp | CLRB a<br>ddr16:bp | SETB<br>dir:bp | SETB<br>a ddr16:bp | BBC<br>dir:bp,rel | BBC ad16<br>:bp,rel | BBS<br>dir:bp,rel | BBS ad16<br>:bp,rel | | | | SBBS<br>a ddr16:bp |
| + 9 | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | | | | ↓ |
| + A | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | | | | ↓ |
| + B | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | | | | ↓ |
| + C | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | | | | ↓ |
| + D | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | | | | ↓ |
| + E | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | | | | ↓ |
| + F | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | | | | ↓ |

### Table B.9-4 Character String Operation Instruction Map (first byte = $6E_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | MOVSI<br>PCB,PCB | MOVSD | MOVSWI | MOVSWD | | | | | SCEQI<br>PCB | SCEQD<br>PCB | SCWEQI<br>PCB | SCWEQD<br>PCB | FILSI<br>PCB | | FILSWI<br>PCB | |
| +1 | PCB,DTB | | | | | | | | DTB | DTB | DTB | DTB | DTB | | DTB | |
| +2 | PCB,ADB | | | | | | | | ADB | ADB | ADB | ADB | ADB | | ADB | |
| +3 | PCB,SPB | | | | | | | | SPB | SPB | SPB | SPB | SPB | | SPB | |
| +4 | DTB,PCB | | | | | | | | | | | | | | | |
| +5 | DTB,DTB | | | | | | | | | | | | | | | |
| +6 | DTB,ADB | | | | | | | | | | | | | | | |
| +7 | DTB,SPB | | | | | | | | | | | | | | | |
| +8 | ADB,PCB | | | | | | | | | | | | | | | |
| +9 | ADB,DTB | | | | | | | | | | | | | | | |
| +A | ADB,ADB | | | | | | | | | | | | | | | |
| +B | ADB,SPB | | | | | | | | | | | | | | | |
| +C | SPB,PCB | | | | | | | | | | | | | | | |
| +D | SPB,DTB | | | | | | | | | | | | | | | |
| +E | SPB,ADB | | | | | | | | | | | | | | | |
| +F | SPB,SPB | | | | | | | | | | | | | | | |

### Table B.9-5 2-byte Instruction Map (first byte = $6F_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | MOV A,DTB | MOV DTB,A | MOVX A, @RL0+d8 | MOV @RL0 +d8,A | MOV A, @RL0+d8 | | | | | | | | | | | |
| +1 | MOV A,ADB | MOV ADB,A | | | | | | | | | | | | | | |
| +2 | MOV A,SSB | MOV SSB,A | MOVX A, @RL1+d8 | MOV @RL1 +d8,A | MOV A, @RL1+d8 | | | | | | | | | | | |
| +3 | MOV A,USB | MOV USB,A | | | | | | | | | | | | | | |
| +4 | MOV A,DPR | MOV DPR,A | MOVX A, @RL2+d8 | MOV @RL2 +d8,A | MOV A, @RL2+d8 | | | | | | | | | | | |
| +5 | MOV A,@A | MOV @AL,AH | | | | | | | | | | | | | | |
| +6 | MOV A,PCB | MOVX A,@A | MOVX A, @RL3+d8 | MOV @RL3 +d8,A | MOV A, @RL3+d8 | | | | | | | | | | | |
| +7 | ROLC A | RORC A | | | | | | | | | | | | | | |
| +8 | | | | MOVW @RL 0+d8,A | MOVW A, @RL0+d8 | | MUL A | | | | | | | | | |
| +9 | | | | | | | MULW A | | | | | | | | | |
| +A | | | | MOVW @RL 1+d8,A | MOVW A, @RL1+d8 | | DIVU A | | | | | | | | | |
| +B | | | | | | | | | | | | | | | | |
| +C | LSLW A,R0 | LSLL A,R0 | LSL A,R0 | MOVW @RL 2+d8,A | MOVW A, @RL2+d8 | | | | | | | | | | | |
| +D | MOVW A,@A | MOVW @AL,AH | NRML A,R0 | | | | | | | | | | | | | |
| +E | ASRW A,R0 | ASRL A,R0 | ASR A,R0 | MOVW @RL 3+d8,A | MOVW A, @RL3+d8 | | | | | | | | | | | |
| +F | LSRW A,R0 | LSRL A,R0 | LSR A,R0 | | | | | | | | | | | | | |

### Table B.9-6 ea Instruction 1 (first byte = $70_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | CWBNE ↓ | CWBNE ↓ | | | | | | | | | CBNE ↓ | CBNE ↓ |
| + 0 | ADDL<br>A,RL0 | ADDL A,<br>@RW0+d8 | SUBL<br>A,RL0 | SUBL A,<br>@RW0+d8 | RW0,<br>#16,rel | @RW0+d8,<br>#16,rel | CMPL<br>A,RL0 | CMPL A,<br>@RW0+d8 | ANDL<br>A,RL0 | ANDL A,<br>@RW0+d8 | ORL<br>A,RL0 | ORL A,<br>@RW0+d8 | XORL<br>A,RL0 | XORL A,<br>@RW0+d8 | R0,<br>#8,rel | @RW0+d8,<br>#8,rel |
| + 1 | ADDL<br>A,RL0 | ADDL A,<br>@RW1+d8 | SUBL<br>A,RL0 | SUBL A,<br>@RW1+d8 | RW1,<br>#16,rel | @RW1+d8,<br>#16,rel | CMPL<br>A,RL0 | CMPL A,<br>@RW1+d8 | ANDL<br>A,RL0 | ANDL A,<br>@RW1+d8 | ORL<br>A,RL0 | ORL A,<br>@RW1+d8 | XORL<br>A,RL0 | XORL A,<br>@RW1+d8 | R1,<br>#8,rel | @RW1+d8,<br>#8,rel |
| + 2 | ADDL<br>A,RL1 | ADDL A,<br>@RW2+d8 | SUBL<br>A,RL1 | SUBL A,<br>@RW2+d8 | RW2,<br>#16,rel | @RW2+d8,<br>#16,rel | CMPL<br>A,RL1 | CMPL A,<br>@RW2+d8 | ANDL<br>A,RL1 | ANDL A,<br>@RW2+d8 | ORL<br>A,RL1 | ORL A,<br>@RW2+d8 | XORL<br>A,RL1 | XORL A,<br>@RW2+d8 | R2,<br>#8,rel | @RW2+d8,<br>#8,rel |
| + 3 | ADDL<br>A,RL1 | ADDL A,<br>@RW3+d8 | SUBL<br>A,RL1 | SUBL A,<br>@RW3+d8 | RW3,<br>#16,rel | @RW3+d8,<br>#16,rel | CMPL<br>A,RL1 | CMPL A,<br>@RW3+d8 | ANDL<br>A,RL1 | ANDL A,<br>@RW3+d8 | ORL<br>A,RL1 | ORL A,<br>@RW3+d8 | XORL<br>A,RL1 | XORL A,<br>@RW3+d8 | R3,<br>#8,rel | @RW3+d8,<br>#8,rel |
| + 4 | ADDL<br>A,RL2 | ADDL A,<br>@RW4+d8 | SUBL<br>A,RL2 | SUBL A,<br>@RW4+d8 | RW4,<br>#16,rel | @RW4+d8,<br>#16,rel | CMPL<br>A,RL2 | CMPL A,<br>@RW4+d8 | ANDL<br>A,RL2 | ANDL A,<br>@RW4+d8 | ORL<br>A,RL2 | ORL A,<br>@RW4+d8 | XORL<br>A,RL2 | XORL A,<br>@RW4+d8 | R4,<br>#8,rel | @RW4+d8,<br>#8,rel |
| + 5 | ADDL<br>A,RL2 | ADDL A,<br>@RW5+d8 | SUBL<br>A,RL2 | SUBL A,<br>@RW5+d8 | RW5,<br>#16,rel | @RW5+d8,<br>#16,rel | CMPL<br>A,RL2 | CMPL A,<br>@RW5+d8 | ANDL<br>A,RL2 | ANDL A,<br>@RW5+d8 | ORL<br>A,RL2 | ORL A,<br>@RW5+d8 | XORL<br>A,RL2 | XORL A,<br>@RW5+d8 | R5,<br>#8,rel | @RW5+d8,<br>#8,rel |
| + 6 | ADDL<br>A,RL3 | ADDL A,<br>@RW6+d8 | SUBL<br>A,RL3 | SUBL A,<br>@RW6+d8 | RW6,<br>#16,rel | @RW6+d8,<br>#16,rel | CMPL<br>A,RL3 | CMPL A,<br>@RW6+d8 | ANDL<br>A,RL3 | ANDL A,<br>@RW6+d8 | ORL<br>A,RL3 | ORL A,<br>@RW6+d8 | XORL<br>A,RL3 | XORL A,<br>@RW6+d8 | R6,<br>#8,rel | @RW6+d8,<br>#8,rel |
| + 7 | ADDL<br>A,RL3 | ADDL A,<br>@RW7+d8 | SUBL<br>A,RL3 | SUBL A,<br>@RW7+d8 | RW7,<br>#16,rel | @RW7+d8,<br>#16,rel | CMPL<br>A,RL3 | CMPL A,<br>@RW7+d8 | ANDL<br>A,RL3 | ANDL A,<br>@RW7+d8 | ORL<br>A,RL3 | ORL A,<br>@RW7+d8 | XORL<br>A,RL3 | XORL A,<br>@RW7+d8 | R7,<br>#8,rel | @RW7+d8,<br>#8,rel |
| + 8 | ADDL<br>A,@RW0 | ADDL A,<br>@RW0+d16 | SUBL<br>A,@RW0 | SUBL A,<br>@RW0+d16 | @RW0,<br>#16,rel | @RW0+d16<br>, #16,rel | CMPL<br>A,@RW0 | CMPL A,<br>@RW0+d16 | ANDL<br>A,@RW0 | ANDL A,<br>@RW0+d16 | ORL<br>A,@RW0 | ORL A,<br>@RW0+d16 | XORL<br>A,@RW0 | XORL A,<br>@RW0+d16 | @RW0,<br>#8,rel | @RW0+d16<br>,#8,rel |
| + 9 | ADDL<br>A,@RW1 | ADDL A,<br>@RW1+d16 | SUBL<br>A,@RW1 | SUBL A,<br>@RW1+d16 | @RW1,<br>#16,rel | @RW1+d16<br>, #16,rel | CMPL<br>A,@RW1 | CMPL A,<br>@RW1+d16 | ANDL<br>A,@RW1 | ANDL A,<br>@RW1+d16 | ORL<br>A,@RW1 | ORL A,<br>@RW1+d16 | XORL<br>A,@RW1 | XORL A,<br>@RW1+d16 | @RW1,<br>#8,rel | @RW1+d16<br>,#8,rel |
| + A | ADDL<br>A,@RW2 | ADDL A,<br>@RW2+d16 | SUBL<br>A,@RW2 | SUBL A,<br>@RW2+d16 | @RW2,<br>#16,rel | @RW2+d16<br>, #16,rel | CMPL<br>A,@RW2 | CMPL A,<br>@RW2+d16 | ANDL<br>A,@RW2 | ANDL A,<br>@RW2+d16 | ORL<br>A,@RW2 | ORL A,<br>@RW2+d16 | XORL<br>A,@RW2 | XORL A,<br>@RW2+d16 | @RW2,<br>#8,rel | @RW2+d16<br>,#8,rel |
| + B | ADDL<br>A,@RW3 | ADDL A,<br>@RW3+d16 | SUBL<br>A,@RW3 | SUBL A,<br>@RW3+d16 | @RW3,<br>#16,rel | @RW3+d16<br>, #16,rel | CMPL<br>A,@RW3 | CMPL A,<br>@RW3+d16 | ANDL<br>A,@RW3 | ANDL A,<br>@RW3+d16 | ORL<br>A,@RW3 | ORL A,<br>@RW3+d16 | XORL<br>A,@RW3 | XORL A,<br>@RW3+d16 | @RW3,<br>#8,rel | @RW3+d16<br>,#8,rel |
| + C | ADDL<br>A,@RW0+ | ADDL A,<br>@RW0+RW7 | SUBL<br>A,@RW0+ | SUBL A,<br>@RW0+RW7 | Use<br>prohibited | @RW0+RW7<br>, #16,rel | CMPL<br>A,@RW0+ | CMPL A,<br>@RW0+RW7 | ANDL<br>A,@RW0+ | ANDL A,<br>@RW0+RW7 | ORL<br>A,@RW0+ | ORL A,<br>@RW0+RW7 | XORL<br>A,@RW0+ | XORL A,<br>@RW0+RW7 | Use<br>prohibited | @RW0+RW7<br>,#8,rel |
| + D | ADDL<br>A,@RW1+ | ADDL A,<br>@RW1+RW7 | SUBL<br>A,@RW1+ | SUBL A,<br>@RW1+RW7 | Use<br>prohibited | @RW1+RW7<br>, #16,rel | CMPL<br>A,@RW1+ | CMPL A,<br>@RW1+RW7 | ANDL<br>A,@RW1+ | ANDL A,<br>@RW1+RW7 | ORL<br>A,@RW1+ | ORL A,<br>@RW1+RW7 | XORL<br>A,@RW1+ | XORL A,<br>@RW1+RW7 | Use<br>prohibited | @RW1+RW7<br>,#8,rel |
| + E | ADDL<br>A,@RW2+ | ADDL A,<br>@PC+d16 | SUBL<br>A,@RW2+ | SUBL A,<br>@PC+d16 | Use<br>prohibited | @PC+d16,<br>#16,rel | CMPL<br>A,@RW2+ | CMPL A,<br>@PC+d16 | ANDL<br>A,@RW2+ | ANDL A,<br>@PC+d16 | ORL<br>A,@RW2+ | ORL A,<br>@PC+d16 | XORL<br>A,@RW2+ | XORL A,<br>@PC+d16 | Use<br>prohibited | @PC+d16,<br>#8,rel |
| + F | ADDL<br>A,@RW3+ | ADDL A,<br>addr16 | SUBL<br>A,@RW3+ | SUBL A,<br>addr16 | Use<br>prohibited | addr16,<br>#16,rel | CMPL<br>A,@RW3+ | CMPL A,<br>addr16 | ANDL<br>A,@RW3+ | ANDL A,<br>addr16 | ORL<br>A,@RW3+ | ORL A,<br>addr16 | XORL<br>A,@RW3+ | XORL A,<br>addr16 | Use<br>prohibited | addr16,<br>#8,rel |

### Table B.9-7  ea Instruction 2 (first byte = $71_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + 0 | JMPP<br>@RL0 | JMPP<br>@@RW0+d8 | CALLP<br>@RL0 | CALLP<br>@@RW0+d8 | INCL<br>RL0 | INCL<br>@RW0+d8 | DECL<br>RL0 | DECL<br>@RW0+d8 | MOVL<br>A,RL0 | MOVL A,<br>@RW0+d8 | MOVL<br>RL0,A | MOVL @R<br>W0+d8,A | MOV<br>R0,#8 | MOV @R<br>W0+d8,#8 | MOVEA<br>A,RW0 | MOVEA A,<br>@RW0+d8 |
| + 1 | JMPP<br>@RL0 | JMPP<br>@@RW1+d8 | CALLP<br>@RL0 | CALLP<br>@@RW1+d8 | INCL<br>RL0 | INCL<br>@RW1+d8 | DECL<br>RL0 | DECL<br>@RW1+d8 | MOVL<br>A,RL0 | MOVL A,<br>@RW1+d8 | MOVL<br>RL0,A | MOVL @R<br>W1+d8,A | MOV<br>R1,#8 | MOV @R<br>W1+d8,#8 | MOVEA<br>A,RW1 | MOVEA A,<br>@RW1+d8 |
| + 2 | JMPP<br>@RL1 | JMPP<br>@@RW2+d8 | CALLP<br>@RL1 | CALLP<br>@@RW2+d8 | INCL<br>RL1 | INCL<br>@RW2+d8 | DECL<br>RL1 | DECL<br>@RW2+d8 | MOVL<br>A,RL1 | MOVL A,<br>@RW2+d8 | MOVL<br>RL1,A | MOVL @R<br>W2+d8,A | MOV<br>R2,#8 | MOV @R<br>W2+d8,#8 | MOVEA<br>A,RW2 | MOVEA A,<br>@RW2+d8 |
| + 3 | JMPP<br>@RL1 | JMPP<br>@@RW3+d8 | CALLP<br>@RL1 | CALLP<br>@@RW3+d8 | INCL<br>RL1 | INCL<br>@RW3+d8 | DECL<br>RL1 | DECL<br>@RW3+d8 | MOVL<br>A,RL1 | MOVL A,<br>@RW3+d8 | MOVL<br>RL1,A | MOVL @R<br>W3+d8,A | MOV<br>R3,#8 | MOV @R<br>W3+d8,#8 | MOVEA<br>A,RW3 | MOVEA A,<br>@RW3+d8 |
| + 4 | JMPP<br>@RL2 | JMPP<br>@@RW4+d8 | CALLP<br>@RL2 | CALLP<br>@@RW4+d8 | INCL<br>RL2 | INCL<br>@RW4+d8 | DECL<br>RL2 | DECL<br>@RW4+d8 | MOVL<br>A,RL2 | MOVL A,<br>@RW4+d8 | MOVL<br>RL2,A | MOVL @R<br>W4+d8,A | MOV<br>R4,#8 | MOV @R<br>W4+d8,#8 | MOVEA<br>A,RW4 | MOVEA A,<br>@RW4+d8 |
| + 5 | JMPP<br>@RL2 | JMPP<br>@@RW5+d8 | CALLP<br>@RL2 | CALLP<br>@@RW5+d8 | INCL<br>RL2 | INCL<br>@RW5+d8 | DECL<br>RL2 | DECL<br>@RW5+d8 | MOVL<br>A,RL2 | MOVL A,<br>@RW5+d8 | MOVL<br>RL2,A | MOVL @R<br>W5+d8,A | MOV<br>R5,#8 | MOV @R<br>W5+d8,#8 | MOVEA<br>A,RW5 | MOVEA A,<br>@RW5+d8 |
| + 6 | JMPP<br>@RL3 | JMPP<br>@@RW6+d8 | CALLP<br>@RL3 | CALLP<br>@@RW6+d8 | INCL<br>RL3 | INCL<br>@RW6+d8 | DECL<br>RL3 | DECL<br>@RW6+d8 | MOVL<br>A,RL3 | MOVL A,<br>@RW6+d8 | MOVL<br>RL3,A | MOVL @R<br>W6+d8,A | MOV<br>R6,#8 | MOV @R<br>W6+d8,#8 | MOVEA<br>A,RW6 | MOVEA A,<br>@RW6+d8 |
| + 7 | JMPP<br>@RL3 | JMPP<br>@@RW7+d8 | CALLP<br>@RL3 | CALLP<br>@@RW7+d8 | INCL<br>RL3 | INCL<br>@RW7+d8 | DECL<br>RL3 | DECL<br>@RW7+d8 | MOVL<br>A,RL3 | MOVL A,<br>@RW7+d8 | MOVL<br>RL3,A | MOVL @R<br>W7+d8,A | MOV<br>R7,#8 | MOV @R<br>W7+d8,#8 | MOVEA<br>A,RW7 | MOVEA A,<br>@RW7+d8 |
| + 8 | JMPP<br>@@RW0 | JMPP @<br>@RW0+d16 | CALLP<br>@@RW0 | CALLP @<br>@RW0+d16 | INCL<br>@RW0 | INCL<br>@RW0+d16 | DECL<br>@RW0 | DECL<br>@RW0+d16 | MOVL<br>A,@RW0 | MOVL A,<br>@RW0+d16 | MOVL<br>@RW0,A | MOVL @R<br>W0+d16,A | MOV<br>@RW0,#8 | MOV @RW<br>0+d16,#8 | MOVEA<br>A,@RW0 | MOVEA A,<br>@RW0+d16 |
| + 9 | JMPP<br>@@RW1 | JMPP @<br>@RW1+d16 | CALLP<br>@@RW1 | CALLP @<br>@RW1+d16 | INCL<br>@RW1 | INCL<br>@RW1+d16 | DECL<br>@RW1 | DECL<br>@RW1+d16 | MOVL<br>A,@RW1 | MOVL A,<br>@RW1+d16 | MOVL<br>@RW1,A | MOVL @R<br>W1+d16,A | MOV<br>@RW1,#8 | MOV @RW<br>1+d16,#8 | MOVEA<br>A,@RW1 | MOVEA A,<br>@RW1+d16 |
| + A | JMPP<br>@@RW2 | JMPP @<br>@RW2+d16 | CALLP<br>@@RW2 | CALLP @<br>@RW2+d16 | INCL<br>@RW2 | INCL<br>@RW2+d16 | DECL<br>@RW2 | DECL<br>@RW2+d16 | MOVL<br>A,@RW2 | MOVL A,<br>@RW2+d16 | MOVL<br>@RW2,A | MOVL @R<br>W2+d16,A | MOV<br>@RW2,#8 | MOV @RW<br>2+d16,#8 | MOVEA<br>A,@RW2 | MOVEA A,<br>@RW2+d16 |
| + B | JMPP<br>@@RW3 | JMPP @<br>@RW3+d16 | CALLP<br>@@RW3 | CALLP @<br>@RW3+d16 | INCL<br>@RW3 | INCL<br>@RW3+d16 | DECL<br>@RW3 | DECL<br>@RW3+d16 | MOVL<br>A,@RW3 | MOVL A,<br>@RW3+d16 | MOVL<br>@RW3,A | MOVL @R<br>W3+d16,A | MOV<br>@RW3,#8 | MOV @RW<br>3+d16,#8 | MOVEA<br>A,@RW3 | MOVEA A,<br>@RW3+d16 |
| + C | JMPP<br>@@RW0+ | JMPP @<br>@RW0+RW7 | CALLP<br>@@RW0+ | CALLP @<br>@RW0+RW7 | INCL<br>@RW0+ | INCL<br>@RW0+RW7 | DECL<br>@RW0+ | DECL<br>@RW0+RW7 | MOVL<br>A,@RW0+ | MOVL A,<br>@RW0+RW7 | MOVL<br>@RW0+,A | MOVL @R<br>W0+RW7,A | MOV<br>@RW0+,#8 | MOV @RW<br>0+RW7,#8 | MOVEA<br>A,@RW0+ | MOVEA A,<br>@RW0+RW7 |
| + D | JMPP<br>@@RW1+ | JMPP @<br>@RW1+RW7 | CALLP<br>@@RW1+ | CALLP @<br>@RW1+RW7 | INCL<br>@RW1+ | INCL<br>@RW1+RW7 | DECL<br>@RW1+ | DECL<br>@RW1+RW7 | MOVL<br>A,@RW1+ | MOVL A,<br>@RW1+RW7 | MOVL<br>@RW1+,A | MOVL @R<br>W1+RW7,A | MOV<br>@RW1+,#8 | MOV @RW<br>1+RW7,#8 | MOVEA<br>A,@RW1+ | MOVEA A,<br>@RW1+RW7 |
| + E | JMPP<br>@@RW2+ | JMPP<br>@@PC+d16 | CALLP<br>@@RW2+ | CALLP<br>@@PC+d16 | INCL<br>@RW2+ | INCL<br>@PC+d16 | DECL<br>@RW2+ | DECL<br>@PC+d16 | MOVL<br>A,@RW2+ | MOVL A,<br>@PC+d16 | MOVL<br>@RW2+,A | MOVL @P<br>C+d16,A | MOV<br>@RW2+,#8 | MOV @P<br>C+d16,#8 | MOVEA<br>A,@RW2+ | MOVEA A,<br>@PC+d16 |
| + F | JMPP<br>@@RW3+ | JMPP<br>@addr16 | CALLP<br>@@RW3+ | CALLP<br>@addr16 | INCL<br>@RW3+ | INCL<br>addr16 | DECL<br>@RW3+ | DECL<br>addr16 | MOVL<br>A,@RW3+ | MOVL A,<br>addr16 | MOVL<br>@RW3+,A | MOVL<br>addr16,A | MOV<br>@RW3+,#8 | MOV a<br>ddr16,#8 | MOVEA<br>A,@RW3+ | MOVEA A,<br>addr16 |

## Table B.9-8  ea Instruction 3 (first byte = $72_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + 0 | ROLC R0 | ROLC @RW0+d8 | RORC R0 | RORC @RW0+d8 | INC R0 | INC @RW0+d8 | DEC R0 | DEC @RW0+d8 | MOV A,R0 | MOV A, @RW0+d8 | MOV R0,A | MOV @RW0+d8,A | MOVX A,R0 | MOVX A, @RW0+d8 | XCH A,R0 | XCH A, @RW0+d8 |
| + 1 | ROLC R1 | ROLC @RW1+d8 | RORC R1 | RORC @RW1+d8 | INC R1 | INC @RW1+d8 | DEC R1 | DEC @RW1+d8 | MOV A,R1 | MOV A, @RW1+d8 | MOV R1,A | MOV @RW1+d8,A | MOVX A,R1 | MOVX A, @RW1+d8 | XCH A,R1 | XCH A, @RW1+d8 |
| + 2 | ROLC R2 | ROLC @RW2+d8 | RORC R2 | RORC @RW2+d8 | INC R2 | INC @RW2+d8 | DEC R2 | DEC @RW2+d8 | MOV A,R2 | MOV A, @RW2+d8 | MOV R2,A | MOV @RW2+d8,A | MOVX A,R2 | MOVX A, @RW2+d8 | XCH A,R2 | XCH A, @RW2+d8 |
| + 3 | ROLC R3 | ROLC @RW3+d8 | RORC R3 | RORC @RW3+d8 | INC R3 | INC @RW3+d8 | DEC R3 | DEC @RW3+d8 | MOV A,R3 | MOV A, @RW3+d8 | MOV R3,A | MOV @RW3+d8,A | MOVX A,R3 | MOVX A, @RW3+d8 | XCH A,R3 | XCH A, @RW3+d8 |
| + 4 | ROLC R4 | ROLC @RW4+d8 | RORC R4 | RORC @RW4+d8 | INC R4 | INC @RW4+d8 | DEC R4 | DEC @RW4+d8 | MOV A,R4 | MOV A, @RW4+d8 | MOV R4,A | MOV @RW4+d8,A | MOVX A,R4 | MOVX A, @RW4+d8 | XCH A,R4 | XCH A, @RW4+d8 |
| + 5 | ROLC R5 | ROLC @RW5+d8 | RORC R5 | RORC @RW5+d8 | INC R5 | INC @RW5+d8 | DEC R5 | DEC @RW5+d8 | MOV A,R5 | MOV A, @RW5+d8 | MOV R5,A | MOV @RW5+d8,A | MOVX A,R5 | MOVX A, @RW5+d8 | XCH A,R5 | XCH A, @RW5+d8 |
| + 6 | ROLC R6 | ROLC @RW6+d8 | RORC R6 | RORC @RW6+d8 | INC R6 | INC @RW6+d8 | DEC R6 | DEC @RW6+d8 | MOV A,R6 | MOV A, @RW6+d8 | MOV R6,A | MOV @RW6+d8,A | MOVX A,R6 | MOVX A, @RW6+d8 | XCH A,R6 | XCH A, @RW6+d8 |
| + 7 | ROLC R7 | ROLC @RW7+d8 | RORC R7 | RORC @RW7+d8 | INC R7 | INC @RW7+d8 | DEC R7 | DEC @RW7+d8 | MOV A,R7 | MOV A, @RW7+d8 | MOV R7,A | MOV @RW7+d8,A | MOVX A,R7 | MOVX A, @RW7+d8 | XCH A,R7 | XCH A, @RW7+d8 |
| + 8 | ROLC @RW0 | ROLC @RW0+d16 | RORC @RW0 | RORC @RW0+d16 | INC @RW0 | INC @RW0+d16 | DEC @RW0 | DEC @RW0+d16 | MOV A,@RW0 | MOV A, @RW0+d16 | MOV @RW0,A | MOV @RW0+d16,A | MOVX A,@RW0 | MOVX A, @RW0+d16 | XCH A,@RW0 | XCH A, @RW0+d16 |
| + 9 | ROLC @RW1 | ROLC @RW1+d16 | RORC @RW1 | RORC @RW1+d16 | INC @RW1 | INC @RW1+d16 | DEC @RW1 | DEC @RW1+d16 | MOV A,@RW1 | MOV A, @RW1+d16 | MOV @RW1,A | MOV @RW1+d16,A | MOVX A,@RW1 | MOVX A, @RW1+d16 | XCH A,@RW1 | XCH A, @RW1+d16 |
| + A | ROLC @RW2 | ROLC @RW2+d16 | RORC @RW2 | RORC @RW2+d16 | INC @RW2 | INC @RW2+d16 | DEC @RW2 | DEC @RW2+d16 | MOV A,@RW2 | MOV A, @RW2+d16 | MOV @RW2,A | MOV @RW2+d16,A | MOVX A,@RW2 | MOVX A, @RW2+d16 | XCH A,@RW2 | XCH A, @RW2+d16 |
| + B | ROLC @RW3 | ROLC @RW3+d16 | RORC @RW3 | RORC @RW3+d16 | INC @RW3 | INC @RW3+d16 | DEC @RW3 | DEC @RW3+d16 | MOV A,@RW3 | MOV A, @RW3+d16 | MOV @RW3,A | MOV @RW3+d16,A | MOVX A,@RW3 | MOVX A, @RW3+d16 | XCH A,@RW3 | XCH A, @RW3+d16 |
| + C | ROLC @RW0+ | ROLC @RW0+RW7 | RORC @RW0+ | RORC @RW0+RW7 | INC @RW0+ | INC @RW0+RW7 | DEC @RW0+ | DEC @RW0+RW7 | MOV A,@RW0+ | MOV A, @RW0+RW7 | MOV @RW0+,A | MOV @RW0+RW7,A | MOVX A,@RW0+ | MOVX A, @RW0+RW7 | XCH A,@RW0+ | XCH A, @RW0+RW7 |
| + D | ROLC @RW1+ | ROLC @RW1+RW7 | RORC @RW1+ | RORC @RW1+RW7 | INC @RW1+ | INC @RW1+RW7 | DEC @RW1+ | DEC @RW1+RW7 | MOV A,@RW1+ | MOV A, @RW1+RW7 | MOV @RW1+,A | MOV @RW1+RW7,A | MOVX A,@RW1+ | MOVX A, @RW1+RW7 | XCH A,@RW1+ | XCH A, @RW1+RW7 |
| + E | ROLC @RW2+ | ROLC @PC+d16 | RORC @RW2+ | RORC @PC+d16 | INC @RW2+ | INC @PC+d16 | DEC @RW2+ | DEC @PC+d16 | MOV A,@RW2+ | MOV A, @PC+d16 | MOV @RW2+,A | MOV @PC+d16,A | MOVX A,@RW2+ | MOVX A, @PC+d16 | XCH A,@RW2+ | XCH A, @PC+d16 |
| + F | ROLC @RW3+ | ROLC addr16 | RORC @RW3+ | RORC addr16 | INC @RW3+ | INC addr16 | DEC @RW3+ | DEC addr16 | MOV A,@RW3+ | MOV A, addr16 | MOV @RW3+,A | MOV addr16,A | MOVX A,@RW3+ | MOVX A, addr16 | XCH A,@RW3+ | XCH A, addr16 |

## Table B.9-9 ea Instruction 4 (first byte = $73_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | JMP @RW0 | JMP @@RW0+d8 | CALL @RW0 | CALL @@RW0+d8 | INCW RW0 | INCW @RW0+d8 | DECW RW0 | DECW @RW0+d8 | MOVW A,RW0 | MOVW A, @RW0+d8 | MOVW RW0,A | MOVW @RW0+d8,A | MOVW RW0,#16 | MOVW @RW0+d8,#16 | XCHW A,RW0 | XCHW A, @RW0+d8 |
| +1 | JMP @RW1 | JMP @@RW1+d8 | CALL @RW1 | CALL @@RW1+d8 | INCW RW1 | INCW @RW1+d8 | DECW RW1 | DECW @RW1+d8 | MOVW A,RW1 | MOVW A, @RW1+d8 | MOVW RW1,A | MOVW @RW1+d8,A | MOVW RW1,#16 | MOVW @RW1+d8,#16 | XCHW A,RW1 | XCHW A, @RW1+d8 |
| +2 | JMP @RW2 | JMP @@RW2+d8 | CALL @RW2 | CALL @@RW2+d8 | INCW RW2 | INCW @RW2+d8 | DECW RW2 | DECW @RW2+d8 | MOVW A,RW2 | MOVW A, @RW2+d8 | MOVW RW2,A | MOVW @RW2+d8,A | MOVW RW2,#16 | MOVW @RW2+d8,#16 | XCHW A,RW2 | XCHW A, @RW2+d8 |
| +3 | JMP @RW3 | JMP @@RW3+d8 | CALL @RW3 | CALL @@RW3+d8 | INCW RW3 | INCW @RW3+d8 | DECW RW3 | DECW @RW3+d8 | MOVW A,RW3 | MOVW A, @RW3+d8 | MOVW RW3,A | MOVW @RW3+d8,A | MOVW RW3,#16 | MOVW @RW3+d8,#16 | XCHW A,RW3 | XCHW A, @RW3+d8 |
| +4 | JMP @RW4 | JMP @@RW4+d8 | CALL @RW4 | CALL @@RW4+d8 | INCW RW4 | INCW @RW4+d8 | DECW RW4 | DECW @RW4+d8 | MOVW A,RW4 | MOVW A, @RW4+d8 | MOVW RW4,A | MOVW @RW4+d8,A | MOVW RW4,#16 | MOVW @RW4+d8,#16 | XCHW A,RW4 | XCHW A, @RW4+d8 |
| +5 | JMP @RW5 | JMP @@RW5+d8 | CALL @RW5 | CALL @@RW5+d8 | INCW RW5 | INCW @RW5+d8 | DECW RW5 | DECW @RW5+d8 | MOVW A,RW5 | MOVW A, @RW5+d8 | MOVW RW5,A | MOVW @RW5+d8,A | MOVW RW5,#16 | MOVW @RW5+d8,#16 | XCHW A,RW5 | XCHW A, @RW5+d8 |
| +6 | JMP @RW6 | JMP @@RW6+d8 | CALL @RW6 | CALL @@RW6+d8 | INCW RW6 | INCW @RW6+d8 | DECW RW6 | DECW @RW6+d8 | MOVW A,RW6 | MOVW A, @RW6+d8 | MOVW RW6,A | MOVW @RW6+d8,A | MOVW RW6,#16 | MOVW @RW6+d8,#16 | XCHW A,RW6 | XCHW A, @RW6+d8 |
| +7 | JMP @RW7 | JMP @@RW7+d8 | CALL @RW7 | CALL @@RW7+d8 | INCW RW7 | INCW @RW7+d8 | DECW RW7 | DECW @RW7+d8 | MOVW A,RW7 | MOVW A, @RW7+d8 | MOVW RW7,A | MOVW @RW7+d8,A | MOVW RW7,#16 | MOVW @RW7+d8,#16 | XCHW A,RW7 | XCHW A, @RW7+d8 |
| +8 | JMP @@RW0 | JMP @@RW0+d16 | CALL @@RW0 | CALL @@RW0+d16 | INCW @RW0 | INCW @RW0+d16 | DECW @RW0 | DECW @RW0+d16 | MOVW A,@RW0 | MOVW A, @RW0+d16 | MOVW @RW0,A | MOVW @RW0+d16,A | MOVW @RW0,#16 | MOVW@RW0+d16,#16 | XCHW A,@RW0 | XCHW A, @RW0+d16 |
| +9 | JMP @@RW1 | JMP @@RW1+d16 | CALL @@RW1 | CALL @@RW1+d16 | INCW @RW1 | INCW @RW1+d16 | DECW @RW1 | DECW @RW1+d16 | MOVW A,@RW1 | MOVW A, @RW1+d16 | MOVW @RW1,A | MOVW @RW1+d16,A | MOVW @RW1,#16 | MOVW@RW1+d16,#16 | XCHW A,@RW1 | XCHW A, @RW1+d16 |
| +A | JMP @@RW2 | JMP @@RW2+d16 | CALL @@RW2 | CALL @@RW2+d16 | INCW @RW2 | INCW @RW2+d16 | DECW @RW2 | DECW @RW2+d16 | MOVW A,@RW2 | MOVW A, @RW2+d16 | MOVW @RW2,A | MOVW @RW2+d16,A | MOVW @RW2,#16 | MOVW@RW2+d16,#16 | XCHW A,@RW2 | XCHW A, @RW2+d16 |
| +B | JMP @@RW3 | JMP @@RW3+d16 | CALL @@RW3 | CALL @@RW3+d16 | INCW @RW3 | INCW @RW3+d16 | DECW @RW3 | DECW @RW3+d16 | MOVW A,@RW3 | MOVW A, @RW3+d16 | MOVW @RW3,A | MOVW @RW3+d16,A | MOVW @RW3,#16 | MOVW@RW3+d16,#16 | XCHW A,@RW3 | XCHW A, @RW3+d16 |
| +C | JMP @@RW0+ | JMP @@RW0+RW7 | CALL @@RW0+ | CALL @@RW0+RW7 | INCW @RW0+ | INCW @RW0+RW7 | DECW @RW0+ | DECW @RW0+RW7 | MOVW A,@RW0+ | MOVW A, @RW0+RW7 | MOVW @RW0+,A | MOVW @RW0+RW7,A | MOVW @RW0+,#16 | MOVW@RW0+RW7,#16 | XCHW A,@RW0+ | XCHW A, @RW0+RW7 |
| +D | JMP @@RW1+ | JMP @@RW1+RW7 | CALL @@RW1+ | CALL @@RW1+RW7 | INCW @RW1+ | INCW @RW1+RW7 | DECW @RW1+ | DECW @RW1+RW7 | MOVW A,@RW1+ | MOVW A, @RW1+RW7 | MOVW @RW1+,A | MOVW @RW1+RW7,A | MOVW @RW1+,#16 | MOVW@RW1+RW7,#16 | XCHW A,@RW1+ | XCHW A, @RW1+RW7 |
| +E | JMP @@RW2+ | JMP @@PC+d16 | CALL @@RW2+ | CALL @@PC+d16 | INCW @RW2+ | INCW @PC+d16 | DECW @RW2+ | DECW @PC+d16 | MOVW A,@RW2+ | MOVW A, @PC+d16 | MOVW @RW2+,A | MOVW @PC+d16,A | MOVW @RW2+,#16 | MOVW @PC+d16,#16 | XCHW A,@RW2+ | XCHW A, @PC+d16 |
| +F | JMP @@RW3+ | JMP @addr16 | CALL @@RW3+ | CALL @addr16 | INCW @RW3+ | INCW addr16 | DECW @RW3+ | DECW addr16 | MOVW A,@RW3+ | MOVW A, addr16 | MOVW @RW3+,A | MOVW addr16,A | MOVW @RW3+,#16 | MOVW addr16,#16 | XCHW A,@RW3+ | XCHW A, addr16 |

### Table B.9-10 ea Instruction 5 (first byte = $74_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + 0 | ADD<br>A,R0 | ADD A,<br>@RW0+d8 | SUB<br>A,R0 | SUB A,<br>@RW0+d8 | ADDC<br>A,R0 | ADDC A,<br>@RW0+d8 | CMP<br>A,R0 | CMP A,<br>@RW0+d8 | AND<br>A,R0 | AND A,<br>@RW0+d8 | OR<br>A,R0 | OR A,<br>@RW0+d8 | XOR<br>A,R0 | XOR A,<br>@RW0+d8 | DBNZ<br>R0,r | DBNZ @<br>RW0+d8,r |
| + 1 | ADD<br>A,R1 | ADD A,<br>@RW1+d8 | SUB<br>A,R1 | SUB A,<br>@RW1+d8 | ADDC<br>A,R1 | ADDC A,<br>@RW1+d8 | CMP<br>A,R1 | CMP A,<br>@RW1+d8 | AND<br>A,R1 | AND A,<br>@RW1+d8 | OR<br>A,R1 | OR A,<br>@RW1+d8 | XOR<br>A,R1 | XOR A,<br>@RW1+d8 | DBNZ<br>R1,r | DBNZ @<br>RW1+d8,r |
| + 2 | ADD<br>A,R2 | ADD A,<br>@RW2+d8 | SUB<br>A,R2 | SUB A,<br>@RW2+d8 | ADDC<br>A,R2 | ADDC A,<br>@RW2+d8 | CMP<br>A,R2 | CMP A,<br>@RW2+d8 | AND<br>A,R2 | AND A,<br>@RW2+d8 | OR<br>A,R2 | OR A,<br>@RW2+d8 | XOR<br>A,R2 | XOR A,<br>@RW2+d8 | DBNZ<br>R2,r | DBNZ @<br>RW2+d8,r |
| + 3 | ADD<br>A,R3 | ADD A,<br>@RW3+d8 | SUB<br>A,R3 | SUB A,<br>@RW3+d8 | ADDC<br>A,R3 | ADDC A,<br>@RW3+d8 | CMP<br>A,R3 | CMP A,<br>@RW3+d8 | AND<br>A,R3 | AND A,<br>@RW3+d8 | OR<br>A,R3 | OR A,<br>@RW3+d8 | XOR<br>A,R3 | XOR A,<br>@RW3+d8 | DBNZ<br>R3,r | DBNZ @<br>RW3+d8,r |
| + 4 | ADD<br>A,R4 | ADD A,<br>@RW4+d8 | SUB<br>A,R4 | SUB A,<br>@RW4+d8 | ADDC<br>A,R4 | ADDC A,<br>@RW4+d8 | CMP<br>A,R4 | CMP A,<br>@RW4+d8 | AND<br>A,R4 | AND A,<br>@RW4+d8 | OR<br>A,R4 | OR A,<br>@RW4+d8 | XOR<br>A,R4 | XOR A,<br>@RW4+d8 | DBNZ<br>R4,r | DBNZ @<br>RW4+d8,r |
| + 5 | ADD<br>A,R5 | ADD A,<br>@RW5+d8 | SUB<br>A,R5 | SUB A,<br>@RW5+d8 | ADDC<br>A,R5 | ADDC A,<br>@RW5+d8 | CMP<br>A,R5 | CMP A,<br>@RW5+d8 | AND<br>A,R5 | AND A,<br>@RW5+d8 | OR<br>A,R5 | OR A,<br>@RW5+d8 | XOR<br>A,R5 | XOR A,<br>@RW5+d8 | DBNZ<br>R5,r | DBNZ @<br>RW5+d8,r |
| + 6 | ADD<br>A,R6 | ADD A,<br>@RW6+d8 | SUB<br>A,R6 | SUB A,<br>@RW6+d8 | ADDC<br>A,R6 | ADDC A,<br>@RW6+d8 | CMP<br>A,R6 | CMP A,<br>@RW6+d8 | AND<br>A,R6 | AND A,<br>@RW6+d8 | OR<br>A,R6 | OR A,<br>@RW6+d8 | XOR<br>A,R6 | XOR A,<br>@RW6+d8 | DBNZ<br>R6,r | DBNZ @<br>RW6+d8,r |
| + 7 | ADD<br>A,R7 | ADD A,<br>@RW7+d8 | SUB<br>A,R7 | SUB A,<br>@RW7+d8 | ADDC<br>A,R7 | ADDC A,<br>@RW7+d8 | CMP<br>A,R7 | CMP A,<br>@RW7+d8 | AND<br>A,R7 | AND A,<br>@RW7+d8 | OR<br>A,R7 | OR A,<br>@RW7+d8 | XOR<br>A,R7 | XOR A,<br>@RW7+d8 | DBNZ<br>R7,r | DBNZ @<br>RW7+d8,r |
| + 8 | ADD<br>A,@RW0 | ADD A,<br>@RW0+d16 | SUB<br>A,@RW0 | SUB A,<br>@RW0+d16 | ADDC<br>A,@RW0 | ADDC A,<br>@RW0+d16 | CMP<br>A,@RW0 | CMP A,<br>@RW0+d16 | AND<br>A,@RW0 | AND A,<br>@RW0+d16 | OR<br>A,@RW0 | OR A,<br>@RW0+d16 | XOR<br>A,@RW0 | XOR A,<br>@RW0+d16 | DBNZ<br>@RW0,r | DBNZ @R<br>W0+d16,r |
| + 9 | ADD<br>A,@RW1 | ADD A,<br>@RW1+d16 | SUB<br>A,@RW1 | SUB A,<br>@RW1+d16 | ADDC<br>A,@RW1 | ADDC A,<br>@RW1+d16 | CMP<br>A,@RW1 | CMP A,<br>@RW1+d16 | AND<br>A,@RW1 | AND A,<br>@RW1+d16 | OR<br>A,@RW1 | OR A,<br>@RW1+d16 | XOR<br>A,@RW1 | XOR A,<br>@RW1+d16 | DBNZ<br>@RW1,r | DBNZ @R<br>W1+d16,r |
| + A | ADD<br>A,@RW2 | ADD A,<br>@RW2+d16 | SUB<br>A,@RW2 | SUB A,<br>@RW2+d16 | ADDC<br>A,@RW2 | ADDC A,<br>@RW2+d16 | CMP<br>A,@RW2 | CMP A,<br>@RW2+d16 | AND<br>A,@RW2 | AND A,<br>@RW2+d16 | OR<br>A,@RW2 | OR A,<br>@RW2+d16 | XOR<br>A,@RW2 | XOR A,<br>@RW2+d16 | DBNZ<br>@RW2,r | DBNZ @R<br>W2+d16,r |
| + B | ADD<br>A,@RW3 | ADD A,<br>@RW3+d16 | SUB<br>A,@RW3 | SUB A,<br>@RW3+d16 | ADDC<br>A,@RW3 | ADDC A,<br>@RW3+d16 | CMP<br>A,@RW3 | CMP A,<br>@RW3+d16 | AND<br>A,@RW3 | AND A,<br>@RW3+d16 | OR<br>A,@RW3 | OR A,<br>@RW3+d16 | XOR<br>A,@RW3 | XOR A,<br>@RW3+d16 | DBNZ<br>@RW3,r | DBNZ @R<br>W3+d16,r |
| + C | ADD<br>A,@RW0+ | ADD A,<br>@RW0+RW7 | SUB<br>A,@RW0+ | SUB A,<br>@RW0+RW7 | ADDC<br>A,@RW0+ | ADDC A,<br>@RW0+RW7 | CMP<br>A,@RW0+ | CMP A,<br>@RW0+RW7 | AND<br>A,@RW0+ | AND A,<br>@RW0+RW7 | OR<br>A,@RW0+ | OR A,<br>@RW0+RW7 | XOR<br>A,@RW0+ | XOR A,<br>@RW0+RW7 | DBNZ<br>@RW0+,r | DBNZ @R<br>W0+RW7,r |
| + D | ADD<br>A,@RW1+ | ADD A,<br>@RW1+RW7 | SUB<br>A,@RW1+ | SUB A,<br>@RW1+RW7 | ADDC<br>A,@RW1+ | ADDC A,<br>@RW1+RW7 | CMP<br>A,@RW1+ | CMP A,<br>@RW1+RW7 | AND<br>A,@RW1+ | AND A,<br>@RW1+RW7 | OR<br>A,@RW1+ | OR A,<br>@RW1+RW7 | XOR<br>A,@RW1+ | XOR A,<br>@RW1+RW7 | DBNZ<br>@RW1+,r | DBNZ @R<br>W1+RW7,r |
| + E | ADD<br>A,@RW2+ | ADD A,<br>@PC+d16 | SUB<br>A,@RW2+ | SUB A,<br>@PC+d16 | ADDC<br>A,@RW2+ | ADDC A,<br>@PC+d16 | CMP<br>A,@RW2+ | CMP A,<br>@PC+d16 | AND<br>A,@RW2+ | AND A,<br>@PC+d16 | OR<br>A,@RW2+ | OR A,<br>@PC+d16 | XOR<br>A,@RW2+ | XOR A,<br>@PC+d16 | DBNZ<br>@RW2+,r | DBNZ @<br>PC+d16,r |
| + F | ADD<br>A,@RW3+ | ADD A,<br>addr16 | SUB<br>A,@RW3+ | SUB A,<br>addr16 | ADDC<br>A,@RW3+ | ADDC A,<br>addr16 | CMP<br>A,@RW3+ | CMP A,<br>addr16 | AND<br>A,@RW3+ | AND A,<br>addr16 | OR<br>A,@RW3+ | OR A,<br>addr16 | XOR<br>A,@RW3+ | XOR A,<br>addr16 | DBNZ<br>@RW3+,r | DBNZ<br>addr16,r |

## Table B.9-11 ea Instruction 6 (first byte = $75_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + 0 | ADD R0,A | ADD @RW0+d8,A | SUB R0,A | SUB @RW0+d8,A | SUBC A,R0 | SUBC A, @RW0+d8 | NEG R0 | NEG @RW0+d8 | AND R0,A | AND @RW0+d8,A | OR R0,A | OR @RW0+d8,A | XOR R0,A | XOR @RW0+d8,A | NOT R0 | NOT @RW0+d8 |
| + 1 | ADD R1,A | ADD @RW1+d8,A | SUB R1,A | SUB @RW1+d8,A | SUBC A,R1 | SUBC A, @RW1+d8 | NEG R1 | NEG @RW1+d8 | AND R1,A | AND @RW1+d8,A | OR R1,A | OR @RW1+d8,A | XOR R1,A | XOR @RW1+d8,A | NOT R1 | NOT @RW1+d8 |
| + 2 | ADD R2,A | ADD @RW2+d8,A | SUB R2,A | SUB @RW2+d8,A | SUBC A,R2 | SUBC A, @RW2+d8 | NEG R2 | NEG @RW2+d8 | AND R2,A | AND @RW2+d8,A | OR R2,A | OR @RW2+d8,A | XOR R2,A | XOR @RW2+d8,A | NOT R2 | NOT @RW2+d8 |
| + 3 | ADD R3,A | ADD @RW3+d8,A | SUB R3,A | SUB @RW3+d8,A | SUBC A,R3 | SUBC A, @RW3+d8 | NEG R3 | NEG @RW3+d8 | AND R3,A | AND @RW3+d8,A | OR R3,A | OR @RW3+d8,A | XOR R3,A | XOR @RW3+d8,A | NOT R3 | NOT @RW3+d8 |
| + 4 | ADD R4,A | ADD @RW4+d8,A | SUB R4,A | SUB @RW4+d8,A | SUBC A,R4 | SUBC A, @RW4+d8 | NEG R4 | NEG @RW4+d8 | AND R4,A | AND @RW4+d8,A | OR R4,A | OR @RW4+d8,A | XOR R4,A | XOR @RW4+d8,A | NOT R4 | NOT @RW4+d8 |
| + 5 | ADD R5,A | ADD @RW5+d8,A | SUB R5,A | SUB @RW5+d8,A | SUBC A,R5 | SUBC A, @RW5+d8 | NEG R5 | NEG @RW5+d8 | AND R5,A | AND @RW5+d8,A | OR R5,A | OR @RW5+d8,A | XOR R5,A | XOR @RW5+d8,A | NOT R5 | NOT @RW5+d8 |
| + 6 | ADD R6,A | ADD @RW6+d8,A | SUB R6,A | SUB @RW6+d8,A | SUBC A,R6 | SUBC A, @RW6+d8 | NEG R6 | NEG @RW6+d8 | AND R6,A | AND @RW6+d8,A | OR R6,A | OR @RW6+d8,A | XOR R6,A | XOR @RW6+d8,A | NOT R6 | NOT @RW6+d8 |
| + 7 | ADD R7,A | ADD @RW7+d8,A | SUB R7,A | SUB @RW7+d8,A | SUBC A,R7 | SUBC A, @RW7+d8 | NEG R7 | NEG @RW7+d8 | AND R7,A | AND @RW7+d8,A | OR R7,A | OR @RW7+d8,A | XOR R7,A | XOR @RW7+d8,A | NOT R7 | NOT @RW7+d8 |
| + 8 | ADD @RW0,A | ADD @RW0+d16,A | SUB @RW0,A | SUB @RW0+d16,A | SUBC A,@RW0 | SUBC A, @RW0+d16 | NEG @RW0 | NEG @RW0+d16 | AND @RW0,A | AND @RW0+d16,A | OR @RW0,A | OR @RW0+d16,A | XOR @RW0,A | XOR @RW0+d16,A | NOT @RW0 | NOT @RW0+d16 |
| + 9 | ADD @RW1,A | ADD @RW1+d16,A | SUB @RW1,A | SUB @RW1+d16,A | SUBC A,@RW1 | SUBC A, @RW1+d16 | NEG @RW1 | NEG @RW1+d16 | AND @RW1,A | AND @RW1+d16,A | OR @RW1,A | OR @RW1+d16,A | XOR @RW1,A | XOR @RW1+d16,A | NOT @RW1 | NOT @RW1+d16 |
| + A | ADD @RW2,A | ADD @RW2+d16,A | SUB @RW2,A | SUB @RW2+d16,A | SUBC A,@RW2 | SUBC A, @RW2+d16 | NEG @RW2 | NEG @RW2+d16 | AND @RW2,A | AND @RW2+d16,A | OR @RW2,A | OR @RW2+d16,A | XOR @RW2,A | XOR @RW2+d16,A | NOT @RW2 | NOT @RW2+d16 |
| + B | ADD @RW3,A | ADD @RW3+d16,A | SUB @RW3,A | SUB @RW3+d16,A | SUBC A,@RW3 | SUBC A, @RW3+d16 | NEG @RW3 | NEG @RW3+d16 | AND @RW3,A | AND @RW3+d16,A | OR @RW3,A | OR @RW3+d16,A | XOR @RW3,A | XOR @RW3+d16,A | NOT @RW3 | NOT @RW3+d16 |
| + C | ADD @RW0+,A | ADD @RW0+RW7,A | SUB @RW0+,A | SUB @RW0+RW7,A | SUBC A,@RW0+ | SUBC A, @RW0+RW7 | NEG @RW0+ | NEG @RW0+RW7 | AND @RW0+,A | AND @RW0+RW7,A | OR @RW0+,A | OR @RW0+RW7,A | XOR @RW0+,A | XOR @RW0+RW7,A | NOT @RW0+ | NOT @RW0+RW7 |
| + D | ADD @RW1+,A | ADD @RW1+RW7,A | SUB @RW1+,A | SUB @RW1+RW7,A | SUBC A,@RW1+ | SUBC A, @RW1+RW7 | NEG @RW1+ | NEG @RW1+RW7 | AND @RW1+,A | AND @RW1+RW7,A | OR @RW1+,A | OR @RW1+RW7,A | XOR @RW1+,A | XOR @RW1+RW7,A | NOT @RW1+ | NOT @RW1+RW7 |
| + E | ADD @RW2+,A | ADD @PC+d16,A | SUB @RW2+,A | SUB @PC+d16,A | SUBC A,@RW2+ | SUBC A, @PC+d16 | NEG @RW2+ | NEG @PC+d16 | AND @RW2+,A | AND @PC+d16,A | OR @RW2+,A | OR @PC+d16,A | XOR @RW2+,A | XOR @PC+d16,A | NOT @RW2+ | NOT @PC+d16 |
| + F | ADD @RW3+,A | ADD addr16,A | SUB @RW3+,A | SUB addr16,A | SUBC A,@RW3+ | SUBC A, addr16 | NEG @RW3+ | NEG addr16 | AND @RW3+,A | AND addr16,A | OR @RW3+,A | OR addr16,A | XOR @RW3+,A | XOR addr16,A | NOT @RW3+ | NOT addr16 |

## Table B.9-12 ea Instruction 7 (first byte = $76_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + 0 | ADDW A,RW0 | ADDW A, @RW0+d8 | SUBW A,RW0 | SUBW A, @RW0+d8 | ADDCW A,RW0 | ADDCW A, @RW0+d8 | CMPW A,RW0 | CMPW A, @RW0+d8 | ANDW A,RW0 | ANDW A, @RW0+d8 | ORW A,RW0 | ORW A, @RW0+d8 | XORW A,RW0 | XORW A, @RW0+d8 | DWBNZ RW0,r | DWBNZ @RW0+d8,r |
| + 1 | ADDW A,RW1 | ADDW A, @RW1+d8 | SUBW A,RW1 | SUBW A, @RW1+d8 | ADDCW A,RW1 | ADDCW A, @RW1+d8 | CMPW A,RW1 | CMPW A, @RW1+d8 | ANDW A,RW1 | ANDW A, @RW1+d8 | ORW A,RW1 | ORW A, @RW1+d8 | XORW A,RW1 | XORW A, @RW1+d8 | DWBNZ RW1,r | DWBNZ @RW1+d8,r |
| + 2 | ADDW A,RW2 | ADDW A, @RW2+d8 | SUBW A,RW2 | SUBW A, @RW2+d8 | ADDCW A,RW2 | ADDCW A, @RW2+d8 | CMPW A,RW2 | CMPW A, @RW2+d8 | ANDW A,RW2 | ANDW A, @RW2+d8 | ORW A,RW2 | ORW A, @RW2+d8 | XORW A,RW2 | XORW A, @RW2+d8 | DWBNZ RW2,r | DWBNZ @RW2+d8,r |
| + 3 | ADDW A,RW3 | ADDW A, @RW3+d8 | SUBW A,RW3 | SUBW A, @RW3+d8 | ADDCW A,RW3 | ADDCW A, @RW3+d8 | CMPW A,RW3 | CMPW A, @RW3+d8 | ANDW A,RW3 | ANDW A, @RW3+d8 | ORW A,RW3 | ORW A, @RW3+d8 | XORW A,RW3 | XORW A, @RW3+d8 | DWBNZ RW3,r | DWBNZ @RW3+d8,r |
| + 4 | ADDW A,RW4 | ADDW A, @RW4+d8 | SUBW A,RW4 | SUBW A, @RW4+d8 | ADDCW A,RW4 | ADDCW A, @RW4+d8 | CMPW A,RW4 | CMPW A, @RW4+d8 | ANDW A,RW4 | ANDW A, @RW4+d8 | ORW A,RW4 | ORW A, @RW4+d8 | XORW A,RW4 | XORW A, @RW4+d8 | DWBNZ RW4,r | DWBNZ @RW4+d8,r |
| + 5 | ADDW A,RW5 | ADDW A, @RW5+d8 | SUBW A,RW5 | SUBW A, @RW5+d8 | ADDCW A,RW5 | ADDCW A, @RW5+d8 | CMPW A,RW5 | CMPW A, @RW5+d8 | ANDW A,RW5 | ANDW A, @RW5+d8 | ORW A,RW5 | ORW A, @RW5+d8 | XORW A,RW5 | XORW A, @RW5+d8 | DWBNZ RW5,r | DWBNZ @RW5+d8,r |
| + 6 | ADDW A,RW6 | ADDW A, @RW6+d8 | SUBW A,RW6 | SUBW A, @RW6+d8 | ADDCW A,RW6 | ADDCW A, @RW6+d8 | CMPW A,RW6 | CMPW A, @RW6+d8 | ANDW A,RW6 | ANDW A, @RW6+d8 | ORW A,RW6 | ORW A, @RW6+d8 | XORW A,RW6 | XORW A, @RW6+d8 | DWBNZ RW6,r | DWBNZ @RW6+d8,r |
| + 7 | ADDW A,RW7 | ADDW A, @RW7+d8 | SUBW A,RW7 | SUBW A, @RW7+d8 | ADDCW A,RW7 | ADDCW A, @RW7+d8 | CMPW A,RW7 | CMPW A, @RW7+d8 | ANDW A,RW7 | ANDW A, @RW7+d8 | ORW A,RW7 | ORW A, @RW7+d8 | XORW A,RW7 | XORW A, @RW7+d8 | DWBNZ RW7,r | DWBNZ @RW7+d8,r |
| + 8 | ADDW A,@RW0 | ADDW A, @RW0+d16 | SUBW A,@RW0 | SUBW A, @RW0+d16 | ADDCW A,@RW0 | ADDCW A, @RW0+d16 | CMPW A,@RW0 | CMPW A, @RW0+d16 | ANDW A,@RW0 | ANDW A, @RW0+d16 | ORW A,@RW0 | ORW A, @RW0+d16 | XORW A,@RW0 | XORW A, @RW0+d16 | DWBNZ @RW0,r | DWBNZ @RW0+d16,r |
| + 9 | ADDW A,@RW1 | ADDW A, @RW1+d16 | SUBW A,@RW1 | SUBW A, @RW1+d16 | ADDCW A,@RW1 | ADDCW A, @RW1+d16 | CMPW A,@RW1 | CMPW A, @RW1+d16 | ANDW A,@RW1 | ANDW A, @RW1+d16 | ORW A,@RW1 | ORW A, @RW1+d16 | XORW A,@RW1 | XORW A, @RW1+d16 | DWBNZ @RW1,r | DWBNZ @RW1+d16,r |
| + A | ADDW A,@RW2 | ADDW A, @RW2+d16 | SUBW A,@RW2 | SUBW A, @RW2+d16 | ADDCW A,@RW2 | ADDCW A, @RW2+d16 | CMPW A,@RW2 | CMPW A, @RW2+d16 | ANDW A,@RW2 | ANDW A, @RW2+d16 | ORW A,@RW2 | ORW A, @RW2+d16 | XORW A,@RW2 | XORW A, @RW2+d16 | DWBNZ @RW2,r | DWBNZ @RW2+d16,r |
| + B | ADDW A,@RW3 | ADDW A, @RW3+d16 | SUBW A,@RW3 | SUBW A, @RW3+d16 | ADDCW A,@RW3 | ADDCW A, @RW3+d16 | CMPW A,@RW3 | CMPW A, @RW3+d16 | ANDW A,@RW3 | ANDW A, @RW3+d16 | ORW A,@RW3 | ORW A, @RW3+d16 | XORW A,@RW3 | XORW A, @RW3+d16 | DWBNZ @RW3,r | DWBNZ @RW3+d16,r |
| + C | ADDW A,@RW0+ | ADDW A, @RW0+RW7 | SUBW A,@RW0+ | SUBW A, @RW0+RW7 | ADDCW A,@RW0+ | ADDCW A, @RW0+RW7 | CMPW A,@RW0+ | CMPW A, @RW0+RW7 | ANDW A,@RW0+ | ANDW A, @RW0+RW7 | ORW A,@RW0+ | ORW A, @RW0+RW7 | XORW A,@RW0+ | XORW A, @RW0+RW7 | DWBNZ @RW0+,r | DWBNZ @RW0+RW7,r |
| + D | ADDW A,@RW1+ | ADDW A, @RW1+RW7 | SUBW A,@RW1+ | SUBW A, @RW1+RW7 | ADDCW A,@RW1+ | ADDCW A, @RW1+RW7 | CMPW A,@RW1+ | CMPW A, @RW1+RW7 | ANDW A,@RW1+ | ANDW A, @RW1+RW7 | ORW A,@RW1+ | ORW A, @RW1+RW7 | XORW A,@RW1+ | XORW A, @RW1+RW7 | DWBNZ @RW1+,r | DWBNZ @RW1+RW7,r |
| + E | ADDW A,@RW2+ | ADDW A, @PC+d16 | SUBW A,@RW2+ | SUBW A, @PC+d16 | ADDCW A,@RW2+ | ADDCW A, @PC+d16 | CMPW A,@RW2+ | CMPW A, @PC+d16 | ANDW A,@RW2+ | ANDW A, @PC+d16 | ORW A,@RW2+ | ORW A, @PC+d16 | XORW A,@RW2+ | XORW A, @PC+d16 | DWBNZ @RW2+,r | DWBNZ @PC+d16,r |
| + F | ADDW A,@RW3+ | ADDW A, addr16 | SUBW A,@RW3+ | SUBW A, addr16 | ADDCW A,@RW3+ | ADDCW A, addr16 | CMPW A,@RW3+ | CMPW A, addr16 | ANDW A,@RW3+ | ANDW A, addr16 | ORW A,@RW3+ | ORW A, addr16 | XORW A,@RW3+ | XORW A, addr16 | DWBNZ @RW3+,r | DWBNZ addr16,r |

### Table B.9-13 ea Instruction 8 (first byte = $77_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + 0 | ADDW<br>RW0,A | ADDW @R<br>W0+d8,A | SUBW<br>RW0,A | SUBW @R<br>W0+d8,A | SUBCW<br>A,RW0 | SUBCW A,<br>@RW0+d8 | NEGW<br>RW0 | NEGW<br>@RW0+d8 | ANDW<br>RW0,A | ANDW @R<br>W0+d8,A | ORW<br>RW0,A | ORW @R<br>W0+d8,A | XORW<br>RW0,A | XORW @R<br>W0+d8,A | NOTW<br>RW0 | NOTW<br>@RW0+d8 |
| + 1 | ADDW<br>RW1,A | ADDW @R<br>W1+d8,A | SUBW<br>RW1,A | SUBW @R<br>W1+d8,A | SUBCW<br>A,RW1 | SUBCW A,<br>@RW1+d8 | NEGW<br>RW1 | NEGW<br>@RW1+d8 | ANDW<br>RW1,A | ANDW @R<br>W1+d8,A | ORW<br>RW1,A | ORW @R<br>W1+d8,A | XORW<br>RW1,A | XORW @R<br>W1+d8,A | NOTW<br>RW1 | NOTW<br>@RW1+d8 |
| + 2 | ADDW<br>RW2,A | ADDW @R<br>W2+d8,A | SUBW<br>RW2,A | SUBW @R<br>W2+d8,A | SUBCW<br>A,RW2 | SUBCW A,<br>@RW2+d8 | NEGW<br>RW2 | NEGW<br>@RW2+d8 | ANDW<br>RW2,A | ANDW @R<br>W2+d8,A | ORW<br>RW2,A | ORW @R<br>W2+d8,A | XORW<br>RW2,A | XORW @R<br>W2+d8,A | NOTW<br>RW2 | NOTW<br>@RW2+d8 |
| + 3 | ADDW<br>RW3,A | ADDW @R<br>W3+d8,A | SUBW<br>RW3,A | SUBW @R<br>W3+d8,A | SUBCW<br>A,RW3 | SUBCW A,<br>@RW3+d8 | NEGW<br>RW3 | NEGW<br>@RW3+d8 | ANDW<br>RW3,A | ANDW @R<br>W3+d8,A | ORW<br>RW3,A | ORW @R<br>W3+d8,A | XORW<br>RW3,A | XORW @R<br>W3+d8,A | NOTW<br>RW3 | NOTW<br>@RW3+d8 |
| + 4 | ADDW<br>RW4,A | ADDW @R<br>W4+d8,A | SUBW<br>RW4,A | SUBW @R<br>W4+d8,A | SUBCW<br>A,RW4 | SUBCW A,<br>@RW4+d8 | NEGW<br>RW4 | NEGW<br>@RW4+d8 | ANDW<br>RW4,A | ANDW @R<br>W4+d8,A | ORW<br>RW4,A | ORW @R<br>W4+d8,A | XORW<br>RW4,A | XORW @R<br>W4+d8,A | NOTW<br>RW4 | NOTW<br>@RW4+d8 |
| + 5 | ADDW<br>RW5,A | ADDW @R<br>W5+d8,A | SUBW<br>RW5,A | SUBW @R<br>W5+d8,A | SUBCW<br>A,RW5 | SUBCW A,<br>@RW5+d8 | NEGW<br>RW5 | NEGW<br>@RW5+d8 | ANDW<br>RW5,A | ANDW @R<br>W5+d8,A | ORW<br>RW5,A | ORW @R<br>W5+d8,A | XORW<br>RW5,A | XORW @R<br>W5+d8,A | NOTW<br>RW5 | NOTW<br>@RW5+d8 |
| + 6 | ADDW<br>RW6,A | ADDW @R<br>W6+d8,A | SUBW<br>RW6,A | SUBW @R<br>W6+d8,A | SUBCW<br>A,RW6 | SUBCW A,<br>@RW6+d8 | NEGW<br>RW6 | NEGW<br>@RW6+d8 | ANDW<br>RW6,A | ANDW @R<br>W6+d8,A | ORW<br>RW6,A | ORW @R<br>W6+d8,A | XORW<br>RW6,A | XORW @R<br>W6+d8,A | NOTW<br>RW6 | NOTW<br>@RW6+d8 |
| + 7 | ADDW<br>RW7,A | ADDW @R<br>W7+d8,A | SUBW<br>RW7,A | SUBW @R<br>W7+d8,A | SUBCW<br>A,RW7 | SUBCW A,<br>@RW7+d8 | NEGW<br>RW7 | NEGW<br>@RW7+d8 | ANDW<br>RW7,A | ANDW @R<br>W7+d8,A | ORW<br>RW7,A | ORW @R<br>W7+d8,A | XORW<br>RW7,A | XORW @R<br>W7+d8,A | NOTW<br>RW7 | NOTW<br>@RW7+d8 |
| + 8 | ADDW<br>@RW0,A | ADDW @R<br>W0+d16,A | SUBW<br>@RW0,A | SUBW @R<br>W0+d16,A | SUBCW<br>A,@RW0 | SUBCW A,<br>@RW0+d16 | NEGW<br>@RW0 | NEGW<br>@RW0+d16 | ANDW<br>@RW0,A | ANDW @R<br>W0+d16,A | ORW<br>@RW0,A | ORW @R<br>W0+d16,A | XORW<br>@RW0,A | XORW @R<br>W0+d16,A | NOTW<br>@RW0 | NOTW<br>@RW0+d16 |
| + 9 | ADDW<br>@RW1,A | ADDW @R<br>W1+d16,A | SUBW<br>@RW1,A | SUBW @R<br>W1+d16,A | SUBCW<br>A,@RW1 | SUBCW A,<br>@RW1+d16 | NEGW<br>@RW1 | NEGW<br>@RW1+d16 | ANDW<br>@RW1,A | ANDW @R<br>W1+d16,A | ORW<br>@RW1,A | ORW @R<br>W1+d16,A | XORW<br>@RW1,A | XORW @R<br>W1+d16,A | NOTW<br>@RW1 | NOTW<br>@RW1+d16 |
| + A | ADDW<br>@RW2,A | ADDW @R<br>W2+d16,A | SUBW<br>@RW2,A | SUBW @R<br>W2+d16,A | SUBCW<br>A,@RW2 | SUBCW A,<br>@RW2+d16 | NEGW<br>@RW2 | NEGW<br>@RW2+d16 | ANDW<br>@RW2,A | ANDW @R<br>W2+d16,A | ORW<br>@RW2,A | ORW @R<br>W2+d16,A | XORW<br>@RW2,A | XORW @R<br>W2+d16,A | NOTW<br>@RW2 | NOTW<br>@RW2+d16 |
| + B | ADDW<br>@RW3,A | ADDW @R<br>W3+d16,A | SUBW<br>@RW3,A | SUBW @R<br>W3+d16,A | SUBCW<br>A,@RW3 | SUBCW A,<br>@RW3+d16 | NEGW<br>@RW3 | NEGW<br>@RW3+d16 | ANDW<br>@RW3,A | ANDW @R<br>W3+d16,A | ORW<br>@RW3,A | ORW @R<br>W3+d16,A | XORW<br>@RW3,A | XORW @R<br>W3+d16,A | NOTW<br>@RW3 | NOTW<br>@RW3+d16 |
| + C | ADDW<br>@RW0+,A | ADDW @R<br>W0+RW7,A | SUBW<br>@RW0+,A | SUBW @R<br>W0+RW7,A | SUBCW<br>A,@RW0+ | SUBCW A,<br>@RW0+RW7 | NEGW<br>@RW0+ | NEGW<br>@RW0+RW7 | ANDW<br>@RW0+,A | ANDW @R<br>W0+RW7,A | ORW<br>@RW0+,A | ORW @R<br>W0+RW7,A | XORW<br>@RW0+,A | XORW @R<br>W0+RW7,A | NOTW<br>@RW0+ | NOTW<br>@RW0+RW7 |
| + D | ADDW<br>@RW1+,A | ADDW @R<br>W1+RW7,A | SUBW<br>@RW1+,A | SUBW @R<br>W1+RW7,A | SUBCW<br>A,@RW1+ | SUBCW A,<br>@RW1+RW7 | NEGW<br>@RW1+ | NEGW<br>@RW1+RW7 | ANDW<br>@RW1+,A | ANDW @R<br>W1+RW7,A | ORW<br>@RW1+,A | ORW @R<br>W1+RW7,A | XORW<br>@RW1+,A | XORW @R<br>W1+RW7,A | NOTW<br>@RW1+ | NOTW<br>@RW1+RW7 |
| + E | ADDW<br>@RW2+,A | ADDW @P<br>C+d16,A | SUBW<br>@RW2+,A | SUBW @P<br>C+d16,A | SUBCW<br>A,@RW2+ | SUBCW A,<br>@PC+d16 | NEGW<br>@RW2+ | NEGW<br>@PC+d16 | ANDW<br>@RW2+,A | ANDW @P<br>C+d16,A | ORW<br>@RW2+,A | ORW @P<br>C+d16,A | XORW<br>@RW2+,A | XORW @P<br>C+d16,A | NOTW<br>@RW2+ | NOTW<br>@PC+d16 |
| + F | ADDW<br>@RW3+,A | ADDW<br>addr16,A | SUBW<br>@RW3+,A | SUBW<br>addr16,A | SUBCW<br>A,@RW3+ | SUBCW A,<br>addr16 | NEGW<br>@RW3+ | NEGW<br>addr16 | ANDW<br>@RW3+,A | ANDW<br>addr16,A | ORW<br>@RW3+,A | ORW<br>addr16,A | XORW<br>@RW3+,A | XORW<br>addr16,A | NOTW<br>@RW3+ | NOTW<br>addr16 |

Table B.9-14 ea Instruction 9 (first byte = $78_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + 0 | MULU A,R0 | MULU A, @RW0+d8 | MULUW A,RW0 | MULUW A, @RW0+d8 | MUL A R0 | MUL A, @RW0+d8 | MULW A,RW0 | MULW A, @RW0+d8 | DIVU A,R0 | DIVU A, @RW0+d8 | DIVUW A,RW0 | DIVUW A, @RW0+d8 | DIV A,R0 | DIV A, @RW0+d8 | DIVW A,RW0 | DIVW A, @RW0+d8 |
| + 1 | MULU A,R1 | MULU A, @RW1+d8 | MULUW A,RW1 | MULUW A, @RW1+d8 | MUL A,R1 | MUL A, @RW1+d8 | MULW A,RW1 | MULW A, @RW1+d8 | DIVU A,R1 | DIVU A, @RW1+d8 | DIVUW A,RW1 | DIVUW A, @RW1+d8 | DIV A,R1 | DIV A, @RW1+d8 | DIVW A,RW1 | DIVW A, @RW1+d8 |
| + 2 | MULU A,R2 | MULU A, @RW2+d8 | MULUW A,RW2 | MULUW A, @RW2+d8 | MUL A,R2 | MUL A, @RW2+d8 | MULW A RW2 | MULW A, @RW2+d8 | DIVU A,R2 | DIVU A, @RW2+d8 | DIVUW A,RW2 | DIVUW A, @RW2+d8 | DIV A,R2 | DIV A, @RW2+d8 | DIVW A,RW2 | DIVW A, @RW2+d8 |
| + 3 | MULU A,R3 | MULU A, @RW3+d8 | MULUW A,RW3 | MULUW A, @RW3+d8 | MUL A,R3 | MUL A, @RW3+d8 | MULW A RW3 | MULW A, @RW3+d8 | DIVU A,R3 | DIVU A, @RW3+d8 | DIVUW A,RW3 | DIVUW A, @RW3+d8 | DIV A,R3 | DIV A, @RW3+d8 | DIVW A,RW3 | DIVW A, @RW3+d8 |
| + 4 | MULU A,R4 | MULU A, @RW4+d8 | MULUW A,RW4 | MULUW A, @RW4+d8 | MUL A,R4 | MUL A, @RW4+d8 | MULW A RW4 | MULW A, @RW4+d8 | DIVU A,R4 | DIVU A, @RW4+d8 | DIVUW A,RW4 | DIVUW A, @RW4+d8 | DIV A,R4 | DIV A, @RW4+d8 | DIVW A,RW4 | DIVW A, @RW4+d8 |
| + 5 | MULU A,R5 | MULU A, @RW5+d8 | MULUW A,RW5 | MULUW A, @RW5+d8 | MUL A,R5 | MUL A, @RW5+d8 | MULW A RW5 | MULW A, @RW5+d8 | DIVU A,R5 | DIVU A, @RW5+d8 | DIVUW A,RW5 | DIVUW A, @RW5+d8 | DIV A,R5 | DIV A, @RW5+d8 | DIVW A,RW5 | DIVW A, @RW5+d8 |
| + 6 | MULU A,R6 | MULU A, @RW6+d8 | MULUW A,RW6 | MULUW A, @RW6+d8 | MUL A R6 | MUL A, @RW6+d8 | MULW A,RW6 | MULW A, @RW6+d8 | DIVU A,R6 | DIVU A, @RW6+d8 | DIVUW A,RW6 | DIVUW A, @RW6+d8 | DIV A,R6 | DIV A, @RW6+d8 | DIVW A,RW6 | DIVW A, @RW6+d8 |
| + 7 | MULU A,R7 | MULU A, @RW7+d8 | MULUW A,RW7 | MULUW A, @RW7+d8 | MUL A,R7 | MUL A, @RW7+d8 | MULW A,RW7 | MULW A, @RW7+d8 | DIVU A,R7 | DIVU A, @RW7+d8 | DIVUW A,RW7 | DIVUW A, @RW7+d8 | DIV A,R7 | DIV A, @RW7+d8 | DIVW A,RW7 | DIVW A, @RW7+d8 |
| + 8 | MULU A,@RW0 | MULU A, @RW0+d16 | MULUW A,@RW0 | MULUW A, @RW0+d16 | MUL A,@RW0 | MUL A, @RW0+d16 | MULW A,@RW0 | MULW A, @RW0+d16 | DIVU A,@RW0 | DIVU A, @RW0+d16 | DIVUW A,@RW0 | DIVUW A, @RW0+d16 | DIV A,@RW0 | DIV A, @RW0+d16 | DIVW A,@RW0 | DIVW A, @RW0+d16 |
| + 9 | MULU A,@RW1 | MULU A, @RW1+d16 | MULUW A,@RW1 | MULUW A, @RW1+d16 | MUL A,@RW1 | MUL A, @RW1+d16 | MULW A @RW1 | MULW A, @RW1+d16 | DIVU A,@RW1 | DIVU A, @RW1+d16 | DIVUW A,@RW1 | DIVUW A, @RW1+d16 | DIV A,@RW1 | DIV A, @RW1+d16 | DIVW A,@RW1 | DIVW A, @RW1+d16 |
| + A | MULU A,@RW2 | MULU A, @RW2+d16 | MULUW A,@RW2 | MULUW A, @RW2+d16 | MUL A,@RW2 | MUL A, @RW2+d16 | MULW A,@RW2 | MULW A, @RW2+d16 | DIVU A,@RW2 | DIVU A, @RW2+d16 | DIVUW A,@RW2 | DIVUW A, @RW2+d16 | DIV A,@RW2 | DIV A, @RW2+d16 | DIVW A,@RW2 | DIVW A, @RW2+d16 |
| + B | MULU A,@RW3 | MULU A, @RW3+d16 | MULUW A,@RW3 | MULUW A, @RW3+d16 | MUL A,@RW3 | MUL A, @RW3+d16 | MULW A,@RW3 | MULW A, @RW3+d16 | DIVU A,@RW3 | DIVU A, @RW3+d16 | DIVUW A,@RW3 | DIVUW A, @RW3+d16 | DIV A,@RW3 | DIV A, @RW3+d16 | DIVW A,@RW3 | DIVW A, @RW3+d16 |
| + C | MULU A,@RW0+ | MULU A, @RW0+RW7 | MULUW A,@RW0+ | MULUW A, @RW0+RW7 | MUL A,@RW0+ | MUL A, @RW0+RW7 | MULW A,@RW0+ | MULW A, @RW0+RW7 | DIVU A,@RW0+ | DIVU A, @RW0+RW7 | DIVUW A,@RW0+ | DIVUW A, @RW0+RW7 | DIV A,@RW0+ | DIV A, @RW0+RW7 | DIVW A,@RW0+ | DIVW A, @RW0+RW7 |
| + D | MULU A,@RW1+ | MULU A, @RW1+RW7 | MULUW A,@RW1+ | MULUW A, @RW1+RW7 | MUL A,@RW1+ | MUL A, @RW1+RW7 | MULW A,@RW1+ | MULW A, @RW1+RW7 | DIVU A,@RW1+ | DIVU A, @RW1+RW7 | DIVUW A,@RW1+ | DIVUW A, @RW1+RW7 | DIV A,@RW1+ | DIV A, @RW1+RW7 | DIVW A,@RW1+ | DIVW A, @RW1+RW7 |
| + E | MULU A,@RW2+ | MULU A, @PC+d16 | MULUW A,@RW2+ | MULUW A, @PC+d16 | MUL A @RW2+ | MUL A, @PC+d16 | MULW A,@RW2+ | MULW A, @PC+d16 | DIVU A,@RW2+ | DIVU A, @PC+d16 | DIVUW A,@RW2+ | DIVUW A, @PC+d16 | DIV A,@RW2+ | DIV A, @PC+d16 | DIVW A,@RW2+ | DIVW A, @PC+d16 |
| + F | MULU A,@RW3+ | MULU A, addr16 | MULUW A,@RW3+ | MULUW A, addr16 | MUL A,@RW3+ | MUL A, addr16 | MULW A,@RW3+ | MULW A, addr16 | DIVU A,@RW3+ | DIVU A, addr16 | DIVUW A,@RW3+ | DIVUW A, addr16 | DIV A,@RW3+ | DIV A, addr16 | DIVW A,@RW3+ | DIVW A, addr16 |

### Table B.9-15 MOVEA RWi, ea Instruction (first byte = $79_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + 0 | MOVEA<br>RW0,RW0 | MOVEA RW0<br>,@RW0+d8 | MOVEA<br>RW1,RW0 | MOVEA RW1<br>,@RW0+d8 | MOVEA<br>RW2,RW0 | MOVEA RW2<br>,@RW0+d8 | MOVEA<br>RW3,RW0 | MOVEA RW3<br>,@RW0+d8 | MOVEA<br>RW4,RW0 | MOVEA RW4<br>,@RW0+d8 | MOVEA<br>RW5,RW0 | MOVEA RW5<br>,@RW0+d8 | MOVEA<br>RW6,RW0 | MOVEA RW6<br>,@RW0+d8 | MOVEA<br>RW7,RW0 | MOVEA RW7<br>,@RW0+d8 |
| + 1 | MOVEA<br>RW0,RW1 | MOVEA RW0<br>,@RW1+d8 | MOVEA<br>RW1,RW1 | MOVEA RW1<br>,@RW1+d8 | MOVEA<br>RW2,RW1 | MOVEA RW2<br>,@RW1+d8 | MOVEA<br>RW3,RW1 | MOVEA RW3<br>,@RW1+d8 | MOVEA<br>RW4,RW1 | MOVEA RW4<br>,@RW1+d8 | MOVEA<br>RW5,RW1 | MOVEA RW5<br>,@RW1+d8 | MOVEA<br>RW6,RW1 | MOVEA RW6<br>,@RW1+d8 | MOVEA<br>RW7,RW1 | MOVEA RW7<br>,@RW1+d8 |
| + 2 | MOVEA<br>RW0,RW2 | MOVEA RW0<br>,@RW2+d8 | MOVEA<br>RW1,RW2 | MOVEA RW1<br>,@RW2+d8 | MOVEA<br>RW2,RW2 | MOVEA RW2<br>,@RW2+d8 | MOVEA<br>RW3,RW2 | MOVEA RW3<br>,@RW2+d8 | MOVEA<br>RW4,RW2 | MOVEA RW4<br>,@RW2+d8 | MOVEA<br>RW5,RW2 | MOVEA RW5<br>,@RW2+d8 | MOVEA<br>RW6,RW2 | MOVEA RW6<br>,@RW2+d8 | MOVEA<br>RW7,RW2 | MOVEA RW7<br>,@RW2+d8 |
| + 3 | MOVEA<br>RW0,RW3 | MOVEA RW0<br>,@RW3+d8 | MOVEA<br>RW1,RW3 | MOVEA RW1<br>,@RW3+d8 | MOVEA<br>RW2,RW3 | MOVEA RW2<br>,@RW3+d8 | MOVEA<br>RW3,RW3 | MOVEA RW3<br>,@RW3+d8 | MOVEA<br>RW4,RW3 | MOVEA RW4<br>,@RW3+d8 | MOVEA<br>RW5,RW3 | MOVEA RW5<br>,@RW3+d8 | MOVEA<br>RW6,RW3 | MOVEA RW6<br>,@RW3+d8 | MOVEA<br>RW7,RW3 | MOVEA RW7<br>,@RW3+d8 |
| + 4 | MOVEA<br>RW0,RW4 | MOVEA RW0<br>,@RW4+d8 | MOVEA<br>RW1,RW4 | MOVEA RW1<br>,@RW4+d8 | MOVEA<br>RW2,RW4 | MOVEA RW2<br>,@RW4+d8 | MOVEA<br>RW3,RW4 | MOVEA RW3<br>,@RW4+d8 | MOVEA<br>RW4,RW4 | MOVEA RW4<br>,@RW4+d8 | MOVEA<br>RW5,RW4 | MOVEA RW5<br>,@RW4+d8 | MOVEA<br>RW6,RW4 | MOVEA RW6<br>,@RW4+d8 | MOVEA<br>RW7,RW4 | MOVEA RW7<br>,@RW4+d8 |
| + 5 | MOVEA<br>RW0,RW5 | MOVEA RW0<br>,@RW5+d8 | MOVEA<br>RW1,RW5 | MOVEA RW1<br>,@RW5+d8 | MOVEA<br>RW2,RW5 | MOVEA RW2<br>,@RW5+d8 | MOVEA<br>RW3,RW5 | MOVEA RW3<br>,@RW5+d8 | MOVEA<br>RW4,RW5 | MOVEA RW4<br>,@RW5+d8 | MOVEA<br>RW5,RW5 | MOVEA RW5<br>,@RW5+d8 | MOVEA<br>RW6,RW5 | MOVEA RW6<br>,@RW5+d8 | MOVEA<br>RW7,RW5 | MOVEA RW7<br>,@RW5+d8 |
| + 6 | MOVEA<br>RW0,RW6 | MOVEA RW0<br>,@RW6+d8 | MOVEA<br>RW1,RW6 | MOVEA RW1<br>,@RW6+d8 | MOVEA<br>RW2,RW6 | MOVEA RW2<br>,@RW6+d8 | MOVEA<br>RW3,RW6 | MOVEA RW3<br>,@RW6+d8 | MOVEA<br>RW4,RW6 | MOVEA RW4<br>,@RW6+d8 | MOVEA<br>RW5,RW6 | MOVEA RW5<br>,@RW6+d8 | MOVEA<br>RW6,RW6 | MOVEA RW6<br>,@RW6+d8 | MOVEA<br>RW7,RW6 | MOVEA RW7<br>,@RW6+d8 |
| + 7 | MOVEA<br>RW0,RW7 | MOVEA RW0<br>,@RW7+d8 | MOVEA<br>RW1,RW7 | MOVEA RW1<br>,@RW7+d8 | MOVEA<br>RW2,RW7 | MOVEA RW2<br>,@RW7+d8 | MOVEA<br>RW3,RW7 | MOVEA RW3<br>,@RW7+d8 | MOVEA<br>RW4,RW7 | MOVEA RW4<br>,@RW7+d8 | MOVEA<br>RW5,RW7 | MOVEA RW5<br>,@RW7+d8 | MOVEA<br>RW6,RW7 | MOVEA RW6<br>,@RW7+d8 | MOVEA<br>RW7,RW7 | MOVEA RW7<br>,@RW7+d8 |
| + 8 | MOVEA<br>RW0,@RW0 | MOVEA RW0<br>,@RW0+d16 | MOVEA<br>RW1,@RW0 | MOVEA RW1<br>,@RW0+d16 | MOVEA<br>RW2,@RW0 | MOVEA RW2<br>,@RW0+d16 | MOVEA<br>RW3,@RW0 | MOVEA RW3<br>,@RW0+d16 | MOVEA<br>RW4,@RW0 | MOVEA RW4<br>,@RW0+d16 | MOVEA<br>RW5,@RW0 | MOVEA RW5<br>,@RW0+d16 | MOVEA<br>RW6,@RW0 | MOVEA RW6<br>,@RW0+d16 | MOVEA<br>RW7,@RW0 | MOVEA RW7<br>,@RW0+d16 |
| + 9 | MOVEA<br>RW0,@RW1 | MOVEA RW0<br>,@RW1+d16 | MOVEA<br>RW1,@RW1 | MOVEA RW1<br>,@RW1+d16 | MOVEA<br>RW2,@RW1 | MOVEA RW2<br>,@RW1+d16 | MOVEA<br>RW3,@RW1 | MOVEA RW3<br>,@RW1+d16 | MOVEA<br>RW4,@RW1 | MOVEA RW4<br>,@RW1+d16 | MOVEA<br>RW5,@RW1 | MOVEA RW5<br>,@RW1+d16 | MOVEA<br>RW6,@RW1 | MOVEA RW6<br>,@RW1+d16 | MOVEA<br>RW7,@RW1 | MOVEA RW7<br>,@RW1+d16 |
| + A | MOVEA<br>RW0,@RW2 | MOVEA RW0<br>,@RW2+d16 | MOVEA<br>RW1,@RW2 | MOVEA RW1<br>,@RW2+d16 | MOVEA<br>RW2,@RW2 | MOVEA RW2<br>,@RW2+d16 | MOVEA<br>RW3,@RW2 | MOVEA RW3<br>,@RW2+d16 | MOVEA<br>RW4,@RW2 | MOVEA RW4<br>,@RW2+d16 | MOVEA<br>RW5,@RW2 | MOVEA RW5<br>,@RW2+d16 | MOVEA<br>RW6,@RW2 | MOVEA RW6<br>,@RW2+d16 | MOVEA<br>RW7,@RW2 | MOVEA RW7<br>,@RW2+d16 |
| + B | MOVEA<br>RW0,@RW3 | MOVEA RW0<br>,@RW3+d16 | MOVEA<br>RW1,@RW3 | MOVEA RW1<br>,@RW3+d16 | MOVEA<br>RW2,@RW3 | MOVEA RW2<br>,@RW3+d16 | MOVEA<br>RW3,@RW3 | MOVEA RW3<br>,@RW3+d16 | MOVEA<br>RW4,@RW3 | MOVEA RW4<br>,@RW3+d16 | MOVEA<br>RW5,@RW3 | MOVEA RW5<br>,@RW3+d16 | MOVEA<br>RW6,@RW3 | MOVEA RW6<br>,@RW3+d16 | MOVEA<br>RW7,@RW3 | MOVEA RW7<br>,@RW3+d16 |
| + C | MOVEA R<br>W0,@RW0+ | MOVEA RW0<br>,@RW0+RW7 | MOVEA R<br>W1,@RW0+ | MOVEA RW1<br>,@RW0+RW7 | MOVEA R<br>W2,@RW0+ | MOVEA RW2<br>,@RW0+RW7 | MOVEA R<br>W3,@RW0+ | MOVEA RW3<br>,@RW0+RW7 | MOVEA R<br>W4,@RW0+ | MOVEA RW4<br>,@RW0+RW7 | MOVEA R<br>W5,@RW0+ | MOVEA RW5<br>,@RW0+RW7 | MOVEA R<br>W6,@RW0+ | MOVEA RW6<br>,@RW0+RW7 | MOVEA R<br>W7,@RW0+ | MOVEA RW7<br>,@RW0+RW7 |
| + D | MOVEA R<br>W0,@RW1+ | MOVEA RW0<br>,@RW1+RW7 | MOVEA R<br>W1,@RW1+ | MOVEA RW1<br>,@RW1+RW7 | MOVEA R<br>W2,@RW1+ | MOVEA RW2<br>,@RW1+RW7 | MOVEA R<br>W3,@RW1+ | MOVEA RW3<br>,@RW1+RW7 | MOVEA R<br>W4,@RW1+ | MOVEA RW4<br>,@RW1+RW7 | MOVEA R<br>W5,@RW1+ | MOVEA RW5<br>,@RW1+RW7 | MOVEA R<br>W6,@RW1+ | MOVEA RW6<br>,@RW1+RW7 | MOVEA R<br>W7,@RW1+ | MOVEA RW7<br>,@RW1+RW7 |
| + E | MOVEA R<br>W0,@RW2+ | MOVEA RW0<br>,@PC+d16 | MOVEA R<br>W1,@RW2+ | MOVEA RW1<br>,@PC+d16 | MOVEA R<br>W2,@RW2+ | MOVEA RW2<br>,@PC+d16 | MOVEA R<br>W3,@RW2+ | MOVEA RW3<br>,@PC+d16 | MOVEA R<br>W4,@RW2+ | MOVEA RW4<br>,@PC+d16 | MOVEA R<br>W5,@RW2+ | MOVEA RW5<br>,@PC+d16 | MOVEA R<br>W6,@RW2+ | MOVEA RW6<br>,@PC+d16 | MOVEA R<br>W7,@RW2+ | MOVEA RW7<br>,@PC+d16 |
| + F | MOVEA R<br>W0,@RW3+ | MOVEA RW0<br>,addr16 | MOVEA R<br>W1,@RW3+ | MOVEA RW1<br>,addr16 | MOVEA R<br>W2,@RW3+ | MOVEA RW2<br>,addr16 | MOVEA R<br>W3,@RW3+ | MOVEA RW3<br>,addr16 | MOVEA R<br>W4,@RW3+ | MOVEA RW4<br>,addr16 | MOVEA R<br>W5,@RW3+ | MOVEA RW5<br>,addr16 | MOVEA R<br>W6,@RW3+ | MOVEA RW6<br>,addr16 | MOVEA R<br>W7,@RW3+ | MOVEA RW7<br>,addr16 |

**Table B.9-16  MOV Ri, ea Instruction (first byte = 7A$_H$)**

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + 0 | MOV R0,R0 | MOV R0, @RW0+d8 | MOV R1,R0 | MOV R1, @RW0+d8 | MOV R2,R0 | MOV R2, @RW0+d8 | MOV R3,R0 | MOV R3, @RW0+d8 | MOV R4,R0 | MOV R4, @RW0+d8 | MOV R5,R0 | MOV R5, @RW0+d8 | MOV R6,R0 | MOV R6, @RW0+d8 | MOV R7,R0 | MOV R7, @RW0+d8 |
| + 1 | MOV R0,R1 | MOV R0, @RW1+d8 | MOV R1,R1 | MOV R1, @RW1+d8 | MOV R2,R1 | MOV R2, @RW1+d8 | MOV R3,R1 | MOV R3, @RW1+d8 | MOV R4,R1 | MOV R4, @RW1+d8 | MOV R5,R1 | MOV R5, @RW1+d8 | MOV R6,R1 | MOV R6, @RW1+d8 | MOV R7,R1 | MOV R7, @RW1+d8 |
| + 2 | MOV R0,R2 | MOV R0, @RW2+d8 | MOV R1,R2 | MOV R1, @RW2+d8 | MOV R2,R2 | MOV R2, @RW2+d8 | MOV R3,R2 | MOV R3, @RW2+d8 | MOV R4,R2 | MOV R4, @RW2+d8 | MOV R5,R2 | MOV R5, @RW2+d8 | MOV R6,R2 | MOV R6, @RW2+d8 | MOV R7,R2 | MOV R7, @RW2+d8 |
| + 3 | MOV R0,R3 | MOV R0, @RW3+d8 | MOV R1,R3 | MOV R1, @RW3+d8 | MOV R2,R3 | MOV R2, @RW3+d8 | MOV R3,R3 | MOV R3, @RW3+d8 | MOV R4,R3 | MOV R4, @RW3+d8 | MOV R5,R3 | MOV R5, @RW3+d8 | MOV R6,R3 | MOV R6, @RW3+d8 | MOV R7,R3 | MOV R7, @RW3+d8 |
| + 4 | MOV R0,R4 | MOV R0, @RW4+d8 | MOV R1,R4 | MOV R1, @RW4+d8 | MOV R2,R4 | MOV R2, @RW4+d8 | MOV R3,R4 | MOV R3, @RW4+d8 | MOV R4,R4 | MOV R4, @RW4+d8 | MOV R5,R4 | MOV R5, @RW4+d8 | MOV R6,R4 | MOV R6, @RW4+d8 | MOV R7,R4 | MOV R7, @RW4+d8 |
| + 5 | MOV R0,R5 | MOV R0, @RW5+d8 | MOV R1,R5 | MOV R1, @RW5+d8 | MOV R2,R5 | MOV R2, @RW5+d8 | MOV R3,R5 | MOV R3, @RW5+d8 | MOV R4,R5 | MOV R4, @RW5+d8 | MOV R5,R5 | MOV R5, @RW5+d8 | MOV R6,R5 | MOV R6, @RW5+d8 | MOV R7,R5 | MOV R7, @RW5+d8 |
| + 6 | MOV R0,R6 | MOV R0, @RW6+d8 | MOV R1,R6 | MOV R1, @RW6+d8 | MOV R2,R6 | MOV R2, @RW6+d8 | MOV R3,R6 | MOV R3, @RW6+d8 | MOV R4,R6 | MOV R4, @RW6+d8 | MOV R5,R6 | MOV R5, @RW6+d8 | MOV R6,R6 | MOV R6, @RW6+d8 | MOV R7,R6 | MOV R7, @RW6+d8 |
| + 7 | MOV R0,R7 | MOV R0, @RW7+d8 | MOV R1,R7 | MOV R1, @RW7+d8 | MOV R2,R7 | MOV R2, @RW7+d8 | MOV R3,R7 | MOV R3, @RW7+d8 | MOV R4,R7 | MOV R4, @RW7+d8 | MOV R5,R7 | MOV R5, @RW7+d8 | MOV R6,R7 | MOV R6, @RW7+d8 | MOV R7,R7 | MOV R7, @RW7+d8 |
| + 8 | MOV R0,@RW0 | MOV R0, @RW0+d16 | MOV R1,@RW0 | MOV R1, @RW0+d16 | MOV R2,@RW0 | MOV R2, @RW0+d16 | MOV R3,@RW0 | MOV R3, @RW0+d16 | MOV R4,@RW0 | MOV R4, @RW0+d16 | MOV R5,@RW0 | MOV R5, @RW0+d16 | MOV R6,@RW0 | MOV R6, @RW0+d16 | MOV R7,@RW0 | MOV R7, @RW0+d16 |
| + 9 | MOV R0,@RW1 | MOV R0, @RW1+d16 | MOV R1,@RW1 | MOV R1, @RW1+d16 | MOV R2,@RW1 | MOV R2, @RW1+d16 | MOV R3,@RW1 | MOV R3, @RW1+d16 | MOV R4,@RW1 | MOV R4, @RW1+d16 | MOV R5,@RW1 | MOV R5, @RW1+d16 | MOV R6,@RW1 | MOV R6, @RW1+d16 | MOV R7,@RW1 | MOV R7, @RW1+d16 |
| + A | MOV R0,@RW2 | MOV R0, @RW2+d16 | MOV R1,@RW2 | MOV R1, @RW2+d16 | MOV R2,@RW2 | MOV R2, @RW2+d16 | MOV R3,@RW2 | MOV R3, @RW2+d16 | MOV R4,@RW2 | MOV R4, @RW2+d16 | MOV R5,@RW2 | MOV R5, @RW2+d16 | MOV R6,@RW2 | MOV R6, @RW2+d16 | MOV R7,@RW2 | MOV R7, @RW2+d16 |
| + B | MOV R0,@RW3 | MOV R0, @RW3+d16 | MOV R1,@RW3 | MOV R1, @RW3+d16 | MOV R2,@RW3 | MOV R2, @RW3+d16 | MOV R3,@RW3 | MOV R3, @RW3+d16 | MOV R4,@RW3 | MOV R4, @RW3+d16 | MOV R5,@RW3 | MOV R5, @RW3+d16 | MOV R6,@RW3 | MOV R6, @RW3+d16 | MOV R7,@RW3 | MOV R7, @RW3+d16 |
| + C | MOV R0, @RW0+ | MOV R0, @RW0+RW7 | MOV R1, @RW0+ | MOV R1, @RW0+RW7 | MOV R2, @RW0+ | MOV R2, @RW0+RW7 | MOV R3, @RW0+ | MOV R3, @RW0+RW7 | MOV R4, @RW0+ | MOV R4, @RW0+RW7 | MOV R5, @RW0+ | MOV R5, @RW0+RW7 | MOV R6, @RW0+ | MOV R6, @RW0+RW7 | MOV R7, @RW0+ | MOV R7, @RW0+RW7 |
| + D | MOV R0, @RW1+ | MOV R0, @RW1+RW7 | MOV R1, @RW1+ | MOV R1, @RW1+RW7 | MOV R2, @RW1+ | MOV R2, @RW1+RW7 | MOV R3, @RW1+ | MOV R3, @RW1+RW7 | MOV R4, @RW1+ | MOV R4, @RW1+RW7 | MOV R5, @RW1+ | MOV R5, @RW1+RW7 | MOV R6, @RW1+ | MOV R6, @RW1+RW7 | MOV R7, @RW1+ | MOV R7, @RW1+RW7 |
| + E | MOV R0, @RW2+ | MOV R0, @PC+d16 | MOV R1, @RW2+ | MOV R1, @PC+d16 | MOV R2, @RW2+ | MOV R2, @PC+d16 | MOV R3, @RW2+ | MOV R3, @PC+d16 | MOV R4, @RW2+ | MOV R4, @PC+d16 | MOV R5, @RW2+ | MOV R5, @PC+d16 | MOV R6, @RW2+ | MOV R6, @PC+d16 | MOV R7, @RW2+ | MOV R7, @PC+d16 |
| + F | MOV R0, @RW3+ | MOV R0, addr16 | MOV R1, @RW3+ | MOV R1, addr16 | MOV R2, @RW3+ | MOV R2, addr16 | MOV R3, @RW3+ | MOV R3, addr16 | MOV R4, @RW3+ | MOV R4, addr16 | MOV R5, @RW3+ | MOV R5, addr16 | MOV R6, @RW3+ | MOV R6, addr16 | MOV R7, @RW3+ | MOV R7, addr16 |

### Table B.9-17 MOVW RWi, ea Instruction (first byte = 7B$_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + 0 | MOVW RW0,RW0 | MOVW RW0,@RW0+d8 | MOVW RW1,RW0 | MOVW RW1,@RW0+d8 | MOVW RW2,RW0 | MOVW RW2,@RW0+d8 | MOVW RW3,RW0 | MOVW RW3,@RW0+d8 | MOVW RW4,RW0 | MOVW RW4,@RW0+d8 | MOVW RW5,RW0 | MOVW RW5,@RW0+d8 | MOVW RW6,RW0 | MOVW RW6,@RW0+d8 | MOVW RW7,RW0 | MOVW RW7,@RW0+d8 |
| + 1 | MOVW RW0,RW1 | MOVW RW0,@RW1+d8 | MOVW RW1,RW1 | MOVW RW1,@RW1+d8 | MOVW RW2,RW1 | MOVW RW2,@RW1+d8 | MOVW RW3,RW1 | MOVW RW3,@RW1+d8 | MOVW RW4,RW1 | MOVW RW4,@RW1+d8 | MOVW RW5,RW1 | MOVW RW5,@RW1+d8 | MOVW RW6,RW1 | MOVW RW6,@RW1+d8 | MOVW RW7,RW1 | MOVW RW7,@RW1+d8 |
| + 2 | MOVW RW0,RW2 | MOVW RW0,@RW2+d8 | MOVW RW1,RW2 | MOVW RW1,@RW2+d8 | MOVW RW2,RW2 | MOVW RW2,@RW2+d8 | MOVW RW3,RW2 | MOVW RW3,@RW2+d8 | MOVW RW4,RW2 | MOVW RW4,@RW2+d8 | MOVW RW5,RW2 | MOVW RW5,@RW2+d8 | MOVW RW6,RW2 | MOVW RW6,@RW2+d8 | MOVW RW7,RW2 | MOVW RW7,, @RW2+d8 |
| + 3 | MOVW RW0,RW3 | MOVW RW0,@RW3+d8 | MOVW RW1,RW3 | MOVW RW1,@RW3+d8 | MOVW RW2,RW3 | MOVW RW2,@RW3+d8 | MOVW RW3,RW3 | MOVW RW3,@RW3+d8 | MOVW RW4,RW3 | MOVW RW4,@RW3+d8 | MOVW RW5,RW3 | MOVW RW5,@RW3+d8 | MOVW RW6,RW3 | MOVW RW6,@RW3+d8 | MOVW RW7,RW3 | MOVW RW7,@RW3+d8 |
| + 4 | MOVW RW0,RW4 | MOVW RW0,@RW4+d8 | MOVW RW1,RW4 | MOVW RW1,@RW4+d8 | MOVW RW2,RW4 | MOVW RW2,@RW4+d8 | MOVW RW3,RW4 | MOVW RW3,@RW4+d8 | MOVW RW4,RW4 | MOVW RW4,@RW4+d8 | MOVW RW5,RW4 | MOVW RW5,@RW4+d8 | MOVW RW6,RW4 | MOVW RW6,@ RW4+d8 | MOVW RW7,RW4 | MOVW RW7,@RW4+d8 |
| + 5 | MOVW RW0,RW5 | MOVW RW0,@RW5+d8 | MOVW RW1,RW5 | MOVW RW1,@RW5+d8 | MOVW RW2,RW5 | MOVW RW2,@RW5+d8 | MOVW RW3,RW5 | MOVW RW3,@RW5+d8 | MOVW RW4,RW5 | MOVW RW4,@RW5+d8 | MOVW RW5,RW5 | MOVW RW5,@RW5+d8 | MOVW RW6,RW5 | MOVW RW6,@RW5+d8 | MOVW RW7,RW5 | MOVW RW7,@RW5+d8 |
| + 6 | MOVW RW0,RW6 | MOVW RW0,@RW6+d8 | MOVW RW1,RW6 | MOVW RW1,@RW6+d8 | MOVW RW2,RW6 | MOVW RW2,@RW6+d8 | MOVW RW3,RW6 | MOVW RW3,@RW6+d8 | MOVW RW4,RW6 | MOVW RW4,@RW6+d8 | MOVW RW5,RW6 | MOVW RW5,@RW6+d8 | MOVW RW6,RW6 | MOVW RW6,@RW6+d8 | MOVW RW7,RW6 | MOVW RW7,@RW6+d8 |
| + 7 | MOVW RW0,RW7 | MOVW RW0,@RW7+d8 | MOVW RW1,RW7 | MOVW RW1,@RW7+d8 | MOVW RW2,RW7 | MOVW RW2,@RW7+d8 | MOVW RW3,RW7 | MOVW RW3,@RW7+d8 | MOVW RW4,RW7 | MOVW RW4,@RW7+d8 | MOVW RW5,RW7 | MOVW RW5,@RW7+d8 | MOVW RW6,RW7 | MOVW RW6,@RW7+d8 | MOVW RW7,RW7 | MOVW RW7,@RW7+d8 |
| + 8 | MOVW RWO,@RW0 | MOVW RWO,@RW0+d16 | MOVW RW1,@RW0 | MOVW RW1,@RW0+d16 | MOVW RW2,@RW0 | MOVW RW2,@RW0+d16 | MOVW RW3,@RW0 | MOVW RW3,@RW0+d16 | MOVW RW4,@RW0 | MOVW RW4,@RW0+d16 | MOVW RW5,@RW0 | MOVW RW5,@RW0+d16 | MOVW RW6,@RW0 | MOVW RW6,@RW0+d16 | MOVW RW7,@RW0 | MOVW RW7,@RW0+d16 |
| + 9 | MOVW RWO,@RW1 | MOVW RWO,@RW1+d16 | MOVW RW1,@RW1 | MOVW RW1,@RW1+d16 | MOVW RW2,@RW1 | MOVW RW2,@RW1+d16 | MOVW RW3,@RW1 | MOVW RW3,@RW1+d16 | MOVW RW4,@RW1 | MOVW RW4,@RW1+d16 | MOVW RW5,@RW1 | MOVW RW5,@RW1+d16 | MOVW RW6,@RW1 | MOVW RW6,@RW1+d16 | MOVW RW7,@RW1 | MOVW RW7,@RW1+d16 |
| + A | MOVW RWO,@RW2 | MOVW RWO,@RW2+d16 | MOVW RW1,@RW2 | MOVW RW1,@RW2+d16 | MOVW RW2,@RW2 | MOVW RW2,@RW2+d16 | MOVW RW3,@RW2 | MOVW RW3,@RW2+d16 | MOVW RW4,@RW2 | MOVW RW4,@RW2+d16 | MOVW RW5,@RW2 | MOVW RW5,@RW2+d16 | MOVW RW6,@RW2 | MOVW RW6,@RW2+d16 | MOVW RW7,@RW2 | MOVW RW7,@RW2+d16 |
| + B | MOVW RWO,@RW3 | MOVW RWO,@RW3+d16 | MOVW RW1,@RW3 | MOVW RW1,@RW3+d16 | MOVW RW2,@RW3 | MOVW RW2,@RW3+d16 | MOVW RW3,@RW3 | MOVW RW3,@RW3+d16 | MOVW RW4,@RW3 | MOVW RW4,@RW3+d16 | MOVW RW5,@RW3 | MOVW RW5,@RW3+d16 | MOVW RW6,@RW3 | MOVW RW6,@RW3+d16 | MOVW RW7,@RW3 | MOVW RW7,@RW3+d16 |
| + C | MOVW RW0,@RW0+ | MOVW RWO,@RW0+RW7 | MOVW RW1,@RW0+ | MOVW RW1,@RW0+RW7 | MOVW RW2,@RW0+ | MOVW RW2,@RW0+RW7 | MOVW RW3,@RW0+ | MOVW RW3,@RW0+RW7 | MOVW RW4,@RW0+ | MOVW RW4,@RW0+RW7 | MOVW RW5,@RW0+ | MOVW RW5,@RW0+RW7 | MOVW RW6,@RW0+ | MOVW À@R @RW0+RW7 | MOVW RW7,@RW0+ | MOVW Å@R @RW0+RW7 |
| + D | MOVW RW0,@RW1+ | MOVW RWO,@RW1+RW7 | MOVW RW1,@RW1+ | MOVW RW1,@RW1+RW7 | MOVW RW2,@RW1+ | MOVW RW2,@RW1+RW7 | MOVW RW3,@RW1+ | MOVW RW3,@RW1+RW7 | MOVW RW4,@RW1+ | MOVW RW4,@RW1+RW7 | MOVW RW5,@RW1+ | MOVW RW5,@RW1+RW7 | MOVW RW6,@RW1+ | MOVW RW6,@RW1+RW7 | MOVW RW7,@RW1+ | MOVW RW7,@RW1+RW7 |
| + E | MOVW RW0,@RW2+ | MOVW RW0,@PC+d16 | MOVW RW1,@RW2+ | MOVW RW1,@PC+d16 | MOVW RW2,@RW2+ | MOVW RW2,@PC+d16 | MOVW RW3,@RW2+ | MOVW RW3,@PC+d16 | MOVW RW4,@RW2+ | MOVW RW4,@PC+d16 | MOVW RW5,@RW2+ | MOVW RW5,@PC+d16 | MOVW RW6,@RW2+ | MOVW RW6,@PC+d16 | MOVW RW7,@RW2+ | MOVW RW7,@PC+d16 |
| + F | MOVW RW0,@RW3+ | MOVW RW0,addr16 | MOVW RW1,@RW3+ | MOVW RW1,addr16 | MOVW RW2,@RW3+ | MOVW RW2,addr16 | MOVW RW3,@RW3+ | MOVW RW3,addr16 | MOVW RW4,@RW3+ | MOVW RW4,addr16 | MOVW RW5,@RW3+ | MOVW RW5,addr16 | MOVW RW6,@RW3+ | MOVW RW6,addr16 | MOVW RW7,@RW3+ | MOVW RW7,addr16 |

### Table B.9-18 MOV ea, Ri Instruction (first byte = $7C_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + 0 | MOV R0,R0 | MOV @RW0+d8,R0 | MOV R0,R1 | MOV @RW0+d8,R1 | MOV R0,R2 | MOV @RW0+d8,R2 | MOV R0,R3 | MOV @RW0+d8,R3 | MOV R0,R4 | MOV @RW0+d8,R4 | MOV R0,R5 | MOV @RW0+d8,R5 | MOV R0,R6 | MOV @RW0+d8,R6 | MOV R0,R7 | MOV @RW0+d8,R7 |
| + 1 | MOV R1,R0 | MOV @RW1+d8,R0 | MOV R1,R1 | MOV @RW1+d8,R1 | MOV R1,R2 | MOV @RW1+d8,R2 | MOV R1,R3 | MOV @RW1+d8,R3 | MOV R1,R4 | MOV @RW1+d8,R4 | MOV R1,R5 | MOV @RW1+d8,R5 | MOV R1,R6 | MOV @RW1+d8,R6 | MOV R1,R7 | MOV @RW1+d8,R7 |
| + 2 | MOV R2,R0 | MOV @RW2+d8,R0 | MOV R2,R1 | MOV @RW2+d8,R1 | MOV R2,R2 | MOV @RW2+d8,R2 | MOV R2,R3 | MOV @RW2+d8,R3 | MOV R2,R4 | MOV @RW2+d8,R4 | MOV R2,R5 | MOV @RW2+d8,R5 | MOV R2,R6 | MOV @RW2+d8,R6 | MOV R2,R7 | MOV @RW2+d8,R7 |
| + 3 | MOV R3,R0 | MOV @RW3+d8,R0 | MOV R3,R1 | MOV @RW3+d8,R1 | MOV R3,R2 | MOV @RW3+d8,R2 | MOV R3,R3 | MOV @RW3+d8,R3 | MOV R3,R4 | MOV @RW3+d8,R4 | MOV R3,R5 | MOV @RW3+d8,R5 | MOV R3,R6 | MOV @RW3+d8,R6 | MOV R3,R7 | MOV @RW3+d8,R7 |
| + 4 | MOV R4,R0 | MOV @RW4+d8,R0 | MOV R4,R1 | MOV @RW4+d8,R1 | MOV R4,R2 | MOV @RW4+d8,R2 | MOV R4,R3 | MOV @RW4+d8,R3 | MOV R4,R4 | MOV @RW4+d8,R4 | MOV R4,R5 | MOV @RW4+d8,R5 | MOV R4,R6 | MOV @RW4+d8,R6 | MOV R4,R7 | MOV @RW4+d8,R7 |
| + 5 | MOV R5,R0 | MOV @RW5+d8,R0 | MOV R5,R1 | MOV @RW5+d8,R1 | MOV R5,R2 | MOV @RW5+d8,R2 | MOV R5,R3 | MOV @RW5+d8,R3 | MOV R5,R4 | MOV @RW5+d8,R4 | MOV R5,R5 | MOV @RW5+d8,R5 | MOV R5,R6 | MOV @RW5+d8,R6 | MOV R5,R7 | MOV @RW5+d8,R7 |
| + 6 | MOV R6,R0 | MOV @RW6+d8,R0 | MOV R6,R1 | MOV @RW6+d8,R1 | MOV R6,R2 | MOV @RW6+d8,R2 | MOV R6,R3 | MOV @RW6+d8,R3 | MOV R6,R4 | MOV @RW6+d8,R4 | MOV R6,R5 | MOV @RW6+d8,R5 | MOV R6,R6 | MOV @RW6+d8,R6 | MOV R6,R7 | MOV @RW6+d8,R7 |
| + 7 | MOV R7,R0 | MOV @RW7+d8,R0 | MOV R7,R1 | MOV @RW7+d8,R1 | MOV R7,R2 | MOV @RW7+d8,R2 | MOV R7,R3 | MOV @RW7+d8,R3 | MOV R7,R4 | MOV @RW7+d8,R4 | MOV R7,R5 | MOV @RW7+d8,R5 | MOV R7,R6 | MOV @RW7+d8,R6 | MOV R7,R7 | MOV @RW7+d8,R7 |
| + 8 | MOV @RW0,R0 | MOV @RW0+d16,R0 | MOV @RW0,R1 | MOV @RW0+d16,R1 | MOV @RW0,R2 | MOV @RW0+d16,R2 | MOV @RW0,R3 | MOV @RW0+d16,R3 | MOV @RW0,R4 | MOV @RW0+d16,R4 | MOV @RW0,R5 | MOV @RW0+d16,R5 | MOV @RW0,R6 | MOV @RW0+d16,R6 | MOV @RW0,R7 | MOV @RW0+d16,R7 |
| + 9 | MOV @RW1,R0 | MOV @RW1+d16,R0 | MOV @RW1,R1 | MOV @RW1+d16,R1 | MOV @RW1,R2 | MOV @RW1+d16,R2 | MOV @RW1,R3 | MOV @RW1+d16,R3 | MOV @RW1,R4 | MOV @RW1+d16,R4 | MOV @RW1,R5 | MOV @RW1+d16,R5 | MOV @RW1,R6 | MOV @RW1+d16,R6 | MOV @RW1,R7 | MOV @RW1+d16,R7 |
| + A | MOV @RW2,R0 | MOV @RW2+d16,R0 | MOV @RW2,R1 | MOV @RW2+d16,R1 | MOV @RW2,R2 | MOV @RW2+d16,R2 | MOV @RW2,R3 | MOV @RW2+d16,R3 | MOV @RW2,R4 | MOV @RW2+d16,R4 | MOV @RW2,R5 | MOV @RW2+d16,R5 | MOV @RW2,R6 | MOV @RW2+d16,R6 | MOV @RW2,R7 | MOV @RW2+d16,R7 |
| + B | MOV @RW3,R0 | MOV @RW3+d16,R0 | MOV @RW3,R1 | MOV @RW3+d16,R1 | MOV @RW3,R2 | MOV @RW3+d16,R2 | MOV @RW3,R3 | MOV @RW3+d16,R3 | MOV @RW3,R4 | MOV @RW3+d16,R4 | MOV @RW3,R5 | MOV @RW3+d16,R5 | MOV @RW3,R6 | MOV @RW3+d16,R6 | MOV @RW3,R7 | MOV @RW3+d16,R7 |
| + C | MOV @RW0+,R0 | MOV @RW0+RW7,R0 | MOV @RW0+,R1 | MOV @RW0+RW7,R1 | MOV @RW0+,R2 | MOV @RW0+RW7,R2 | MOV @RW0+,R3 | MOV @RW0+RW7,R3 | MOV @RW0+,R4 | MOV @RW0+RW7,R4 | MOV @RW0+,R5 | MOV @RW0+RW7,R5 | MOV @RW0+,R6 | MOV @RW0+RW7,R6 | MOV @RW0+,R7 | MOV @RW0+RW7,R7 |
| + D | MOV @RW1+,R0 | MOV @RW1+RW7,R0 | MOV @RW1+,R1 | MOV @RW1+RW7,R1 | MOV @RW1+,R2 | MOV @RW1+RW7,R2 | MOV @RW1+,R3 | MOV @RW1+RW7,R3 | MOV @RW1+,R4 | MOV @RW1+RW7,R4 | MOV @RW1+,R5 | MOV @RW1+RW7,R5 | MOV @RW1+,R6 | MOV @RW1+RW7,R6 | MOV @RW1+,R7 | MOV @RW1+RW7,R7 |
| + E | MOV @RW2+,R0 | MOV @PC+d16,R0 | MOV @RW2+,R1 | MOV @PC+d16,R1 | MOV @RW2+,R2 | MOV @PC+d16,R2 | MOV @RW2+,R3 | MOV @PC+d16,R3 | MOV @RW2+,R4 | MOV @PC+d16,R4 | MOV @RW2+,R5 | MOV @PC+d16,R5 | MOV @RW2+,R6 | MOV @PC+d16,R6 | MOV @RW2+,R7 | MOV @PC+d16,R7 |
| + F | MOV @RW3+,R0 | MOV addr16,R0 | MOV @RW3+,R1 | MOV addr16,R1 | MOV @RW3+,R2 | MOV addr16,R2 | MOV @RW3+,R3 | MOV addr16,R3 | MOV @RW3+,R4 | MOV addr16,R4 | MOV @RW3+,R5 | MOV addr16,R5 | MOV @RW3+,R6 | MOV addr16,R6 | MOV @RW3+,R7 | MOV addr16,R7 |

Table B.9-19 MOVW ea, Rwi Instruction (first byte = 7D$_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + 0 | MOVW<br>RW0,RW0 | MOVW @RW<br>0+d8,RW0 | MOVW<br>RW0,RW1 | MOVW @RW<br>0+d8,RW1 | MOVW<br>RW0,RW2 | MOVW @RW<br>0+d8,RW2 | MOVW<br>RW0,RW3 | MOVW @RW<br>0+d8,RW3 | MOVW<br>RW0,RW4 | MOVW @RW<br>0+d8,RW4 | MOVW<br>RW0,RW5 | MOVW @RW<br>0+d8,RW5 | MOVW<br>RW0,RW6 | MOVW @RW<br>0+d8,RW6 | MOVW<br>RW0,RW7 | MOVW @RW<br>0+d8,RW7 |
| + 1 | MOVW<br>RW1,RW0 | MOVW @RW<br>1+d8,RW0 | MOVW<br>RW1,RW1 | MOVW @RW<br>1+d8,RW1 | MOVW<br>RW1,RW2 | MOVW @RW<br>1+d8,RW2 | MOVW<br>RW1,RW3 | MOVW @RW<br>1+d8,RW3 | MOVW<br>RW1,RW4 | MOVW @RW<br>1+d8,RW4 | MOVW<br>RW1,RW5 | MOVW @RW<br>1+d8,RW5 | MOVW<br>RW1,RW6 | MOVW @RW<br>1+d8,RW6 | MOVW<br>RW1,RW7 | MOVW @RW<br>1+d8,RW7 |
| + 2 | MOVW<br>RW2,RW0 | MOVW @RW<br>2+d8,RW0 | MOVW<br>RW2,RW1 | MOVW @RW<br>2+d8,RW1 | MOVW<br>RW2,RW2 | MOVW @RW<br>2+d8,RW2 | MOVW<br>RW2,RW3 | MOVW @RW<br>2+d8,RW3 | MOVW<br>RW2,RW4 | MOVW @RW<br>2+d8,RW4 | MOVW<br>RW2,RW5 | MOVW @RW<br>2+d8,RW5 | MOVW<br>RW2,RW6 | MOVW @RW<br>2+d8,RW6 | MOVW<br>RW2,RW7 | MOVW @RW<br>2+d8,RW7 |
| + 3 | MOVW<br>RW3,RW0 | MOVW @RW<br>3+d8,RW0 | MOVW<br>RW3,RW1 | MOVW @RW<br>3+d8,RW1 | MOVW<br>RW3,RW2 | MOVW @RW<br>3+d8,RW2 | MOVW<br>RW3,RW3 | MOVW @RW<br>3+d8,RW3 | MOVW<br>RW3,RW4 | MOVW @RW<br>3+d8,RW4 | MOVW<br>RW3,RW5 | MOVW @RW<br>3+d8,RW5 | MOVW<br>RW3,RW6 | MOVW @RW<br>3+d8,RW6 | MOVW<br>RW3,RW7 | MOVW @RW<br>3+d8,RW7 |
| + 4 | MOVW<br>RW4,RW0 | MOVW @RW<br>4+d8,RW0 | MOVW<br>RW4,RW1 | MOVW @RW<br>4+d8,RW1 | MOVW<br>RW4,RW2 | MOVW @RW<br>4+d8,RW2 | MOVW<br>RW4,RW3 | MOVW @RW<br>4+d8,RW3 | MOVW<br>RW4,RW4 | MOVW @RW<br>4+d8,RW4 | MOVW<br>RW4,RW5 | MOVW @RW<br>4+d8,RW5 | MOVW<br>RW4,RW6 | MOVW @RW<br>4+d8,RW6 | MOVW<br>RW4,RW7 | MOVW @RW<br>4+d8,RW7 |
| + 5 | MOVW<br>RW5,RW0 | MOVW @RW<br>5+d8,RW0 | MOVW<br>RW5,RW1 | MOVW @RW<br>5+d8,RW1 | MOVW<br>RW5,RW2 | MOVW @RW<br>5+d8,RW2 | MOVW<br>RW5,RW3 | MOVW @RW<br>5+d8,RW3 | MOVW<br>RW5,RW4 | MOVW @RW<br>5+d8,RW4 | MOVW<br>RW5,RW5 | MOVW @RW<br>5+d8,RW5 | MOVW<br>RW5,RW6 | MOVW @RW<br>5+d8,RW6 | MOVW<br>RW5,RW7 | MOVW @RW<br>5+d8,RW7 |
| + 6 | MOVW<br>RW6,RW0 | MOVW @RW<br>6+d8,RW0 | MOVW<br>RW6,RW1 | MOVW @RW<br>6+d8,RW1 | MOVW<br>RW6,RW2 | MOVW @RW<br>6+d8,RW2 | MOVW<br>RW6,RW3 | MOVW @RW<br>6+d8,RW3 | MOVW<br>RW6,RW4 | MOVW @RW<br>6+d8,RW4 | MOVW<br>RW6,RW5 | MOVW @RW<br>6+d8,RW5 | MOVW<br>RW6,RW6 | MOVW @RW<br>6+d8,RW6 | MOVW<br>RW6,RW7 | MOVW @RW<br>6+d8,RW7 |
| + 7 | MOVW<br>RW7,RW0 | MOVW @RW<br>7+d8,RW0 | MOVW<br>RW7,RW1 | MOVW @RW<br>7+d8,RW1 | MOVW<br>RW7,RW2 | MOVW @RW<br>7+d8,RW2 | MOVW<br>RW7,RW3 | MOVW @RW<br>7+d8,RW3 | WOVW<br>RW7,RW4 | MOVW @RW<br>7+d8,RW4 | WOVW<br>RW7,RW5 | MOVW @RW<br>7+d8,RW5 | WOVW<br>RW7,RW6 | MOVW @RW<br>7+d8,RW6 | WOVW<br>RW7,RW7 | MOVW @RW<br>7+d8,RW7 |
| + 8 | MOVW<br>@RW0,RW0 | MOVW @RW0<br>+d16,RW0 | MOVW<br>@RW0,RW1 | MOVW @RW0<br>+d16,RW1 | MOVW<br>@RW0,RW2 | MOVW @RW0<br>+d16,RW2 | MOVW<br>@RW0,RW3 | MOVW @RW0<br>+d16,RW3 | MOVW<br>@RW0,RW4 | MOVW @RW0<br>+d16,RW4 | MOVW<br>@RW0,RW5 | MOVW @RW0<br>+d16,RW5 | MOVW<br>@RW0,RW6 | MOVW @RW0<br>+d16,RW6 | MOVW<br>@RW0,RW7 | MOVW @RW0<br>+d16,RW7 |
| + 9 | MOVW<br>@RW1,RW0 | MOVW @RW1<br>+d16,RW0 | MOVW<br>@RW1,RW1 | MOVW @RW1<br>+d16,RW1 | MOVW<br>@RW1,RW2 | MOVW @RW1<br>+d16,RW2 | MOVW<br>@RW1,RW3 | MOVW @RW1<br>+d16,RW3 | MOVW<br>@RW1,RW4 | MOVW @RW1<br>+d16,RW4 | MOVW<br>@RW1,RW5 | MOVW @RW1<br>+d16,RW5 | MOVW<br>@RW1,RW6 | MOVW @RW1<br>+d16,RW6 | MOVW<br>@RW1,RW7 | MOVW @RW1<br>+d16,RW7 |
| + A | MOVW<br>@RW2,RW0 | MOVW @RW2<br>+d16,RW0 | MOVW<br>@RW2,RW1 | MOVW @RW2<br>+d16,RW1 | MOVW<br>@RW2,RW2 | MOVW @RW2<br>+d16,RW2 | MOVW<br>@RW2,RW3 | MOVW @RW2<br>+d16,RW3 | MOVW<br>@RW2,RW4 | MOVW @RW2<br>+d16,RW4 | MOVW<br>@RW2,RW5 | MOVW @RW2<br>+d16,RW5 | MOVW<br>@RW2,RW6 | MOVW @RW2<br>+d16,RW6 | MOVW<br>@RW2,RW7 | MOVW @RW2<br>+d16,RW7 |
| + B | MOVW<br>@RW3,RW0 | MOVW @RW3<br>+d16,RW0 | MOVW<br>@RW3,RW1 | MOVW @RW3<br>+d16,RW1 | MOVW<br>@RW3,RW2 | MOVW @RW3<br>+d16,RW2 | MOVW<br>@RW3,RW3 | MOVW @RW3<br>+d16,RW3 | MOVW<br>@RW3,RW4 | MOVW @RW3<br>+d16,RW4 | MOVW<br>@RW3,RW5 | MOVW @RW3<br>+d16,RW5 | MOVW<br>@RW3,RW6 | MOVW @RW3<br>+d16,RW6 | MOVW<br>@RW3,RW7 | MOVW @RW3<br>+d16,RW7 |
| + C | MOVW @<br>RW0+,RW0 | MOVW @RW0<br>+RW7,RW0 | MOVW @<br>RW0+,RW1 | MOVW @RW0<br>+RW7,RW1 | MOVW @<br>RW0+,RW2 | MOVW @RW0<br>+RW7,RW2 | MOVW @<br>RW0+,RW3 | MOVW @RW0<br>+RW7,RW3 | MOVW @<br>RW0+,RW4 | MOVW @RW0<br>+RW7,RW4 | MOVW @<br>RW0+,RW5 | MOVW @RW0<br>+RW7,RW5 | MOVW @<br>RW0+,RW6 | MOVW @RW0<br>+RW7,RW6 | MOVW @<br>RW0+,RW7 | MOVW @RW0<br>+RW7,RW7 |
| + D | MOVW @<br>RW1+,RW0 | MOVW @RW1<br>+RW7,RW0 | MOVW @<br>RW1+,RW1 | MOVW @RW1<br>+RW7,RW1 | MOVW @<br>RW1+,RW2 | MOVW @RW1<br>+RW7,RW2 | MOVW @<br>RW1+,RW3 | MOVW @RW1<br>+RW7,RW3 | MOVW @<br>RW1+,RW4 | MOVW @RW1<br>+RW7,RW4 | MOVW @<br>RW1+,RW5 | MOVW @RW1<br>+RW7,RW5 | MOVW @<br>RW1+,RW6 | MOVW @RW1<br>+RW7,RW6 | MOVW @<br>RW1+,RW7 | MOVW @RW1<br>+RW7,RW7 |
| + E | MOVW @<br>RW2+,RW0 | MOVW @PC+<br>d16,RW0 | MOVW @<br>RW2+,RW1 | MOVW @PC+<br>d16,RW1 | MOVW @<br>RW2+,RW2 | MOVW @PC+<br>d16,RW2 | MOVW @<br>RW2+,RW3 | MOVW @PC+<br>d16,RW3 | MOVW @<br>RW2+,RW4 | MOVW @PC+<br>d16,RW4 | MOVW @<br>RW2+,RW5 | MOVW @PC+<br>d16,RW5 | MOVW @<br>RW2+,RW6 | MOVW @PC+<br>d16,RW6 | MOVW @<br>RW2+,RW7 | MOVW @PC+<br>d16,RW7 |
| + F | MOVW @<br>RW3+,RW0 | MOVW addr<br>16,RW0 | MOVW @<br>RW3+,RW1 | MOVW addr<br>16,RW1 | MOVW @<br>RW3+,RW2 | MOVW addr<br>16,RW2 | MOVW @<br>RW3+,RW3 | MOVW addr<br>16,RW3 | MOVW @<br>RW3+,RW4 | MOVW addr<br>16,RW4 | MOVW @<br>RW3+,RW5 | MOVW addr<br>16,RW5 | MOVW @<br>RW3+,RW6 | MOVW addr<br>16,RW6 | MOVW @<br>RW3+,RW7 | MOVW addr<br>16,RW7 |

### Table B.9-20 XCH Ri, ea Instruction (first byte = 7E$_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + 0 | XCH<br>R0,R0 | XCH R0,<br>@RW0+d8 | XCH<br>R1,R0 | XCH R1,<br>@RW0+d8 | XCH<br>R2,R0 | XCH R2,<br>@RW0+d8 | XCH<br>R3,R0 | XCH R3,<br>@RW0+d8 | XCH<br>R4,R0 | XCH R4,<br>@RW0+d8 | XCH<br>R5,R0 | XCH R5,<br>@RW0+d8 | XCH<br>R6,R0 | XCH R6,<br>@RW0+d8 | XCH<br>R7,R0 | XCH R7,<br>@RW0+d8 |
| + 1 | XCH<br>R0,R1 | XCH R0,<br>@RW1+d8 | XCH<br>R1,R1 | XCH R1,<br>@RW1+d8 | XCH<br>R2,R1 | XCH R2,<br>@RW1+d8 | XCH<br>R3,R1 | XCH R3,<br>@RW1+d8 | XCH<br>R4,R1 | XCH R4,<br>@RW1+d8 | XCH<br>R5,R1 | XCH R5,<br>@RW1+d8 | XCH<br>R6,R1 | XCH R6,<br>@RW1+d8 | XCH<br>R7,R1 | NOTW R7,<br>@RW1+d8 |
| + 2 | XCH<br>R0,R2 | XCH R0,<br>@RW2+d8 | XCH<br>R1,R2 | XCH R1,<br>@RW2+d8 | XCH<br>R2,R2 | XCH R2,<br>@RW2+d8 | XCH<br>R3,R2 | XCH R3,<br>@RW2+d8 | XCH<br>R4,R2 | XCH R4,<br>@RW2+d8 | XCH<br>R5,R2 | XCH R5,<br>@RW2+d8 | XCH<br>R6,R2 | XCH R6,<br>@RW2+d8 | XCH<br>R7,R2 | XCH R7,<br>@RW2+d8 |
| + 3 | XCH<br>R0,R3 | XCH R0,<br>@RW3+d8 | XCH<br>R1,R3 | XCH R1,<br>@RW3+d8 | XCH<br>R2,R3 | XCH R2,<br>@RW3+d8 | XCH<br>R3,R3 | XCH R3,<br>@RW3+d8 | XCH<br>R4,R3 | XCH R4,<br>@RW3+d8 | XCH<br>R5,R3 | XCH R5,<br>@RW3+d8 | XCH<br>R6,R3 | XCH R6,<br>@RW3+d8 | XCH<br>R7,R3 | NOTW R7,<br>@RW3+d8 |
| + 4 | XCH<br>R0,R4 | XCH R0,<br>@RW4+d8 | XCH<br>R1,R4 | XCH R1,<br>@RW4+d8 | XCH<br>R2,R4 | XCH R2,<br>@RW4+d8 | XCH<br>R3,R4 | XCH R3,<br>@RW4+d8 | XCH<br>R4,R4 | XCH R4,<br>@RW4+d8 | XCH<br>R5,R4 | XCH R5,<br>@RW4+d8 | XCH<br>R6,R4 | XCH R6,<br>@RW4+d8 | XCH<br>R7,R4 | NOTW R7,<br>@RW4+d8 |
| + 5 | XCH<br>R0,R5 | XCH R0,<br>@RW5+d8 | XCH<br>R1,R5 | XCH R1,<br>@RW5+d8 | XCH<br>R2,R5 | XCH R2,<br>@RW5+d8 | XCH<br>R3,R5 | XCH R3,<br>@RW5+d8 | XCH<br>R4,R5 | XCH R4,<br>@RW5+d8 | XCH<br>R5,R5 | XCH R5,<br>@RW5+d8 | XCH<br>R6,R5 | XCH R6,<br>@RW5+d8 | XCH<br>R7,R5 | XCH R7,<br>@RW5+d8 |
| + 6 | XCH<br>R0,R6 | XCH R0,<br>@RW6+d8 | XCH<br>R1,R6 | XCH R1,<br>@RW6+d8 | XCH<br>R2,R6 | XCH R2,<br>@RW6+d8 | XCH<br>R3,R6 | XCH R3,<br>@RW6+d8 | XCH<br>R4,R6 | XCH R4,<br>@RW6+d8 | XCH<br>R5,R6 | XCH R5,<br>@RW6+d8 | XCH<br>R6,R6 | XCH R6,<br>@RW6+d8 | XCH<br>R7,R6 | XCH R7,<br>@RW6+d8 |
| + 7 | XCH<br>R0,R7 | XCH R0,<br>@RW7+d8 | XCH<br>R1,R7 | XCH R1,<br>@RW7+d8 | XCH<br>R2,R7 | XCH R2,<br>@RW7+d8 | XCH<br>R3,R7 | XCH R3,<br>@RW7+d8 | XCH<br>R4,R7 | XCH R4,<br>@RW7+d8 | XCH<br>R5,R7 | XCH R5,<br>@RW7+d8 | XCH<br>R6,R7 | XCH R6,<br>@RW7+d8 | XCH<br>R7,R7 | XCH R7,<br>@RW7+d8 |
| + 8 | XCH<br>R0,@RW0 | XCH R0,<br>@RW0+d16 | XCH<br>R1,@RW0 | XCH R1,<br>@RW0+d16 | XCH<br>R2,@RW0 | XCH R2,<br>@RW0+d16 | XCH<br>R3,@RW0 | XCH R3,<br>@RW0+d16 | XCH<br>R4,@RW0 | XCH R4,<br>@RW0+d16 | XCH<br>R5,@RW0 | XCH R5,<br>@RW0+d16 | XCH<br>R6,@RW0 | XCH R6,<br>@RW0+d16 | XCH<br>R7,@RW0 | XCH R7,<br>@RW0+d16 |
| + 9 | XCH<br>R0,@RW1 | XCH R0,<br>@RW1+d16 | XCH<br>R1,@RW1 | XCH R1,<br>@RW1+d16 | XCH<br>R2,@RW1 | XCH R2,<br>@RW1+d16 | XCH<br>R3,@RW1 | XCH R3,<br>@RW1+d16 | XCH<br>R4,@RW1 | XCH R4,<br>@RW1+d16 | XCH<br>R5,@RW1 | XCH R5,<br>@RW1+d16 | XCH<br>R6,@RW1 | XCH R6,<br>@RW1+d16 | XCH<br>R7,@RW1 | XCH R7,<br>@RW1+d16 |
| + A | XCH<br>R0,@RW2 | XCH R0,<br>W2+d16,A | XCH<br>R1,@RW2 | XCH R1,<br>W2+d16,A | XCH<br>R2,@RW2 | XCH R2,<br>W2+d16,A | XCH<br>R3,@RW2 | XCH R3,<br>W2+d16,A | XCH<br>R4,@RW2 | XCH R4,<br>W2+d16,A | XCH<br>R5,@RW2 | XCH R5,<br>W2+d16,A | XCH<br>R6,@RW2 | XCH R6,<br>W2+d16,A | XCH<br>R7,@RW2 | XCH R7,<br>W2+d16,A |
| + B | XCH<br>R0,@RW3 | XCH R0,<br>@RW3+d16 | XCH<br>R1,@RW3 | XCH R1,<br>@RW3+d16 | XCH<br>R2,@RW3 | XCH R2,<br>@RW3+d16 | XCH<br>R3,@RW3 | XCH R3,<br>@RW3+d16 | XCH<br>R4,@RW3 | XCH R4,<br>@RW3+d16 | XCH<br>R5,@RW3 | XCH R5,<br>@RW3+d16 | XCH<br>R6,@RW3 | XCH R6,<br>@RW3+d16 | XCH<br>R7,@RW3 | XCH R7,<br>@RW3+d16 |
| + C | XCH<br>R0,@RW0+ | XCH R0,<br>@RW0+RW7 | XCH<br>R1,@RW0+ | XCH R1,<br>@RW0+RW7 | XCH<br>R2,@RW0+ | XCH R2,<br>@RW0+RW7 | XCH<br>R3,@RW0+ | XCH R3,<br>@RW0+RW7 | XCH<br>R4,@RW0+ | XCH R4,<br>@RW0+RW7 | XCH<br>R5,@RW0+ | XCH R5,<br>@RW0+RW7 | XCH<br>R6,@RW0+ | XCH R6,<br>@RW0+RW7 | XCH<br>R7,@RW0+ | XCH R7,<br>@RW0+RW7 |
| + D | XCH<br>R0,@RW1+ | XCH R0,<br>@RW1+RW7 | XCH<br>R1,@RW1+ | XCH R1,<br>@RW1+RW7 | XCH<br>R2,@RW1+ | XCH R2,<br>@RW1+RW7 | XCH<br>R3,@RW1+ | XCH R3,<br>@RW1+RW7 | XCH<br>R4,@RW1+ | XCH R4,<br>@RW1+RW7 | XCH<br>R5,@RW1+ | XCH R5,<br>@RW1+RW7 | XCH<br>R6,@RW1+ | XCH R6,<br>@RW1+RW7 | XCH<br>R7,@RW1+ | XCH R7,<br>@RW1+RW7 |
| + E | XCH<br>R0,@RW2+ | XCH R0,<br>@PC+d16 | XCH<br>R1,@RW2+ | XCH R1,<br>@PC+d16 | XCH<br>R2,@RW2+ | XCH R2,<br>@PC+d16 | XCH<br>R3,@RW2+ | XCH R3,<br>@PC+d16 | XCH<br>R4,@RW2+ | XCH R4,<br>@PC+d16 | XCH<br>R5,@RW2+ | XCH R5,<br>@PC+d16 | XCH<br>R6,@RW2+ | XCH R6,<br>@PC+d16 | XCH<br>R7,@RW2+ | XCH R7,<br>@PC+d16 |
| + F | XCH<br>R0,@RW3+ | XCH R0,<br>addr16 | XCH<br>R1,@RW3+ | XCH R1,<br>addr16 | XCH<br>R2,@RW3+ | XCH R2,<br>addr16 | XCH<br>R3,@RW3+ | XCH R3,<br>addr16 | XCH<br>R4,@RW3+ | XCH R4,<br>addr16 | XCH<br>R5,@RW3+ | XCH R5,<br>addr16 | XCH<br>R6,@RW3+ | XCH R6,<br>addr16 | XCH<br>R7,@RW3+ | XCH R7,<br>addr16 |

## Table B.9-21  XCHW RWi, ea Instruction (first byte = 7F$_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| + 0 | XCHW<br>RW0,RW0 | XCHW RW0,<br>@RW0+d8 | XCHW<br>RW1,RW0 | XCHW RW1,<br>@RW0+d8 | XCHW<br>RW2,RW0 | XCHW RW2,<br>@RW0+d8 | XCHW<br>RW3,RW0 | XCHW RW3,<br>@RW0+d8 | XCHW<br>RW4,RW0 | XCHW RW4,<br>@RW0+d8 | XCHW<br>RW5,RW0 | XCHW RW5,<br>@RW0+d8 | XCHW<br>RW6,RW0 | XCHW RW6,<br>@RW0+d8 | XCHW<br>RW7,RW0 | XCHW RW7,<br>@RW0+d8 |
| + 1 | XCHW<br>RW0,RW1 | XCHW RW0,<br>@RW1+d8 | XCHW<br>RW1,RW1 | XCHW RW1,<br>@RW1+d8 | XCHW<br>RW2,RW1 | XCHW RW2,<br>@RW1+d8 | XCHW<br>RW3,RW1 | XCHW RW3,<br>@RW1+d8 | XCHW<br>RW4,RW1 | XCHW RW4,<br>@RW1+d8 | XCHW<br>RW5,RW1 | XCHW RW5,<br>@RW1+d8 | XCHW<br>RW6,RW1 | XCHW RW6,<br>@RW1+d8 | XCHW<br>RW7,RW1 | XCHW RW7,<br>@RW1+d8 |
| + 2 | XCHW<br>RW0,RW2 | XCHW RW0,<br>@RW2+d8 | XCHW<br>RW1,RW2 | XCHW RW1,<br>@RW2+d8 | XCHW<br>RW2,RW2 | XCHW RW2,<br>@RW2+d8 | XCHW<br>RW3,RW2 | XCHW RW3,<br>@RW2+d8 | XCHW<br>RW4,RW2 | XCHW RW4,<br>@RW2+d8 | XCHW<br>RW5,RW2 | XCHW RW5,<br>@RW2+d8 | XCHW<br>RW6,RW2 | XCHW RW6,<br>@RW2+d8 | XCHW<br>RW7,RW2 | XCHW RW7,<br>@RW2+d8 |
| + 3 | XCHW<br>RW0,RW3 | XCHW RW0,<br>@RW3+d8 | XCHW<br>RW1,RW3 | XCHW RW1,<br>@RW3+d8 | XCHW<br>RW2,RW3 | XCHW RW2,<br>@RW3+d8 | XCHW<br>RW3,RW3 | XCHW RW3,<br>@RW3+d8 | XCHW<br>RW4,RW3 | XCHW RW4,<br>@RW3+d8 | XCHW<br>RW5,RW3 | XCHW RW5,<br>@RW3+d8 | XCHW<br>RW6,RW3 | XCHW RW6,<br>@RW3+d8 | XCHW<br>RW7,RW3 | XCHW RW7,<br>@RW3+d8 |
| + 4 | XCHW<br>RW0,RW4 | XCHW RW0,<br>@RW4+d8 | XCHW<br>RW1,RW4 | XCHW RW1,<br>@RW4+d8 | XCHW<br>RW2,RW4 | XCHW RW2,<br>@RW4+d8 | XCHW<br>RW3,RW4 | XCHW RW3,<br>@RW4+d8 | XCHW<br>RW4,RW4 | XCHW RW4,<br>@RW4+d8 | XCHW<br>RW5,RW4 | XCHW RW5,<br>@RW4+d8 | XCHW<br>RW6,RW4 | XCHW RW6,<br>@RW4+d8 | XCHW<br>RW7,RW4 | XCHW RW7,<br>@RW4+d8 |
| + 5 | XCHW<br>RW0,RW5 | XCHW RW0,<br>@RW5+d8 | XCHW<br>RW1,RW5 | XCHW RW1,<br>@RW5+d8 | XCHW<br>RW2,RW5 | XCHW RW2,<br>@RW5+d8 | XCHW<br>RW3,RW5 | XCHW RW3,<br>@RW5+d8 | XCHW<br>RW4,RW5 | XCHW RW4,<br>@RW5+d8 | XCHW<br>RW5,RW5 | XCHW RW5,<br>@RW5+d8 | XCHW<br>RW6,RW5 | XCHW RW6,<br>@RW5+d8 | XCHW<br>RW7,RW5 | XCHW RW7,<br>@RW5+d8 |
| + 6 | XCHW<br>RW0,RW6 | XCHW RW0,<br>@RW6+d8 | XCHW<br>RW1,RW6 | XCHW RW1,<br>@RW6+d8 | XCHW<br>RW2,RW6 | XCHW RW2,<br>@RW6+d8 | XCHW<br>RW3,RW6 | XCHW RW3,<br>@RW6+d8 | XCHW<br>RW4,RW6 | XCHW RW4,<br>@RW6+d8 | XCHW<br>RW5,RW6 | XCHW RW5,<br>@RW6+d8 | XCHW<br>RW6,RW6 | XCHW RW6,<br>@RW6+d8 | XCHW<br>RW7,RW6 | XCHW RW7,<br>@RW6+d8 |
| + 7 | XCHW<br>RW0,RW7 | XCHW RW0,<br>@RW7+d8 | XCHW<br>RW1,RW7 | XCHW RW1,<br>@RW7+d8 | XCHW<br>RW2,RW7 | XCHW RW2,<br>@RW7+d8 | XCHW<br>RW3,RW7 | XCHW RW3,<br>@RW7+d8 | XCHW<br>RW4,RW7 | XCHW RW4,<br>@RW7+d8 | XCHW<br>RW5,RW7 | XCHW RW5,<br>@RW7+d8 | XCHW<br>RW6,RW7 | XCHW RW6,<br>@RW7+d8 | XCHW<br>RW7,RW7 | XCHW RW7,<br>@RW7+d8 |
| + 8 | XCHW<br>RW0,@RW0 | XCHW RW0,<br>@RW0+d16 | XCHW<br>RW1,@RW0 | XCHW RW1,<br>@RW0+d16 | XCHW<br>RW2,@RW0 | XCHW RW2,<br>@RW0+d16 | XCHW<br>RW3,@RW0 | XCHW RW3,<br>@RW0+d16 | XCHW<br>RW4,@RW0 | XCHW RW4,<br>@RW0+d16 | XCHW<br>RW5,@RW0 | XCHW RW5,<br>@RW0+d16 | XCHW<br>RW6,@RW0 | XCHW RW6,<br>@RW0+d16 | XCHW<br>RW7,@RW0 | XCHW RW7,<br>@RW0+d16 |
| + 9 | XCHW<br>RW0,@RW1 | XCHW RW0,<br>@RW1+d16 | XCHW<br>RW1,@RW1 | XCHW RW1,<br>@RW1+d16 | XCHW<br>RW2,@RW1 | XCHW RW2,<br>@RW1+d16 | XCHW<br>RW3,@RW1 | XCHW RW3,<br>@RW1+d16 | XCHW<br>RW4,@RW1 | XCHW RW4,<br>@RW1+d16 | XCHW<br>RW5,@RW1 | XCHW RW5,<br>@RW1+d16 | XCHW<br>RW6,@RW1 | XCHW RW6,<br>@RW1+d16 | XCHW<br>RW7,@RW1 | XCHW RW7,<br>@RW1+d16 |
| + A | XCHW<br>RW0,@RW2 | XCHW RW0,<br>@RW2+d16 | XCHW<br>RW1,@RW2 | XCHW RW1,<br>@RW2+d16 | XCHW<br>RW2,@RW2 | XCHW RW2,<br>@RW2+d16 | XCHW<br>RW3,@RW2 | XCHW RW3,<br>@RW2+d16 | XCHW<br>RW4,@RW2 | XCHW RW4,<br>@RW2+d16 | XCHW<br>RW5,@RW2 | XCHW RW5,<br>@RW2+d16 | XCHW<br>RW6,@RW2 | XCHW RW6,<br>@RW2+d16 | XCHW<br>RW7,@RW2 | XCHW RW7,<br>@RW2+d16 |
| + B | XCHW<br>RW0,@RW3 | XCHW RW0,<br>@RW3+d16 | XCHW<br>RW1,@RW3 | XCHW RW1,<br>@RW3+d16 | XCHW<br>RW2,@RW3 | XCHW RW2,<br>@RW3+d16 | XCHW<br>RW3,@RW3 | XCHW RW3,<br>@RW3+d16 | XCHW<br>RW4,@RW3 | XCHW RW4,<br>@RW3+d16 | XCHW<br>RW5,@RW3 | XCHW RW5,<br>@RW3+d16 | XCHW<br>RW6,@RW3 | XCHW RW6,<br>@RW3+d16 | XCHW<br>RW7,@RW3 | XCHW RW7,<br>@RW3+d16 |
| + C | XCHW R<br>W0,@RW0+ | XCHW RW0,<br>@RW0+RW7 | XCHW R<br>W1,@RW0+ | XCHW RW1,<br>@RW0+RW7 | XCHW R<br>W2,@RW0+ | XCHW RW2,<br>@RW0+RW7 | XCHW R<br>W3,@RW0+ | XCHW RW3,<br>@RW0+RW7 | XCHW R<br>W4,@RW0+ | XCHW RW4,<br>@RW0+RW7 | XCHW R<br>W5,@RW0+ | XCHW RW5,<br>@RW0+RW7 | XCHW R<br>W6,@RW0+ | XCHW RW6,<br>@RW0+RW7 | XCHW R<br>W7,@RW0+ | XCHW RW7,<br>@RW0+RW7 |
| + D | XCHW R<br>WO,@RW1+ | XCHW RW0,<br>@RW1+RW7 | XCHW R<br>W1,@RW1+ | XCHW RW1,<br>@RW1+RW7 | XCHW R<br>W2,@RW1+ | XCHW RW2,<br>@RW1+RW7 | XCHW R<br>W3,@RW1+ | XCHW RW3,<br>@RW1+RW7 | XCHW R<br>W4,@RW1+ | XCHW RW4,<br>@RW1+RW7 | XCHW R<br>W5,@RW1+ | XCHW RW5,<br>@RW1+RW7 | XCHW R<br>W6,@RW1+ | XCHW RW6,<br>@RW1+RW7 | XCHW R<br>W7,@RW1+ | XCHW RW7,<br>@RW1+RW7 |
| + E | XCHW R<br>WO,@RW2+ | XCHW RW0,<br>@PC+d16 | XCHW R<br>W1,@RW2+ | XCHW RW1,<br>@PC+d16 | XCHW R<br>W2,@RW2+ | XCHW RW2,<br>@PC+d16 | XCHW R<br>W3,@RW2+ | XCHW RW3,<br>@PC+d16 | XCHW R<br>W4,@RW2+ | XCHW RW4,<br>@PC+d16 | XCHW R<br>W5,@RW2+ | XCHW RW5,<br>@PC+d16 | XCHW R<br>W6,@RW2+ | XCHW RW6,<br>@PC+d16 | XCHW R<br>W7,@RW2+ | XCHW RW7,<br>@PC+d16 |
| + F | XCHW R<br>WO,@RW3+ | XCHW RW0,<br>addr16 | XCHW R<br>W1,@RW3+ | XCHW RW1,<br>addr16 | XCHW R<br>W2,@RW3+ | XCHW RW2,<br>addr16 | XCHW R<br>W3,@RW3+ | XCHW RW3,<br>addr16 | XCHW R<br>W4,@RW3+ | XCHW RW4,<br>addr16 | XCHW R<br>W5,@RW3+ | XCHW RW5,<br>addr16 | XCHW R<br>W6,@RW3+ | XCHW RW6,<br>addr16 | XCHW R<br>W7,@RW3+ | XCHW RW7,<br>addr16 |

# APPENDIX C Index of Registers

**This section provides an index for finding the page containing the description of an MB90M405 series register from the register address, peripheral resource name, abbreviation, or register name.**

## ■ Index of Registers

**Table C-1 Index of Registers**

| Address | Abbreviation | Register | Peripheral resource | Page number |
|---|---|---|---|---|
| 000008$_H$ | PDR8 | Port 8 data register | Port 8 | 158 |
| 000009$_H$ | PDR9 | Port 9 data register | Port 9 | 163 |
| 00000A$_H$ | PDRA | Port A data register | Port A | 168 |
| 00000B$_H$ | PDRB | Port B data register | Port B | 174 |
| 000018$_H$ | DDR8 | Port 8 direction register | Port 8 | 158 |
| 000019$_H$ | DDR9 | Port 9 direction register | Port 9 | 163 |
| 00001A$_H$ | DDRA | Port A direction register | Port A | 168 |
| 00001B$_H$ | DDRB | Port B direction register | Port B | 174 |
| 00001E$_H$ | ADER0 | Analog input enable register 0 | Port A, A/D | 168 |
| 00001F$_H$ | ADER1 | Analog input enable register 1 | Port B, A/D | 174 |
| 000020$_H$ | SMR(0) | Mode register ch0 | UART ch0 | 275 |
| 000021$_H$ | SCR(0) | Control register ch0 | | 272 |
| 000022$_H$ | SIDR(0)/ SODR(0) | Input data register ch0/Output data register ch0 | | 280, 280 |
| 000023$_H$ | SSR(0) | Status register ch0 | | 277 |
| 000024$_H$ | SMR(1) | Mode register ch1 | UART ch1 | 275 |
| 000025$_H$ | SCR(1) | Control register ch1 | | 272 |
| 000026$_H$ | SIDR(1)/ SODR(1) | Input data register ch0/Output data register ch1 | | 280, 280 |
| 000027$_H$ | SSR(1) | Status register ch1 | | 277 |
| 000028$_H$ | CDCR(0) | Communication prescaler control register ch0 | Communication prescaler 0 | 282 |
| 000029$_H$ | CDCR(1) | Communication prescaler control register ch1 | Communication prescaler 1 | 282 |

**Table C-1 Index of Registers (Continued)**

| Address | Abbreviation | Register | Peripheral resource | Page number |
|---|---|---|---|---|
| 00002A$_H$ | IBSR | I$^2$C status register | I$^2$C interface | 336 |
| 00002B$_H$ | IBCR | I$^2$C control register | | 338 |
| 00002C$_H$ | ICCR | I$^2$C clock control register | | 341 |
| 00002D$_H$ | IADR | I$^2$C address register | | 344 |
| 00002E$_H$ | IDAR | I$^2$C data register | | 345 |
| 00002F$_H$ | ISEL | I$^2$C port selection register | | 346 |
| 000030$_H$ | ENIR | DTP/Interrupt enable register | DTP/external interrupt | 319 |
| 000031$_H$ | EIRR | DTP/interrupt cause register | | 316 |
| 000032$_H$ | ELVR | Request level setting register | | 321 |
| 000034$_H$ | ADCS0 | A/D control status register | A/D converter | 365 |
| 000035$_H$ | ADCS1 | | | 363 |
| 000036$_H$ | ADCR0 | A/D data register | | 367 |
| 000037$_H$ | ADCR1 | | | |
| 000039$_H$ | ADMR | A/D conversion channel setting register | | 369 |
| 000040$_H$ | TCCS | Timer counter control status register | 16-bit free-running timer | 250 |
| 000042$_H$ | TCDT | Timer counter data register | | 249 |
| 000043$_H$ | | | | |
| 000044$_H$ | IPC(0) | Input capture data register ch0 | Input capture | 255 |
| 000045$_H$ | | | | |
| 000046$_H$ | IPC(1) | Input capture data register ch1 | | 255 |
| 000047$_H$ | | | | |
| 000048$_H$ | ICSO1 | Input capture control status register | | 256 |
| 00004A$_H$ | OCCP0 | Output compare register ch0 | Output compare | 252 |
| 00004B$_H$ | | | | |
| 00004C$_H$ | OCS0 | Output compare control status register ch0 | | 253 |
| 000050$_H$ | TMCSR(0) | Timer control status register ch0 | 16-bit reload timer ch0 | 229 to 231 |
| 000051$_H$ | | | | |
| 000052$_H$ | TMR(0)/ TMRLR(0) | 16-bit timer register/16-bit reload register ch0 | | 233 to 234 |
| 000053$_H$ | | | | |

**Table C-1 Index of Registers (Continued)**

| Address | Abbreviation | Register | | Peripheral resource | Page number |
|---|---|---|---|---|---|
| $000054_H$ | TMCSR(1) | Timer control status register ch1 | | 16-bit reload timer ch1 | 229 to 231 |
| $000055_H$ | | | | | |
| $000056_H$ | TMR(1)/<br>TMRLR(1) | 16-bit timer register/16-bit reload register ch1 | | | 233 to 234 |
| $000057_H$ | | | | | |
| $000058_H$ | TMCSR(2) | Timer control status register ch2 | | 16-bit reload timer ch2 | 229 to 231 |
| $000059_H$ | | | | | |
| $00005A_H$ | TMR(2)/<br>TMRLR(2) | 16-bit timer register/16-bit reload register ch2 | | | 233 to 234 |
| $00005B_H$ | | | | | |
| $000060_H$ | SMCR(2) | Serial mode control status register ch2 | | Serial I/O ch2 | 184 |
| $000061_H$ | | | | | |
| $000062_H$ | SDR(2) | Serial shift data register ch2 | | | 188 |
| $000064_H$ | SMCR(3) | Serial mode control status register ch3 | | Serial I/O ch3 | 184 |
| $000065_H$ | | | | | |
| $000066_H$ | SDR(3) | Serial shift data register ch3 | | | 188 |
| $000068_H$ | FLC1 | Display control register 1 | | FL control circuit | 385 |
| $000069_H$ | FLC2 | Display control register 2 | | | 387 |
| $00006A_H$ | FLDG | Digit setting register | | | 389 |
| $00006B_H$ | FLDC | Digit count register | | | 391 |
| $00006D_H$ | FLST | Status/settlement register | | | 394 |
| $00006F_H$ | ROMM | ROM mirroring function selection register | | ROM mirroring function selection module | 425 |
| $000070_H$<br>to<br>$000070_H$ | SEGD0 to<br>SEGD7 | Segment dimmer setting register | | FL control circuit | 397 |
| $000078_H$ | FLPD0 | Port register | FIP36 to 43 | | 393 |
| $000079_H$ | FLPD1 | | FIP44 to 51 | | |
| $00007A_H$ | FLPD2 | | FIP52 to 59 | | |
| $00009E_H$ | PACSR | Program address detection control status register | | Address match detection circuit | 417 |
| $00009F_H$ | DIRR | Delayed interrupt cause generation/release register | | Delayed interrupt | 409 |

**Table C-1 Index of Registers (Continued)**

| Address | Abbreviation | Register | Peripheral resource | Page number |
|---|---|---|---|---|
| 0000A0$_H$ | LPMCR | Low power consumption mode control register | Low power consumption control circuit | 86 |
| 0000A1$_H$ | CKSCR | Clock selection register | | 75 |
| 0000A8$_H$ | WDTC | Watchdog timer control register | Watchdog timer | 215 |
| 0000A9$_H$ | TBTC | Timebase timer control register | Timebase timer | 204 |
| 0000AE$_H$ | FMCS | Flash memory control status register | 1M-bit flash memory | 430 |
| 0000AF$_H$ | TMCS | Watch clock output control register | Watch clock division | 406 |
| 0000B0$_H$ to 0000BF$_H$ | ICR00 to ICR15 | Interrupt control register ch0 to ch15 | Interrupt controller | 108 |
| 000100$_H$ to 0010FF$_H$ | RAM area | RAM memory | RAM | |
| 001100$_H$ to 0011FF$_H$ | FL000 to FL255 | Display data RAM | FL control circuit | 396 |
| 001FF0$_H$ | PADR0 | Program address detection register (lower) | Address match detection function | 416 |
| 001FF1$_H$ | | Program address detection register (middle) | | |
| 001FF2$_H$ | | Program address detection register (upper) | | |
| 001FF3$_H$ | PADR1 | Program address detection register (lower) | | |
| 001FF4$_H$ | | Program address detection register (middle) | | |
| 001FF5$_H$ | | Program address detection register (upper) | | |
| FE0000$_H$ (FE8000$_H$) to FFFFFF$_H$ | ROM area | ROM | ROM | |

# APPENDIX D   Index of Pin Functions

**This section provides an index for finding the page containing a block diagram and description from the MB90M405 series package pin number, pin name, circuit type, or peripheral resource name.**

## ■ Index of Pin Functions

**Table D-1  Index of Index of Pin Functions**

| Pin number | Pin name | Circuit type | Peripheral resource name and function name | Functional description | Block diagram |
|---|---|---|---|---|---|
| 82, 83 | X0, X1 | A | Oscillation pin | 10 | 14 |
| 77 | $\overline{\text{RST}}$ | B | Reset input | 10 | 14 |
| 85 to 100<br>1 | FIP0 to FIP16 | C | FL control circuit | 383 | 383 |
| | LED0 to LED16 | | | 383 | 383 |
| 2 to 10<br>12 to 19 | FIP17 to FIP33 | | | 383 | 383 |
| 20 to 22<br>24 to 41<br>43 to 47 | FIP34 to FIP59 | D | FL control circuit | 384 | 384 |
| 52 | P80 | E | General-purpose I/O port 8 bit 0 | 158 | 159 |
| | IC0 | | Input capture ch0 | 158 | 159 |
| | INT0 | | External interrupt input ch0 | 314 | 314 |
| 53 | P81 | | General-purpose I/O port 8 bit 1 | 158 | 159 |
| | IC1 | | Input capture ch0 | 158 | 159 |
| | INT1 | | External interrupt input ch1 | 314 | 314 |
| 54 | P82 | | General-purpose I/O port 8 bit 2 | 158 | 159 |
| | SI0 | | UART data input ch0 | 269 | 270 |
| 55 | P83 | | General-purpose I/O port 8 bit 3 | 158 | 159 |
| | SC0 | | UART clock I/O ch0 | 269 | 270 |
| 56 | P84 | | General-purpose I/O port 8 bit 4 | 158 | 159 |
| | SO0 | | UART data output ch0 | 269 | 270 |
| 57 | P85 | | General-purpose I/O port 8 bit 5 | 158 | 159 |
| | SI1 | | UART data input ch1 | 269 | 270 |

Table D-1 Index of Index of Pin Functions (Continued)

| Pin number | Pin name | Circuit type | Peripheral resource name and function name | Functional description | Block diagram |
|---|---|---|---|---|---|
| 58 | P86 | E | General-purpose I/O port 8 bit 6 | 158 | 159 |
| | SC1 | | UART clock I/O ch1 | 269 | 270 |
| 59 | P87 | | General-purpose I/O port 8 bit 7 | 158 | 159 |
| | SO1 | | UART data output ch1 | 269 | 270 |
| 60 | P90 | G | General-purpose I/O port 9 bit 0 | 163 | 164 |
| | SDA | | I$^2$C | 333 | 333 |
| | SO3 | | Serial I/O data output ch3 | 163 | 164 |
| 61 | P91 | | General-purpose I/O port 9 bit 1 | 163 | 164 |
| | SCL | | I$^2$C | 333 | 333 |
| | SC3 | | Serial I/O clock I/O ch3 | 163 | 164 |
| 64 | PA0 | F | General-purpose I/O port A bit 0 | 168 | 169 |
| | AN0 | | A/D converter analog input ch0 | 360 | 361 |
| | TMCK | | Watch clock output pin | - | 169 |
| 65 to 71 | PA1 to PA7 | | General-purpose I/O port A bit 1 to bit 7 | 168 | 169 |
| | AN1 to AN7 | | A/D converter analog input ch1 to ch7 | 360 | 361 |
| 72 to 74 | PB0 to PB2 | | General-purpose I/O port B bit 0 to bit 2 | 172 | 172 |
| | AN8 to AN10 | | A/D converter analog input ch8 to ch10 | 358 | 359 |
| 75 | PB3 | | General-purpose I/O port B bit 3 | 174 | 175 |
| | AN11 | | A/D converter analog input ch11 | 360 | 361 |
| | SI2 | | Serial I/O data input ch2 | 174 | 175 |
| 76 | PB4 | | General-purpose I/O port B bit 4 | 174 | 175 |
| | AN12 | | A/D converter analog input ch12 | 360 | 361 |
| | SC2 | | Serial I/O clock I/O ch2 | 174 | 175 |
| | TIN0 | | Reload timer event input ch0 | 227 | 227 |
| 78 | PB5 | | General-purpose I/O port B bit 5 | 174 | 175 |
| | AN13 | | A/D converter analog input ch13 | 360 | 361 |
| | SO2 | | Serial I/O data output ch2 | 174 | 175 |
| 79, 80 | PB6 to PB7 | F | General-purpose I/O port B bit 6 to bit 7 | 174 | 175 |
| | AN14 to AN15 | | A/D converter analog input ch14 to ch15 | 360 | 361 |
| | INT2 to INT3 | | External interrupt input ch2 to ch3 | 314 | 314 |

**Table D-1  Index of Index of Pin Functions (Continued)**

| Pin number | Pin name | Circuit type | Peripheral resource name and function name | Functional description | Block diagram |
|---|---|---|---|---|---|
| 62 | $AV_{CC}$ | H | Analog $V_{CC}$ power input pin | 10 | 15 |
| 63 | $AV_{SS}$ | | Analog $V_{SS}$ power input pin | 10 | 15 |
| 48 | $V_{KK}$ | - | High withstand voltage output pull-down power pin | 10 | - |
| 49 | MD0 | - | Power mode specification input pin 0 | 10 | - |
| 50 | MD1 | - | Power mode specification input pin 1 | 10 | - |
| 51 | MD2 | - | Operating mode specification input pin 2 | 10 | - |
| 11, 42 | $V_{SS}$-IO | - | I/O power (GND) input pin | 10 | - |
| 23 | $V_{DD}$-FIP | - | FIP power (3V) input pin | 10 | - |
| 81 | $V_{SS}$-CPU | - | Control circuit power (GND) input pin | 10 | - |
| 84 | $V_{CC}$-CPU | - | Control circuit power (3V) input pin | 10 | - |

# INDEX

**The index follows on the next page.**
**This is listed in alphabetic order.**

# Index

CM44-10116-2E

**FUJITSU SEMICONDUCTOR •CONTROLLER MANUAL**

F$^2$MC-16LX
16-BIT MICROCONTROLLER
MB90M405 Series
HARDWARE MANUAL

October 2001 the second edition

Published **FUJITSU LIMITED** Electronic Devices
Edited Technical Information Dept.