USER'S MANUAL

NEC

µPD17145 SUB-SERIES

4 BIT SINGLE-CHIP MICRO CONTROLLER

µPD17145
µPD17147
µPD17149
µPD17P149

Document No. U10261EJ2V0UM00 (2nd edition)
(Previous No. IEU-1383)
Date Published December 1995 P
Printed in Japan

Cautions on CMOS Devices

<1> Countermeasures against static electricity for all MOSs

Caution When handling MOS devices, take care so that they are not electrostatically charged.

Strong static electricity may cause dielectric breakdown in gates. When transporting or storing MOS devices, use conductive trays, magazine cases, shock absorbers, or metal cases that NEC uses for packaging and shipping. Be sure to ground MOS devices during assembling. Do not allow MOS devices to stand on plastic plates or do not touch pins.

Also handle boards on which MOS devices are mounted in the same way.

<2> CMOS-specific handling of unused input pins

Caution Hold CMOS devices at a fixed input level.

Unlike bipolar or NMOS devices, if a CMOS device is operated with no input, an intermediate-level input may be caused by noise. This allows current to flow in the CMOS device, resulting in a malfunction. Use a pull-up or pull-down resistor to hold a fixed input level. Since unused pins may function as output pins at unexpected times, each unused pin should be separately connected to the VDD or GND pin through a resistor.

If handling of unused pins is documented, follow the instructions in the document.

<3> Statuses of all MOS devices at initialization

Caution The initial status of a MOS device is unpredictable when power is turned on.

Since characteristics of a MOS device are determined by the amount of ions implanted in molecules, the initial status cannot be determined in the manufacture process. NEC has no responsibility for the output statuses of pins, input and output settings, and the contents of registers at power on. However, NEC assures operation after reset and items for mode setting if they are defined.

When you turn on a device having a reset function, be sure to reset the device first.

The export of this product from Japan is regulated by the Japanese government. To export this product may be prohibited without governmental license, the need for which must be judged by the customer. The export or re-export of this product from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

The information in this document is subject to change without notice.

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Corporation. NEC Corporation assumes no responsibility for any errors which may appear in this document.

NEC Corporation does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from use of a device described herein or any other liability arising from use of such device. No license, either express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Corporation or others.

While NEC Corporation has been making continuous effort to enhance the reliability of its semiconductor devices, the possibility of defects cannot be eliminated entirely. To minimize risks of damage or injury to persons or property arising from a defect in an NEC semiconductor device, customer must incorporate sufficient safety measures in its design, such as redundancy, fire-containment, and anti-failure features.

NEC devices are classified into the following three quality grades:

“Standard“, “Special“, and “Specific“. The Specific quality grade applies only to devices developed based on a customer designated “quality assurance program“ for a specific application. The recommended applications of a device depend on its quality grade, as indicated below. Customers must check the quality grade of each device before using it in a particular application.

Standard: Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots

Special: Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support)

Specific: Aircrafts, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems or medical equipment for life support, etc.

The quality grade of NEC devices in “Standard“ unless otherwise specified in NEC's Data Sheets or Data Books.

If customers intend to use NEC devices for applications other than those specified for Standard quality grade, they should contact NEC Sales Representative in advance.

Anti-radioactive design is not implemented in this product.

Major Changes

Page	Description
Throughout	The µPD17P149 has been developed.
p.110	**Figure 13-1** has been modified.
p.116	**Section 13.1.6** has been added.
p.117	**Section 13.1.7** has been added.
p.139	**Figure 13-21** has been changed.
p.163	**Caution 2** for **Table 15-1** has been modified.
p.170	**Section 15.3.3** has been changed.
p.177	**Section 17.3** has been added.
p.192	**Note** has been addded to **Section 20.3**.
p.263	**Appendix B** has been modified.
p.271	**Appendix F** has been added.

The mark ⋆ shows major revised points.

PREFACE

Users:

This manual is for users who intend to learn the capabilities of the μPD17145 sub-series for application program development.

Purpose:

This manual describes the functions of the μPD17145 sub-series. The purpose of this manual is to help users generate a program.

Guidance:

Before using this manual, the user should have a general knowledge of the electronics, logical circuit, and microcomputer fields.

- **To understand the general functions of the μPD17145 sub-series:**
  Read the entire manual in the order of the table of contents.
- **To check instructions if you know their mnemonics:** See **Appendix E**.
- **To check instructions when you don't know their mnemonics, but you know their functions roughly:**
  Check the mnemonics of the instructions in **Section 20.3**, then check their functions in **Section 20.5**.
- **To check the electrical characteristics of the μPD17145 sub-series:**
  Refer to the separate data sheet.

Notation:

Data weight:	Higher digits on the left side	
	Lower digits on the right side	
Active low:	$\overline{\text{xxx}}$ (Pins and signal names are overscored.)	
Memory map address:	Low-order address on the upper side	
	High-order address on the lower side	
Note:	Explanation of the indicated part of the text	
Caution:	Information requesting the user's special attention	
Remarks:	Supplementary information	
Numeric value:	Binary:	xxxx or xxxxB
	Decimal:	xxxx or xxxxD
	Hexadecimal:	xxxxH

In this manual, the product described is the μPD17149, which is one of the μPD17145 sub-series, unless otherwise specified.

Related documents:

The following documents are provided for reference. The numbers listed in the table are document numbers. Some related documents may be preliminary versions. Note that, however, what documents are preliminary is not indicated in this document.

Product Document	µPD17145	µPD17147	µPD17149	µPD17P149
Data Sheet	IC-8793 [IC-3283]			IC-8978 [IC-3505]
User's Manual	The manual you are reading			
IE-17K (Ver. 1.6) User's Manual	EEU-929 [EEU-1467]			
IE-17K-ET (Ver. 1.6) User's Manual	EEU-931 [EEU-1466]			
SIMPLEHOST™ User's Manual	EEU-723 [EEU-1336] (Tutorial) EEU-724 [EEU-1337] (Reference)			
AS17K (Ver. 1.11) User's Manual	EEU-603 [EEU-1287]			
Device File User's Manual	EEU-949 [EEU-1486]			
SE Board Preliminary User's Manual	EEU-945 [EEU-1475]			

Remark The numbers in brackets are English document numbers.

CONTENTS

LIST OF FIGURES (1/3)

LIST OF FIGURES (2/3)

LIST OF FIGURES (3/3)

LIST OF TABLES (1/2)

LIST OF TABLES (2/2)

[MEMO]

CHAPTER 1 OVERVIEW

The µPD17149 is a 4-bit single-chip microcontroller containing an 8-bit A/D converter (four channels), three timers, and a serial interface. The µPD17149 can incorporate a POC circuit by using a mask option.

Since the µPD17P149 is a one-time PROM product, it is suited for program evaluation in developing a system and for low-volume production. Its features are as follows.

• 17K architecture	General registers; fixed, 16-bit instruction length
• Instruction execution time	2 µs (When operating at f_X = 8 MHz with ceramic oscillation)
• Program memory (ROM)	µPD17145 : 2K bytes (1024 x 16 bits) µPD17147 : 4K bytes (2048 x 16 bits) µPD17149 : 8K bytes (4096 x 16 bits) µPD17P149: 8K bytes (4096 x 16 bits, one-time PROM)
• Data memory (RAM)	110 x 4 bits
• A/D converter	4 channels (8-bit resolution, successive approximation system)
• Timer function	3 channels (8-bit timer counter x 2 channels and basic interval timer[Note])
• Serial interface	1 channel (three-wire synchronous mode)
• POC circuit (mask option)	
• Supply voltage	V_{DD} = 4.5 to 5.5 V (at f_X = 400 kHz to 8 MHz) V_{DD} = 2.7 to 5.5 V (at f_X = 400 kHz to 2 MHz)

Note An internal reset signal can be generated by using a basic interval timer. (Watchdog timer function)

The µPD17149 is suited for control including analog voltage measurement, and for submicrocontrollers. It can be used in the following units.

- Electric appliances
- Battery chargers
- Cameras
- Electronic measuring instruments

1.1 FUNCTIONS

Item / Product	µPD17145	µPD17147	µPD17149	µPD17P149
ROM	Mask ROM			One-time PROM
	2K bytes (1024 x 16 bits)	4K bytes (2048 x 16 bits)	8K bytes (4096 x 16 bits)	
RAM	110 x 4 bits			
Stack	5 address stacks, 3 interrupt stacks			
Number of I/O ports	23 { • 20 I/O ports • 2 general input ports • 1 sensor input port (INT pin[Note])			
A/D converter input	4 channels (shared with ports) with an absolute accuracy of ±1.5 LSB or less			
Timer	3 channels { • 2 channels for 8-bit timer counter (They can be used together as one 16-bit timer.) • 1 channel for 7-bit basic interval timer (can be used as a watchdog timer)			
Serial interface	1 channel (3-wire type)			
Interrupt	• Up to 3 levels of multiple hardware interrupt • 1 external interrupt (INT) { Detection of the rising edge Detection of the falling edge Detection of both edges } Selectable • 4 internal interrupts { • Timer 0 (TM0) • Timer 1 (TM1) • Basic interval timer (BTM) • Serial interface (SIO)			
Execution time of an instruction	2 µs (when operating at f_X = 8 MHz with ceramic oscillation)			
Standby function	HALT/STOP			
* POC circuit	Mask option (Can be used in an application circuit where V_{DD} is 5 V ±10 % and the clock frequency ranges from 400 kHz to 4 MHz)			None
Supply voltage	V_{DD} = 2.7 V to 5.5 V (at f_X = 400 kHz to 2 MHz) V_{DD} = 4.5 V to 5.5 V (at f_X = 400 kHz to 8 MHz)			
Package	28-pin plastic shrink DIP (400 mil) 28-pin plastic SOP (375 mil)			

* **Note** The INT pin can be used as an input pin (sensor input) when the external interrupt function is not used. The status of the pin is read with the INT flag of the control register, not with the port register.

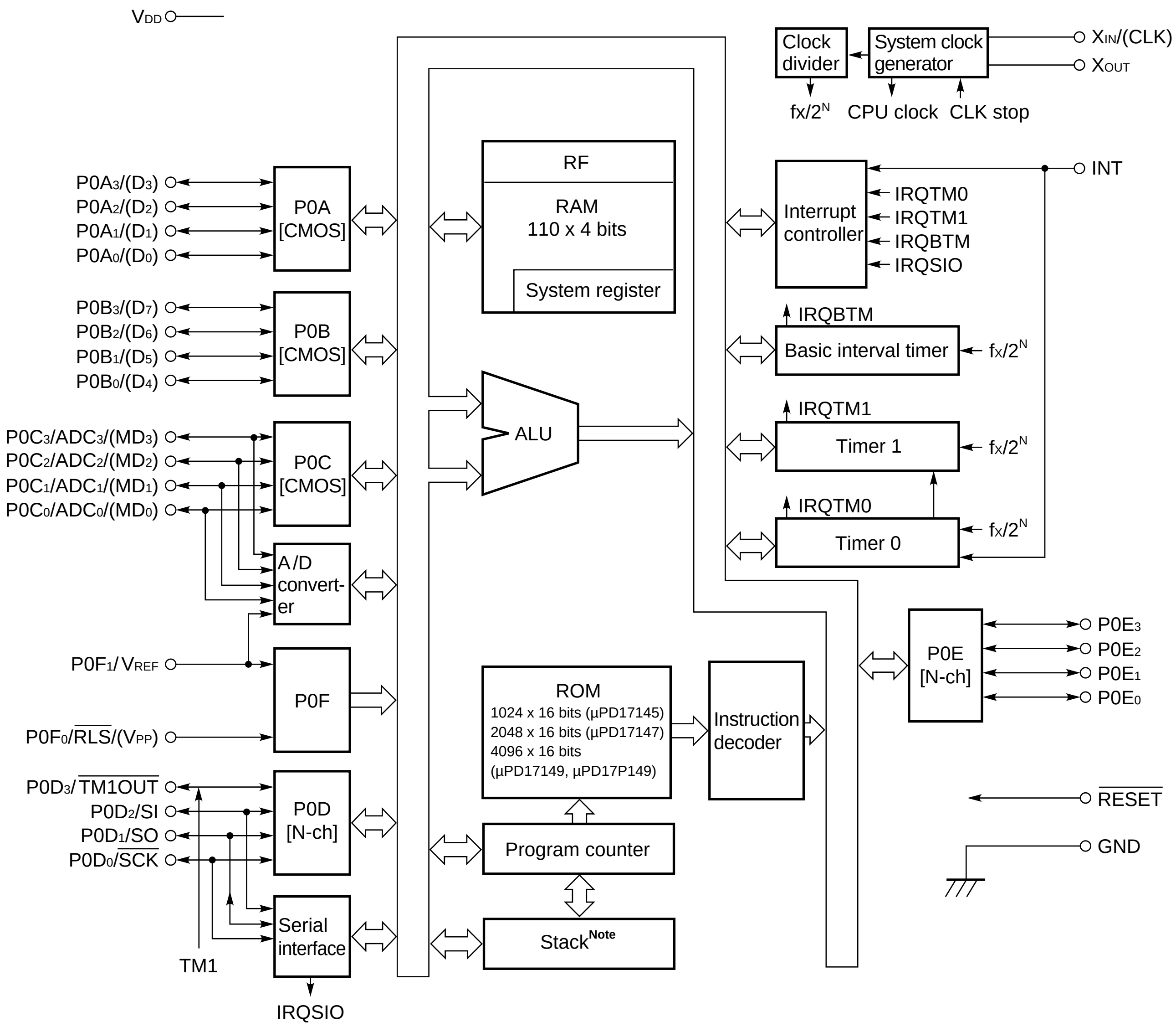
* **Caution Although a PROM product is highly compatible with a mask ROM product in respect of functions, they differ in internal ROM circuits and part of electrical characteristics. Before changing the PROM product to the mask ROM product in an application system, evaluate the system carefully using the mask ROM product.**

1.2 ORDERING INFORMATION

Part number	Package		Built-in ROM
µPD17145CT-xxx	28-pin plastic shrink DIP	(400 mil)	Mask ROM
µPD17145GT-xxx	28-pin plastic SOP	(375 mil)	Mask ROM
µPD17147CT-xxx	28-pin plastic shrink DIP	(400 mil)	Mask ROM
µPD17147GT-xxx	28-pin plastic SOP	(375 mil)	Mask ROM
µPD17149CT-xxx	28-pin plastic shrink DIP	(400 mil)	Mask ROM
µPD17149GT-xxx	28-pin plastic SOP	(375 mil)	Mask ROM
µPD17P149CT	28-pin plastic shrink DIP	(400 mil)	One-time PROM
µPD17P149GT	28-pin plastic SOP	(375 mil)	One-time PROM

Remark xxx is a ROM code number.

1.3 BLOCK DIAGRAM



Note The capacity of the stack depends on the product.

Remarks 1. The values in parentheses are valid only when the µPD17P149 is in program memory write/verify mode.

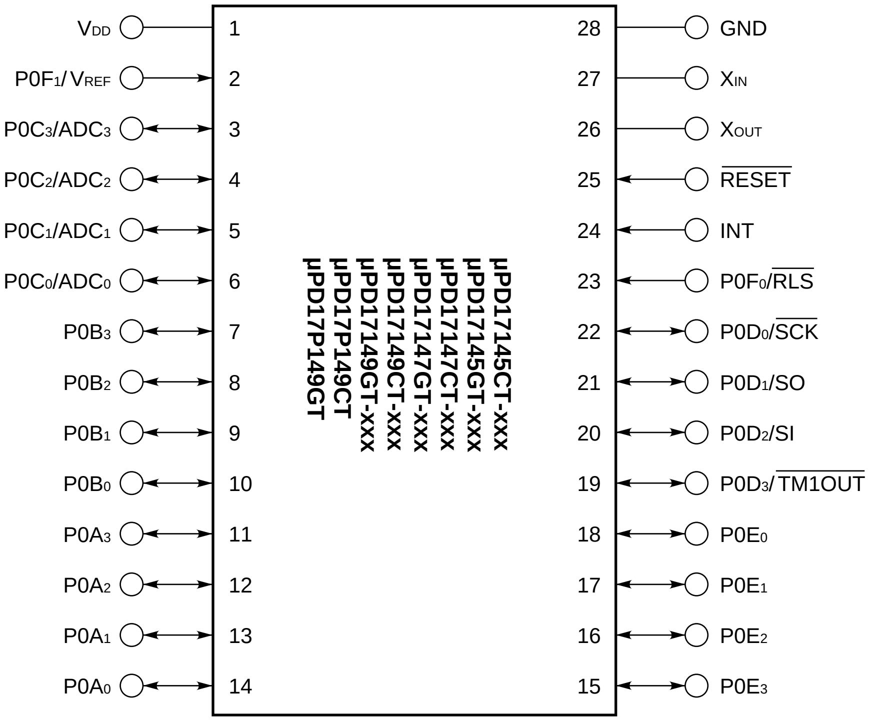
2. The terms CMOS and N-ch in brackets indicate the output form of the port.

CMOS: CMOS push-pull output
N-ch : N-channel open-drain output

1.4 PIN CONFIGURATION (TOP VIEW)

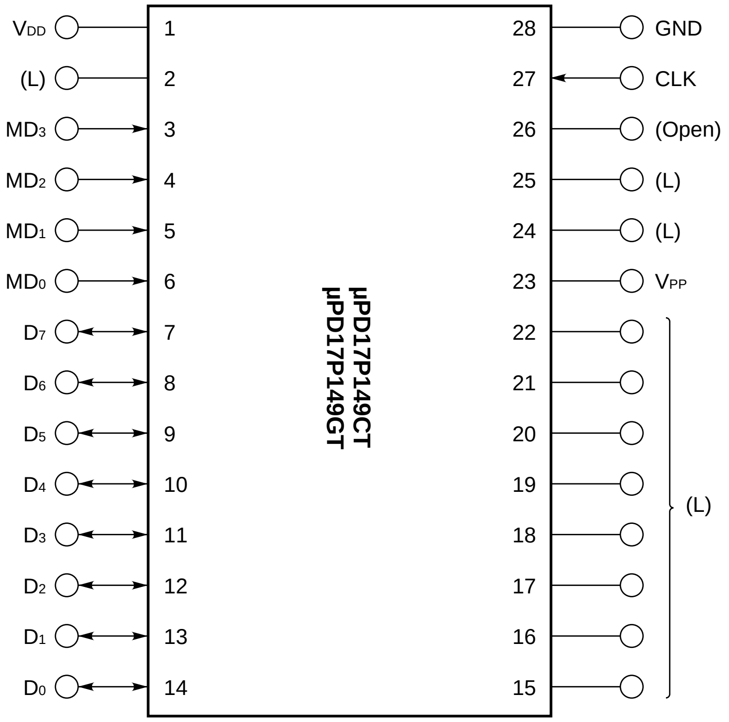
(1) Normal operation mode

28-pin plastic shrink DIP (400 mil)
28-pin plastic SOP (375 mil)



ADC_0 - ADC_3	: Analog input	$\overline{\text{RESET}}$	: Reset input
GND	: Ground	$\overline{\text{RLS}}$	: Standby release signal input
INT	: External interrupt input	$\overline{\text{SCK}}$	: Serial clock input/output
$P0A_0$ - $P0A_3$	: Port 0A	SI	: Serial data input
$P0B_0$ - $P0B_3$	: Port 0B	SO	: Serial data output
$P0C_0$ - $P0C_3$	: Port 0C	$\overline{\text{TM1OUT}}$	: Timer 1 output
$P0D_0$ - $P0D_3$	: Port 0D	V_{DD}	: Power supply
$P0E_0$ - $P0E_3$	: Port 0E	V_{REF}	: Reference voltage for the A/D converter
$P0F_0$ and $P0F_1$	: Port 0F	X_{IN}, X_{OUT}	: System clock oscillation

(2) Program memory write/verify mode



CLK	: Input clock for address update	MD_0 - MD_3	: Operating mode selection input
D_0 - D_7	: Data I/O	V_{DD}	: Power supply
GND	: Ground	V_{PP}	: Programming power supply

Caution Symbols in parentheses denote processing for pins not used in the program memory write/ verify mode.

L : Connect these pins separately to the GND pin through pull-down resistors.

Open: Nothing should be connected on these pins.

CHAPTER 2 PIN FUNCTIONS

2.1 EXPLANATION OF PIN FUNCTIONS

2.1.1 Normal Operating Mode

Pin No.	Symbol	Function	Output	Upon reset
1	V_{DD}	Power supply	–	–
2	$P0F_1/V_{REF}$	Port 0F. The reference voltage is supplied to the A/D converter through this pin. ● Pull-up resistor incorporation specifiable with the mask option**Note** ● $P0F_1$ • Bit 1 of 2-bit input port P0F ● V_{REF} ● Reference voltage input for the A/D converter	Input	Input ($P0F_1$)
3 - 6	$P0C_3/ADC_3$ - $P0C_0/ADC_0$	Port 0C. Analog voltage is supplied to the A/D converter through these pins. ● $P0C_3$ - $P0C_0$ • 4-bit input/output port • Input/output setting allowed in units of 1 bit ● ADC_3 - ADC_0 • Analog input for the A/D converter	CMOS push-pull	Input (P0C)
7 8 9 10	$P0B_3$ $P0B_2$ $P0B_1$ $P0B_0$	Port 0B • 4-bit input/output port • Input/output setting allowed in units of 4 bits • Pull-up resistor incorporation specifiable by software in units of 4 bits.	CMOS push-pull	Input
11 12 13 14	$P0A_3$ $P0A_2$ $P0A_1$ $P0A_0$	Port 0A • 4-bit input/output port • Input/output setting allowed in units of 4 bits • Pull-up resistor incorporation specifiable by software in units of 4 bits.	CMOS push-pull	Input
15 16 17 18	$P0E_3$ $P0E_2$ $P0E_1$ $P0E_0$	Port 0E • 4-bit input/output port • Input/output setting allowed in units of 4 bits • Pull-up resistor incorporation specifiable by software in units of 4 bits.	N-ch open drain	Input

Note For the µPD17P149, a pull-up resistor with the mask option is not incorporated. *

Pin No.	Symbol	Function	Output	Upon reset
19	$P0D_3/\overline{TM1OUT}$	Pin for port 0D, timer 1 output, serial data input, serial data output, and serial clock input/output • Pull-up resistor incorporation specifiable by software in units of 1 bit • $P0D_3$ - $P0D_0$ • 4-bit input/output port • Input/output setting allowed in units of 1 bit • $\overline{TM1OUT}$ • Timer 1 output	N-ch open drain	Input
20	$P0D_2/SI$	• SI • Serial data input		
21	$P0D_1/SO$	• SO • Serial data output		
22	$P0D_0/\overline{SCK}$	• $\overline{SCK}$ • Serial clock input/output		
23	$P0F_0/\overline{RLS}$	Pin for port 0F and input for standby mode release signal • Pull-up resistor incorporation specifiable with the mask option**Note** • $P0F_0$ • Bit 0 of 2-bit input port P0F • $\overline{RLS}$ • Input for standby mode release signal	Input	Input
24	INT	Input for an external interrupt request signal and standby mode release signal. • Pull-up resistor incorporation specifiable with the mask option**Note**	Input	Input
25	$\overline{RESET}$	System reset input pin • Pull up resistor incorporation specifiable with the mask option**Note**	Input	Input
26 27	X_{OUT} X_{IN}	For system clock oscillation The ceramic resonator is connected between X_{IN} and X_{OUT}.	–	–
28	GND	Ground	–	–

* **Note** For the µPD17P149, a pull-up resistor is not incorporated with the mask option.

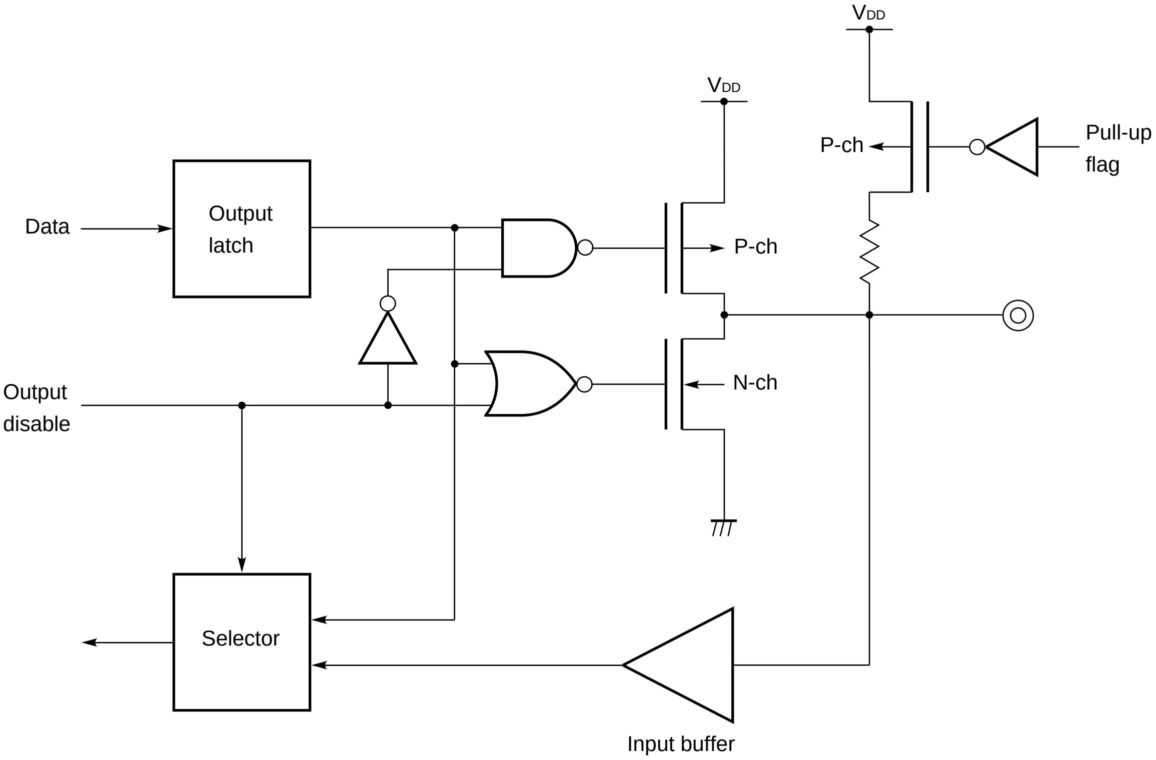
2.1.2 Program Memory Write/Verify Mode (Only for the µPD17P149)

Pin No.	Pin name	Function	Input/output
1	V_{DD}	Positive power supply pin. +6 V is applied to this pin when writing to program memory or verifying its contents.	–
3 to 6	MD_3 to MD_0	Input pins that select an operation mode when writing to program memory or verifying its contents	Input
7 to 14	D_7 to D_0	Input/output pins for 8-bit data used when writing to program memory or verifying its contents	Input/output
23	V_{PP}	Voltage (+12.5 V) is applied to this pin when writing to program memory or verifying its contents.	–
27	CLK	Input pin for address update clocks used when writing to program memory or verifying its contents	Input
28	GND	Ground	–

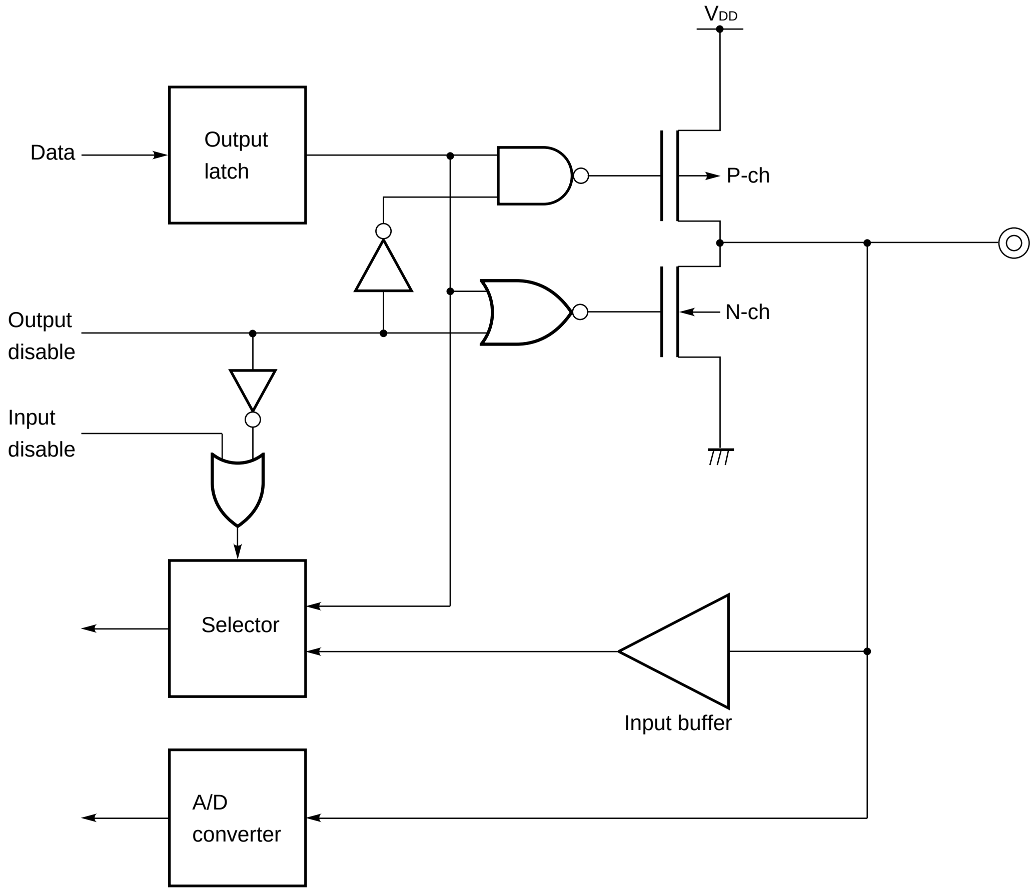
2.2 EQUIVALENT INPUT/OUTPUT CIRCUITS

Below are simplified diagrams of the input/output circuits for each pin.

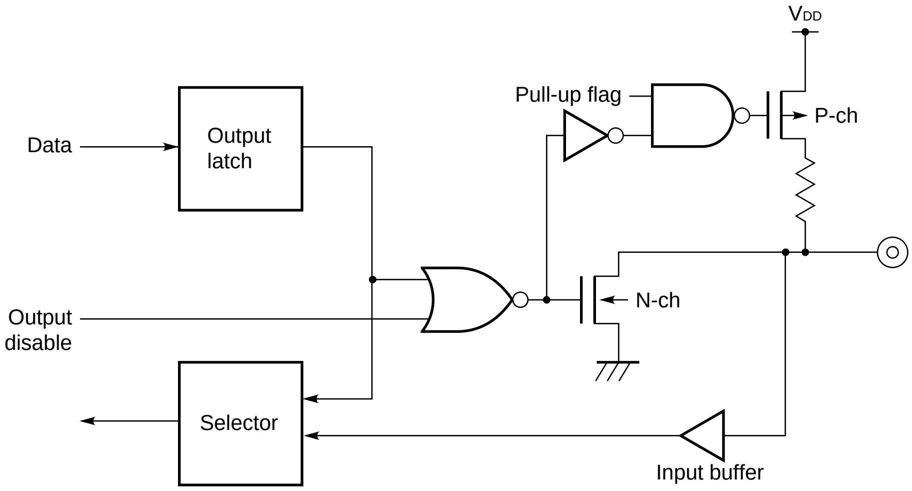
(1) P0A$_0$ - P0A$_3$, P0B$_0$ - P0B$_3$



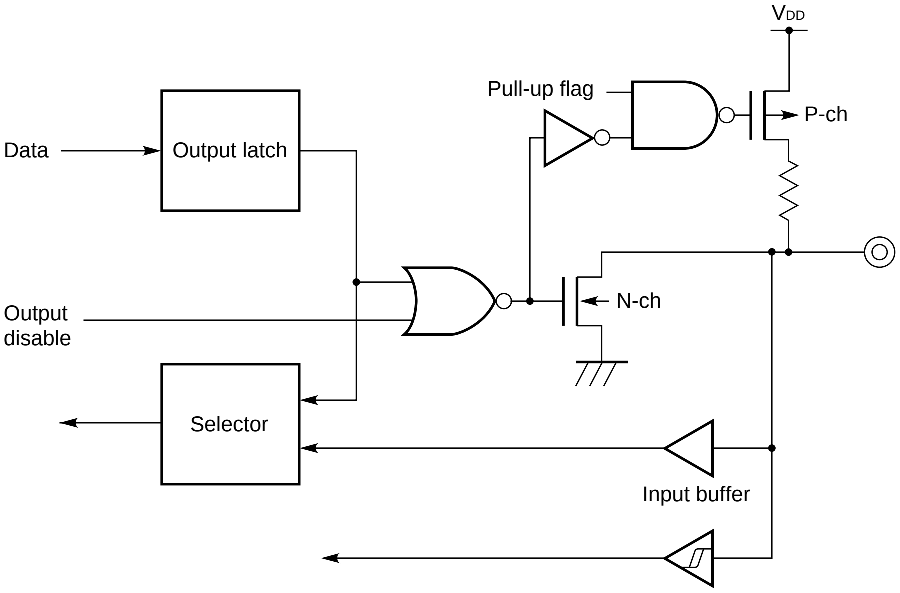
(2) P0C$_0$/ADC$_0$ - P0C$_3$/ADC$_3$



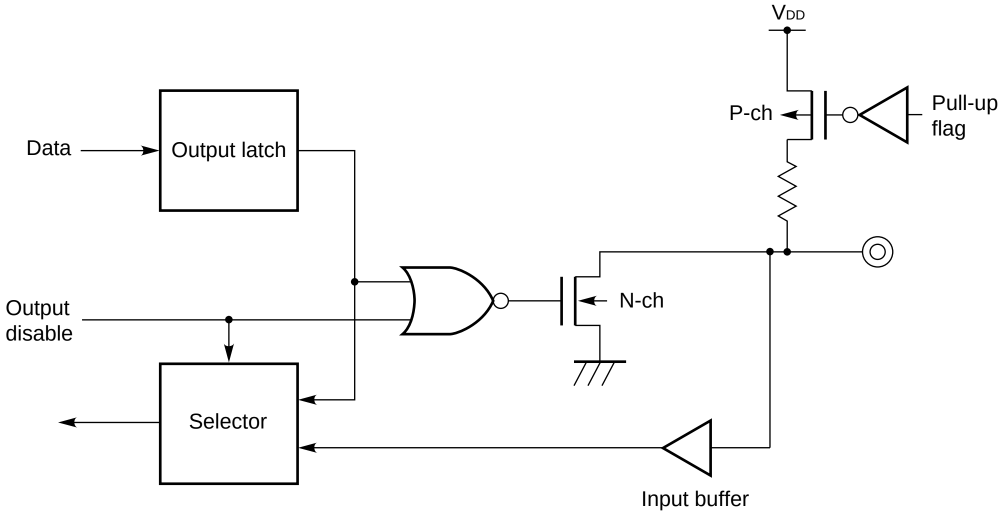
(3) P0D$_3$/$\overline{\text{TM1OUT}}$, P0D$_1$/SO



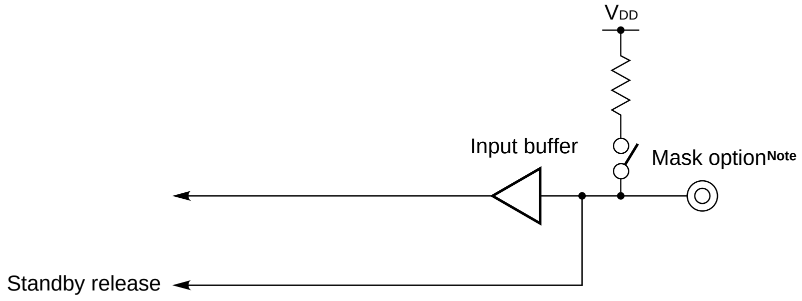
(4) P0D$_2$/SI, P0D$_0$/$\overline{\text{SCK}}$



(5) P0E$_0$ - P0E$_3$

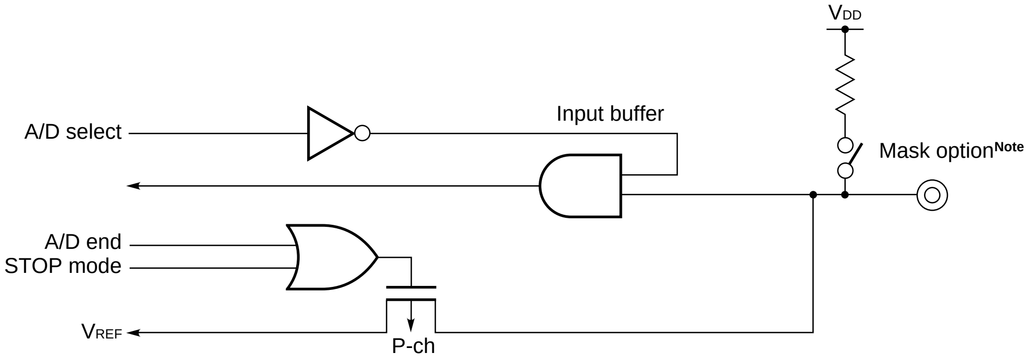


(6) P0F$_0$/$\overline{\text{RLS}}$

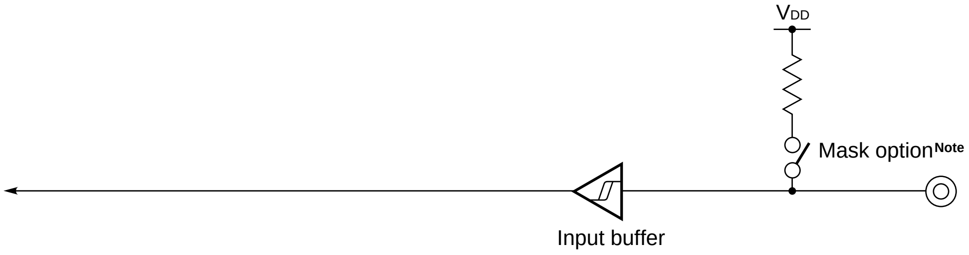


(7) P0F$_1$/V$_{\text{REF}}$

*



(8) $\overline{\text{RESET}}$, INT



Note For the µPD17P149, a pull-up resistor is not incorporated with the mask option.

*

2.3 CONNECTION OF UNUSED PINS

Connect unused pins as follows:

*

Table 2-1. Connection of Unused Pins

Pin			Conditions and handling	
			Internal	External
Port	Input mode	P0A, P0B, P0D, P0E	Pull-up resistors that can be specified by software are incorporated.	Leave open.
		P0C	–	Connect to V_{DD} through pull-up resistors. Or, connect to ground through pull-down resistors.**Note 1**
		$P0F_1$	Pull-up resistors that can be specified with the mask option are not incorporated.	Connect directly to V_{DD} or ground.
			Pull-up resistors that can be specified with the mask option are incorporated.	Leave open.
		$P0F_0$**Note 2**	Pull-up resistors that can be specified with the mask option are not incorporated.	Connect directly to ground.
	Output mode	P0A, P0B, P0C (CMOS ports)	–	Leave open.
		P0D (N-ch open-drain port)	Outputs low level.	
		P0E (N-ch open-drain port)	Outputs low level without pull-up resistors that can be specified by software.	
			Outputs high level with pull-up resistors that can be specified by software.	
External interrupt (INT)			Pull-up resistors that can be specified with the mask option are not incorporated.	Connect directly to V_{DD} or ground.
			Pull-up resistors that can be specified with the mask option are incorporated.	Leave open.
$\overline{\text{RESET}}$**Note 3** (When use only internal POC circuit)			Pull-up resistors that can be specified with the mask option are not incorporated.	Connect directly to V_{DD}.
			Pull-up resistors that can be specified with the mask option are incorporated.	

Notes 1. When a pin is pulled up to V_{DD} or pulled down to ground outside the chip, take the driving capacity and maximum current consumption of a port into consideration. When using high-resistance pull-up or pull-down resistors, apply appropriate countermeasures to ensure that noise is not attracted by the resistors. Although the optimum pull-up or pull-down resistor varies with the application circuit, in general, a resistor of 10 to 100 kilohms is suitable.

2. Since the $\mathrm{P0F_0}/\overline{\mathrm{RLS}}$ pin is also used for setting the test mode, connect it directly to ground without incorporating a pull-up resistor that can be specified with the mask option, when the pin is not used.

3. When designing an application circuit which requires high reliability, be sure to design a circuit to which an external $\overline{\mathrm{RESET}}$ signal can be input. Since the $\overline{\mathrm{RESET}}$ pin is also used for setting the test mode, connect it to V_{DD} directly when not used.

Caution **To fix the I/O mode, pull-up resistors that can be specified with the software, and output level of a pin, it is recommended that they should be specified repeatedly within a loop in a program.**

Remark For the µPD17P149, a pull-up resistor and POC circuit are not incorporated with the mask option.

2.4 NOTES ON USE OF THE $\overline{\text{RESET}}$ AND P0F$_0$/$\overline{\text{RLS}}$ PINS (ONLY FOR NORMAL OPERATION MODE)

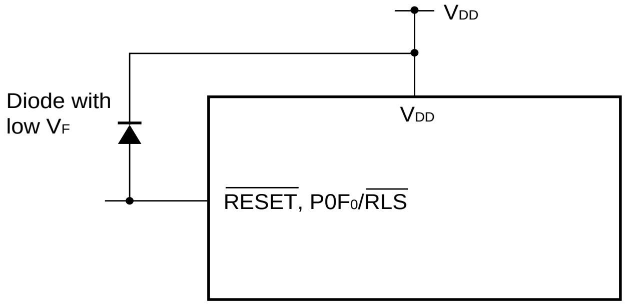
The $\overline{\text{RESET}}$ and P0F$_0$/$\overline{\text{RLS}}$ pins have the test mode selecting function for testing the internal operation of the µPD17149 (IC test), besides the functions shown in **Section 2.1**.

Applying a voltage exceeding V_{DD} to the $\overline{\text{RESET}}$ and/or P0F$_0$ pin causes the µPD17149 to enter the test mode. When noise exceeding V_{DD} comes in during normal operation, the device is switched to the test mode.

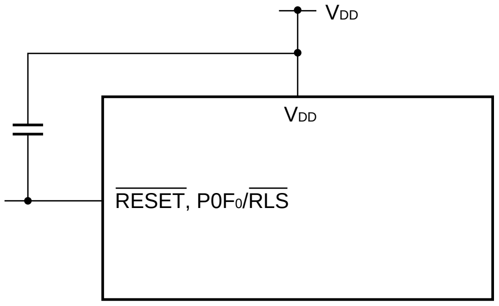
For example, if the wiring from the $\overline{\text{RESET}}$ or P0F$_0$/$\overline{\text{RLS}}$ pin is too long, noise may be induced on the wiring, causing this mode switching.

When installing the wiring, lay the wiring in such a way that noise is suppressed as much as possible. If noise yet arises, use an external part to suppress it as shown below.

- Connect a diode with low V_F between the pin and V_{DD}.



- Connect a capacitor between the pin and V_{DD}.



CHAPTER 3 PROGRAM MEMORY (ROM)

Table 3-1 lists the program memory configuration for the µPD17145, µPD17147, µPD17149, and µPD17P149.

Table 3-1. Program Memory Configuration

Product	Program memory capacity	Address range
µPD17145	2K bytes (1024 x 16 bits)	0000H - 03FFH
µPD17147	4K bytes (2048 x 16 bits)	0000H - 07FFH
µPD17149	8K bytes (4096 x 16 bits)	0000H - 0FFFH
µPD17P149		

Program memory stores the program and the constant data table. The reset address and interrupt vector address are stored at the top of the program memory. *

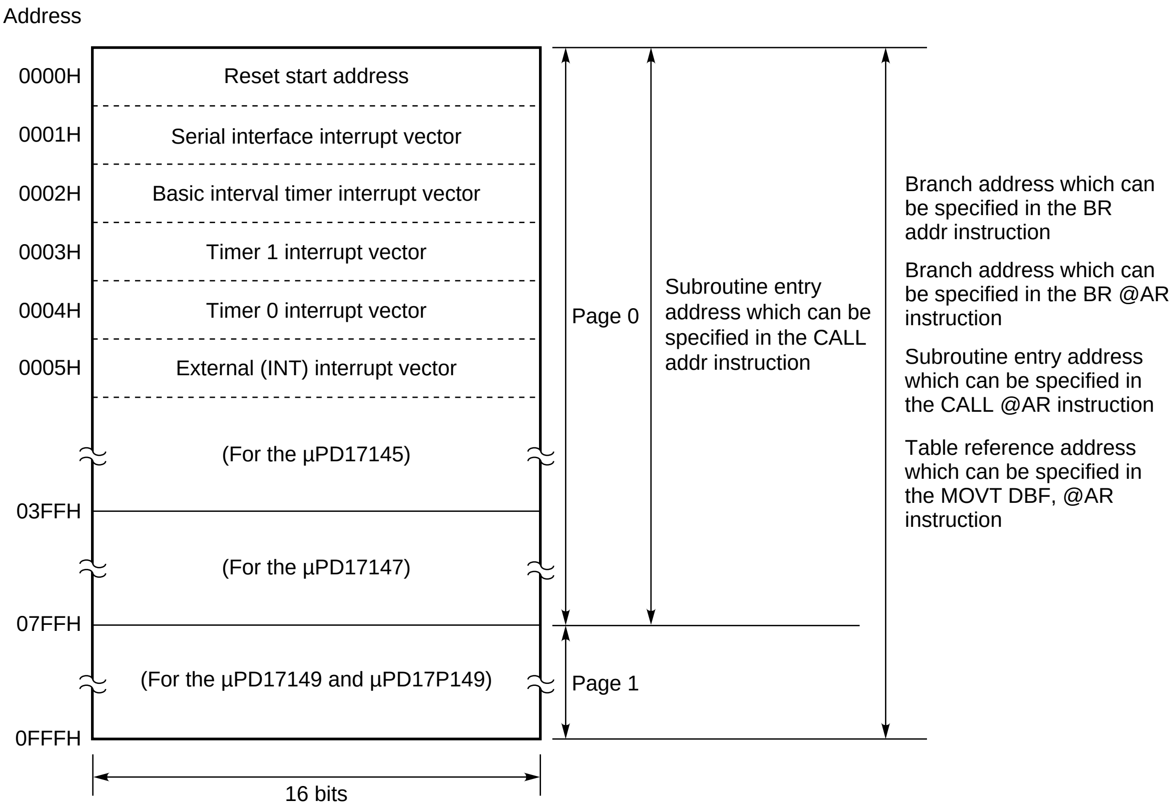
The program memory address is specified by the program counter.

3.1 PROGRAM MEMORY CONFIGURATION

Figure 3-1 shows the program memory map. A step consists of 16 bits of program memory. A 2K-step area is called a page.

Direct subroutine calls can specify address 0000H to 07FFH (page 0) in program memory. Branch instructions, indirect subroutine calls, and table references can specify any address in each entire program memory.

Figure 3-1. Program Memory Map


Address
0000H
Reset start address
0001H
Serial interface interrupt vector
0002H
Basic interval timer interrupt vector
0003H
Timer 1 interrupt vector
0004H
Timer 0 interrupt vector
0005H
External (INT) interrupt vector
(For the µPD17145)
03FFH
(For the µPD17147)
07FFH
(For the µPD17149 and µPD17P149)
0FFFH
16 bits
Page 0
Page 1
Subroutine entry address which can be specified in the CALL addr instruction
Branch address which can be specified in the BR addr instruction
Branch address which can be specified in the BR @AR instruction
Subroutine entry address which can be specified in the CALL @AR instruction
Table reference address which can be specified in the MOVT DBF, @AR instruction

3.2 PROGRAM MEMORY USAGE

Program memory has the following two main functions:

(1) Storage of the program
(2) Storage of constant data

The program is made up of the instructions which operate the CPU (Central Processing Unit). The CPU executes sequential processing according to the instructions stored in the program. In other words, the CPU reads each instruction in the order stored by the program in program memory and executes it.

Since all instructions are 16-bit long words, each instruction is stored in a single location in program memory.

Constant data, such as display output patterns, are set beforehand. The MOVT instruction is used for reading constant data in program memory. This instruction is used to transfer data from program memory to the data buffer (DBF) in data memory. Reading the constant data in program memory is called table reference.

Program memory is read-only (ROM: Read Only Memory) and therefore cannot be changed by any instructions.

3.2.1 Flow of the Program

The program is usually stored in program memory starting from memory location 0000H and executed sequentially one memory location at a time. However, if for some reason a different kind of program is to be executed, it will be necessary to change the flow of the program. In this case, the branch instruction (BR instruction) is used.

If the same section of program code is going to appear in a number of places, reproducing the code each time it needs to be used will decrease the efficiency of the program. In this case, this section of program code should be stored in only one place in memory. Then, the same section of program code is called by using the CALL instruction. Such a piece of code is called a subroutine. As opposed to a subroutine, code used during normal operation is called the main routine.

For cases unrelated to the flow of the program (in which a section of code is to be executed when a certain condition arises), the interrupt function is used. Whenever a condition arises that is unrelated to the flow of the program, the interrupt function can be used to branch the program to a prechosen memory location (called a vector address).

Items (1) to (4) explain branching of the program using the interrupt function and CPU instructions.

(1) Vector address

Table 3-2 shows the address to which the program is branched (vector address) when a reset or interrupt occurs.

Table 3-2. Vector Address

Vector address	Cause of the interrupt
0000H	Reset
0001H	Serial interface interrupt
0002H	Basic interval timer interrupt
0003H	Timer 1 interrupt
0004H	Timer 0 interrupt
0005H	External interrupt (INT pin)

(2) Direct branch

When executing a direct branch (BR addr) instruction, the program branches to the address specified by the value of the operand (addr).

For the µPD17145, the 10 low-order bits of the operand are used to specify a branch destination address in program memory. (Note, however, that addresses exceeding 03FFH cannot be specified. If an address is specified outside of this range, an error will occur in the assembler.)

For the µPD17147, all bits (11 bits) of the operand are used to specify a branch destination address in program memory. (Note, however, that addresses exceeding 07FFH cannot be specified. If an address is specified outside of this range, an error will occur in the assembler.)

For the µPD17149, the least significant bit of the operation code (hereinafter opcode) and all bits (11 bits) of the operand (12 bits in total) are used to specify a branch destination address in program memory. (Note, however, that addresses exceeding 0FFFH cannot be specified. If an address is specified outside of this range, an error will occur in the assembler.)

A BR addr instruction can thus be used to branch to any address in program memory.

- **Precautions in debugging the µPD17149**

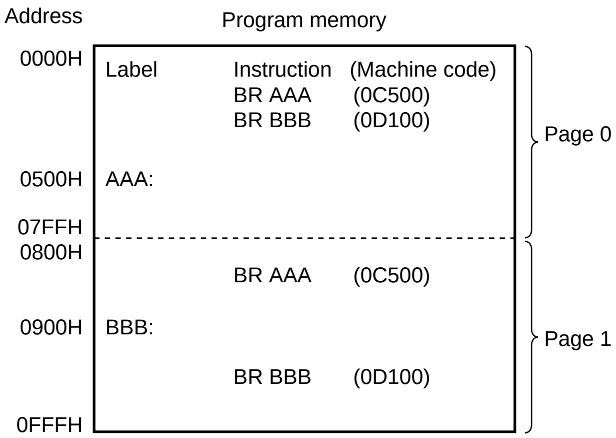
  The machine code for a BR addr instruction has only 11 bits for specifying the program memory address. Different opcodes are assigned for BR addr instructions depending on the page which contains the destination address. When the destination address of a direct branch is in page 1, the machine code does not correspond to the actual address in the program.

  * Normally, the assembler automatically controls which opcode should be used. Therefore, take care when patching is performed using machine code directly rather than the assembler.

Table 3-3. Correspondence between the Destination Address and the Machine Code of a BR addr Instruction

Destination address	Machine code of BR addr
0000H - 07FFH (page 0)	0C000H - 0C7FFH
0800H - 0FFFH (page 1)	0D000H - 0D7FFH

Figure 3-2. Example of a Machine Code of a BR addr instruction



(3) Indirect branch

When executing an indirect branch instruction (BR @AR), the program branches to the address specified by the contents of the address register (AR). A BR @AR instruction can be used to branch to any address in program memory.

Also see **Section 8.2**.

(4) Subroutine

The subroutine call instructions (CALL) are used for branching to a subroutine.

Two types of CALL instructions are supported: direct subroutine call instruction (CALL addr) which branches to the address specified by the addr operand; and indirect subroutine call instruction (CALL @AR) which branches to the address specified in the address register.

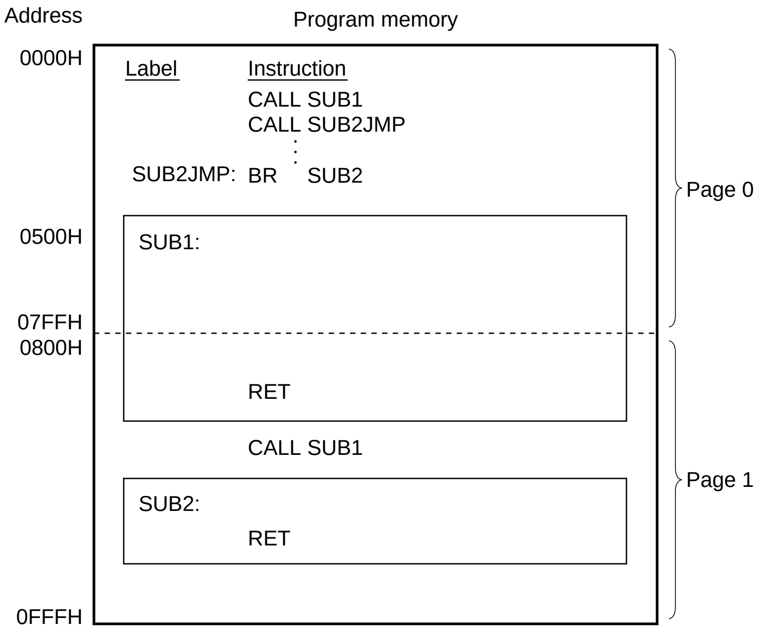
The RET or RETSK instruction is used for return from a subroutine. Executing the RET or RETSK instruction returns to the program memory address next to the CALL instruction.

The RETSK instruction treats the first instruction after return as a NOP instruction.

<1> Direct subroutine call

When using a direct subroutine call (CALL addr), all bits (11 bits) of the operand are used to specify a program memory address of the called subroutine. When a CALL addr instruction is used, the starting address of the subroutine must be in page 0 (addresses 0000H to 07FFH). Otherwise, the subroutine cannot be called directly. To branch to a subroutine whose starting address is located at an area other than page 0, place a branch instruction (BR) in page 0, as shown in Figure 3-3, to call the subroutine (SUB2) in the following example.

Figure 3-3. CALL addr Instruction



<2> Indirect subroutine call

When using an indirect subroutine call (CALL @AR), the value in the address register (AR) should be an address of the called subroutine. This instruction can be used to branch to any address in program memory. See **Section 8.2**.

3.3 TABLE REFERENCE

Table reference is used to reference constant data in program memory.

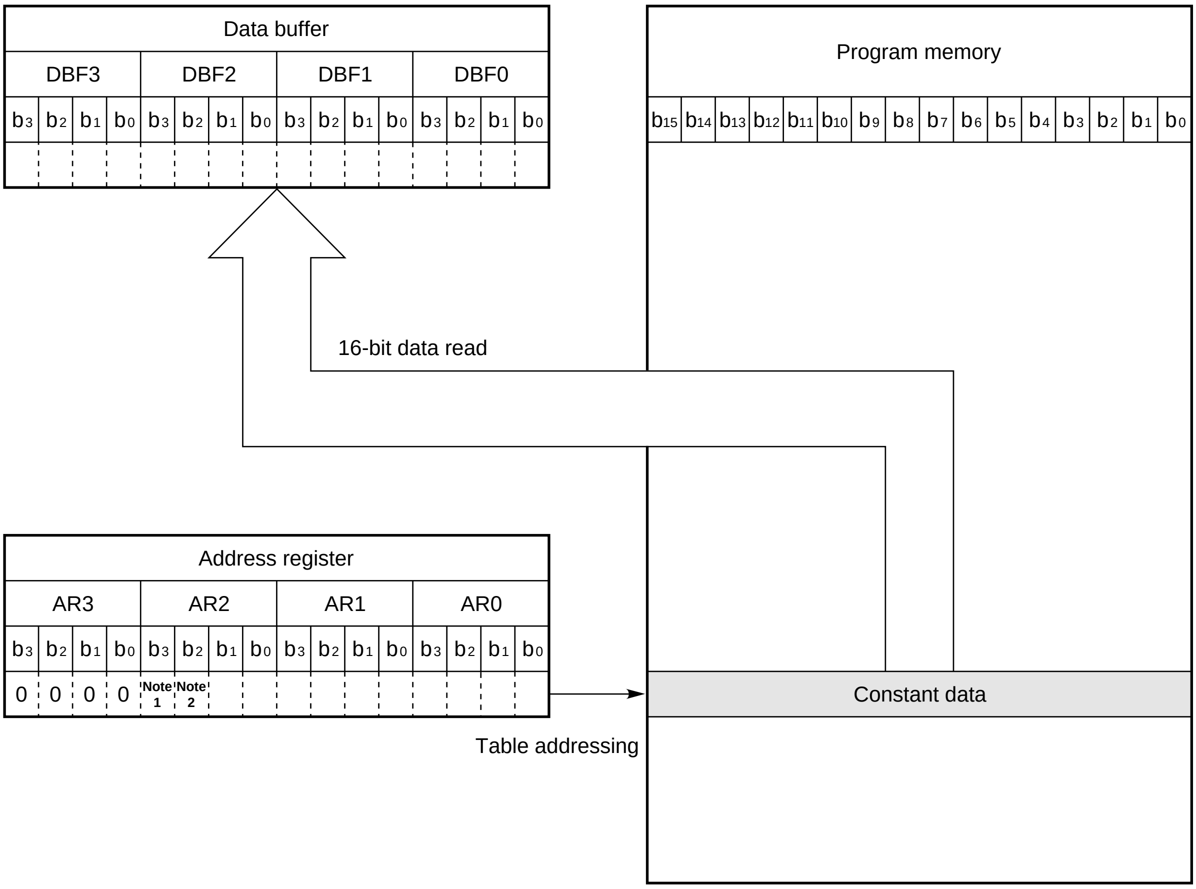
The table reference instruction (MOVT DBF, @AR) is used to store the contents of the program memory address specified by the address register in the data buffer.

Since each location in program memory contains 16 bits of information, the MOVT instruction causes 16 bits of data to be stored in the data buffer. The address register can be used to table reference any location in program memory.

Caution When referencing a table, be careful not to exceed the usable stack level; one level of the stack is temporarily used. Also see Section 8.2 and Chapter 10.

Remark Exceptionally, execution of a table reference instruction requires two instruction cycles. *

Figure 3-4. MOVT DBF, @AR Instruction


Data buffer
DBF3
DBF2
DBF1
DBF0
b3 b2 b1 b0 b3 b2 b1 b0 b3 b2 b1 b0 b3 b2 b1 b0
Program memory
b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0
16-bit data read
Address register
AR3
AR2
AR1
AR0
b3 b2 b1 b0 b3 b2 b1 b0 b3 b2 b1 b0 b3 b2 b1 b0
0 0 0 0 Note 1 Note 2
Table addressing
Constant data

Notes 1. Always 0 for the µPD17145 and µPD17147

2. Always 0 for the µPD17145

(1) Constant data table

Example 1 shows an example of code used to reference a constant data table.

Example 1. Code used for reading the data in a constant data table.

```
OFFSET          MEM  0.00H            ; Storage area for an offset address.

ROMREF:
          ; BANK0
                                      ; Stores the start address of the  constant data
                                      ; table in the address register (AR).
            MOV       AR3, #.DL.TABLE SHR 12 AND 0FH
            MOV       AR2, #.DL.TABLE SHR 8 AND 0FH
            MOV       AR1, #.DL.TABLE SHR 4 AND 0FH
            MOV       AR0, #.DL.TABLE AND 0FH

          ; MOV       RPH, #0         ; Set the register pointer to row
            MOV       RPL, #7 SHL 1   ; address 7.

            ADD       AR0, OFFSET     ; Adds the offset address.
            ADDC      AR1, #0
            ADDC      AR2, #0
            ADDC      AR3, #0
            MOVT      DBF, @AR        ; Reads the constant data.
TABLE:
            DW        0001H           ; When OFFSET = 0H
            DW        0002H
            DW        0004H
            DW        0008H
            DW        0010H
            DW        0020H
            DW        0040H
            DW        0080H
            DW        0100H
            DW        0200H
            DW        0400H
            DW        0800H
            DW        1000H
            DW        2000H
            DW        4000H
            DW        8000H           ; When OFFSET = 0FH

            END
```

(2) Branch address table

Example 2 shows an example of code used to reference a branch address table.

Example 2. Code used for branching to the address in a branch address table.

```
OFFSET      MEM       0.00H            ; Storage are for an offset address.

ROMREF:
          ; BANK0                      ; Stores the start address of the constant data
                                       ; table in the address register (AR).
            MOV       AR3, #.DL.TABLE SHR 12 AND 0FH
            MOV       AR2, #.DL.TABLE SHR 8 AND 0FH
            MOV       AR1, #.DL.TABLE SHR 4 AND 0FH
            MOV       AR0, #.DL.TABLE AND 0FH

          ; MOV       RPH, #0          ; Sets the register pointer to row
            MOV       RPL, #7 SHL 1    ; address 7.

            ADD       AR0, OFFSET      ; Adds the offset address.
            ADDC      AR1, #0
            MOVT      DBF, @AR         ; Reads the branch address.
            PUT       AR, DBF          ; AR <– Branch address
            BR        @AR
TABLE:
            DW        0001H            ; When OFFSET = 0H
            DW        0002H
            DW        0004H
            DW        0008H
            DW        0010H
            DW        0020H
            DW        0040H
            DW        0080H
            DW        0100H
            DW        0200H            ; When OFFSET = 9H

            END
```

[MEMO]

CHAPTER 4   PROGRAM COUNTER (PC)

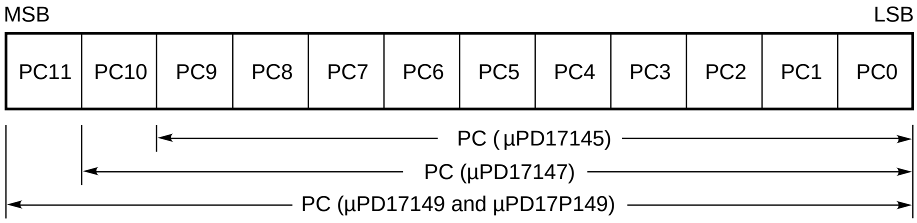
The program counter is used to specify an address in program memory.

4.1   PROGRAM COUNTER CONFIGURATION

As shown in Figure 4-1, the program counter is a 12-bit binary counter.

Remark The size of the program counter depends on the product. The µPD17145 has the 10-bit program counter and the µPD17147 has the 11-bit program counter.

Figure 4-1. Program Counter



4.2 PROGRAM COUNTER OPERATION

Normally, the program counter is automatically incremented each time a command is executed. The memory address at which the next instruction to be executed is stored is assigned to the program counter under the following conditions: At reset; when a branch, subroutine call, return, or table referencing instruction is executed; or when an interrupt is received.

Sections 4.2.1 to **4.2.7** explain program counter operation during execution of each instruction.

Figure 4-2. Value of the Program Counter After an Instruction Is Executed

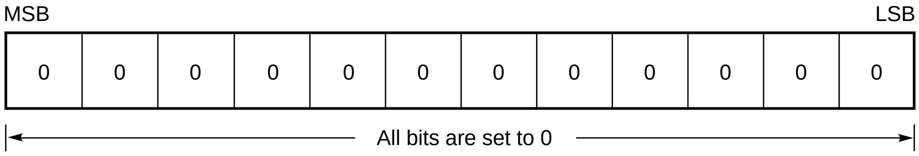
Program counter bit Instruction	Program counter value											
	PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
During reset	0	0	0	0	0	0	0	0	0	0	0	0
BR addr	0	Value specified in addr										
	1											
CALL addr	0											
BR @AR CALL @AR (MOVT DBF, @AR)	Value in the address register (AR)											
RET RETSK RETI	Value in the address stack location pointed to by the stack pointer (return address)											
During interrupt	Vector address for the interrupt											

Remark The µPD17145 does not have PC11 or PC10. The µPD17147 does not have PC11.

4.2.1 Program Counter at Reset

By setting the $\overline{\text{RESET}}$ terminals to low, the program counter is set to 000H.

Figure 4-3. Value in the Program Counter after Reset



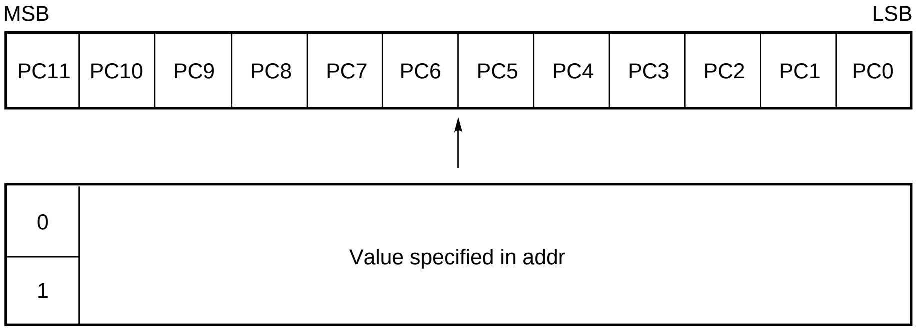
0	0	0	0	0	0	0	0	0	0	0	0

4.2.2 Program Counter during Execution of the Branch Instruction (BR)

There are two ways to specify branching using the branch instruction. One is branch to the address specified in the operand using the direct branch instruction (BR addr). The other is branch to the address specified by the address register using the indirect branch instruction (BR @AR).

The address specified by a BR addr instruction is placed in the program counter.

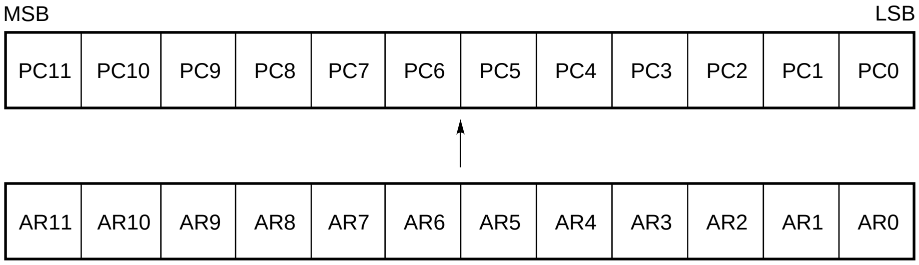
Figure 4-4. Value in the Program Counter during Execution of a BR addr Instruction



Remark The µPD17145 does not have PC11 or PC10. The µPD17147 does not have PC11.

A BR @AR instruction causes the address in the address counter to be placed in the program counter.

Figure 4-5. Value in the Program Counter during Execution of a BR @AR Instruction



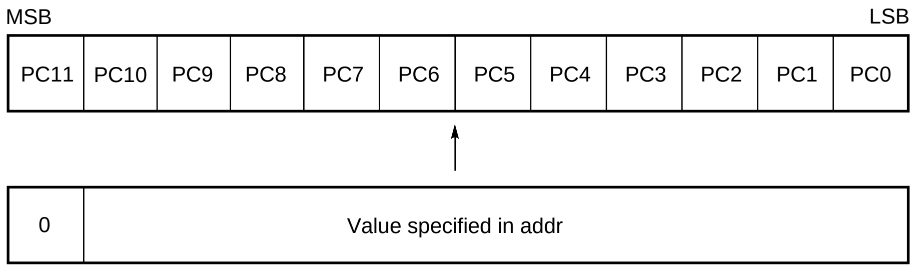
Remark The µPD17145 does not have PC11 or PC10. The µPD17147 does not have PC11.

4.2.3 Program Counter during Execution of Subroutine Calls (CALL)

There are two ways to specify branching using subroutine calls. One is to branch to the address specified in the operand using the direct subroutine call (CALL addr). The other is branch to the address specified by the address register using the indirect subroutine call (CALL @AR).

A CALL addr instruction causes the value in the program counter to be saved in the address stack register and then the address specified in the operand to be placed in the program counter. CALL addr instructions can specify addresses 0000H to 07FFH.

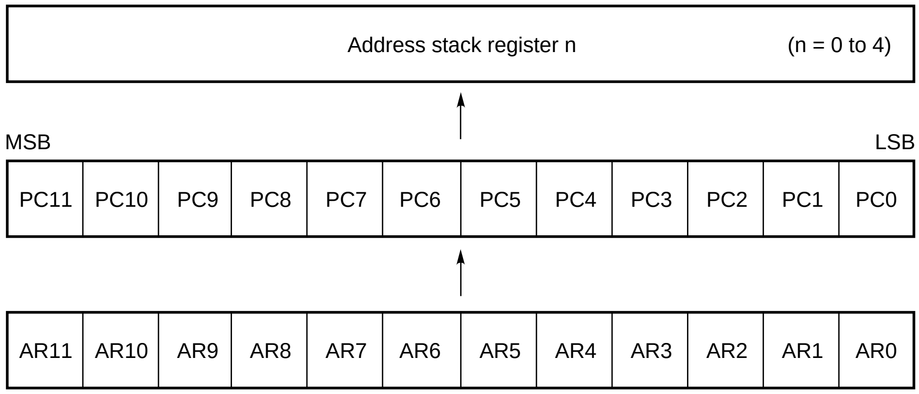
Figure 4-6. Value in the Program Counter during Execution of a CALL addr Instruction



Remark The μPD17145 does not have PC11 or PC10. The μPD17147 does not have PC11.

A CALL @AR instruction causes the value in the program counter to be saved in the address stack register and then the value in the address register to be placed in the program counter.

Figure 4-7. Value in the Program Counter during Execution of a CALL @AR Instruction

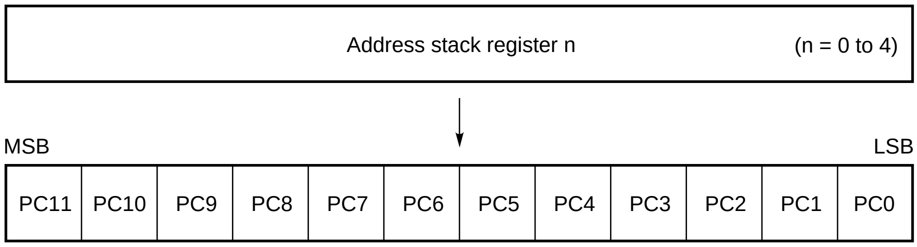


Remark The μPD17145 does not have PC11 or PC10. The μPD17147 does not have PC11.

4.2.4 Program Counter during Execution of Return Instructions (RET, RETSK, RETI)

During execution of a RET, RETSK, or RETI instruction, the program counter is restored to the value saved in the address stack register.

Figure 4-8. Value in the Program Counter during Execution of a RET, RETSK, or RETI Instruction



Remark The µPD17145 does not have PC11 or PC10. The µPD17147 does not have PC11.

4.2.5 Program Counter during Table Reference (MOVT)

During execution of MOVT DBF, @AR instruction, the value in the program counter is saved in the stack, the address register is set by the program counter, then the contents stored at that program memory location is read into the data buffer (DBF). After the program memory contents are read into DBF, the program counter is restored to the value saved in the address stack register.

Caution One level of the address stack is temporarily used during execution of table reference. Be careful of the stack level.

4.2.6 Program Counter during Execution of Skip Instructions (SKE, SKGE, SKLT, SKNE, SKT, SKF)

When skip conditions are met and a skip instruction is executed, the instruction immediately following the skip instruction is treated as a NOP instruction. Therefore, whether skip conditions are met or not, the number of instructions executed and instruction execution time remain the same.

4.2.7 Program Counter When an Interrupt Is Received

When an interrupt is received, the value in the program counter is saved in the address stack. Next, the vector address for the interrupt received is placed in the program counter.

[MEMO]

CHAPTER 5 STACK

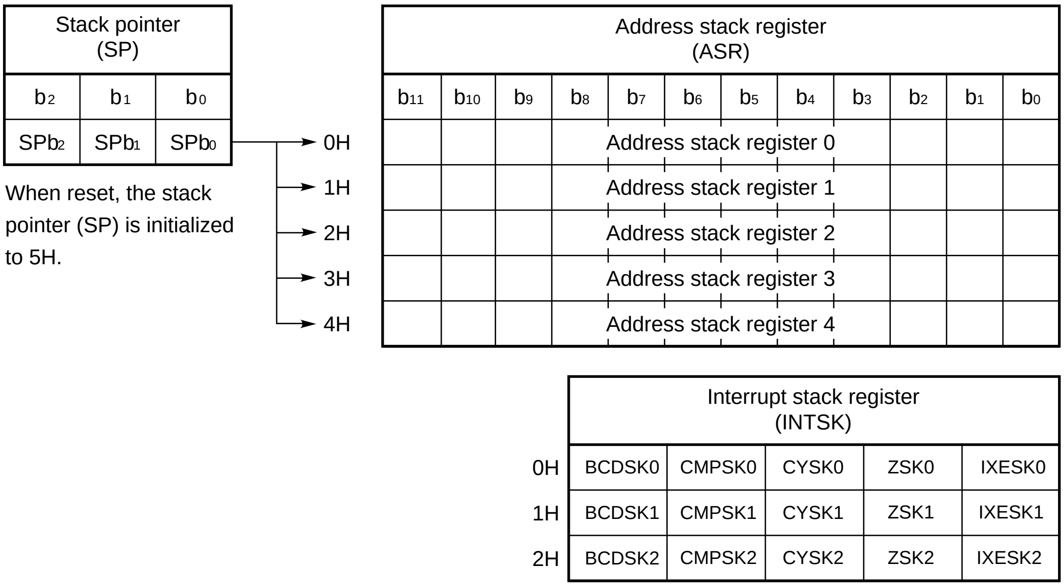
The stack is a register used to save information such as the program return address and the contents of the system register during execution of subroutine calls, interrupts and similar operations.

5.1 STACK CONFIGURATION

Figure 5-1 shows the configuration of the stack.

The stack consists of the following parts: one 3-bit binary counter stack pointer (SP), five 10-bit (µPD17145)/ 11-bit (µPD17147)/12-bit (µPD17149) address stack registers (ASR), and three 5-bit interrupt stack registers (INTSK).

Figure 5-1. Stack Configuration


Stack pointer (SP)
b2 b1 b0
SPb2 SPb1 SPb0
When reset, the stack pointer (SP) is initialized to 5H.
0H
1H
2H
3H
4H
Address stack register (ASR)
b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0
Address stack register 0
Address stack register 1
Address stack register 2
Address stack register 3
Address stack register 4
Interrupt stack register (INTSK)
0H BCDSK0 CMPSK0 CYSK0 ZSK0 IXESK0
1H BCDSK1 CMPSK1 CYSK1 ZSK1 IXESK1
2H BCDSK2 CMPSK2 CYSK2 ZSK2 IXESK2

5.2 FUNCTIONS OF THE STACK

The stack is used to save the return address during execution of subroutine calls and table reference instructions. When an interrupt occurs, the program return address and the program status word (PSWORD) are automatically saved in the stack. Then, all bits of the PSWORD are cleared to 0.

5.3 ADDRESS STACK REGISTER (ASR)

As shown in Figure 5-1, the address stack register consists of five consecutive 12-bit registers.

- A return address is stored in the address stack register when a CALL addr instruction or CALL @AR instruction, or the first cycle of a MOVT DBF, @AR instruction is executed, or an interrupt is received.
- When a PUSH AR instruction is executed, the contents of the address register (AR) is stored in the ASR at the address pointed to by the stack pointer minus 1.
- When a RET, RETSK, or RETI instruction, or the second cycle of a MOVT DBF, @AR instruction is executed, the contents of the ASR (return address) pointed to by the stack pointer is restored to the program counter and the stack pointer is incremented.
- When a POP AR instruction is executed, the value in the ASR pointed to by the stack pointer is transferred to the address register and the stack pointer is incremented.

Caution If the stack pointer causes an underflow when a CALL addr or CALL @AR instruction has been executed, or an interrupt has been handled, the underflow is determined to be a software crush. The controller generates an internal reset signal and initializes all internal hardware to its initial state. Then, the program starts at address 0000H.

Remark The size of the ASR depends on products. The ASRs of the µPD17145 and µPD17147 consist of five consecutive 10-bit and 11-bit registers respectively.

5.4 INTERRUPT STACK REGISTER (INTSK)

As shown in Figure 5-1, the interrupt stack register (INTSK) consists of three 5-bit registers.

- When an interrupt is received, five bits in the system register (SYSREG) (mentioned later) that is, each of the 5-bit program status word (PSWORD) flags (BCD, CMP, CY, Z, IXE), are saved in the INTSK. All the bits of the PSWORD are cleared to 0 after they are saved.
- When the RETI instruction is executed, the PSWORD is restored from the contents of the INTSK.
- In the INTSK, every time an interrupt is received, necessary data is saved.

Caution When more than three interrupts are received, the data from the first interrupt is lost.

5.5 STACK POINTER (SP) AND INTERRUPT STACK REGISTER

As shown in Figure 5-1, the stack pointer (SP) is a 3-bit binary counter used to point to addresses in the five address stack registers. The stack pointer (SP) is located at address 01H in the register file and initialized to 5H when reset.

- As shown in Table 5-1, the SP is decremented when a CALL addr or CALL @AR instruction, the first cycle of a MOVT DBF, @AR instruction, or a PUSH AR instruction is executed, or an interrupt is accepted.
- The SP is incremented when a RET or RETSK instruction is executed, the second instruction cycle of a MOVT DBF, @AR instruction, a POP AR instruction, or a RETI instruction is executed.

The interrupt stack counter as well as the stack pointer (SP) is decremented when an interrupt is accepted. The interrupt stack counter is incremented by a RETI only.

Table 5-1. Operation of the Stack Pointer

Instruction	Stack pointer value	Counter of interrupt stack register
CALL addr CALL @AR MOVT DBF, @AR (first instruction cycle) PUSH AR	–1	Does not change
RET RETSK MOVT DBF, @AR (second instruction cycle) POP AR	+1	
Interrupt receipt	–1	–1
RETI	+1	+1

Remark Exceptionally, execution of a MOVT DBF, @AR instruction requires two instruction cycles. *

As mentioned above, the stack pointer (SP) is a 3-bit counter and therefore can conceivably store any of the eight values from 0H to 7H. Since there are only five address stack registers, however, a stack pointer (SP) value that is greater than five will cause an **internal reset signal to be generated** (to prevent a software crash).

Since the stack pointer (SP) is located in the register file, it can be written to directly by using the POKE instruction to manipulate the register file. When this is done, the stack pointer (SP) value will change but the values in the address stack register will not be affected. The stack pointer (SP) can also be read by using the PEEK instruction.

When reset, the stack pointer (SP) is set to 5H.

5.6 STACK OPERATION

Stack operation during execution of each command is explained in **Sections 5.6.1** to **5.6.3**.

5.6.1 Stack Operation during CALL, RET, or RETSK Instruction

Table 5-2 shows operation of the stack pointer (SP), address stack register, and the program counter (PC) during execution of the CALL, RET, or RETSK instruction.

Table 5-2. Operation of the Stack Pointer during Execution of the CALL, RET, or RETSK Instruction

Instruction	Operation
CALL addr CALL @AR	**<1>** Stack pointer (SP) is decremented. **<2>** Program counter (PC) is saved in the address stack register pointed to by the stack pointer (SP). **<3>** Value specified by the instruction operand (addr or @AR) is transferred to the program counter.
RET RETSK	**<1>** Value in the address stack register pointed to by the stack pointer (SP) is restored to the program counter (PC). **<2>** Stack pointer (SP) is incremented.

When the RETSK instruction is executed, the first command after data restoration becomes a NOP instruction.

5.6.2 Stack Operation during Table Reference (MOVT DBF, @AR)

Table 5-3 shows stack operation during table reference.

Table 5-3. Stack Operation during Execution of the MOVT DBF, @AR Instruction

Instruction	Instruction cycle	Operation
MOVT DBF, @AR	First	**<1>** Stack pointer (SP) is decremented. **<2>** Program counter (PC) is saved in the address stack register pointed to by the stack pointer (SP). **<3>** Value in the address register (AR) is transferred to the program counter (PC).
	Second	**<4>** Contents of the program memory (ROM) pointed to by the program counter (PC) is transferred to the data buffer (DBF). **<5>** Value in the address stack register pointed to by the stack pointer (SP) is restored to the program counter (PC). **<6>** Stack pointer (SP) is incremented.

* **Caution When executing a MOVT DBF, @AR instruction, be careful not to exceed the usable stack level; one level of the stack is temporarily used.**

* **Remark** Exceptionally, execution of a MOVT DBF, @AR instruction requires two instruction cycles.

5.6.3 Stack Operation during Interrupt Receipt and Execution of a RETI Instruction

Table 5-4 shows stack operation during interrupt receipt and execution of a RETI instruction.

Table 5-4. Stack Operation during Interrupt Receipt and Execution of the RETI Instruction

Instruction	Operation
Receipt of interrupt	**<1>** Stack pointer (SP) is decremented. **<2>** Value in the program counter (PC) is saved in the address stack register pointed to by the stack pointer (SP). **<3>** Values in the PSWORD flags (BCD, CMP, CY, Z, IXE) are saved in the interrupt stack register. **<4>** Vector address is transferred to the program counter (PC)
RETI	**<1>** Values in the interrupt stack register are restored to the PSWORD flags (BCD, CMP, CY, Z, IXE). **<2>** Value in the address stack register pointed to by the stack pointer (SP) is restored to the program counter (PC). **<3>** Stack pointer (SP) is incremented.

5.7 STACK NESTING LEVELS AND THE PUSH AND POP INSTRUCTIONS

During execution of operations such as subroutine calls and returns, the stack pointer (SP) simply functions as a 3-bit counter which is incremented and decremented. When the value in the stack pointer (SP) is 0H and a CALL or MOVT instruction is executed or an interrupt is received, the stack pointer (SP) is decremented to 7H. The µPD17149 treats this condition as a fault and generates an internal reset signal.

In order to avoid this condition, when the address stack register is being used frequently, use the PUSH and POP instructions to save or restore the address stack register if necessary.

Table 5-5 shows stack operation during execution of the PUSH and POP instructions.

Table 5-5. Stack Operation during Execution of the PUSH and POP Instructions

Instruction	Operation
PUSH	**<1>** Stack pointer (SP) is decremented. **<2>** Value in the address register (AR) is transferred to the address stack register pointed to by the stack pointer (SP).
POP	**<1>** Value in the address stack register pointed to by the stack pointer (SP) is transferred to the address register (AR). **<2>** Stack pointer (SP) is incremented.

[MEMO]

CHAPTER 6 DATA MEMORY (RAM)

Data memory (RAM) stores data such as operation and control data. Data can be read from or written to data memory with an instruction during normal operation.

6.1 DATA MEMORY CONFIGURATION

Data memory locations have 7-bit addresses. The three high-order bits of each address are called the row address, and the four low-order bits are called the column address.

For example, the row address of address 1AH is 1H. The column address is 0AH.

Each addressed memory location is 4-bits (one nibble) long.

Remark A “byte” is a unit of eight bits. A “nibble” is a unit of four bits.

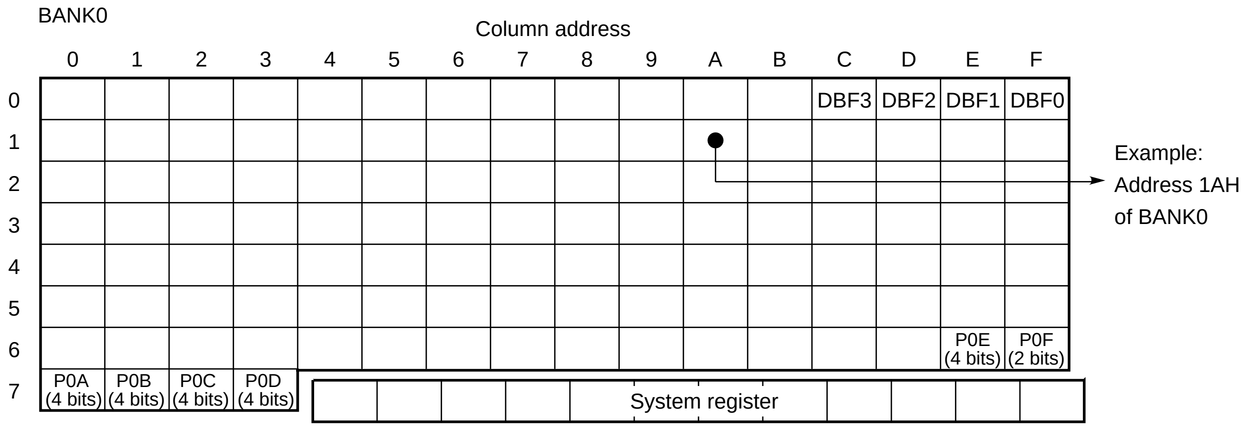
Four bits	=	one nibble	
Eight bits	=	two nibbles	= one byte
16 bits	=	four nibbles	= two bytes

Data memory contains an area to which the user is allowed to store data freely, as well as areas which are reserved for the use of specific functions.

The areas reserved for specific functions are as follows:

- System register (SYSREG) (See **Chapter 8**.)
- Data buffer (DBF) (See **Chapter 10**.)
- Port registers (See **Chapter 12**.)

Figure 6-1. Organization of Data Memory



[MEMO]

CHAPTER 7 GENERAL REGISTER (GR)

The general register, as the name implies, is a general register used for data transfer and manipulation. In the 17K series, the location of the general register is not fixed. The area used for the general register is in data memory, as specified by the general register pointer (RP). Thus, part of the data memory area can be specified as the general register as required, allowing data transfer in data memory and data memory manipulation to be performed with a single instruction.

7.1 GENERAL REGISTER POINTER (RP)

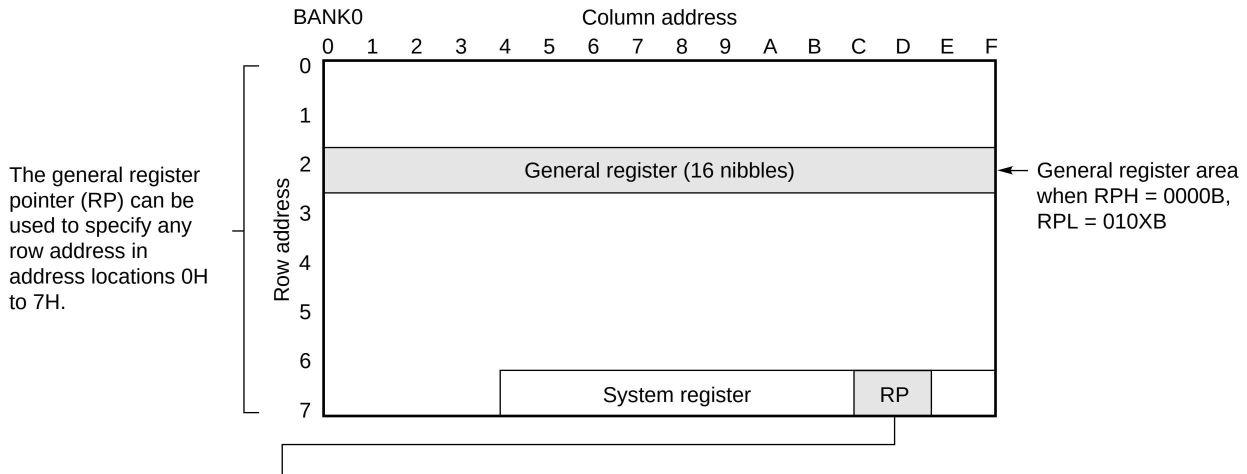
RP is a pointer used to specify part of data memory as the general register. In RP, specify a desired data memory bank and row address for the general register. RP consists of seven bits: 7DH (RPH), and the three high-order bits of 7EH (RPL) in the system register (see **Chapter 8**).

Set a bank in RPH, and a data memory row address in RPL.

Caution The least significant bit of RPL is the BCD flag (see Section 8.7).

Remark RPH of the µPD17149 is always 0, indicating bank 0. (This prevents other than bank 0 from being specified.)

Figure 7-1. General Register Pointer Configuration


BANK0
Column address
0 1 2 3 4 5 6 7 8 9 A B C D E F
Row address
0
1
2
3
4
5
6
7
General register (16 nibbles)
General register area when RPH = 0000B, RPL = 010XB
The general register pointer (RP) can be used to specify any row address in address locations 0H to 7H.
System register
RP

Address	7DH				7EH			
Name	General register pointer (RP)							
Symbol	RPH				RPL			
Bits	b3	b2	b1	b0	b3	b2	b1	b0
Data	0	0	0	0				B C D
Reset	0	0	0	0	0	0	0	

[MEMO]

CHAPTER 8 SYSTEM REGISTER (SYSREG)

The system register (SYSREG), located in data memory, is used for direct control of the CPU.

8.1 SYSTEM REGISTER CONFIGURATION

Figure 8-1 shows the allocation address of the system register in data memory. As shown in Figure 8-1, the system register is allocated in addresses 74H to 7FH of data memory.

Since the system register is allocated in data memory, it can be manipulated using any of the instructions available for manipulating data memory. Therefore, it is also possible to put the system register in the general register.

Figure 8-1. Allocation of System Register in Data Memory

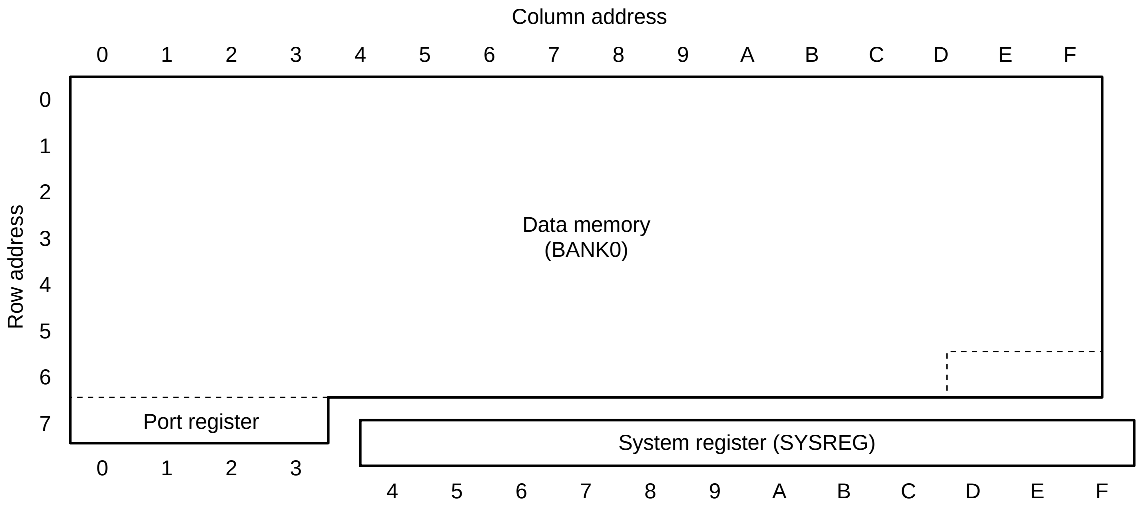


Figure 8-2 shows the configuration of the system register. As shown in Figure 8-2, the system register consists of the following seven registers.

• Address register	(AR)
• Window register	(WR)
• Bank register	(BANK)
• Index register	(IX)
• Data memory row address pointer	(MP)
• General register pointer	(RP)
• Program status word	(PSWORD)

Figure 8-2. System Register Configuration

Address	74H				75H				76H				77H				78H				79H				7AH				7BH				7CH				7DH				7EH				7FH			
Name	Address register (AR)																Window register (WR)				Bank register (BANK)				Index register (IX)	Data memory row address pointer (MP)											General register pointer (RP)							Program status word (PSWORD)				
Symbol	AR3				AR2				AR1				AR0				WR				BANK				IXH MPH				IXM MPL				IXL				RPH				RPL				PSW			
Bit	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Data[Note 1]																										← (IX) →																						
	0	0	0	0	Note 2												← →				0	0	0	0	MPE	0	0	0	0								0	0	0	0				BCD	CMP	CY	Z	IXE
	← (AR) →																				← (BANK) →					← (MP) →											← (RP) →											
Initial value when reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Not defined				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Notes 1. A bit indicating zero is fixed to zero.

2. Bit b_3 and b_2 of AR2 are always 0 for the μPD17145. Bit b_3 of AR2 is always 0 also for the μPD17147.

8.2 ADDRESS REGISTER (AR)

8.2.1 Address Register Configuration

Figure 8-3 shows the configuration of the address register.

As shown in Figure 8-3, the address register consists of the sixteen bits in address 74H to 77H of the system register. However, since the 4/5/6 high-order bits are always set to 0, the address register is actually 12/11/10 bits. When the system is reset, all sixteen bits of the address register are reset to 0.

Figure 8-3. Address Register Configuration

Address	74H				75H				76H				77H			
Name	Address register (AR)															
Symbol	AR3				AR2				AR1				AR0			
Bit	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Data	(AR)															
	0	0	0	0	**Note**	**Note**										
Initial value when reset	0				0				0				0			

Note Bit b_3 and b_2 of AR2 are always 0 for the μPD17145. Bit b_3 of AR2 is always 0 also for the μPD17147.

8.2.2 Address Register Functions

The address register (AR) is used to specify an address in program memory when executing a BR @AR instruction, CALL @AR instruction, or MOVT DBF, @AR instruction. The contents of the AR can be written to the address stack register (ASR) by the PUSH AR stack manipulation instruction. Also, the contents of the ASR can be read into the AR by the POP AR stack manipulation instruction.

Items (1) to (4) explain address register operation during execution of each instruction.

The address register can be incremented by using the dedicated increment instruction (INC AR).

(1) Indirect branch instruction (BR @AR)

When the BR @AR instruction is executed, the program branches to the address in program memory specified by the contents of the address register.

(2) Indirect subroutine call (CALL @AR)

When the CALL @AR instruction is executed, the subroutine located at the address in program memory specified by the contents of the address register is called.

(3) Table reference (MOVT DBF, @AR)

When the MOVT DBF, @AR instruction is executed, the contents of the program memory (16-bit data) located at the address specified by the contents of the address register are read into the data buffer (addresses 0CH to 0FH of BANK0 in data memory).

(4) Stack manipulation instructions (PUSH AR, POP AR)

When the PUSH AR instruction is executed, the stack pointer (SP) is first decremented and then the address register is stored in the address stack register pointed to by the stack pointer.

When the POP AR instruction is executed, the contents of the address stack register pointed to by the stack pointer is transferred to the address register and then the stack pointer is incremented.

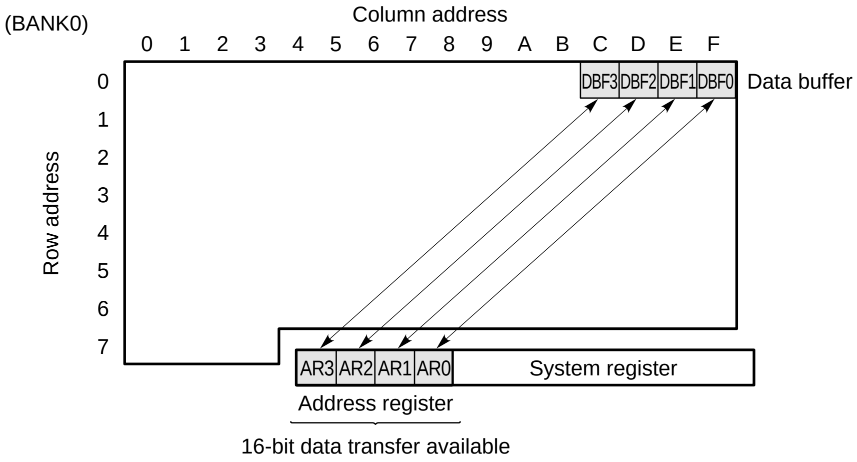
Also see **Chapter 5**.

(5) Address register used as a peripheral hardware register

The address register can be manipulated four bits at a time. The address register can also be used as a peripheral hardware register for transferring 16-bit data to the data buffer. In other words, by using the PUT AR, DBF and GET DBF, AR instructions, the address register can be used to transfer 16-bit data to the data buffer.

Note that the data buffer is allocated in addresses 0CH to 0FH of BANK0 in data memory.

Figure 8-4. Address Register Used as a Peripheral Hardware Register



8.3 WINDOW REGISTER (WR)

8.3.1 Window Register Configuration

Figure 8-5 shows the configuration of the window register.

As shown in Figure 8-5, the window register consists of four bits allocated at address 78H of the system register. The contents of the window register is undefined after a system reset. However, when the $\overline{\text{RESET}}$ input is used to release the system from HALT or STOP mode, the previous state of the window register is maintained.

Figure 8-5. Window Register Configuration

Address	78H			
Name	Window register			
Symbol	WR			
Bit	b_3	b_2	b_1	b_0
Data	←――――――→			
Initial value when reset	Not defined			

8.3.2 Window Register Functions

The window register is used to transfer data to and from the register file (RF).

Data is transferred to and from the register file using instructions PEEK WR, rf and POKE rf, WR.

See **Section 9.2.3** for details.

8.4 BANK REGISTER (BANK)

Figure 8-6 shows the configuration of the bank register.

The bank register consists of four bits at address 79H of the system register. However, all bits are set to 0.

Figure 8-6. Bank Register Configuration

Address	79H			
Name	Bank register			
Symbol	BANK			
Bit	b_3	b_2	b_1	b_0
Data	0	0	0	0
	← (BANK) →			
Initial value when reset	0			

8.5 INDEX REGISTER (IX) AND DATA MEMORY ROW ADDRESS POINTER (MEMORY POINTER: MP)

8.5.1 Index Register (IX)

IX is used for data memory address modification. IX differs from MP in that IX is used to modify the bank and the address specified by operand m.

As shown in Figure 8-7, IX consists of 12 bits in the system register: 7AH (IXH), 7BH (IXM), and 7CH (IXL). The index register enable flag (IXE) that enables address modification with IX is assigned to the least significant bit of PSW.

When IXE = 1, the data memory address specified by operand m is modified by ORing it with the contents of IXM and IXL. The specified bank is also modified by ORing it with the contents of BANK and IXH.

Remark IXH of the µPD17149 is always 0. Even when IXE = 1, the bank is not modified. (This prevents other than bank 0 from being specified.)

8.5.2 Data Memory Row Address Pointer (Memory Pointer: MP)

MP is used for data memory address modification. It differs from IX in that MP is used to modify the bank and the row address of the address specified indirectly by operand @r.

As shown in Figure 8-7, MPH and IXH, as well as MPL and IXM, are located at the same addresses, respectively. (Both MPH and IXH are at 7AH in the system register, while both MPL and IXM are at 7BH in the system register). The seven bits, consisting of the three low-order bits of MPH and MPL, are actually used as MP. The least significant bit of MPH is the memory pointer enable flag (MPE) that enables address modification by MP.

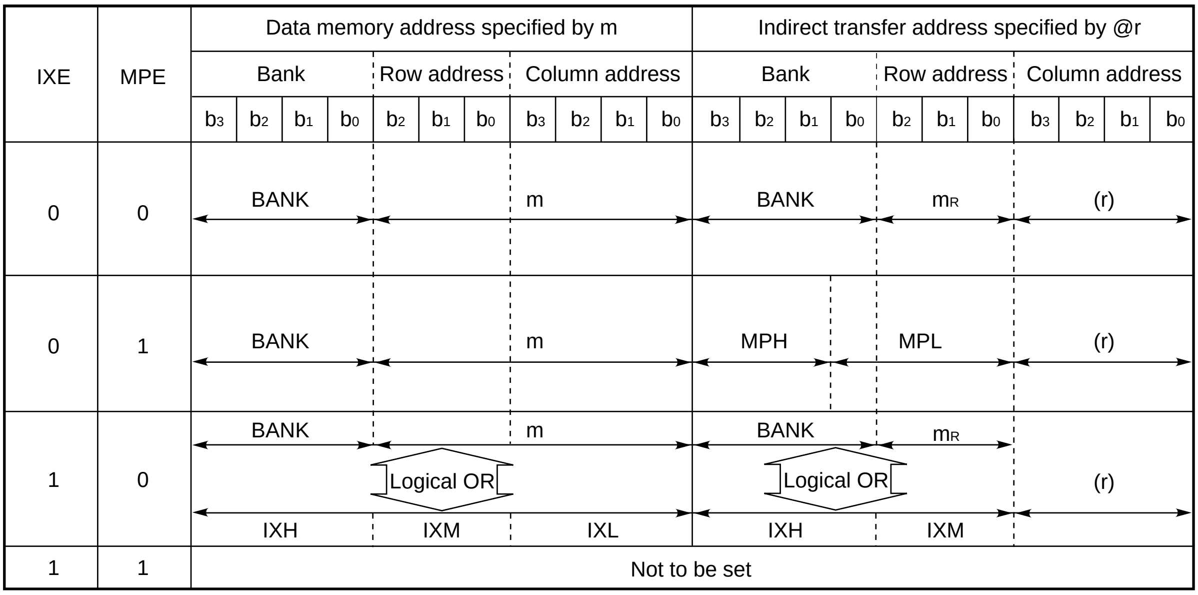
When MPE = 1, the data memory bank and row address specified indirectly by operand @r are not the contents of BANK and m_R; the address specified by MP is used. (The column address is specified by r, regardless of the setting of MPE.) In this case, the four bits consisting of the three low-order bits of MPH and the most significant bit of MPL specify a bank, while the three low-order bits of MPL specify a row address.

Remark For the µPD17149, the three low-order bits of MPH and the most significant bit of MPL are always 0. Even when MPE = 1, bank 0 is selected. (This prevents other than bank 0 from being specified).

Figure 8-7. Configuration of Index Register and Memory Pointer

Address	7AH				7BH				7CH				7FH			
Name	Index register (IX)												Four low-order bits of the program status word (PSWORD)			
		Memory pointer (MP)														
Symbolic name	IXH				IXM				IXL				PSW			
	MPH				MPL											
Bit	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Flag name	MPE															IXE
Data		← (IX) →														
		← (MP) →														
		0	0	0	0											
Initial value when reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-8. Modification of Data Memory Addresses Using the Index Register and Memory Pointer



BANK : Bank register
IX : Index register
IXE : Index enable flag
IXH : Index register bits 8 to 10
IXM : Index register bits 4 to 7
IXL : Index register bits 0 to 3
m : Data memory address specified by m_R and m_C
m_R : Data memory row address
m_C : Data memory column address

MP : Memory pointer
MPE : Memory pointer enable flag
MPH : Three high-order bits of the memory pointer
MPL : Four low-order bits of the memory pointer
r : General register column address
RP : General register pointer
(x) : Contents of x
x: direct address such as r

Table 8-1. Instructions for Which Address Modification Is Performed

Arithmetic operation	ADD ADDC SUB SUBC	r, m m, #n4
Logical operation	AND OR XOR	r, m m, #n4
Evaluation	SKT SKF	m, #n
Comparison	SKE SKGE SKLT SKNE	m, #n4
Transfer	LD	r, m
	ST	m, r
	MOV	m, #n4 @r, m m, @r

8.5.3 IXE = 0 and MPE = 0 (No Data Memory Modification)

As shown in Figure 8-8, data memory addresses are not affected by the index register and the data memory row address pointer.

(1) Data memory manipulation instructions

Example 1. Execution of ADD r, m when the general register is in row address 0

```
R003    MEM    0.03H
M061    MEM    0.61H
        ADD    R003, M061      ; Memory-to-memory addition  (0.03H) <– (0.03H)
                               ; + (0.61H)
```

As shown in Figure 8-9, when this instruction is executed, the data in general register address R003 and data memory address M061 are added together and the result is stored in general register address R003.

(2) Indirect transfer of data in the general register (horizontal indirect transfer)

Example 2. Execution of MOV @r, m when the general register is in row address 0

```
R005    MEM     0.05H
M034    MEM     0.34H
        MOV     R005,  #8          ; R005 <– 8 (column address setting of @r)
        MOV     @R005, M034        ; Indirect transfer of data in the register
                                   ; (0.38H) <– (0.34H)
```

As shown in Figure 8-9, when this instruction is executed, the data stored in data memory address M034 is transferred to data memory location 38H.

The MOV @r, m instruction transfers the contents of the data memory location addressed by m to a data memory location whose column address is specified by @r and whose row address is the same as that specified by m.

In the above example, data at M034 is transferred to 38H, where the row address is the same as that of M034 (= 3), and the column address is specified by the contents of R005 (= 8).

Example 3. Execution of MOV m, @r when the general register is in row address 0

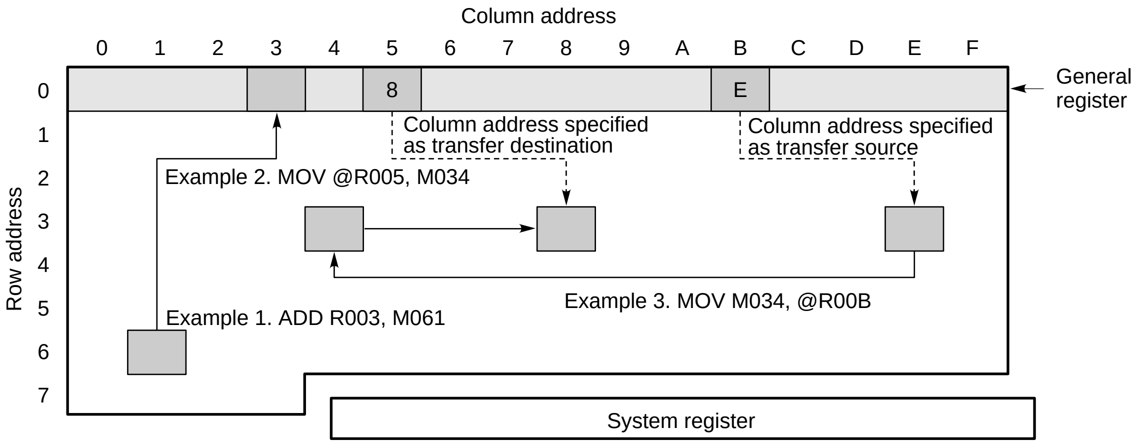
```
R00B    MEM     0.0BH
M034    MEM     0.34H
        MOV     R00B, #0EH         ; R00B <– 0EH (column address setting of @r)
        MOV     M034, @R00B        ; Indirect transfer of data in the register
                                   ; (0.34H) <– (0.3EH)
```

As shown in Figure 8-9, when this instruction is executed, the contents of data memory stored at address 3EH is transferred to data memory location M034.

The MOV m, @r instruction transfers the contents of a data memory location whose row address is the same as the row address of m, and whose column address is specified by @r, to a data memory location addressed by m.

In the above example, the data at 3EH, where the row address is the same as that of M034 (= 3) and the column address is specified by the contents of R00B (= 0EH), is transferred to M034.

Figure 8-9. Example of Operation When IXE = 0 and MPE = 0


Column address
0 1 2 3 4 5 6 7 8 9 A B C D E F
Row address
0 1 2 3 4 5 6 7
8
E
General register
Column address specified as transfer destination
Column address specified as transfer source
Example 2. MOV @R005, M034
Example 3. MOV M034, @R00B
Example 1. ADD R003, M061
System register

Addresses in Example 1

ADD R003, M061

	Bank	Row address	Column address
Data memory address M	0000	110	0001
General register address R	0000	000	0011

Addresses in Example 2

MOV @R005, M034

	Bank	Row address	Column address
Data memory address M	0000	011	0100
General register address R	0000	000	0101
Indirect transfer address @R	0000	011	1000
	Same as M		Contents of R

8.5.4 IXE = 0 and MPE = 1 (Diagonal Indirect Data Transfer)

As shown in Figure 8-8, the indirect data transfer bank and row address specified by @r become the data memory row address pointer value only when general register indirect data transfer instructions (MOV @r, m and MOV m, @r) are used.

Example 1. Execution of MOV @r, m when the general register is in row address 0

```
R005    MEM    0.05H
M034    MEM    0.34H
        MOV    MPL, #0110B      ; MP <– 6 (row address setting of @r)
        MOV    MPH, #1000B      ; MPE <– 1, bank <– 0
        MOV    R005, #8         ; R005 <– 8 (column address setting of @r)
        MOV    @R005, M034      ; Indirect transfer of data in the register
                                ; (0.68H) <– (0.34H)
```

As shown in Figure 8-10, when this instruction is executed, the contents of data memory address M034 is transferred to data memory location 68H.

When the MOV @r, m instruction is executed when MPE = 1, the contents of the data memory address specified by m is transferred to the column address pointed to by the row address @r being pointed to by the memory pointer.

In this case, the indirect address specified by @r becomes the value used for the bank and row address data memory pointer (above example uses row address 6). The column address is the value in the general register address specified by r (above example uses column address 8).

Therefore the address in the above example is 68H.

This example is different from Example 2 in **Section 8.5.3** when MPE = 0 for the following reasons: In this example, the data memory row address pointer is used to point to the indirect address bank and row address specified by @r. (In Example 2 in**Section 8.5.3**, the indirect address bank and row address are the same as m.)

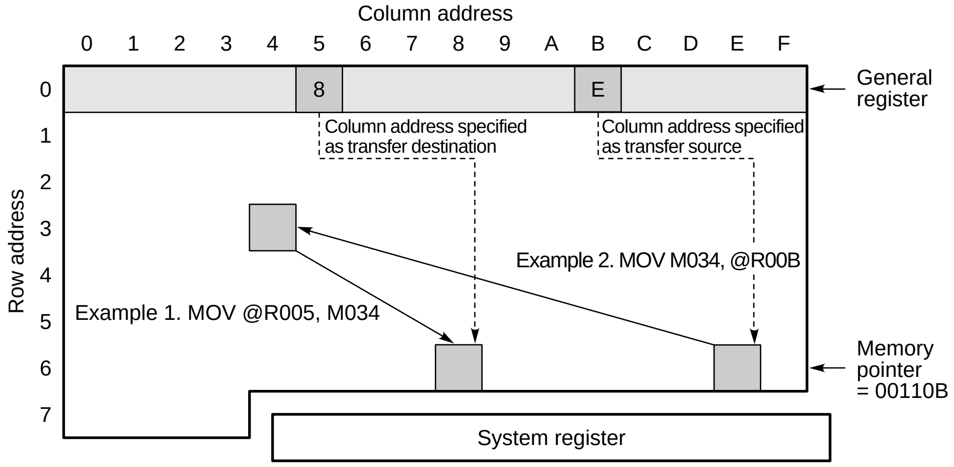
By setting MPE = 1, diagonal indirect data transfer can be performed using the general register.

Example 2. Execution of MOV m, @r when the general register is in row address 0

```
R00B    MEM     0.0BH
M034    MEM     0.34H
        MOV     MPL, #0110B      ; MP <– 6 (row address setting of @r)
        MOV     MPH, #1000B      ; MPE <– 1, bank <– 0
        MOV     R00B, #0EH       ; R00B <– 0EH (column address setting of @r)
        MOV     M034, @R00B      ; Indirect transfer of data in the register
                                 ; (0.34H) <– (0.6EH)
```

As shown in Figure 8-10, when this instruction is executed, the data stored in address 6EH is transferred to data memory location M034.

Figure 8-10. Example of Operation When IXE = 0 and MPE = 1


Column address
0 1 2 3 4 5 6 7 8 9 A B C D E F
Row address
0 1 2 3 4 5 6 7
8
E
General register
Column address specified as transfer destination
Column address specified as transfer source
Example 2. MOV M034, @R00B
Example 1. MOV @R005, M034
Memory pointer = 00110B
System register

Addresses in Example 1
MOV @R005, M034

	Bank	Row address	Column address
Data memory address M	0000	011	0100
General register address R	0000	000	0101
Indirect transfer address @R	0000	110	1000
	Contents of MP		Contents of R

Addresses in Example 2
MOV M034, @R00B

	Bank	Row address	Column address
Data memory address M	0000	011	0100
General register address R	0000	000	1011
Indirect transfer address @R	0000	110	1110
	Contents of MP		Contents of R

8.5.5 IXE = 1 and MPE = 0 (Index Modification)

As shown in Figure 8-8, when a data memory manipulation instruction is executed, any bank or address in data memory specified by m can be modified using the index register.

When indirect data transfer using the general register (MOV @r, m or MOV m, @r) is executed, the indirect transfer bank and address specified by @r can be modified using the index register.

Address modification is done by performing an OR operation on the data memory address and the index register. The data memory manipulation instruction being executed manipulates data in the memory location pointed to by the result of the operation (called the real address).

Examples are shown below.

Example 1. Execution of ADD r, m when the general register is in row address 0

```
R003    MEM    0.03H
M061    MEM    0.61H
        MOV    IXL, #0010B              ; IX <– 00000010010B
        MOV    IXM, #0001B              ;
        MOV    IXH, #0000B              ; MPE <– 0
        OR     PSW, #.DF.IXE AND 0FH    ; IXE <– 1
        ADD    R003, M061               ; (0.03H) <– (0.03H) + (0.73H)
```

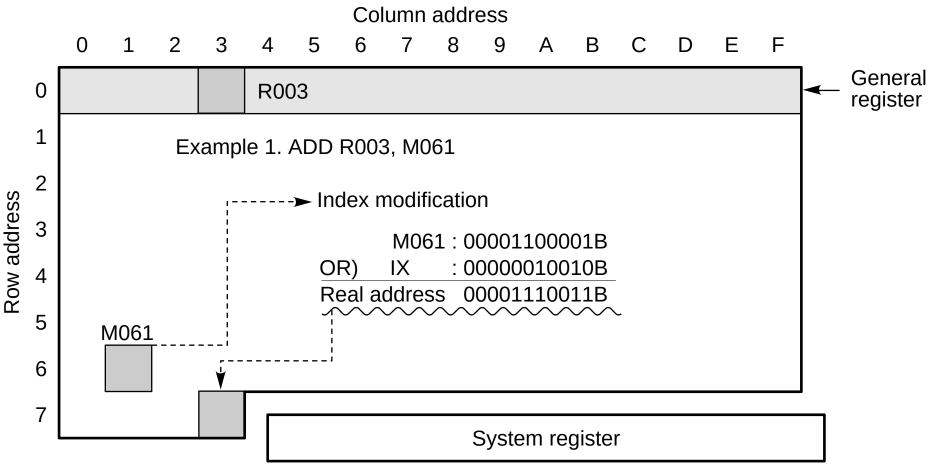
As shown in Figure 8-11, when this instruction is executed, the value in data memory address 73H (real address) and the value in general register address R003 (address location 03H) are added together and the result is stored in general register address R003.

When the ADD r, m instruction is executed, the data memory address specified by m (address 61H in above example) is index modified.

Modification is done by performing an OR operation on data memory location M061 (address 61H, binary 00001100001B) and the index register (00000010010B in the above example). The result of the operation (00001110011B) is used as a real address (address location 73H) by the instruction being executed.

As compared to when IXE = 0 (Examples in **Section 8.5.3**), in this example the data memory address being directly specified by m is modified by performing an OR operation on m and the index register.

Figure 8-11. Example of Operation When IXE = 1 and MPE = 0



Addresses in Example 1

ADD R003, M061

		Bank	Row address	Column address
Data memory address M		0000	110	0001
General register address R		0000	000	0011
Index modification	M061	0000	110	0001
		BANK	m	
	IX	0000	001	0010
		IXH	IXM	IXL
	Real address (OR operation)	0000	111	0011

Instruction is executed using this address.

Example 2. Indirect data transfer using the general register (execution of MOV @r, m)

```
R005    MEM     0.05H
M034    MEM     0.34H
        MOV     IXL, #0001B              ; Column address <– 5 (ORing 4 with 1)
        MOV     IXM, #0000B              ; Row address <– 3 (ORing 3 with 0)
        MOV     IXH, #0000B              ; MPE <– 0, bank <– 0 (ORing 0 with 0)
        OR      PSW, #.DF.IXE AND 0FH    ; IXE <– 1
        MOV     R005, #8                 ; R005 <– 8 (column address setting of @r)
        MOV     @R005, M034              ; Indirect data transfer using the register
                                         ; (0.38H) <– (0.35H)
```

As shown in Figure 8-12, when this instruction is executed, the contents of data memory address 35H is transferred to data memory location 38H.

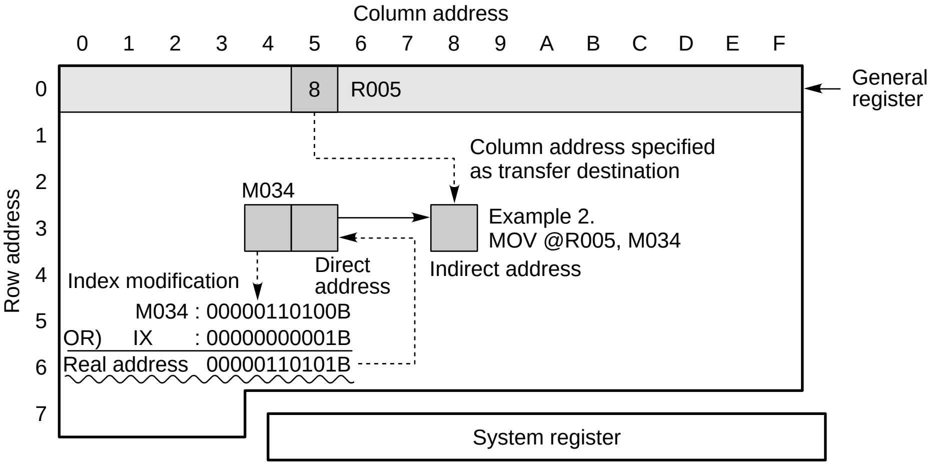
When the MOV @r, m instruction is executed when IXE = 1, the data memory address specified by m (direct address) is modified using the contents of the index register. The bank and row address of the indirect address specified by @r are also modified using the index register.

The bank, row address, and column address specified by m (direct address) are all modified, and the bank and row address specified by @r (indirect address) are modified.

Therefore, in the above example the direct address m is 35H and the indirect address @r is 38H.

This example is different from Example 3 in **Section 8.5.3** when IXE = 0 for the following reasons: In this example, the bank, row address and column address of the direct address specified by m are modified using the index register. The general register is transferred to the address specified by the column address of the modified data memory address and the same row address. (In Example 3 in **Section 8.5.3**, the direct address is not modified.)

Figure 8-12. Example of Operation When IXE = 1 and MPE = 0 (Indirect Data Transfer Using the General Register)



Example 3. Clearing data memory from 00H to 6FH (setting to 0)

```
M000    MEM     0.00H
        MOV     IXL, #0                   ; IX <– 0
        MOV     IXM, #0                   ;
        MOV     IXH, #0                   ; MPE <– 0
LOOP:
        OR      PSW, #.DF.IXE AND 0FH     ; IXE <– 1
        MOV     M000, #0                  ; Set data memory specified by IX to 0
        INC     IX                        ; IX <– IX + 1
        AND     PSW, #1110B               ; IXE <– 0, IXE is set to 0 so that address
                                          ; 7FH is unchanged even if modified by IX.
        SKE     IXM, #7                   ; Row address 7?
        BR      LOOP                      ; If not 7 then LOOP (row address is not
                                          ; cleared)
```

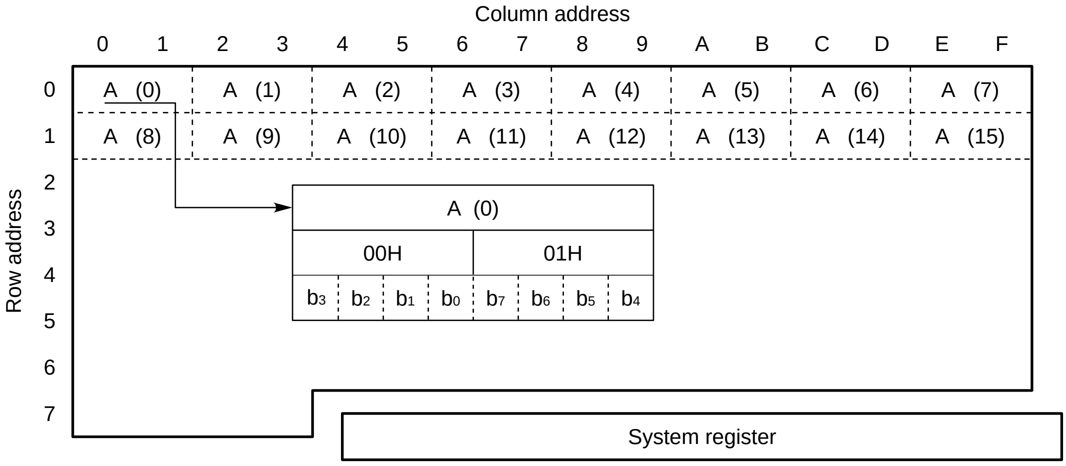
4. Processing an array

To perform operation "A (N) = A (N) + 4 (0 - N - 15)" for one-dimensional array A (N) consisting of 8-bit elements as shown in Figure 8-13, execute the following instructions:

```
M000    MEM     0.00H
M001    MEM     0.01H
        MOV     IXH, #0
        MOV     IXM, #N SHR3              ; Sets the offset of the row address.
        MOV     IXL, #N SHL 1 AND 0FH     ; Sets the offset of the column address.
        OR      PSW, #.DF.IXE AND 0FH     ; IXE <– 1
        ADD     M000, #4
        ADDC    M001, #0                  ; A(N) <– A(N) + 4
```

In the above example, one element consists of 8 bits. Therefore, the value resulting from shifting N one bit to the left (value two times that of N) is set in the index register.

Figure 8-13. Example of Operation When IXE = 1 and MPE = 0 (Array Processing)



8.6 GENERAL REGISTER POINTER (RP)

General register pointer (RP) is used to specify the bank and the row address of the general register. (See **Chapter 7**.)

8.7 PROGRAM STATUS WORD (PSWORD)

8.7.1 Program Status Word Configuration

Figure 8-14 shows the configuration of the program status word.

Figure 8-14. Program Status Word Configuration

Address	7EH				7FH			
Name	(RP)			Program status word (PSWORD)				
Symbol	RPL				PSW			
Bit	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Data				BCD	CMP	CY	Z	IXE
Initial value when reset	0				0			

As shown in Figure 8-14, the program status word consists of five bits; the least significant bit of system register address 7EH (RPL) and all four bits of system register address 7FH (PSW).

The program status word is divided into the following 1-bit flags: Binary coded decimal flag (BCD), compare flag (CMP), carry flag (CY), zero flag (Z), and the index enable flag (IXE).

All the bits are cleared to 0 when it is reset or when it is saved in the interrupt stack register.

8.7.2 Functions of the Program Status Word

The flags of the program status word are used for setting conditions for arithmetic/logical operations and data transfer instructions and for reflecting the status of operation results. Figure 8-15 shows an outline of the functions of the program status word.

Figure 8-15. Outline of Functions of the Program Status Word

Address	7EH				7FH			
Bit	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Symbol	RPL				PSW			
Flag				BCD	CMP	CY	Z	IXE

Flag	Function of PSWORD
IXE	Used to specify that index modification be performed on the data memory address used when a data memory manipulation instruction is executed. 0: Index modification disabled. 1: Index modification enabled.
Z	Set when the result of an arithmetic operation is 0. 0: Indicates that the result of the arithmetic operation is a value other than 0. 1: Indicates that the result of the arithmetic operation is 0.
CY	Set when there is a carry in the result of an addition operation or a borrow in the result of a subtraction operation. 0: Indicates there was no carry or borrow. 1: Indicates there was a carry or borrow.
CMP	Used to specify that the result of an arithmetic operation not be stored in data memory or the general register but just be reflected in the CY and Z flags. 0: Results of arithmetic operations are stored. 1: Results of arithmetic operations are not stored.
BCD	Used to specify how arithmetic operations are performed. 0: Arithmetic operations are performed in 4-bit binary. 1: Arithmetic operations are performed in BCD.

8.7.3 Index Enable Flag (IXE)

The IXE flag is used to specify whether index modification is to be performed on the data memory address. For details, see **Section 8.5.1**.

8.7.4 Zero Flag (Z) and Compare Flag (CMP)

The Z flag indicates whether the result of an arithmetic operation is 0. The CMP flag is used to specify that the result of an arithmetic operation not be stored in data memory or the general register.

Table 8-2 shows how the CMP flag affects the setting and resetting of the Z flag.

Table 8-2. The Zero (Z) Flag and the Compare Flag (CMP)

Conditions	CMP = 0	CMP = 1
When arithmetic operation results in 0	Z <– 1	Z flag does not change
When arithmetic operation results in a non-zero value	Z <– 0	Z <– 0

The Z flag and the CMP flag are used for comparing values in the general register and data memory. The Z flag is only affected by arithmetic operations. The CMP flag is only affected by bit evaluation.

Example of 12-bit data comparison

```
 ; Is the 12-bit data stored in M001, M002, and M003 equal to 456H?
 CMP456:

             SET2    CMP, Z
             SUB     M001, #4    ; The data stored in M001, M002, and M003 is not damaged.
             SUB     M002, #5
             SUB     M003, #6
             ; CLR1  CMP
             SKT1    Z           ; Resets CMP automatically when the bit test instruction is executed.
             BR      DIFFER      ; • 456H
             BR      AGREE       ; = 456H
```

8.7.5 Carry Flag (CY)

The CY flag shows whether there is a carry in the result of an addition operation or a borrow in the result of a subtraction operation.

The CY flag is set (CY = 1) when there is a carry or borrow in the result and reset (CY = 0) when there is no carry or borrow in the result.

When the RORC r instruction (contents in the general register pointed to by r is shifted right one bit) is executed, the following occurs: the value in the CY flag just before execution of the instruction is shifted to the most significant bit of the general register and the least significant bit is shifted to the CY flag.

The CY flag is also useful for when the user wants to skip the next instruction when there is a carry or borrow in the result of an operation.

The CY flag is only affected by arithmetic operations and rotations. It is not affected by the CMP flag.

8.7.6 Binary-Coded Decimal Flag (BCD)

The BCD flag is used to specify BCD operations.

When the BCD flag is set (BCD = 1), all arithmetic operations will be performed in BCD. When the BCD flag is reset (BCD = 0), arithmetic operations are performed in 4-bit binary.

The BCD flag does not affect logical operations, bit evaluation, comparison evaluations or rotations.

8.7.7 Warning Concerning Use of Arithmetic Operations on the Program Status Word

When performing arithmetic operations (addition and subtraction) on the program status word (PSWORD), the following point should be kept in mind.

When an arithmetic operation is performed on the program status word and the result is stored in the program status word, the result may be different than anticipated.

Below is an example.

Example

```
MOV     PSW, #0001B
ADD     PSW, #1111B
```

When the above instructions are executed, a carry is generated which should cause bit 2 (CY flag) of PSW to be set. However, the result of the operation (0000B) is stored in PSW, meaning that CY does not get set.

8.8 WARNINGS CONCERNING USE OF THE SYSTEM REGISTER

8.8.1 Reserved Words for Use with the System Register

Because the system register is allocated in data memory, it can be used in any of the data memory manipulation instructions. As shown in Example 1 (using a 17K Series Assembler), because a data memory address can not be directly specified in an instruction operand, it needs to be defined as a symbol beforehand.

The system register is data memory, but has specialized functions which make it different from general-purpose data memory. Because of this, the system register is used by defining it beforehand with symbols (used as reserved words) in the assembler (AS17K).

Reserved words for use with the system register are allocated in address locations 74H to 7FH. They are defined by the symbols (AR3, AR2, ..., PSW) shown in **Figure 8-2**.

As shown in Example 2, if these reserved words are used, it is not necessary to define symbols.

For information concerning reserved words, see **Chapter 21**.

Example 1.

```
                MOV    34H, #0101B      ; Using a data memory address like 34H or 76H will
                MOV    76H, #1010B      ; cause an error in  the assembler.
        M037    MEM    0.37H            ; Addresses in general data memory need to be
                MOV    M037, #0101B     ; defined as symbols using the MEM pseudo
                                        ; instruction.
```

2.

```
                MOV    AR1, #1010B      ; By using the reserved word AR1 (address 76H),
                                        ; there is no need to define the address as a symbol.
                                        ; Reserved word AR1 is defined in a device with
                                        ; the pseudo instruction “AR1 MEM 0.76H”).
```

Assembler AS17K has the below flag symbol handling instructions defined internally as macros.

SETn	: Set a flag to 1
CLRn	: Reset a flag to 0
SKTn	: Skip when all flags are 1
SKFn	: Skip when all flags are 0
NOTn	: Invert a flag
INITFLG	: Initialize a flag

By using these macro instructions, data memory can be handled as flags as shown below in Example 3.

The functions of the program status word and the memory pointer enable flag are defined in bit units (flag units) and each bit has a reserved word (BCD, CMP, CY, Z, IXE, or MPE) defined for it.

If these flag reserved words are used, the incorporated macro instructions can be used as shown in Example 4.

Example 3.

```
F0003   FLG     0.00.3          ; Flag symbol definition
        SET1    F0003           ; Incorporated macro
        ┌ Expanded macro ─────────────────────────────────
        │ OR .MF.F0003 SHR 4, #.DF.F0003 AND 0FH
        │                       ; Set bit 3 of address 00H of BANK0
        └─────────────────────────────────────────────────
```

4.

```
        SET1 BCD                ; Incorporated macro
        ┌ Expanded macro ─────────────────────────────────
        │ OR .MF.BCD SHR 4, #.DF.BCD AND 0FH
        │                       ; Set the BCD flag
        │                       ; BCD is defined as “BCD FLG 0.7EH.0”
        └─────────────────────────────────────────────────

        CLR2 Z, CY              ; Identical address flag
        ┌ Expanded macro ─────────────────────────────────
        │ AND .MF.Z SHR 4, #.DF. (NOT (Z OR CY) AND 0FH)
        └─────────────────────────────────────────────────

        CLR2 Z, BCD             ; Different address flag
        ┌ Expanded macro ─────────────────────────────────
        │ AND .MF.Z SHR 4, #.DF. (NOT Z AND 0FH)
        │ AND .MF.BCD SHR 4, #.DF. (NOT BCD AND 0FH)
        └─────────────────────────────────────────────────
```

8.8.2 Notes on Handling of System Register Addresses Fixed at 0

In dealing with system register area fixed at 0 (see **Figure 8-2**), there are a few points for which caution should be taken with regard to device, emulator and assembler operation.

Items (1), (2) and (3) explain these points.

(1) Device operation

Trying to write data to an address fixed at 0 will not change the value (0) at that address. Any attempt to read an address fixed at 0 will result in the value 0 being read.

(2) When using a 17K series in-circuit emulator (IE-17K or IE-17K-ET)

An error will be generated if a write instruction attempts to write the value 1 to an address fixed at 0. The type of instructions shown below will cause the in-circuit emulator to generate an error.

Example 1.

```
MOV     BANK, #0100B    ; Attempts to write the value 1 to bit 2 (an address fixed at 0).
```

2.

```
MOV     IXL, #1111B     ;
MOV     IXM, #0111B     ;
ADD     IXL, #1         ;
ADDC    IXM, #0         ;
```

However, when all valid bits are set to 1 as shown in Example 2, executing the instructions INC AR or INC IX will not cause an error to be generated by the in-circuit emulator. This is because when all valid bits of the address register and index register are set to 1, executing the INC instruction causes all bits to be set to 0.

The only time the in-circuit emulator will not generate an error when an attempt is made to write the value 1 to the data fixed at 0 is when the address being written to is in the address register.

(3) When using a 17K series assembler

No error is output when an attempt is made to write the value 1 to the data fixed at 0. The instruction shown in Example 1

```
MOV   BANK, #0100B
```

will not cause an assembler error. However, when the instruction is executed in the in-circuit emulator, an error is generated.

The assembler does not generate an error because it does not judge the correspondence between symbols (including reserved words) and data memory addresses used by data memory manipulation instructions.

The assembler generates an error when the value n in the incorporated macro BANKn is a value greater than or equal to 1.

This is because the assembler judges that incorporated macro instructions other than BANK0 cannot be used in the µPD17145 sub-series.

[MEMO]

CHAPTER 9 REGISTER FILE (RF)

The register file is a register used mainly for specifying conditions for peripheral hardware.

9.1 REGISTER FILE CONFIGURATION

9.1.1 Configuration of the Register File

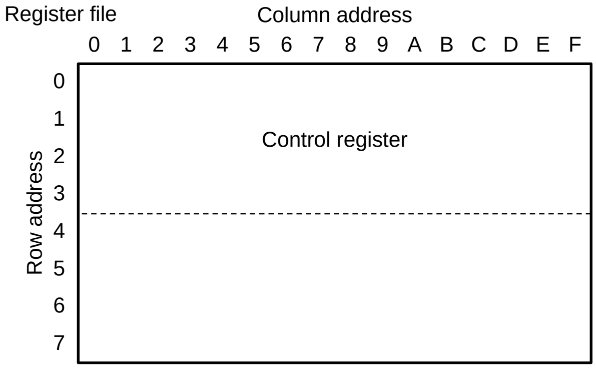
Figure 9-1 shows the configuration of the register file.

As shown in Figure 9-1, the register file is a register consisting of 128 nibbles (128 x 4 bits).

In the same way as with data memory, the register file is divided into addresses in units of four bits. It has a total of 128 nibbles specified in row addresses from 0H to 7H and column addresses from 0H to 0FH.

Address locations 00H to 3FH define an area called the control register.

Figure 9-1. Register File Configuration



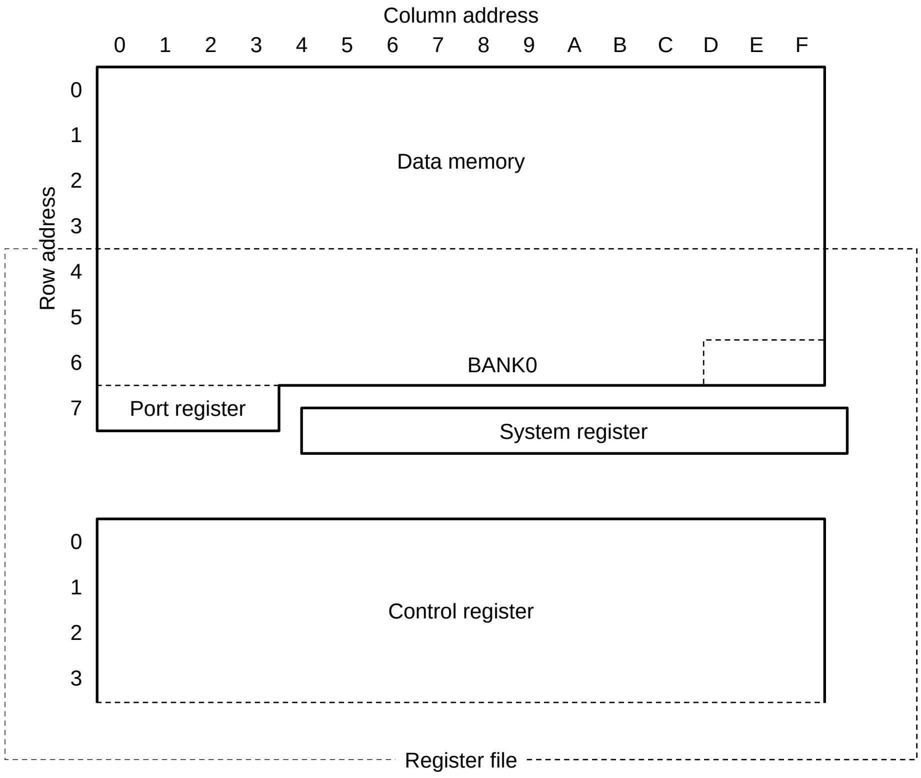
9.1.2 Relationship between the Register File and Data Memory

Figure 9-2 shows the relationship between the register file and data memory.

As shown in Figure 9-2, the register file overlaps with data memory at addresses 40H to 7FH.

This means that a program identifies register file addresses 40H to 7FH also as data memory addresses 40H to 7FH.

Figure 9-2. Relationship Between the Register File and Data Memory



9.2 FUNCTIONS OF THE REGISTER FILE

9.2.1 Functions of the Register File

The register file is a collection of registers in which peripheral hardware conditions are set with the PEEK instruction or POKE instruction.

The register used to control the peripheral hardware is located at addresses 00H to 3FH. This area is called the control register.

Addresses 40H to 7FH of the register file constitute normal data memory. Thus, not only the MOV instruction, but also the PEEK and POKE instructions, can be used to enable this part to perform read and write operations.

9.2.2 Control Register Functions

The peripheral hardware whose conditions can be controlled by control registers is listed below.

For details concerning peripheral hardware and the control register, see the section for the peripheral hardware concerned.

- Ports
- 8-bit timer counter (TM0, TM1)
- Basic interval timer (BTM)
- A/D converter
- Serial interface (SIO)
- Interrupt function
- Stack pointer (SP)

9.2.3 Register File Manipulation Instructions

Reading and writing data from and to the register file is done using the window register (WR: address 78H) located in the system register.

Reading and writing of data is performed using the following dedicated instructions:

PEEK WR, rf: Read the data in the address specified by rf and put it into WR.
POKE rf, WR: Write the data in WR into the address specified by rf.

Below is an example of register file operation.

Example

```
RF02    MEM     0.82H           ; Symbol definition
RF1F    MEM     0.9FH           ; Register file addresses 00H to 3FH must be defined
                                ; with symbols
RF53    MEM     0.53H           ; as BANK0 addresses 80H to BFH.
RF6D    MEM     0.6DH           ; See Section 9.4 for details.
RF70    MEM     1.70H           ;
RF71    MEM     1.71H           ;
        ; BANK0
<1>     PEEK    WR, RF02        ;
<2>     POKE    RF1F, WR        ;
<3>     PEEK    WR, RF53        ;
<4>     POKE    RF6D, WR        ;
```

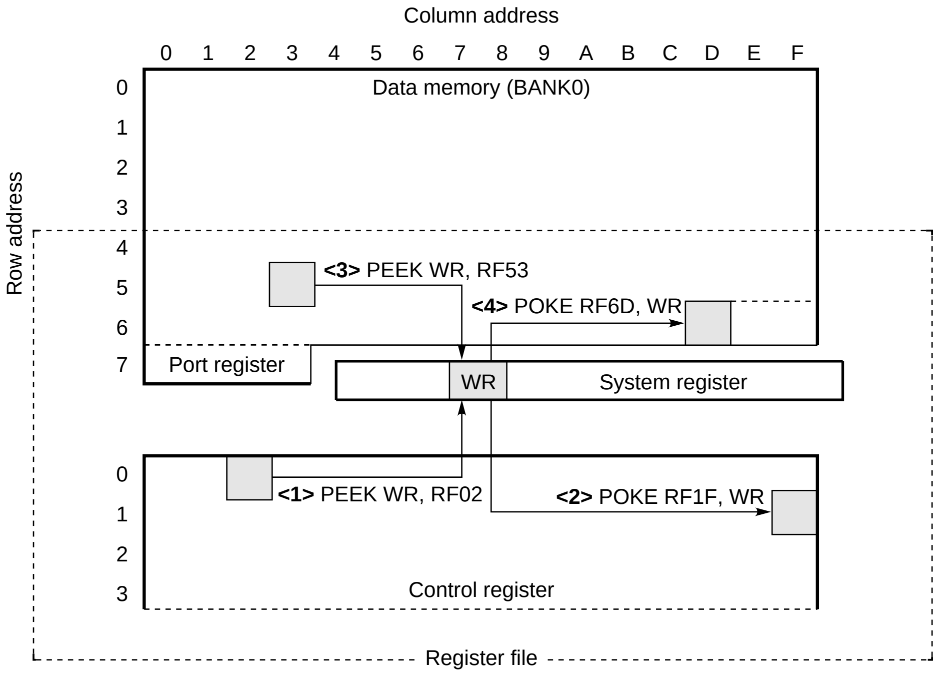
Figure 9-3 shows an example of register file operation.

As shown in Figure 9-3, reading and writing of data to and from the control register (address locations 00H to 3FH) is performed using the PEEK WR, rf and POKE rf, WR instructions. Data within the control register specified using rf can be read from and written to the control register, only by using these instructions with the window register.

The fact that the register file overlaps with data memory in addresses 40H to 7FH has the following effect: When a PEEK WR, rf or POKE rf, WR instruction is executed, the effect is the same as if they were being executed on the data memory address specified by rf.

Addresses 40H to 7FH of the register file can be operated by normal memory manipulation instructions.

Figure 9-3. Accessing the Register File Using the PEEK and POKE Instructions



9.3 CONTROL REGISTER

9.3.1 Control Register Configuration

The control register consists of 64 nibbles (64 x 4 bits) allocated in register file address locations 00H to 3FH.

However, only 25 nibbles are actually used. The remaining 39 nibbles are allocated for registers which have not yet been implemented. Data should not be read from or written to this area.

There are two types of registers, both of which occupy one nibble of memory. One type is read/write (R/W), and the other is read-only (R).

Note that within the read/write (R/W) flags, there exists a flag that will always be read as 0.

The following read/write (R/W) flags are those flags which will always be read as 0:

- WDTRES (RF: 03H, bit 3)
- WDTEN (RF: 03H, bit 0)
- TM0RES (RF: 11H, bit 2)
- TM1RES (RF: 12H, bit 2)
- BTMRES (RF: 13H, bit 2)
- ADCSTRT (RF: 20H, bit 0)

Within the four bits of data in a nibble, there are bits which are fixed at 0 and will therefore always be read as 0. These bits remain fixed at 0 even when an attempt is made to write to them.

Attempting to read data in the unused register address area (39 nibbles) will yield unpredictable values. In addition, attempting to write to this area has no effect.

See **Figure 21-2** for the configuration of the control registers.

9.4 WARNINGS CONCERNING USE OF THE REGISTER FILE

9.4.1 Warnings Concerning Operation of the Control Register (Read-Only and Unused Registers)

It is necessary to take note of the following warnings concerning device operation and use of the 17K Series assembler and in-circuit emulator (IE-17K or IE-17K-ET) with regard to the read-only (R) and unused registers in the control register (register file address locations 00H to 3FH).

(1) Device operation

Writing to a read-only register has no effect.
Attempting to read data from an address in the unused data area will yield an unpredictable value.
Attempting to write to an address in the unused data area has no effect.

(2) During use of the assembler

An error will be generated if an attempt is made to write to a read-only register.
An error will also be generated if an attempt is made to read from or write to an address in the unused data area.

(3) During use of the in-circuit emulator (IE-17K or IE-17K-ET) (operation during patch processing and similar operations)

Attempting to write to a read-only register has no effect. Also note that no error is generated.
Attempting to read data from an address in the unused data area will yield an unpredictable value.
Attempting to write to an address in the unused data area has no effect. No errors are generated.

9.4.2 Register File Symbol Definitions and Reserved Words

Attempting to use a numerical value in a 17K Series assembler to specify a register file address in the rf operand of the PEEK WR, rf or POKE rf, WR instruction will cause an error to be generated.

Therefore, as shown in Example 1, register file addresses need to be defined beforehand as symbols.

Example 1. Case which causes an error to be generated

```
PEEK    WR, 02H          ;
POKE    21H, WR          ;
```

Case in which no error is generated

```
RF71    MEM     0.71H    ; Symbol definition
        PEEK    WR, RF71 ;
```

Caution should especially be taken with regard to the following point:

- When using a symbol to define the control register as an address in data memory, it needs to be defined as addresses 80H to BFH of BANK0.

Since the control register is manipulated using the window register, any attempt to manipulate the control register other than by using the PEEK and POKE commands needs to cause an error to be generated in the assembler.

However, note that any address in the area of the register file overlapping with data memory (address locations 40H to 7FH) can be defined as a symbol in the same manner as with normal data memory.

An example is given below.

Example 2.

```
RF71    MEM     0.71H       ; Address in register file overlapping with data memory    *
RF02    MEM     0.82H       ; Control register

        PEEK    WR, RF71    ; RF71 becomes address 71H in data memory.
        PEEK    WR, RF02    ; RF02 becomes address 02H in the control register.
```

The assembler (AS17K) has the below flag symbol handling instructions defined internally as macros.

SETn	: Set a flag to 1
CLRn	: Reset a flag to 0
SKTn	: Skip when all flags are 1
SKFn	: Skip when all flags are 0
NOTn	: Invert a flag
INITFLG	: Initialize a flag (in units of 4 bits)

By using these incorporated macro instructions, the contents of the register file can be manipulated one bit at a time.

Due to the fact that most of control register consists of 1-bit flags, the assembler has reserved words (predefined symbols) for use with these flags.

However, note that the control register has no reserved word for the stack pointer for its use as a flag. The reserved word used for the stack pointer is the reserved word SP, for its use as data memory. For this reason, none of the flag manipulation instructions using reserved words can be used with the stack pointer.

[MEMO]

CHAPTER 10 DATA BUFFER (DBF)

The data buffer consists of four nibbles allocated in addresses 0CH to 0FH in BANK0.

The data buffer acts as a data storage area for the CPU peripheral hardware (address register, serial interface, timer 0, timer 1, and A/D converter) through use of the GET and PUT instructions. It also acts as data storage used for receiving and transferring data. By using the MOVT DBF, @AR instruction, fixed data in program memory can be read into the data buffer.

10.1 DATA BUFFER CONFIGURATION

Figure 10-1 shows the allocation of the data buffer in data memory.

As shown in Figure 10-1, the data buffer is allocated in address locations 0CH to 0FH in data memory and consists of four nibbles (4 x 4 bits).

Figure 10-1. Allocation of the Data Buffer

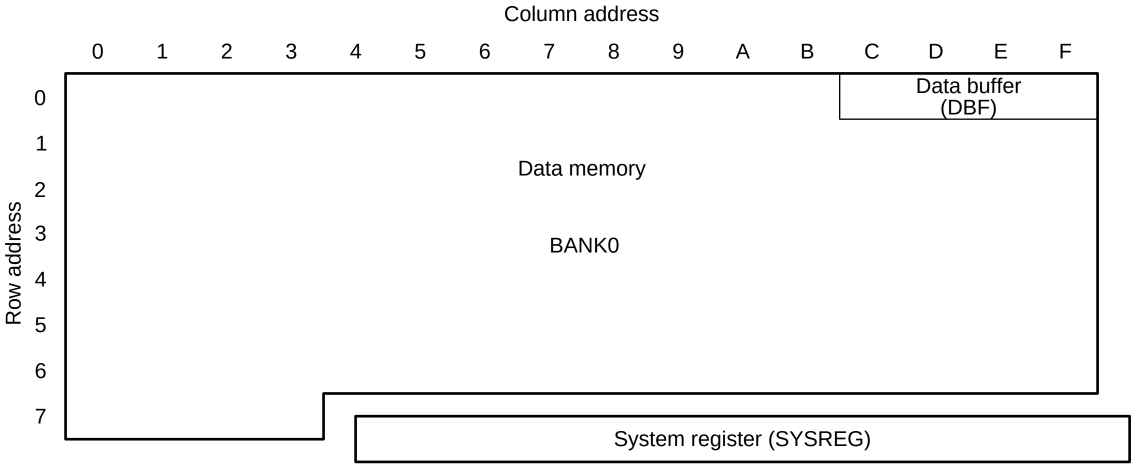


Figure 10-2 shows the configuration of the data buffer. As shown in Figure 10-2, the data buffer is made up of sixteen bits with its least significant bit in bit 0 of address 0FH and its most significant bit in bit 3 of address 0CH.

Figure 10-2. Data Buffer Configuration

Data memory BANK0	Address	0CH				0DH				0EH				0FH			
	Bit	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Data buffer	Bit	b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
	Symbol	DBF3				DBF2				DBF1				DBF0			
	Data	^ M S B v				Data											^ L S B v

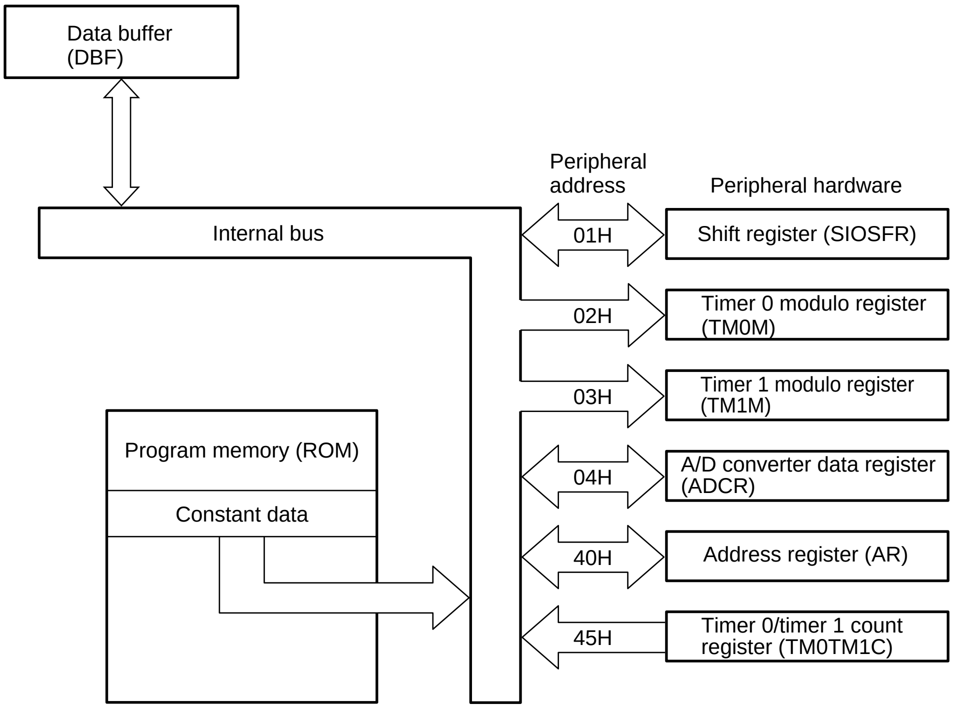
Because the data buffer is allocated in data memory, it can be used in any of the data memory manipulation instructions. Upon reset, all 16 bits are undefined.

10.2 FUNCTIONS OF THE DATA BUFFER

The data buffer has two separate functions.

The data buffer is used for data transfer with peripheral hardware. The data buffer is also used for reading constant data in program memory. Figure 10-3 shows the relationship between the data buffer and peripheral hardware.

Figure 10-3. Relationship between the Data Buffer and Peripheral Hardware



10.2.1 Data Buffer and Peripheral Hardware

Table 10-1 shows data transfer with peripheral hardware using the data buffer.

Each unit of peripheral hardware has an individual address (called its peripheral address). By using this peripheral address and instructions GET and PUT, data can be transferred between each unit of peripheral hardware and the data buffer.

Instruction	Operation
GET DBF, p	Read the data in the peripheral hardware address specified by p into the data buffer.
PUT p, DBF	Write the data in the data buffer to the peripheral hardware address specified by p.

There are three types of peripheral hardware units: read/write, write-only and read-only.

The following describes what happens when a GET instruction is used with write-only hardware and when a PUT instruction is used with read-only hardware.

- Reading (GET) from write-only peripheral hardware will yield an unpredictable value.
- Writing (PUT) to read-only peripheral hardware has no effect (same as a NOP instruction).

Table 10-1. Peripheral Hardware

(1) Peripheral hardware with input/output in 8-bit units

Peripheral address	Name	Peripheral hardware	Direction of data		Actual bit length
			PUT	GET	
01H	SIOSFR	SIO shift register	o	o	8 bits
02H	TM0M	Timer 0 modulo register	o	x	8 bits
03H	TM1M	Timer 1 modulo register	o	x	8 bits
04H	ADCR	A/D converter data register	o	o	8 bits

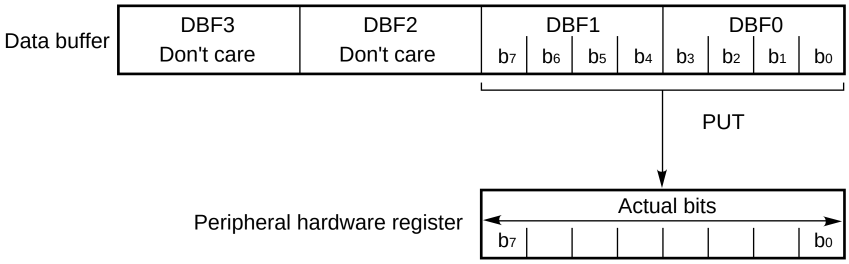
(2) Peripheral hardware with input/output in 16-bit units

Peripheral address	Name	Peripheral hardware	Direction of data		Actual bit length
			PUT	GET	
40H	AR	Address register	o	o	10 bits (μPD17145) 11 bits (μPD17147) 12 bits (μPD17149 μPD17P149)
45H	TM0TM1C	Timer 0/timer 1 count register	x	o	16 bits

10.2.2 Data Transfer with Peripheral Hardware

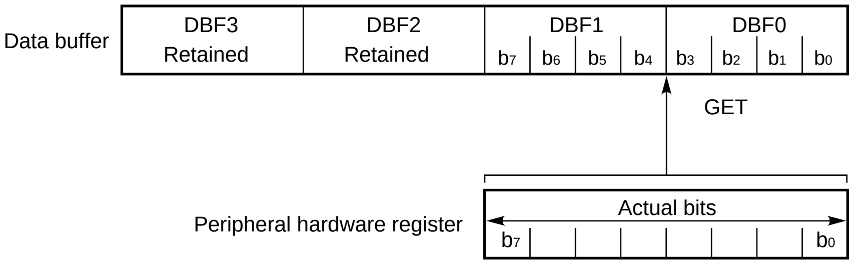
Data can be transferred between the data buffer and peripheral hardware in 8- or 16-bit units. Instruction cycle for a single PUT or GET instruction is the same regardless of whether eight or sixteen bits are being transferred.

Example 1. PUT instruction (when the actual bits in peripheral hardware are the eight bits from 7 to 0)



When only eight bits of data are being written from the data buffer, the upper eight bits of the data buffer (DBF3, DBF2) are irrelevant.

Example 2. GET instruction (when the actual bits in peripheral hardware are the eight bits from 7 to 0)



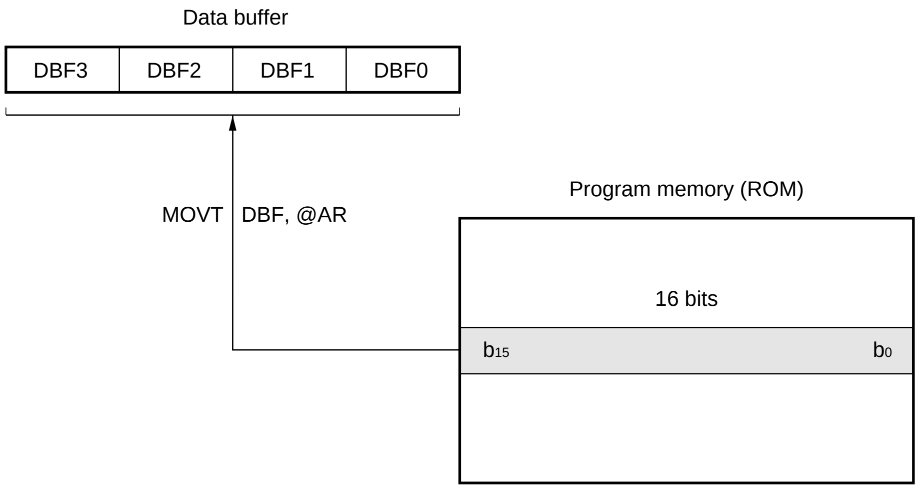
When only eight bits of data are being read into the data buffer, the values in the upper eight bits of the data buffer (DBF3, DBF2) remain unchanged.

10.2.3 Table Reference

By using the MOVT instruction, constant data in program memory (ROM) can be read into the data buffer.

The MOVT instruction is explained below.

MOVT DBF, @AR: The contents of the program memory being pointed to by the address register (AR) is read into the data buffer (DBF).



[MEMO]

CHAPTER 11 ALU BLOCK

The ALU is used for performing arithmetic operations, logical operations, bit evaluations, comparison evaluations, and rotations on 4-bit data.

11.1 ALU BLOCK CONFIGURATION

Figure 11-1 shows the configuration of the ALU block.

As shown in Figure 11-1, the ALU block consists of the main 4-bit data processor, temporary registers A and B, the status flip-flop for controlling the status of the ALU, and the decimal conversion circuit for use during arithmetic operations in BCD.

As shown in Figure 11-1, the status flip-flop consists of the following flags: Zero flag flip-flop, carry flag flip-flop, compare flag flip-flop, and the BCD flag flip-flop.

Each flag in the status flip-flop corresponds directly to a flag in the program status word (PSWORD: addresses 7EH, 7FH) located in the system register. The flags in the program status word are the following: Zero flag (Z), carry flag (CY), compare flag (CMP), and the BCD flag (BCD).

11.2 FUNCTIONS OF THE ALU BLOCK

Arithmetic operations, logical operations, bit evaluations, comparison evaluations, and rotations are performed using the instructions in the ALU block. Table 11-1 lists each arithmetic/logical instruction, evaluation instruction, and rotation instruction.

By using the instructions listed in Table 11-1, 4-bit arithmetic/logical operations, evaluations and rotations can be performed in a single instruction. Arithmetic operations in decimal can also be performed in a single instruction.

11.2.1 Functions of the ALU

The arithmetic operations consist of addition and subtraction. Arithmetic operations can be performed on the contents of the general register and data memory or on immediate data and the contents of data memory. Operations in binary are performed on four bits of data and operations in decimal are performed on one place (BCD operation).

Logical operations include ANDing, ORing, and XORing. Their operands can be general register contents and data memory contents, or data memory contents and immediate data.

Bit evaluation is used to determine whether bits in 4-bit data in data memory are 0 or 1.

Comparison evaluation is used to compare contents of data memory with immediate data. It is used to determine whether one value is equal to or greater than the other, less than the other, or if both values are equal or not equal.

Rotation is used to shift 4-bit data in the general register one bit in the direction of its least significant bit (rotation to the right).

Table 11-1. List of ALU Instructions (1/2)

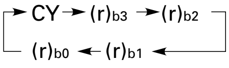
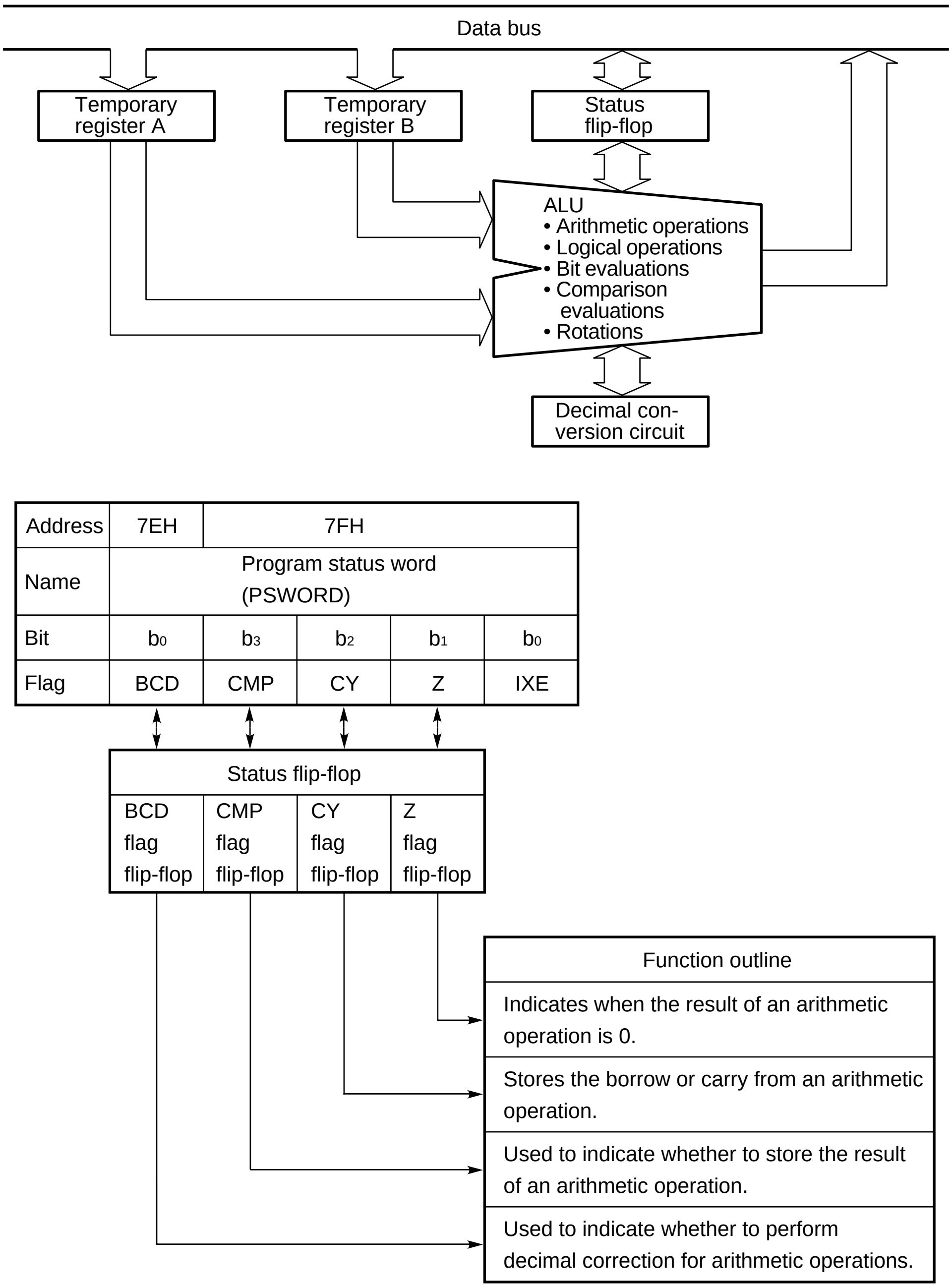
ALU function	Instruction		Operation	Explanation
Arithmetic operations	Addition	ADD r, m	(r) <– (r) + (m)	Adds contents of general register and data memory. Result is stored in general register.
		ADD m, #n4	(m) <– (m) + n4	Adds immediate data to contents of data memory. Result is stored in data memory.
		ADDC r, m	(r) <– (r) + (m) + CY	Adds contents of general register, data memory and CY flag. Result is stored in general register.
		ADDC m, #n4	(m) <– (m) + n4 + CY	Adds immediate data, contents of data memory and CY flag. Result is stored in data memory.
	Subtraction	SUB r, m	(r) <– (r) – (m)	Subtracts contents of data memory from contents of general register. Result is stored in general register.
		SUB m, #n4	(m) <– (m) – n4	Subtracts immediate data from data memory. Result is stored in data memory.
		SUBC r, m	(r) <– (r) – (m) – CY	Subtracts contents of data memory and CY flag from contents of general register. Result is stored in general register.
		SUBC m, #n4	(m) <– (m) – n4 – CY	Subtracts immediate data and CY flag from data memory. Result is stored in data memory.
Logical operations	Logical OR	OR r, m	(r) <– (r) ∨ (m)	OR operation is performed on contents of general register and data memory. Result is stored in general register.
		OR m, #n4	(m) <– (m) ∨ n4	OR operation is performed on immediate data and contents of data memory. Result is stored in data memory.
	Logical AND	AND r, m	(r) <– (r) ∧ (m)	AND operation is performed on contents of general register and data memory. Result is stored ingeneral register.
		AND m, #n4	(m) <– (m) ∧ n4	AND operation is performed on immediate data and contents of data memory. Result is stored in data memory.
	Logical XOR	XOR r, m	(r) <– (r) ⊻ (m)	XOR operation is performed on contents of general register and data memory. Result is stored ingeneral register.
		XOR m, #n4	(m) <– (m) ⊻ n4	XOR operation is performed on immediate data and contents of data memory. Result is stored in data memory.
Bit evaluation	True	SKT m, #n	CMP <– 0, if(m) ∧ n = n, then skip	Skips next instruction if all bits in data memory specified by n are TRUE (1). Result is not stored.
	False	SKF m, #n	CMP <– 0, if(m) ∧ n = 0, then skip	Skips next instruction if all bits in data memory specified by n are FALSE (0). Result is not stored.
Comparison evaluation	Equal	SKE m, #n4	(m) – n4, skip if zero	Skips next instruction if immediate data equals contents of data memory. Result is not stored.
	Not equal	SKNE m, #n4	(m) – n4, skip if not zero	Skips next instruction if immediate data is not equal to contents of data memory. Result is not stored.
	•	SKGE m, #n4	(m) – n4, skip if not borrow	Skips next instruction if contents of data memory is greater than or equal to immediate data. Result is not stored.
	<	SKLT m, #n4	(m) – n4, skip if borrow	Skips next instruction if contents of data memory is less than immediate data. Result is not stored.
Rotation	Rotate to the right	RORC r		Rotate contents of the general register along with the CY flag to the right. Result is stored in general register.

Table 11-1. List of ALU Instructions (2/2)

ALU function	Operation depending on the program status word (PSWORD)					
	Value in BCD flag	Value in CMP flag	Operation	CY flag	Z flag	Modification by IXE = 1
Arithmetic operation	0	0	Store result of binary operation	Set (1) when carry or borrow is generated, otherwise reset (0).	Set (1) when result of operation is 0000B, otherwise reset (0).	Yes
	0	1	Do not store result of binary operation		Status maintained when result of operation is 0000B, otherwise reset (0).	
	1	0	Store result of BCD operation		Set (1) when result of operation is 0000B, otherwise reset (0).	
	1	1	Do not store result of BCD operation		Status maintained when result of operation is 0000B, otherwise reset (0).	
Logical operations	Don’t care (maintained)	Don’t care (maintained)	No change	Don't care (maintained)	Don’t care (maintained)	Yes
Bit evaluation	Don’t care (maintained)	Reset	No change	Don't care (maintained)	Don’t care (maintained)	Yes
Comparison evaluation	Don’t care (maintained)	Don’t care (maintained)	No change	Don't care (maintained)	Don’t care (maintained)	Yes
Rotation	Don’t care (maintained)	Don’t care (maintained)	No change	Value in b_0 of the general register	Don’t care (maintained)	Yes

Figure 11-1. Configuration of the ALU



Address	7EH	7FH			
Name	Program status word (PSWORD)				
Bit	b_0	b_3	b_2	b_1	b_0
Flag	BCD	CMP	CY	Z	IXE

Status flip-flop			
BCD flag flip-flop	CMP flag flip-flop	CY flag flip-flop	Z flag flip-flop

Function outline
Indicates when the result of an arithmetic operation is 0.
Stores the borrow or carry from an arithmetic operation.
Used to indicate whether to store the result of an arithmetic operation.
Used to indicate whether to perform decimal correction for arithmetic operations.

11.2.2 Functions of Temporary Registers A and B

Temporary registers A and B are needed for processing of 4-bit data. These registers are used for temporary storage of the first and second data operands of an instruction.

11.2.3 Functions of the Status Flip-flop

The status flip-flop is used for controlling operation of the ALU and for storing data which has been processed. Each flag in the status flip-flop corresponds directly to a flag in the program status word (PSWORD) located in the system register. This means that when a flag in the system register is manipulated it is the same as manipulating a flag in the status flip-flop. Each flag in the program status word is described below.

(1) Z flag

This flag is set (1) when the result of an arithmetic operation is 0000B, otherwise it is reset (0). However, as described below, depending on the status of the CMP flag, the conditions which cause this flag to be set (1) can be changed.

(i) When CMP = 0

Z flag is set (1) when the result of an arithmetic operation is 0000B, otherwise it is reset (0).

(ii) When CMP = 1

The previous state of the Z flag is maintained when the result of an arithmetic operation is 0000B, otherwise it is reset (0). Only affected by arithmetic operations.

(2) CY flag

This flag is set (1) when a carry or borrow is generated in the result of an arithmetic operation, otherwise it is reset (0).

When an arithmetic operation is being performed using a carry or borrow, the operation is performed using the CY flag as the least significant bit.

When a rotation (RORC instruction) is performed, the contents of the CY flag becomes the most significant bit (bit b_3) of the general register and the least significant bit of the general register is stored in the CY flag.

Only affected by arithmetic operations and rotations.

(3) CMP flag

When the CMP flag is set (1), the result of an arithmetic operation is not stored in either the general register or data memory.

When the bit evaluation instruction is performed, the CMP flag is reset (0).

The CMP flag does not affect comparison evaluations, logical operations, or rotations.

(4) BCD flag

When the BCD flag is set (1), decimal correction is performed for all arithmetic operations. When the flag is reset (0), all operations are performed in 4-bit binary.

The BCD flag does not affect logical operations, bit evaluations, comparison evaluations, or rotations.

These flags can also be set through direct manipulation of the values in the program status word (PSWORD). When the flags in the program status word are manipulated, the corresponding flag in the status flip-flop is also manipulated.

11.2.4 Performing Operations in 4-Bit Binary

When the BCD flag is set to 0, arithmetic operations are performed in 4-bit binary.

11.2.5 Performing Operations in BCD

When the BCD flag is set to 1, decimal correction is performed for arithmetic operations performed in 4-bit binary. Table 11-2 shows the differences in the results of operations performed in 4-bit binary and in BCD. When the result of an addition after decimal correction is equal to or greater than 20, or the result of a subtraction after decimal correction is outside of the range –10 to +9, a value of 1010B (0AH) or higher is stored as the result (shaded area in Table 11-2).

Table 11-2. Results of Arithmetic Operations Performed in 4-Bit Binary and BCD

Operation result	Addition in 4-bit binary		Addition in BCD	
	CY	Operation result	CY	Operation result
0	0	0000	0	0000
1	0	0001	0	0001
2	0	0010	0	0010
3	0	0011	0	0011
4	0	0100	0	0100
5	0	0101	0	0101
6	0	0110	0	0110
7	0	0111	0	0111
8	0	1000	0	1000
9	0	1001	0	1001
10	0	1010	1	0000
11	0	1011	1	0001
12	0	1100	1	0010
13	0	1101	1	0011
14	0	1110	1	0100
15	0	1111	1	0101
16	1	0000	1	0110
17	1	0001	1	0111
18	1	0010	1	1000
19	1	0011	1	1001
20	1	0100	1	1110
21	1	0101	1	1111
22	1	0110	1	1100
23	1	0111	1	1101
24	1	1000	1	1110
25	1	1001	1	1111
26	1	1010	1	1100
27	1	1011	1	1101
28	1	1100	1	1010
29	1	1101	1	1011
30	1	1110	1	1100
31	1	1111	1	1101

Operation result	Subtraction in 4-bit binary		Subtraction in BCD	
	CY	Operation result	CY	Operation result
0	0	0000	0	0000
1	0	0001	0	0001
2	0	0010	0	0010
3	0	0011	0	0011
4	0	0100	0	0100
5	0	0101	0	0101
6	0	0110	0	0110
7	0	0111	0	0111
8	0	1000	0	1000
9	0	1001	0	1001
10	0	1010	1	1100
11	0	1011	1	1101
12	0	1100	1	1110
13	0	1101	1	1111
14	0	1110	1	1100
15	0	1111	1	1101
–16	1	0000	1	1110
–15	1	0001	1	1111
–14	1	0010	1	1100
–13	1	0011	1	1101
–12	1	0100	1	1110
–11	1	0101	1	1111
–10	1	0110	1	0000
–9	1	0111	1	0001
–8	1	1000	1	0010
–7	1	1001	1	0011
–6	1	1010	1	0100
–5	1	1011	1	0101
–4	1	1100	1	0110
–3	1	1101	1	0111
–2	1	1110	1	1000
–1	1	1111	1	1001

11.2.6 Performing Operations in the ALU Block

When arithmetic operations, logical operations, bit evaluations, comparison evaluations or rotations in a program are executed, the first data operand is stored in temporary register A and the second data operand is stored in temporary register B.

The first data operand is four bits of data used to specify the contents of an address in the general register or data memory. The second data operand is four bits of data used to either specify the contents of an address in data memory or to be used as an immediate value. For example, in the instruction

```
ADD r, m
     |  └──── Second data operand
     └──────        First data operand
```

the first data operand, r, is used to specify the contents of an address in the general register. The second data operand, m, is used to specify the contents of an address in data memory. In the instruction

ADD m, #n4

the first data operand, m, is used to specify an address in data memory. The second operand, #n4, is immediate data. In the rotation instruction

RORC r

only the first data operand, r (used to specify the contents of an address in the general register) is used.

Next, using the data stored in temporary registers A and B, the ALU executes the operation specified by the instruction (arithmetic operation, logical operation, bit evaluation, comparison evaluation, or rotation). When the instruction being executed is an arithmetic operation, logical operation, or rotation, the data processed by the ALU is stored in the location specified by the first data operand (general register address or data memory address) and the operation terminates. When the instruction being executed is a bit evaluation or comparison evaluation, the result processed by the ALU is used to determine whether or not to skip the next instruction (whether to treat next instruction as an NOP instruction) and the operation terminates.

Caution should be taken with regard to the following points:

(1) Arithmetic operations are affected by the CMP and BCD flags in the program status word.

(2) Logical operations are not affected by the CMP or BCD flag in the program status word. Logical operations do not affect the Z or CY flags.

(3) Bit evaluation causes the CMP flag in the program status word to be reset.

(4) When an arithmetic operation, logical operation, bit evaluation, comparison evaluation, or rotation is being executed and the IXE flag in the program status word is set (1), address modification is performed using the index register.

11.3 ARITHMETIC OPERATIONS (ADDITION AND SUBTRACTION IN 4-BIT BINARY AND BCD)

As shown in Table 11-3, arithmetic operations consist of addition, subtraction, addition with carry, and subtraction with borrow. These instructions are ADD, ADDC, SUB, and SUBC.

The ADD, ADDC, SUB, and SUBC instructions are further divided into addition and subtraction of the general register and data memory and addition and subtraction of data memory and immediate data. When the operands r and m are used, addition or subtraction is performed using the general register and data memory. When the operands m and #n4 are used, addition or subtraction is performed using data memory and immediate data.

Arithmetic operations are affected by the status flip-flop and the program status word (PSWORD) in the system register. The BCD flag in the program status word (PSWORD) is used to specify whether arithmetic operations are to be performed in 4-bit binary or in BCD. The CMP flag is used to specify whether or not the results of arithmetic operations are to be stored.

Sections 11.3.1 to **11.3.4** explain the relationship between each command and the program status word (PSWORD).

Table 11-3. Types of Arithmetic Operations

Arithmetic operation	Addition	Without carry ADD	General register and data memory	ADD r, m
			Data memory and immediate data	ADD m, #n4
		With carry ADDC	General register and data memory	ADDC r, m
			Data memory and immediate data	ADDC m, #n4
	Subtraction	Without borrow SUB	General register and data memory	SUB r, m
			Data memory and immediate data	SUB m, #n4
		With borrow SUBC	General register and data memory	SUBC r, m
			Data memory and immediate data	SUBC m, #n4

11.3.1 Addition and Subtraction When CMP = 0 and BCD = 0

Addition and subtraction are performed in 4-bit binary and the result is stored in the general register or data memory.

When the result of the operation is greater than 1111B (carry generated) or less than 0000B (borrow generated), the CY flag is set (1); otherwise it is reset (0).

When the result of the operation is 0000B, the Z flag is set (1) regardless of whether there is carry or borrow; otherwise it is reset (0).

11.3.2 Addition and Subtraction When CMP = 1 and BCD = 0

Addition and subtraction are performed in 4-bit binary.

However, because the CMP flag is set (1), the result of the operation is not stored in either the general register or data memory.

When there is a carry or borrow in the result of the operation, the CY flag is set (1); otherwise it is reset (0).

When the result of the operation is 0000B, the previous state of the Z flag is maintained; otherwise it is reset (0).

11.3.3 Addition and Subtraction When CMP = 0 and BCD = 1

BCD operations are performed.

The result of the operation is stored in the general register or data memory. When the result of the operation is greater than 1001B (9D) or less than 0000B (0D), the CY flag is set (1), otherwise it is reset (0).

When the result of the operation is 0000B (0D), the Z flag is set (1), otherwise it is reset (0).

Operations in BCD are performed by first computing the result in binary and then by using the decimal conversion circuit to convert the result to decimal. For information concerning the binary to decimal conversion, see Table 11-2 in **Section 11.2.5**.

In order for operations in BCD to be performed properly, note the following:

(1) Result of an addition must be in the range 0D to 19D.

(2) Result of a subtraction must be in the range 0D to 9D, or in the range –10D to –1D.

The following shows which value is considered the CY flag in the range 0D to 19D (shown in 4-bit binary):

$\underbrace{0}_{\text{CY}}$, 0000B to $\underbrace{1}_{\text{CY}}$, 0011B

The following shows which value is considered the CY flag in the range –10D to –1D (shown in 4-bit binary):

$\underbrace{1}_{\text{CY}}$, 0110B to $\underbrace{1}_{\text{CY}}$, 1111B

When operations in BCD are performed outside of the limits of (1) and (2) stated above, the CY flag is set (1) and the result of operation is output as a value greater than or equal to 1010B (0AH).

11.3.4 Addition and Subtraction When CMP = 1 and BCD = 1

BCD operations are performed.

The result is not stored in either the general register or data memory.

In other words, the operations specified by CMP = 1 and BCD = 1 are both performed at the same time.

Example

```
MOV   RPL,     #0001B; Sets the BCD flag (BCD = 1).
MOV   PSW,     #1010B; Sets the CMP and Z flag (CMP = 1, Z = 1) and resets the
                     ; CY flag (CY = 0).
SUB   M1,      #0001B; <1>
SUBC  M2,      #0010B; <2>
SUBC  M3,      #0011B; <3>
```

By executing the instructions in steps numbered **<1>** , **<2>**, and **<3>**, the twelve bits in memory locations M1, M2, and M3 and the immediate data (321) can be compared in decimal.

11.3.5 Warnings Concerning Use of Arithmetic Operations

When performing arithmetic operations with the program status word (PSWORD), caution should be taken with regard to the result of the operation being stored in the program status word.

Normally, the CY and Z flags in the program status word are set (1) or reset (0) according to the result of the arithmetic operation being executed. However, when an arithmetic operation is performed on the program status word itself, the result is stored in the program status word. This means that there is no way to determine if there is a carry or borrow in the result of the operation nor if the result of the operation is zero.

However, when the CMP flag is set (1), results of arithmetic operations are not stored. Therefore, even in the above case, the CY and Z flags will be properly set (1) or reset (0) according to the result of the operation.

11.4 LOGICAL OPERATIONS

As shown in Table 11-4, logical operations consist of logical OR, logical AND, and logical XOR. Accordingly, the logical operation instructions are OR, AND, and XOR.

The OR, AND, and XOR instructions can be performed on either the general register and data memory, or on data memory and immediate data. The operands of these instructions are specified in the same way as for arithmetic operations (“r, m” or “m, #n4”).

Logical operations are not affected by the BCD or CMP flags in the program status word (PSWORD). Logical operations do not affect the CY and Z flags. However, when the index enable flag (IXE) is set (1), index modification is performed using the index register.

Table 11-4. Logical Operations

Logical operation	Logical OR	General register and data memory	OR r, m
		Data memory and immediate data	OR m, #n4
	Logical AND	General register and data memory	AND r, m
		Data memory and immediate data	AND m, #n4
	Logical XOR	General register and data memory	XOR r, m
		Data memory and immediate data	XOR m, #n4

Table 11-5. Table of True Values for Logical Operations

Logical AND C = A AND B			Logical OR C = A OR B			Logical XOR C = A XOR B		
A	B	C	A	B	C	A	B	C
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

11.5 BIT EVALUATIONS

As shown in Table 11-6, there are both TRUE (1) and FALSE (0) bit evaluation instructions. The SKT instruction skips the next instruction when a bit is evaluated as TRUE (1) and the SKF instruction skips the next instruction when a bit is evaluated as FALSE (0).

The SKT and SKF instructions can only be used with data memory.

Bit evaluations are not affected by the BCD flag in the program status word (PSWORD) and bit evaluations do not cause either the CY or Z flags in the program status word (PSWORD) to be set. However, when an SKT or SKF instruction is executed, the CMP flag is reset (0). When the index enable flag (IXE) is set (1), index modification is performed using the index register. For information concerning index modification using the index register, see **Chapter 8**.

Sections 11.5.1 and **11.5.2** explain TRUE (1) and FALSE (0) bit evaluations.

Table 11-6. Bit Evaluation Instructions

Bit evaluation	TRUE (1) bit evaluation SKT m, #n
	FALSE (0) bit evaluation SKF m, #n

11.5.1 TRUE (1) Bit Evaluation

The TRUE (1) bit evaluation instruction (SKT m, #n) is used to determine whether or not the bits specified by n in the four bits of data memory m are TRUE (1). When all bits specified by n are TRUE (1), this instruction causes the next instruction to be skipped.

Example

```
MOV    M1,  #1011B
SKT    M1,  #1011B ; <1>
BR     A
BR     B
SKT    M1,  #1101B ; <2>
BR     C
BR     D
```

In this example, bits 3, 1, and 0 of data memory M1 are evaluated in step number **<1>**. Because all the bits are TRUE (1), the program branches to B. In step number **<2>**, bits 3, 2, and 0 of data memory M1 are evaluated. Since bit 2 of data memory M1 is FALSE (0), the program branches to C.

11.5.2 FALSE (0) Bit Evaluation

The FALSE (0) bit evaluation instruction (SKF m, #n) is used to determine whether or not the bits specified by n in the four bits of data memory m are FALSE (0). When all bits specified by n are FALSE (0), this instruction causes the next instruction to be skipped.

Example

```
MOV    M1,   #1001B ;
SKF    M1,   #0110B ; <1>
BR     A            ;
BR     B            ;
SKF    M1,   #1110B ; <2>
BR     C            ;
BR     D            ;
```

In this example, bits 2 and 1 of data memory M1 are evaluated in step number **<1>**. Because both bits are FALSE (0), the program branches to B. In step number **<2>**, bits 3, 2, and 1 of data memory M1 are evaluated. Since bit 3 of data memory M1 is TRUE (1), the program branches to C.

11.6 COMPARISON EVALUATIONS

As shown in Table 11-7, there are comparison evaluation instructions for determining if one value is “equal to”, “not equal to”, “greater than or equal to”, or “less than” another.

The SKE instruction is used to determine if two values are equal. The SKNE instruction is used to determine two values are not equal. The SKGE instruction is used to determine if one value is greater than or equal to another and the SKLT instruction is used to determine if one value is less than another.

The SKE, SKNE, SKGE, and SKLT instructions perform comparisons between a value in data memory and immediate data. In order to compare values in the general register and data memory, a subtraction instruction is performed according to the values in the CMP and Z flags in the program status word (PSWORD). For more information concerning comparison of the general register and data memory, see **Section 11.3**.

Comparison evaluations are not affected by the BCD or CMP flags in the program status word (PSWORD) and comparison evaluations do not cause either the CY or Z flags in the program status word (PSWORD) to be set.

Sections 11.6.1 to **11.6.4** explain the “equal”, “not equal”, “greater than or equal”, and “less than” comparison evaluations.

Table 11-7. Comparison Evaluation Instructions

Comparison evaluation	Equal SKE m, #n4
	Not equal SKNE m, #n4
	Greater than or equal SKGE m, #n4
	Less than SKLT m, #n4

11.6.1 "Equal" Evaluation

The "equal" evaluation instruction (SKE m, #n4) is used to determine if immediate data and the contents of a location in data memory are equal.

This instruction causes the next instruction to be skipped when the immediate data and the contents of data memory are equal.

Example

```
MOV   M1,  #1010B
SKE   M1,  #1010B ; <1>
BR    A
BR    B
;
SKE   M1,  #1000B ; <2>
BR    C
BR    D
```

In this example, because the contents of data memory M1 and immediate data 1010B in step number **<1>** are equal, the program branches to B. In step number **<2>**, because the contents of data memory M1 and immediate data 1000B are not equal, the program branches to C.

11.6.2 "Not Equal" Evaluation

The "not equal" evaluation instruction (SKNE m, #n4) is used to determine if immediate data and the contents of a location in data memory are not equal.

This instruction causes the next instruction to be skipped when the immediate data and the contents of data memory are not equal.

Example

```
MOV   M1,  #1010B
SKNE  M1,  #1000B ; <1>
BR    A
BR    B
;
SKNE  M1,  #1010B ; <2>
BR    C
BR    D
```

In this example, because the contents of data memory M1 and immediate data 1000B in step number **<1>** are not equal, the program branches to B. In step number **<2>**, because the contents of data memory M1 and immediate data 1010B are equal, the program branches to C.

11.6.3 “Greater Than or Equal” Evaluation

The “greater than or equal” evaluation instruction (SKGE m, #n4) is used to determine if the contents of a location in data memory is a value greater than or equal to the value of the immediate data operand. If the value in data memory is greater than or equal to that of the immediate data, this instruction causes the next instruction to be skipped.

Example

```
MOV    M1,   #1000B
SKGE   M1,   #0111B; <1>
BR     A
BR     B
;
SKGE   M1,   #1000B; <2>
BR     C
BR     D
;
SKGE   M1,   #1001B; <3>
BR     E
BR     F
```

In this example, the program will first branch to B since the value in data memory is larger than that of the immediate data **<1>**. Next it will branch to D since the value in data memory is equal to that of the immediate data **<2>**. Last it will branch to E since the value in data memory is less than that of the immediate data **<3>**.

11.6.4 “Less Than” Evaluation

The “less than” evaluation instruction (SKLT m, #n4) is used to determine if the contents of a location in data memory is a value less than that of the immediate data operand. If the value in data memory is less than that of the immediate data, this instruction causes the next instruction to be skipped.

Example

```
MOV    M1,   #1000B
SKLT   M1,   #1001B; <1>
BR     A
BR     B
;
SKLT   M1,   #1000B; <2>
BR     C
BR     D
;
SKLT   M1,   #0111B; <3>
BR     E
BR     F
```

In this example, the program will first branch to B since the value in data memory is less than that of the immediate data **<1>**. Next it will branch to C since the value in data memory is equal to that of the immediate data **<2>**. Last it will branch to E since the value in data memory is greater than that of the immediate data **<3>**.

11.7 ROTATIONS

There are rotation instructions for rotation to the right and for rotation to the left.

The RORC instruction is used for rotation to the right.

The RORC instruction can only be used with the general register.

Rotation using the RORC instruction is not affected by the BCD or CMP flags in the program status word (PSWORD) and does not affect the Z flag in the program status word (PSWORD).

Rotation to the left is performed by using the addition instruction ADDC.

Sections 11.7.1 and **11.7.2** explain rotation.

11.7.1 Rotation to the Right

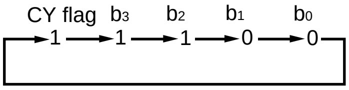
The instruction used for rotation to the right (RORC r) rotates the contents of the general register in the direction of its least significant bit.

When this instruction is executed, the contents of the CY flag becomes the most significant bit of the general register (bit 3) and the least significant bit (bit 0) of the general register is placed in the CY flag.

* **Example 1.**

```
MOV   PSW,   #0100B; Sets CY flag to 1.
MOV   R1,    #1100B
RORC  R1
```

When these instructions are executed, the following operation is performed.



Basically, when rotation to the right is performed, the following operation is executed:
CY flag –> b_3, b_3 –> b_2, b_2 –> b_1, b_1 –> b_0, b_0 –> CY flag.

2.

```
MOV   PSW,   #0000B; Resets CY flag to 0.
MOV   R1,    #1000B; Most significant bit
MOV   R2,    #0100B
MOV   R3,    #0010B; Least significant bit
RORC  R1
RORC  R2
RORC  R3
```

* The program code above rotates the 13 bits in CY, R1, R2, and R3 to the right.

11.7.2 Rotation to the Left

Rotation to the left is performed by using the addition instruction, “ADDC r, m”.

Example

```
MOV     PSW,    #0000B; Resets CY flag to 0.        *
MOV     R1,     #1000B; Most significant bit
MOV     R2,     #0100B
MOV     R3,     #0010B; Least significant bit
ADDC    R3, R3
ADDC    R2, R2
ADDC    R1, R1
SKF1    CY
OR      R3,     #0001B
```

The program code above rotates the 13 bits in CY, R1, R2, and R3 to the left.

[MEMO]

CHAPTER 12 PORTS

12.1 PORT 0A (P0A$_0$, P0A$_1$, P0A$_2$, P0A$_3$)

Port 0A is a 4-bit input/output port with an output latch. It is mapped into address 70H of BANK0 in data memory. The output format is CMOS push-pull output.

Input or output can be specified in 4-bit units. Input/output is specified by P0AGIO (bit 0 at address 2CH) in the register file.

When P0AGIO is 0, each pin of port 0A is used as input port. If a read instruction is executed for the port register, pin statuses are read.

When P0AGIO is 1, each pin of port 0A is used as output port and the contents written in the output latch are output to pins. If a read instruction is executed when pins are output ports, the contents of the output latch, rather than pin statuses, are fetched.

Port 0A contains a software controlled pull-up resistor. P0AGPU (bit 0 at address 0CH) of the register file is used to determine whether port 0A contains the pull-up resistor. When P0AGPU is 1, all 4-bit pins are pulled up. If P0AGPU is 0, the pull-up resistor is not contained.

At reset, P0AGIO and P0AGPU are set to 0 and all P0A pins become input ports without a pull-up resistor. The contents of the port output latch are 0.

Table 12-1. Writing into and Reading from the Port Register (0.70H)

P0AGIO	Pin input/output	BANK0 70H	
RF: 2CH, bit 0		Write	Read
0	Input	Possible	P0A pin status
1	Output	Write to the P0A latch	Data in P0A latch

12.2 PORT 0B ($P0B_0$, $P0B_1$, $P0B_2$, $P0B_3$)

Port 0B is a 4-bit input/output port with an output latch. It is mapped into address 71H of BANK0 in data memory. The output format is CMOS push-pull output.

Input or output can be specified in 4-bit units. Input/ output is specified by P0BGIO (bit 1 at address 2CH) in the register file.

When P0BGIO is 0, all pins of port 0B are used as input ports. If a read instruction is executed for the port register, pin statuses are read.

When P0BGIO is 1, all pins of port 0B are used as output ports. The contents written in the output latch are output to pins. If a read instruction is executed when pins are used as output ports, the contents of the output latch, rather than pin statuses, are fetched.

Port 0B contains a software controlled pull-up resistor. P0BGPU (bit 1 at address 0CH) is used to determine whether or not port 0B contains a pull-up resistor. When P0BGPU is 1, all 4-bit pins are pulled up. When P0BGPU is 0, a pull-up resistor is not contained.

At reset, P0BGIO and P0BGPU are 0 and all P0B pins are input ports without a pull-up resistor. The value of the port 0B output latch is 0.

Table 12-2. Writing into and Reading from the Port Register (0.71H)

P0BGIO	Pin input/output	BANK0 71H	
RF: 2CH, bit 1		Write	Read
0	Input	Possible Write to the P0B latch	P0B pin status
1	Output		Data in P0B latch

12.3 PORT 0C ($P0C_0/ADC_0$, $P0C_1/ADC_1$, $P0C_2/ADC_2$, $P0C_3/ADC_3$)

Port 0C is a 4-bit input/output port with an output latch. It is mapped into address 72H of BANK0 in data memory. The output format is CMOS push-pull output.

Input or output can be specified bit-by-bit. Input/output can be specified by P0CBIO0 to P0CBIO3 (address 1CH) in the register file.

If P0CBIOn is 0 (n = 0 to 3), the P0Cn pins are used as input port. If a data read instruction is executed for the port register, the pin statuses are read. If P0CBIOn is 1 (n = 0 to 3), the P0Cn pins are used as output port and the contents written in the output latch are output to pins. If a read instruction is executed when pins are used as output ports, the contents of the latch, rather than pin statuses, are fetched.

At reset, P0CBIO0 to P0CBIO3 are 0 and all P0C pins are input ports. The contents of the port output latch are 0.

Port 0C can also be used as an analog input to the A/D converter. P0C0IDI to P0C3IDI (1BH address) in the register file are used to switch the port and analog input pin.

If P0CnIDI is 0 (n = 0 to 3), the $P0C_n/ADC_n$ pin functions as a port. If P0CnIDI is 1 (n = 0 to 3), the $P0C_n/ADC_n$ pin functions as the analog input pin for the A/D converter. If any bit of P0CnIDI (n = 0 to 3) is 1, the $P0F_1/V_{REF}$ pin is used as the V_{REF} pin.

When using these pins for the analog input for the A/D converter, set P0CnIDI to 1 for the pins to which analog voltage is input, immediately after reset. This setting disables the port function for the pins. Then clear P0CBIOn (n = 0 to 3) to 0 to use the pins for input. Select which pins are used for analog input, using ADCCH0 and ADCCH1 (bits 0 and 1 at address 22H) in the register file.

At reset, P0CBIO0 to P0CBIO3, P0C0IDI to P0C3IDI, ADCCH0, and ADCCH1 are set to 0 and the P0C pins are used as input ports.

Table 12-3. Switching the Port and A/D Converter

(n = 0 to 3)

P0CnIDI RF: 1BH	P0CBIOn RF: 1CH	Function	BANK0 72H	
			Write	Read
0	0	Input port	Possible P0C latch	Pin status
	1	Output port	Possible P0C latch	Data in P0C latch
1	0	A/D converter analog input**Note 1**	Possible P0C latch	Data in P0C latch
	1	Output port and A/D converter analog input**Note 2**	Possible P0C latch	Data in P0C latch

Notes 1. Normal setting when the pins are used as A/D converter analog input pins.

2. Functions as an output port. The analog input voltage varies with the output from the port. To use pins as analog input pins, be sure to set P0CBIOn to 0. *

12.4 PORT 0D (P0D$_0$/$\overline{\text{SCK}}$, P0D$_1$/SO, P0D$_2$/SI, P0D$_3$/$\overline{\text{TM1OUT}}$)

Port 0D is a 4-bit input/output port with an output latch. It is mapped into address 73H of BANK0 in data memory. The output format is N-ch open-drain output.

Input or output can be specified bit-by-bit. Input/output is specified with P0DBIO0 to P0DBIO3 (address 2BH) in the register file.

If P0DBIOn is 0 (n = 0 to 3), the P0Dn pins are used as input port. Pin statuses are read if a data read instruction is executed for the port register. If P0DBIOn is 1, the P0Dn pins are used as output port and the value written in the output latch are output to pins. If a data read instruction is executed when pins are used as output ports, the output latch value, rather than pin statuses, is fetched.

Port 0D contains a software controlled pull-up resistor. P0DBPU0 to P0DBPU3 (address 0DH) of the register file are used to determine whether each bit of port 0D contains the pull-up resistor. When P0DBPUn is 1, the P0Dn pin is pulled up. If P0DBPUn is 0, the pull-up resistor is not contained.

At reset, P0DBIOn is set to 0 and all P0D pins become input ports. The contents of the port output latch become 0. The output latch contents remain unchanged even if P0DBIOn changes from 1 to 0.

Port 0D can also be used for serial interface input/output or timer 1 output. SIOEN (0BH bit 0) in the register file is used to switch ports (P0D$_0$ to P0D$_2$) to serial interface input/output ($\overline{\text{SCK}}$, SO, SI) and vice versa. TM1OSEL (bit 3 at address 0BH) in the register file is used to switch a port (P0D$_3$) to timer 1 output ($\overline{\text{TM1OUT}}$) and vice versa. If TM1OSEL = 1 is selected, 1 is output at timer 1 reset. This output is inverted every time a timer 1 count value matches the modulo register contents.

Table 12-4. Register File Contents and Pin Functions

(n = 0 to 3)

Register file value			Pin function			
TM1OSEL RF: 0BH Bit 3	SIOEN RF: 0BH Bit 0	P0DBIOn RF: 2BH Bit n	P0D$_0$/$\overline{\text{SCK}}$	P0D$_1$/SO	P0D$_2$/SI	P0D$_3$/$\overline{\text{TM1OUT}}$
0	0	0	Input port			
		1	Output port			
	1	0	$\overline{\text{SCK}}$	SO	SI	Input port
		1				Output port
1	0	0	Input port			$\overline{\text{TM1OUT}}$
		1	Output port			
	1	0	$\overline{\text{SCK}}$	SO	SI	
		1				

Table 12-5. Data Read from the Port Register (0.73H)

Port mode		Data read from the port register (0.73H)
Input port		Pin status
Output port		Data in output latch
$\overline{\text{SCK}}$	An internal clock is selected as a serial clock.	Data in output latch
	An external clock is selected as a serial clock.	Pin status
SI		Pin status
SO		Data in output latch
$\overline{\text{TM1OUT}}$		Data in output latch

12.5 PORT 0E (P0E$_0$, P0E$_1$, P0E$_2$, P0E$_3$)

Port 0E is a 4-bit input/output port with an output latch. It is mapped into address 6EH of BANK0 in data memory. The output format is N-ch open-drain output.

Input or output can be specified in units of four bits. Input/output is specified by P0EGIO (bit 2 at address 2CH) in the register file.

When P0EGIO is 0, each pin of port 0E is used as input port. If a read instruction is executed for the port register, pin statuses are read. When P0EGIO is 1, each pin of port 0E is used as output port and the contents written in the output latch are output to pins. If a read instruction is executed when pins are output ports, the contents of the output latch, rather than pin statuses, are fetched.

Port 0E contains a software controlled pull-up resistor. P0EGPU (bit 2 at address 0CH) of the register file in used to determine whether port 0E contains the pull-up resistor. When P0EGPU is 1, all 4-bit pins are pulled up. If P0EGPU is 0, the pull-up resistor is not contained.

At reset, P0EGIO is set to 0 and all P0E pins become input ports. The contents of the port output latch are 0.

Table 12-6. Writing into and Reading from the Port Register (0.6EH)

(n = 0 to 3)

P0EGIOn	Pin input/output	BANK0 6EH	
RF: 2CH, bit 2		Write	Read
0	Input	Possible	P0E pin status
1	Output	Write to the P0E latch	Data in P0E latch

12.6 PORT 0F (P0F$_0$/$\overline{\text{RLS}}$, P0F$_1$/V$_{\text{REF}}$)

Port 0F is a 2-bit input-dedicated port. It is mapped into address 6FH of BANK0 in data memory. Mask option can be used to specify whether each pin uses a built-in pull-up resistor.

If a pin of port 0F is used as an input port, a pin status is read in the two low-order bits of the port register when a data read instruction is executed for the port register (the two high-order bits are always 0). A data write instruction does not affect the port register.

The P0F$_0$/$\overline{\text{RLS}}$ pin can also be used for the input pin for the signal for releasing the standby mode.

The P0F$_1$/V$_{\text{REF}}$ pin is used as the V$_{\text{REF}}$ pin (reference voltage input pin for the A/D converter) when any bit of P0CnIDI (address 1BH in the register file, n = 0 to 3) is 1. If the P0F$_1$/V$_{\text{REF}}$ pin functions as the V$_{\text{REF}}$ pin, bit 1 at address 6FH is unpredictable and only bit 0 is valid when a data read instruction is executed for the port register.

12.7 PORT CONTROL REGISTER

12.7.1 Input/Output Switching by Group I/O

Ports which switch input/output in units of four bits are called group I/O. Port 0A, port 0B, and port 0E are used as group I/O. The register shown in the figure below is used for input/output switching.

Figure 12-1. Input/Output Switching by Group I/O

RF: 2CH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	P0EGIO	P0BGIO	P0AGIO
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

P0AGIO	Function
0	Sets port 0A to input mode.
1	Sets port 0A to output mode.

P0BGIO	Function
0	Sets port 0B to input mode.
1	Sets port 0B to output mode.

P0EGIO	Function
0	Sets port 0E to input mode.
1	Sets port 0E to output mode.

12.7.2 Input/Output Switching by Bit I/O

Ports which switch input/output bit-by-bit are called bit I/O. Port 0C and port 0D are used as bit I/O. The register shown in the figure below is used for input/output switching.

Figure 12-2. Port Control Registers for Bit I/O (1/2)

RF: 1CH

	Bit 3	Bit 2	Bit 1	Bit 0
	P0CBIO3	P0CBIO2	P0CBIO1	P0CBIO0
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

P0CBIO0	Function
0	Sets P0C$_0$ to input mode.
1	Sets P0C$_0$ to output mode.

P0CBIO1	Function
0	Sets P0C$_1$ to input mode.
1	Sets P0C$_1$ to output mode.

P0CBIO2	Function
0	Sets P0C$_2$ to input mode.
1	Sets P0C$_2$ to output mode

P0CBIO3	Function
0	Sets P0C$_3$ to input mode.
1	Sets P0C$_3$ to output mode.

Figure 12-2. Port Control Registers for Bit I/O (2/2)

RF: 2BH

	Bit 3	Bit 2	Bit 1	Bit 0
	P0DBIO3	P0DBIO2	P0DBIO1	P0DBIO0
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

P0DBIO0	Function
0	Sets P0D$_0$ to input mode.
1	Sets P0D$_0$ to output mode.

P0DBIO1	Function
0	Sets P0D$_1$ to input mode.
1	Sets P0D$_1$ to output mode.

P0DBIO2	Function
0	Sets P0D$_2$ to input mode.
1	Sets P0D$_2$ to output mode.

P0DBIO3	Function
0	Sets P0D$_3$ to input mode.
1	Sets P0D$_3$ to output mode.

12.7.3 Specifying the Incorporation of Pull-Up Resistors for Group Pull-Up Ports

Ports for which incorporating pull-up resistors can be specified in units of four bits are called group pull-up ports. Ports 0A, 0B and 0E are group pull-up ports.

P0AGPU, P0BGPU, and P0EGPU (RF: 0CH, bits 2, 1, and 0) are used for specifying the incorporation of pull-up resistors for the group pull-up ports.

Figure 12-3. Register for Specifying the Incorporation of Pull-Up Resistors for Group Pull-Up Ports

RF: 0CH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	P0EGPU	P0BGPU	P0AGPU
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

P0AGPU	Function
0	Does not contain pull-up resistor in port 0A.
1	Contains pull-up resistor in port 0A.

P0BGPU	Function
0	Does not contain pull-up resistor in port 0B.
1	Contains pull-up resistor in port 0B.

P0EGPU	Function
0	Does not contain pull-up resistor in port 0E.
1	Contains pull-up resistor in port 0E.

12.7.4 Specifying the Incorporation of Pull-Up Resistors for the Bit Pull-Up Port

Ports for which incorporating pull-up resistors can be specified in units of one bit are called bit pull-up ports. Port 0D is a bit pull-up port.

P0DBPU0 to P0DBPU3 (RF: 0DH) are used for specifying the incorporation of pull-up resistors for the bit pull-up port.

Figure 12-4. Register for Specifying the Incorporation of Pull-Up Resistors for the Bit Pull-Up Port

RF: 0DH

	Bit 3	Bit 2	Bit 1	Bit 0
	P0DBPU3	P0DBPU2	P0DBPU1	P0DBPU0
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

P0DBPU0	Function
0	Does not contain pull-up resistor in $P0D_0$.
1	Contains pull-up resistor in $P0D_0$.

P0DBPU1	Function
0	Does not contain pull-up resistor in port $P0D_1$.
1	Contains pull-up resistor in $P0D_1$.

P0DBPU2	Function
0	Does not contain pull-up resistor in $P0D_2$.
1	Contains pull-up resistor in $P0D_2$.

P0DBPU3	Function
0	Does not conatain pull-up resistor in $P0D_3$.
1	Contains pull-up resistor in $P0D_3$.

CHAPTER 13 PERIPHERAL HARDWARES

13.1 8-BIT TIMER COUNTER (TM0, TM1)

The µPD17149's 8-bit timer counter has two channel timers: timer 0 (TM0) and timer 1 (TM1).

By using timer 0 count up signal as timer 1 count pulses, these two 8-bit timers can be used as a one-channel 16-bit timer.

The timers are controlled by hardware operation using the PUT/GET instruction or by register operation in the register file using the PEEK/POKE instruction.

13.1.1 Configuration of 8-Bit Timer Counter

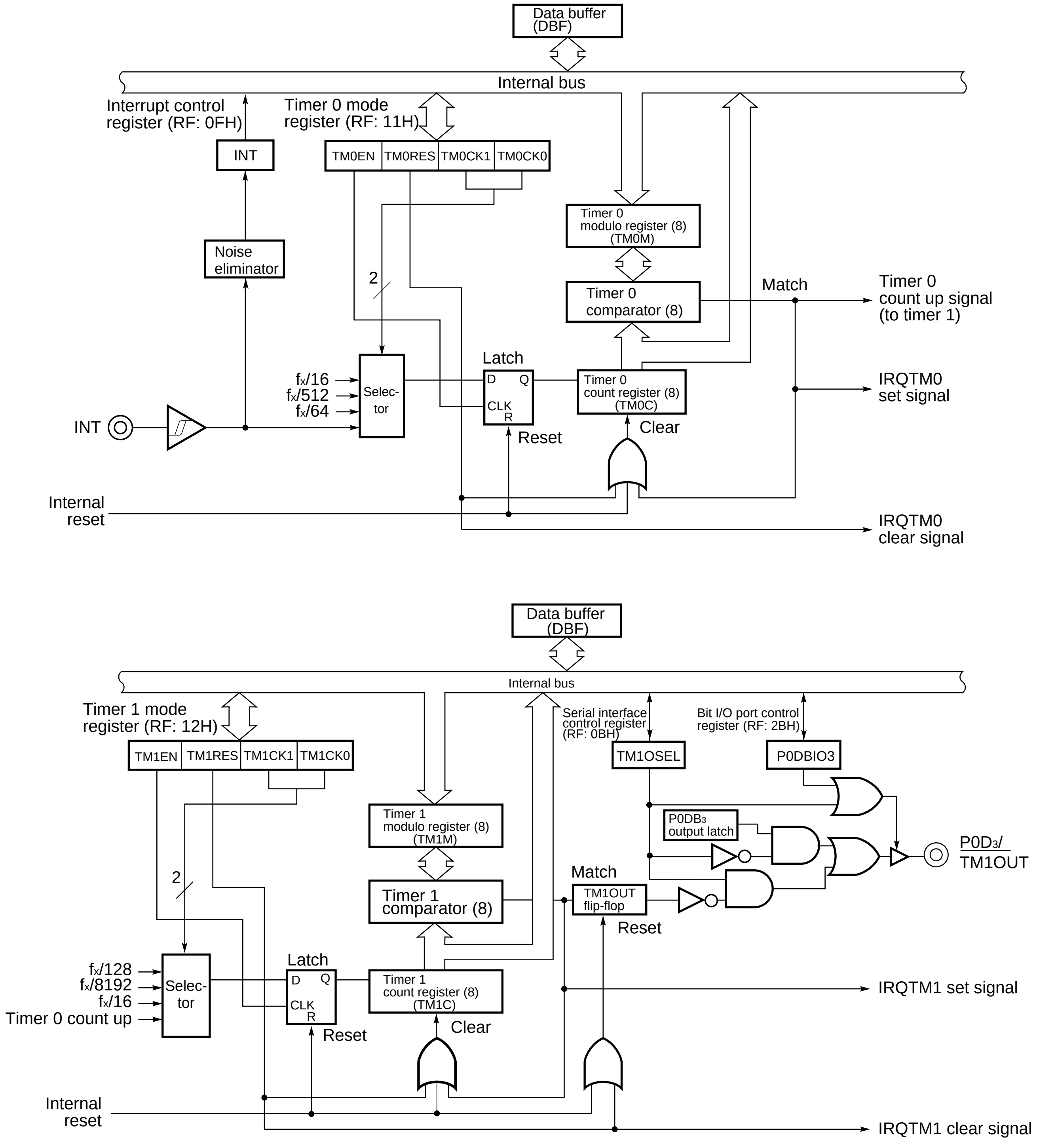
Figure 13-1 shows the configuration of the 8-bit timer counter. An 8-bit timer counter consists of an 8-bit count register, 8-bit modulo register, comparator (compares count register values and modulo register values), and selector which selects count pulse.

Cautions 1. The modulo register is a write-only register.

2. The count register is a read-only register.

*

Figure 13-1. Configuration of the 8-Bit Timer Counters



Remark f_X: System clock oscillation frequency

Figure 13-2. Timer 0 Mode Register

RF: 11H

	Bit 3	Bit 2	Bit 1	Bit 0
	TM0EN	TM0RES	TM0CK1	TM0CK0
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

TM0CK1	TM0CK0	Selection of timer 0 count pulse
0	0	f_X/16
0	1	f_X/512
1	0	f_X/64
1	1	External clock from the INT pin

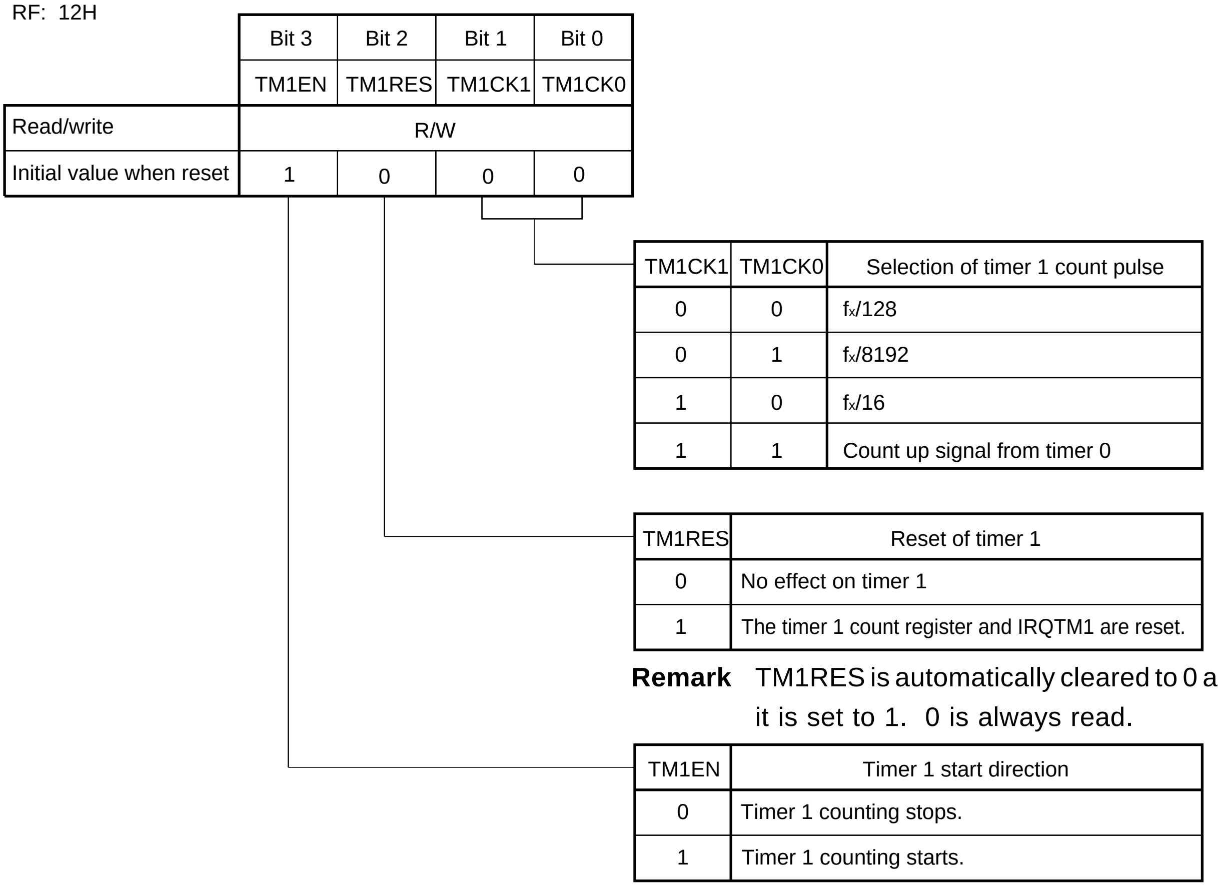
TM0RES	Reset of timer 0
0	No effect on timer 0
1	The timer 0 count register and IRQTM0 are reset.

Remark TM0RES is automatically cleared to 0 after it is set to 1. 0 is always read.

TM0EN	Timer 0 start direction
0	Timer 0 counting stops.
1	Timer 0 counting starts.

Remark TM0EN can be used as a status flag for detecting the counting status of timer 0. (1: Counting, 0: Not counting)

Figure 13-3. Timer 1 Mode Register



RF: 12H

	Bit 3	Bit 2	Bit 1	Bit 0
	TM1EN	TM1RES	TM1CK1	TM1CK0
Read/write	R/W			
Initial value when reset	1	0	0	0

TM1CK1	TM1CK0	Selection of timer 1 count pulse
0	0	$f_x/128$
0	1	$f_x/8192$
1	0	$f_x/16$
1	1	Count up signal from timer 0

TM1RES	Reset of timer 1
0	No effect on timer 1
1	The timer 1 count register and IRQTM1 are reset.

Remark TM1RES is automatically cleared to 0 after it is set to 1. 0 is always read.

TM1EN	Timer 1 start direction
0	Timer 1 counting stops.
1	Timer 1 counting starts.

Remark TM1EN can be used as a status flag for detecting the counting status of timer 1. (1: Counting, 0: Not counting)

13.1.2 Operation of 8-Bit Timer Counters

(1) Count register

The timer 0 and timer 1 count registers are 8-bit up counters whose initial values are 00H. They are incremented each time a count pulse is entered.

A count register is initialized to 00H when:

<1> This product is reset. (See **Chapter 16**.)

<2> The contents of the 8-bit modulo register match a count register value and the comparator generates a match signal.

<3> For timer 0, 1 is written into TM0RES of the register file.
For timer 1, 1 is written into TM1RES of the register file.

(2) Modulo register

The modulo registers of timer 0 and timer 1 determine the count value of the count register. They are initialized to FFH.

A value is set in a modulo register via the data buffer (DBF) using the PUT instruction.

(3) Comparator

The comparators of timer 0 and timer 1 output match signals when the values of the count register and modulo register match and when the next count pulse is input. That is, if the value of the modulo register is initial value FFH, the comparator outputs a match signal when 256 is counted.

The match signal from the comparator clears the count register contents to 0 and automatically sets an interrupt request flag (IRQTM0 or IRQTM1) to 1. An interrupt is accepted when the EI instruction (interrupt acceptance enable instruction) is executed and an interrupt enable flag (IPTM0 or IPTM1) is set. When an interrupt is accepted, an interrupt request flag (IRQTM0 or IRQTM1) is set to 0 and program control is transferred to an interrupt handling.

13.1.3 Selecting the Count Pulse

The count pulse for timer 0 is selected with TM0CK0 or TM0CK1.

One of four count pulses is selected: system clock (f_X)/512, system clock (f_X)/64, system clock (f_X)/16, and an external count pulse input from the INT pin.

At reset, TM0CK0 and TM0CK1 are 0 and f_X/16 is selected.

The count pulse for timer 1 is selected with TM1CK0 or TM1CK1.

One of four count pulse is selected: f_X/8192, f_X/128, f_X/16, and count up signals from timer 0.

At power start-up or reset, timer 1 is used to generate an oscillation setting time. For this purpose, the initial values are TM1CK0 = 0 and TM1CK1 = 0 and f_X/128 is selected for the count pulse. Since TM1EN = 1 is set as the initial value, when f_X = 4 MHz, the µPD17149 starts at address 0000H approx. 8 ms after a reset occurs (see **Chapter 16**).

13.1.4 Setting the Count Value in a Modulo Register

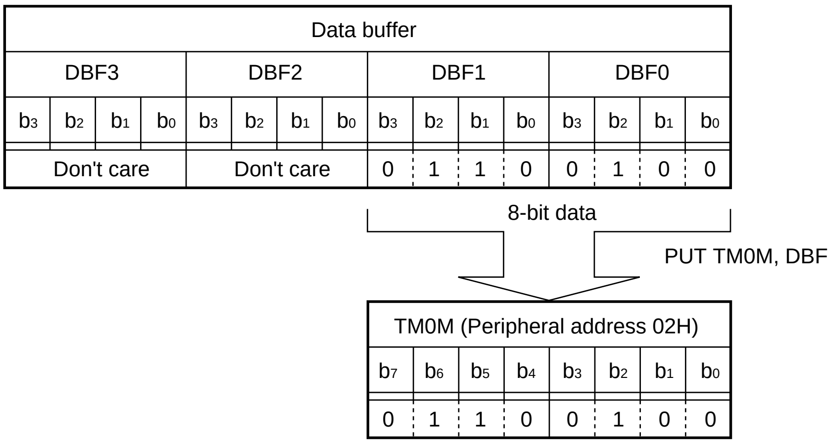
A value is set in a modulo register via the data buffer (DBF) using the PUT instruction. The peripheral address of the modulo register is 02H for timer 0 and 03H for timer 1.

When a value is sent by the PUT instruction, data in the eight low-order bits (DBF1 and DBF0) of DBF is sent to the modulo register. Figure 13-4 shows an example of timer 0.

Figure 13-4. Setting the Count Value in a Modulo Register

Example of setting count value 64H in timer 0 modulo register

```
CONTDATL   DAT  4H               ; CONTDATL is assigned to 4H using the symbol definition
                                   instruction.
CONTDATH   DAT  6H               ; CONTDATH is assigned to 6H using the symbol definition
                                   instruction.
           MOV  DBF0, #CONTDATL ;
           MOV  DBF1, #CONTDATH ;
           PUT  TM0M, DBF        ; The value is transferred with reserved word TM0M.
```



Caution A value of between 01H and FFH can be set in the modulo register. When 00H is set, normal counting is not done.

The modulo register is a write-only register. No count value set in the modulo register can be read from it. Counting does not stop during 8-bit timer counter operation when the PUT TM0M, DBF or PUT TM1M, DBF instruction is executed.

13.1.5 Reading Count Register Values

The count register values of timer 0 and timer 1 are read at the same time via DBF (data buffer) using the GET instruction.

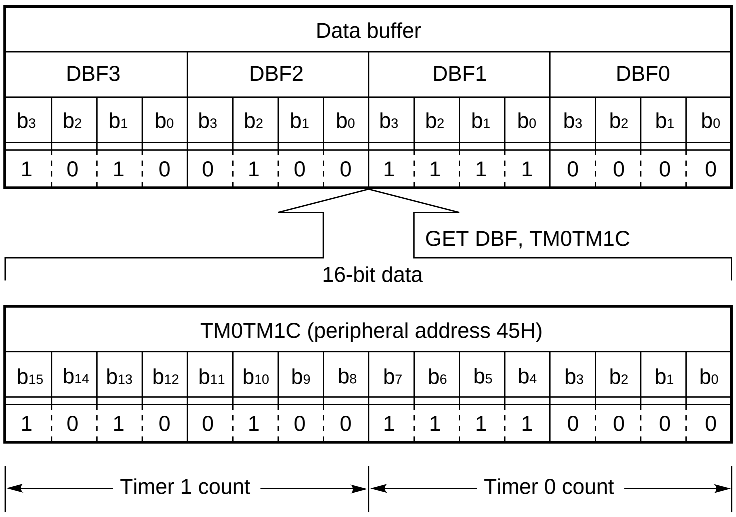
The count register values of timer 0 and timer 1 are assigned to peripheral address 45H. The eight high-order bits are assigned to the timer 1 count value. The eight low-order bits are assigned to the timer 0 count value.

The count register values can be read into DBF by using the GET instruction. During execution of the GET instruction, the count register stops counting and a count value is retained. When a count pulse enters the timer in use during execution of the GET instruction, the count register is retained. After execution of the GET instruction, the count register is incremented by one and continues counting.

The scheme prevents miscounting, even when the GET instruction is executed during timer operation unless two or more count pulses are input during single instruction cycle.

Figure 13-5. Example of Reading 8-Bit Counter Count Values

The timer 0 count value is F0H and the timer 1 count value is A4H.
GET DBF, TM0TM1C; Example of using reserved words DBF and TM0TM1C



Data buffer															
DBF3				DBF2				DBF1				DBF0			
b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
1	0	1	0	0	1	0	0	1	1	1	1	0	0	0	0

TM0TM1C (peripheral address 45H)															
b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
1	0	1	0	0	1	0	0	1	1	1	1	0	0	0	0

* 13.1.6 Setting the Interval Time

The interval at which the comparator outputs a match signal is determined by the value set in the modulo register. The following equations represent how to determine value N to be set in the modulo register from interval time T [s].

$T = (N + 1)/f_{CP} = (N + 1) \times T_{CP}$
$N = T \times f_{CP} - 1$ or $N = T/T_{CP} - 1$ (where N = 1 to 255)
where f_{CP} : Count pulse frequency [Hz]
T_{CP}: Count pulse period [s] ($1/f_{CP}$ = resolution)

- Example of calculating the count value from the interval time, and program example

  - Example in which an interval time of 7 ms is assumed for timer 1 (system clock: f_X = 4 MHz)
    It is impossible to set the interval time to 7 ms exactly, because of the resolution of the timer. To select a value nearest to 7 ms, it is necessary to select the count pulse (f_X/128, having a resolution of 32 μs) and use it to obtain the count value.

    (Example of calculation) T = 7 ms and resolution = 32 μs

    N = T/resolution – 1
    $= 7 \times 10^{-3}/(32 \times 10^{-5}) - 1$
    $= 217.75 \fallingdotseq 218$ (= DAH)

    When the interval time becomes nearest to 7 ms, the modulo register value becomes DAH, in which case the interval time is 7.008 ms.

    (Program example)

```
MOV     DBF0,   #0AH    ; Store DAH in the DBF using reserved words DBF0 and DBF1.
MOV     DBF1,   #0DH    ;
PUT     TMM,    DBF     ; Transfer the contents of the DBF using reserved word TMM.

INITFLG TM1EN, TM1RES, NOT TM1CK1, NOT TM1CK0
                        ; Set TM1EN and TM1RES using the built-in macro instruction INITFLG,
                        ; set the timer 1 count pulse to fX/128, and
                        ; start counting.
```

13.1.7 Interval Time Error

*

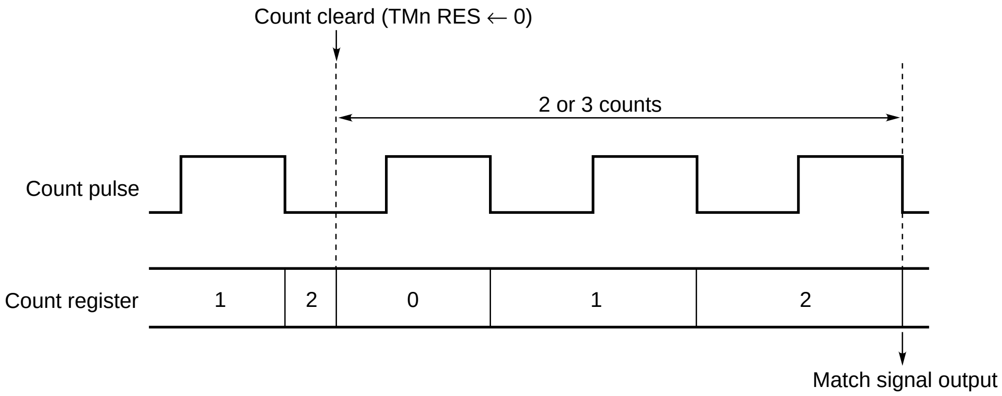
The interval time may have an error of up to –1.5 counts. Be careful when the value set in the modulo register is small.

(1) Error that occurs if the count register is cleared to 0 when counting (maximum error: –1 count)

The count register for the 8-bit timer counter is cleared to 0 by setting the TMnRES flag (to 1). The frequency division circuit for generating the count pulse from the system clock is not reset, however.

If the count is cleared to 0 by setting the TMnRES flag (to 1) when counting, an error for one pulse period occurs at the timing of the first count. The following example shows how counting proceeds if 2 is set in the modulo register.

Figure 13-6. Error that Occurs If the Count Register is Cleared to 0 When Counting



In this example, a match signal is supposed to be output at every 3 counts. After the count is cleared, however, a match signal is output at the second count (minimum).

This error occurs if TMnRES <– 1 occurs simultaneously with TMnEN=1 <– 0.

(2) Error that occurs when counting is resumed from a point of break (maximum error: –1.5 counts)

The count register for the 8-bit timer counter is cleared to 0 by setting the TMnRES flag (to 1). The frequency division circuit for generating the count pulse from the system clock is not reset, however.

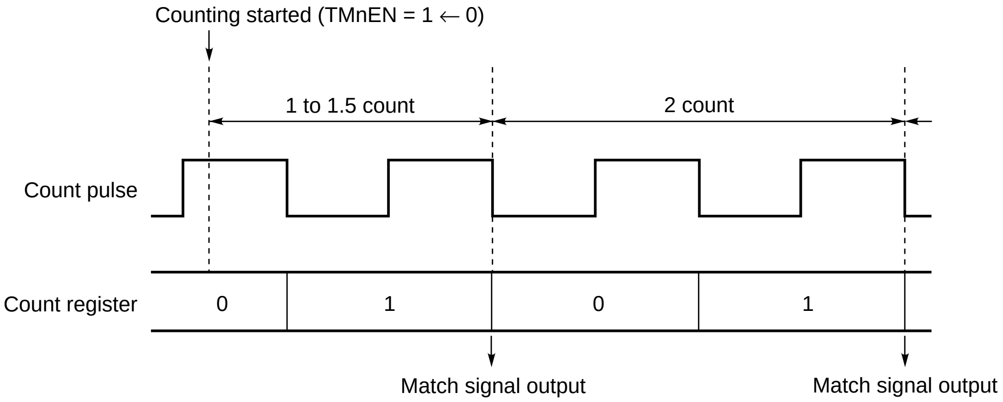
If counting is resumed by setting the TMnEN flag (to 1), the timing of the first count varies as shown below, depending on whether the count pulse begins at a low or high level.

If the count pulse begins at a high level, the first count occurs when the count pulse falls next time. If the count pulse begins at a low level, the first count occurs when counting starts.

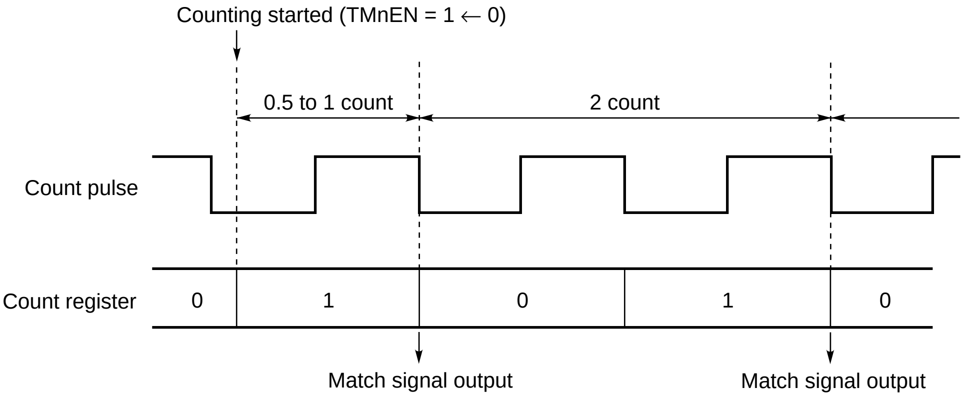
For the first count after counting begins, there occurs an error of –0.5 to –1.5 counts before a match signal is output. The following example shows how counting proceeds if 1 is set in the modulo register.

Figure 13-7. Error that Occurs When Counting is Resumed from a Point of Break.

(a) If the count pulse begins at a high level (error: –0.5 to –1 count)



(b) If the count pulse begins at a low level (error: –1 to –1.5 counts)



In this example, a match signal is supposed to be output at every 2 counts, but the first match signal (immediately after counting is resumed), is output after 1.5 counts (maximum) or 0.5 counts (minimum) (error: –0.5 to –1.5 counts).

The same error occurs also for the oscillation settling time, because the timer is used also for oscillation settling time generation.

13.1.8 Outputting a Timer 1

The $\text{P0D}_3/\overline{\text{TM1OUT}}$ pin functions as a timer 1 output pin when the TM1OSEL flag is set to 1. The P0DBIO3 value has nothing to do with this setting.

Timer 1 contains a flip-flop for a match signal output. It reverses the output each time the comparator outputs a match signal. When the TM1OSEL flag is set to 1, the contents of this flip-flop are output to the $\text{P0D}_3/\overline{\text{TM1OUT}}$ pin.

The $\text{P0D}_3/\overline{\text{TM1OUT}}$ pin is an N-ch open-drain output pin. The mask option enables this pin to contain a pull-up resistor. If this pin does not contain a pull-up resistor, its initial status is high impedance.

An internal timer 1 output flip-flop starts operating when TM1EN is set to 1. To make timer 1 start output beginning at an initial status of the pin, set 1 in TM1RES and reset the flip-flop.

Figure 13-8. Timer 1 Output Setting Register

RF: 0BH

	Bit 3	Bit 2	Bit 1	Bit 0
	TM1OSEL	0	0	SIOEN
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

SIOEN	Function
0	$\text{P0D}_0/\overline{\text{SCK}}$, P0D_1/SO, and P0D_2/SI function as ports.
1	$\text{P0D}_0/\overline{\text{SCK}}$, P0D_1/SO, and P0D_2/SI function as the serial interface.

Caution The SIOEN bit has no relationship with the timer 1 output setting.

TM1OSEL	Function
0	The $\text{P0D}_3/\overline{\text{TM1OUT}}$ pin is used as a port.
1	The $\text{P0D}_3/\overline{\text{TM1OUT}}$ pin is used for timer 1 match signal output.

13.2 BASIC INTERVAL TIMER (BTM)

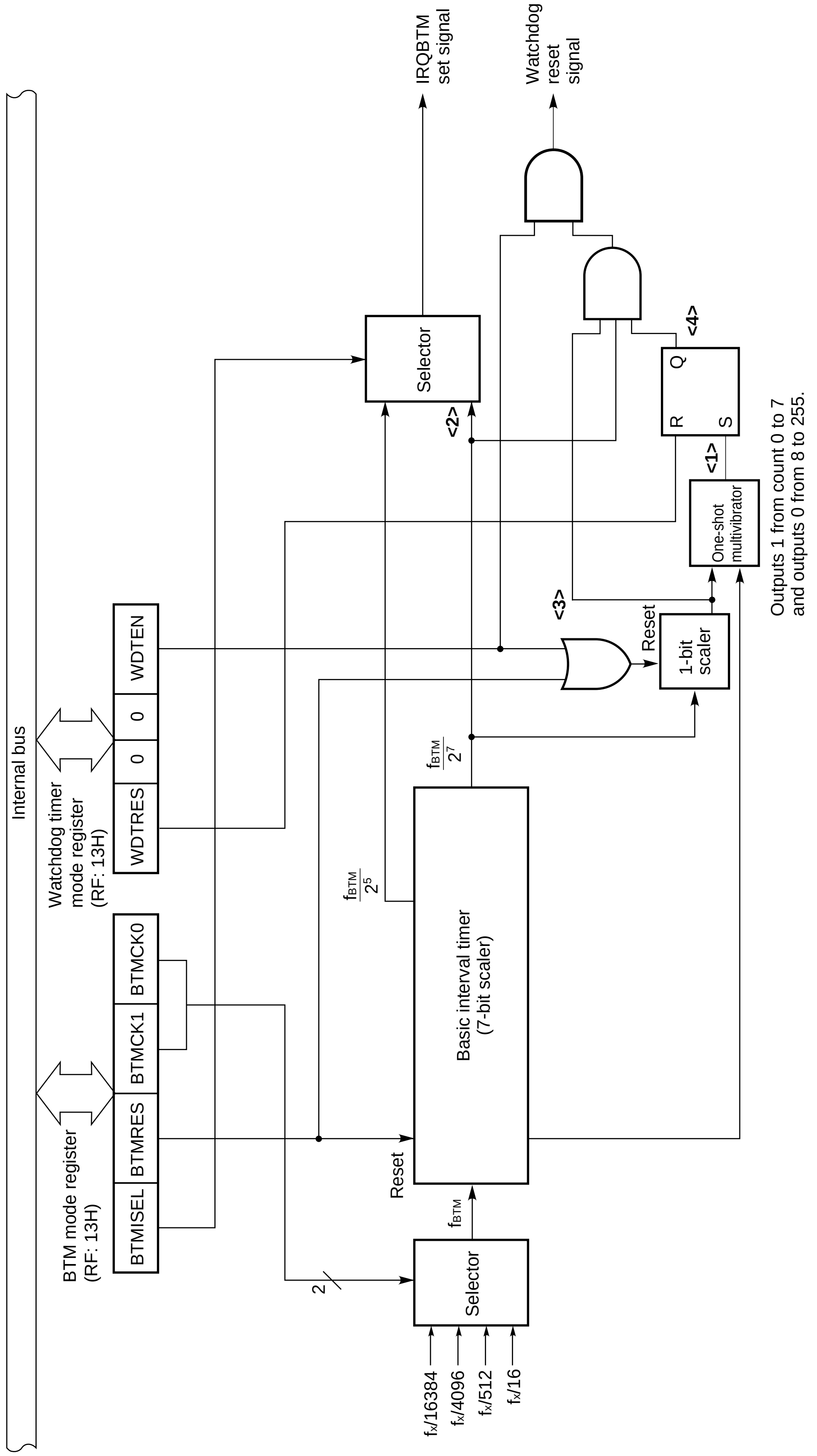
The µPD17149 provides a 7-bit basic interval timer.
This timer has the following functions:

(1) Reference time generation
(2) Selection and counting of a wait time when standby mode is released
(3) Watchdog timer operation for detecting software errors (infinite loops, etc.)

13.2.1 Configuration of the Basic Interval Timer

Figure 13-9 shows the configuration of the basic interval timer.

Figure 13-9. Configuration of the Basic Interval Timer



Remark <1> to <4> in the figure indicate the signals in the timing chart in Figure 13-12.

13.2.2 Registers for Controlling the Basic Interval Timer

The basic interval timer is controlled by the BTM mode register and watchdog timer mode register.
Figure 13-10 and 13-11 show the configurations of the registers.

Figure 13-10. BTM Mode Register

RF: 13H

	Bit 3	Bit 2	Bit 1	Bit 0
	BTMISEL	BTMRES	BTMCK1	BTMCK0
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

BTMCK1	BTMCK0	Selection of BTM count pulse
0	0	$f_X/16$ (1 instruction execution time)
0	1	$f_X/16384$ (1024 instruction execution time)
1	0	$f_X/4096$ (256 instruction execution time)
1	1	$f_X/512$ (32 instruction execution time)

BTMRES	Reset of BTM
0	Does not affect the basic interval timer (BTM).
1	Resets the binary counter of the basic interval timer (BTM).

Remark BTMRES is automatically cleared to 0 after it is set to 1. 0 is always read.

BTMISEL	Selection of interval
0	Sets the interval for the 128-count pulse.
1	Sets the interval for the 32-count pulse.

Figure 13-11. Watchdog Timer Mode Register

RF: 03H

	Bit 3	Bit 2	Bit 1	Bit 0
	WDTRES	0	0	WDTEN
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

WDTEN	Enabling watchdog timer function
0	Puts the watchdog timer in stop status.
1	Starts watchdog timer operation.

Remarks 1. WDTEN cannot be cleared to 0 by the program.

2. WDTEN is automatically cleared to 0 after it is set to 1. 0 is always read.

WDTRES	Reset of the watchdog timer
0	Does not affect the watchdog timer.
1	Resets the flip-flop used to retain a BTM overflow carry used for the watchdog timer.

Remark WDTRES is automatically cleared to 0 after it is set to 1. 0 is always read.

13.2.3 Operation of the Basic Interval Timer

The basic interval timer is a 7-bit binary counter that is always incremented by a count pulse specified in the BTM mode register. It cannot stop counting.

The interval time for the basic interval timer can be switched using BTMISEL. When BTMISEL = 0, the interval time is the time ($128/f_{BTM}$) obtained by dividing the count pulse frequency by 128. When BTMISEL = 1, it is the time ($32/f_{BTM}$) obtained by dividing the count pulse frequency by 32.

Switching the interval time does not cause the contents of the counter to be cleared.

13.2.4 Watchdog Timer Function

The basic interval timer can also be used as a watchdog timer which detects a system hang.

(1) Overview of the watchdog timer

The watchdog timer is a counter that generates a reset signal at constant intervals. When the generation of a reset signal is being disabled every time by the program, this function enables the system to be reset (starting from address 0000H) when the system hangs up (the watchdog timer is not reset within the expected time) for some reason, such as due to external noise.

Even if a program branches to an unexpected routine due to external noise and enters an infinite loop, the system can be recovered within a certain time by the reset signal that is generated by the watchdog timer.

(2) Operation of the watchdog timer

When WDTEN is set to 1, the 1-bit scaler starts operating, causing the basic interval timer to operate as an 8-bit watchdog timer.

Once the watchdog timer runs, the watchdog timer function can be stopped only when the device is reset and WDTEN is cleared to 0.

The generation of a reset by the watchdog timer can be disabled in the following two ways:

<1> Repeatedly setting WDTRES in a program
<2> Repeatedly setting BTMRES in a program

For **<1>**, it is necessary to set WDTRES within the time during which the watchdog timer counts from 8 to 191 (immediately before 192), as shown in Figure 13-12. So, a program must be developed which executes SET1 WDTRES at least once within the time in which the watchdog timer counts 184.

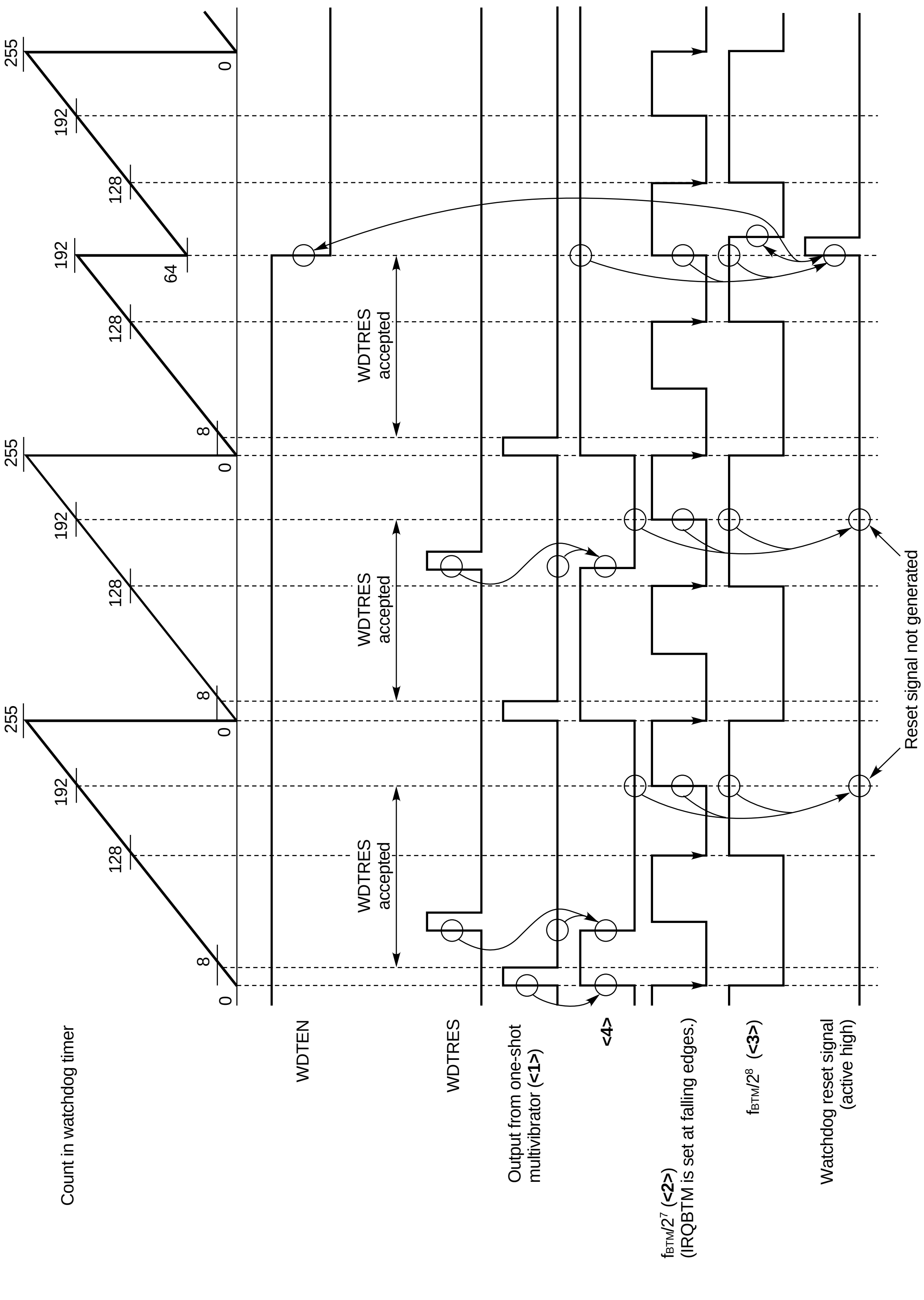
For **<2>**, it is necessary to set BTMRES within the time in which the basic interval timer (BTM) counts 128. So, a program must be developed which executes SET1 BTMRES at least once within the time in which the BTM counts 128. This method, however, disables BTM interrupt handling.

Caution Setting WDTEN does not cause the BTM to be reset. So, it is necessary to set BTMRES before setting WDTEN for the first time, thereby resetting the BTM.

Example

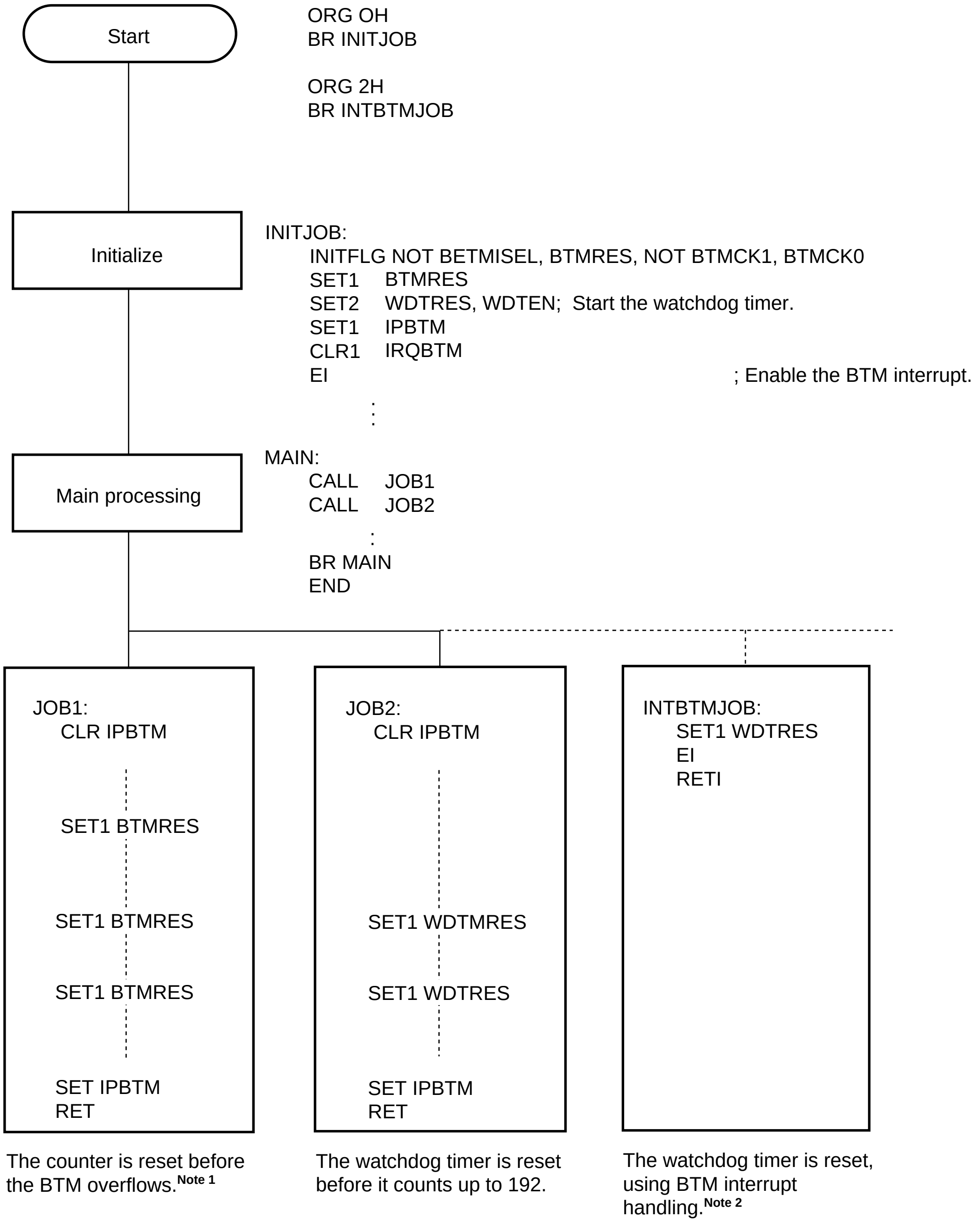
```
:
SET1  BTMRES
SET2  WDTEN, WDTRES
:
```

Figure 13-12. Timing Chart for the Watchdog Timer (When the WDTRES Flag is Used) *


Count in watchdog timer
WDTEN
WDTRES
Output from one-shot multivibrator (<1>)
<4>
fBTM/2^7 (<2>) (IRQBTM is set at falling edges.)
fBTM/2^8 (<3>)
Watchdog reset signal (active high)
0
8
128
192
255
0
8
128
192
255
0
8
128
192
64
128
192
255
0
WDTRES accepted
WDTRES accepted
WDTRES accepted
Reset signal not generated

* (3) Watchdog timer program example

(Program example)



```
ORG OH
BR INITJOB

ORG 2H
BR INTBTMJOB
```

```
INITJOB:
      INITFLG NOT BETMISEL, BTMRES, NOT BTMCK1, BTMCK0
      SET1    BTMRES
      SET2    WDTRES, WDTEN;  Start the watchdog timer.
      SET1    IPBTM
      CLR1    IRQBTM
      EI                                       ; Enable the BTM interrupt.
              :
```

```
MAIN:
      CALL    JOB1
      CALL    JOB2
              :
      BR MAIN
      END
```

```
JOB1:
    CLR IPBTM

    SET1 BTMRES

    SET1 BTMRES

    SET1 BTMRES

    SET IPBTM
    RET
```

The counter is reset before the BTM overflows.[Note 1]

```
JOB2:
    CLR IPBTM

    SET1 WDTMRES

    SET1 WDTRES

    SET IPBTM
    RET
```

The watchdog timer is reset before it counts up to 192.

```
INTBTMJOB:
    SET1 WDTRES
    EI
    RETI
```

The watchdog timer is reset, using BTM interrupt handling.[Note 2]

Notes 1. A method of resetting the counter before the BTM overflows does not enable BTM interrupt handling.

2. Programming is easy with a method in which BTM interrupt handling is used to reset the watchdog timer, but its program crash detection ratio is low, compared with the other two methods.

13.3 A/D CONVERTER

µPD17149 contains an 8-bit resolution A/D converter with 4-channel analog input ($P0C_0/ADC_0$ - $P0C_3/ADC_3$).

The A/D converter uses the successive approximation method. The following two operation modes are available:

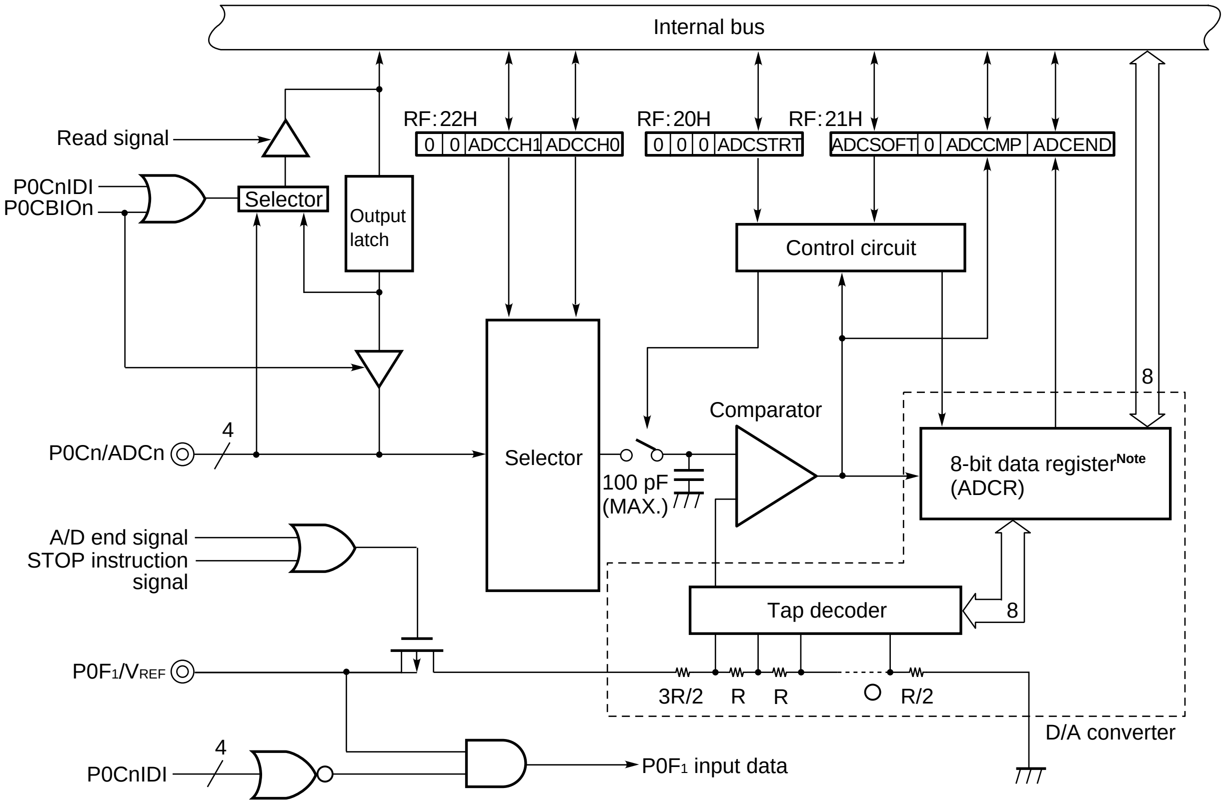
<1> Continuous mode: 8-bit A/D conversion occurs starting at high-order bits.
<2> Single mode : Comparison occurs with an arbitrary voltage value set in the 8-bit data register.

13.3.1 A/D Converter Configuration

Figure 13-13 shows the A/D converter configuration.

Figure 13-13. Block Diagram for the A/D Converter *

Remark n = 0 to 3



Note When the stop instruction is executed, the 8-bit data register (ADCR) is cleared to 00H. *

Remark A/D conversion continues even if the HALT instruction is executed. When A/D conversion ends, the current flowing into the V_{REF} pin is cut off. *

13.3.2 A/D Converter Functions

(1) ADC_0 - ADC_3 pins

These pins are used to input 4-channel analog voltage to the A/D converter. The A/D converter contains a sample hold circuit. Analog input voltage is internally retained during A/D conversion.

(2) V_{REF} pin

This pin is used to input the reference voltage for the A/D converter.
A signal input to ADC_0 to ADC_3 is converted to a digital signal based on voltage applied across V_{REF} and GND. To reduce the current consumption of the microcontroller, the A/D converter has a function for automatically stopping the current which flows into the V_{REF} pin when the converter is not operating. Current flows into the V_{REF} pin in the following cases.

<1> Continuous mode (ADCSOFT = 0)
From when the ADCSTRT flag is set (1) until the ADCEND flag is set (1).

<2> Single mode (ADCSOFT = 1)
From when the ADCSTRT flag is set (1) or from when a value of the 8-bit data register is written until the result of comparison by the comparator is written in the ADCCMP flag.

Remarks 1. Even when the HALT instruction is executed during the A/D conversion, the A/D converter operates until the ADCEND flag is set in continuous mode or until the result of comparison is saved in the ADCCMP flag in single mode. A current flows into the V_{REF} pin while the A/D converter is operating.

2. The A/D conversion stops when the STOP instruction is executed. The A/D converter is initialized and a current does not flow into the V_{REF} pin. (The A/D converter remains stopped after the STOP mode is released.)

(3) 8-bit data register (ADCR)

In the continuous mode, this 8-bit data register stores A/D conversion results for successive approximation. It is read by the GET instruction. In the single mode, the data in this register is converted to analog voltage by the internal D/A converter and the comparator compares this voltage with an analog signal input from the ADCn pin. A value can be written in this register by using the PUT instruction.

(4) Comparator

The comparator compares an analog input voltage with the voltage output from the D/A converter. Value 1 is output if the analog input voltage is higher. Value 0 is output if the input voltage is lower. The comparison result is stored in the 8-bit data register (ADCR) in the continuous mode. It is stored in the ADCCMP flag in the single mode.

(5) A/D converter control register

Figure 13-14 shows the A/D converter control register.

Figure 13-14. A/D Converter Control Register (1/2)

RF: 21H

	Bit 3	Bit 2	Bit 1	Bit 0
	ADCSOFT	0	ADCCMP	ADCEND
Read/write	R/W		R	
Initial value when reset	0	0	0	0

Read = R, write = W

ADCEND	End of A/D conversion
0	Initial status or during A/D conversion.
1	Indicates the end of A/D conversion in the continuous mode. Automatically cleared to 0 when A/D conversion (continuous or single mode) starts.

*

ADCCMP	Comparison result (valid only in the single mode)
0	Analog input voltage is lower than output voltage of the internal D/A converter.
1	Analog input voltage is higher than output voltage of the internal D/A converter.

Remarks 1. In the single mode, the flag content is valid for the third and subsequent instructions after ADCSTRT is set (1) or data is set in ADCR until ADCSTRT or ADCR is set again.

2. In the continuous mode, a value changes according to an A/D conversion value. However, the bit for this value cannot be identified.

ADCSOFT	A/D operation mode selection flag
0	Continuous mode
1	Single mode

Figure 13-14. A/D Converter Control Register (2/2)

RF: 20 H

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	0	ADCSTRT
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

ADCSTRT	Function
0	Automatically cleared to 0 when A/D conversion.
1	Automatically cleared to 0 when A/D conversion (continuous or single mode) starts.

Remark ADCSTRT is a read/write flag. However, 0 is always read.

RF: 22H

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	ADCCH1	ADCCH0
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

ADCCH1	ADCCH0	Analog input channel selection
0	0	ADC$_0$ is selected.
0	1	ADC$_1$ is selected.
1	0	ADC$_2$ is selected.
1	1	ADC$_3$ is selected.

13.3.3 Setting Values in the 8-Bit Data Register (ADCR)

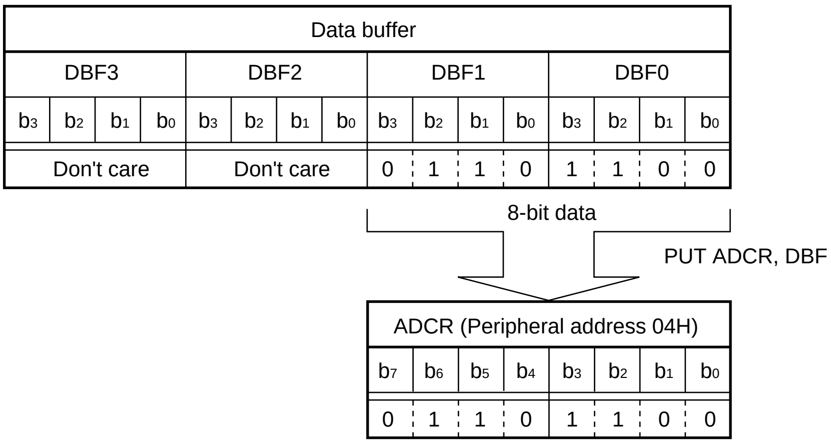
A value is set in the 8-bit data register via the data buffer (DBF) using the PUT instruction in the same way as for comparison voltage setting in the single mode.

The peripheral address for the 8-bit data register (ADCR) of the A/D converter is assigned to 04H. If a value is sent to ADCR by the PUT instruction, only the eight low-order bits (DBF1, DBF0) of DBF are valid. DBF3 and DBF2 values do not affect ADCR.

Figure 13-15. Setting a Value in the 8-Bit Data Register (ADCR)

Example of setting 6CH in ADCR

```
CONTDATL    DAT   CH          ; CONTDATL is assigned to CH using the symbol definition instruction.
CONTDATH    DAT   6H          ; CONTDATH is assigned to 6H using the symbol definition instruction.
            MOV   DBF0, #CONTDATL ;
            MOV   DBF1, #CONTDATH ;
            PUT   ADCR, DBF;  Data is transferred using reserved words ADCR and DBF.
```



Data buffer															
DBF3				DBF2				DBF1				DBF0			
b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Don't care				Don't care				0	1	1	0	1	1	0	0

ADCR (Peripheral address 04H)							
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	1	1	0	1	1	0	0

13.3.4 Reading Values from the 8-Bit Data Register (ADCR)

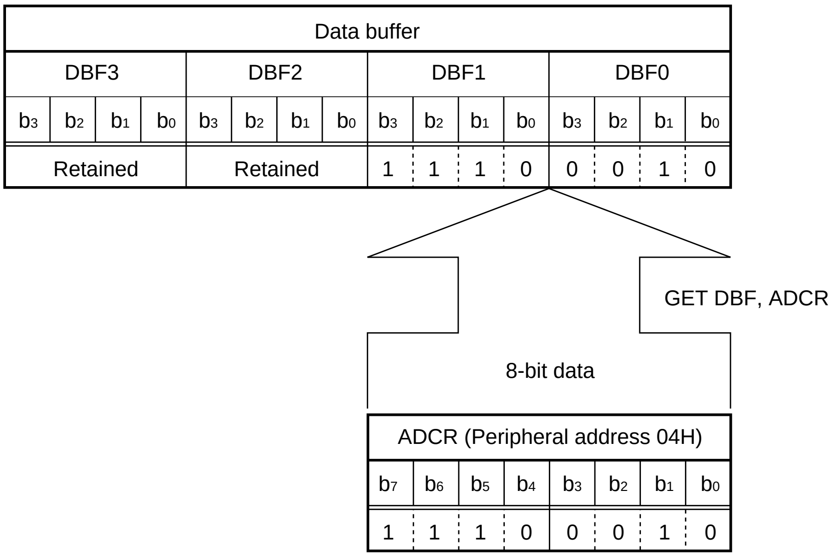
A value is read from the 8-bit data register (ADCR) via the data buffer (DBF) using the GET instruction.

The 8-bit data register (ADCR) of the A/D converter has peripheral address 04H and only its eight low-order bits (DBF1, DBF0) are valid. Execution of the GET instruction does not affect the eight high-order bits of DBF.

Figure 13-16. Reading Values from the 8-Bit Data Register (ADCR)

The result from 8-bit A/D conversion is E2H.

```
GET    DBF, ADCR       ; Example of using reserved words DBF and ADCR
```



Data buffer															
DBF3				DBF2				DBF1				DBF0			
b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Retained				Retained				1	1	1	0	0	0	1	0

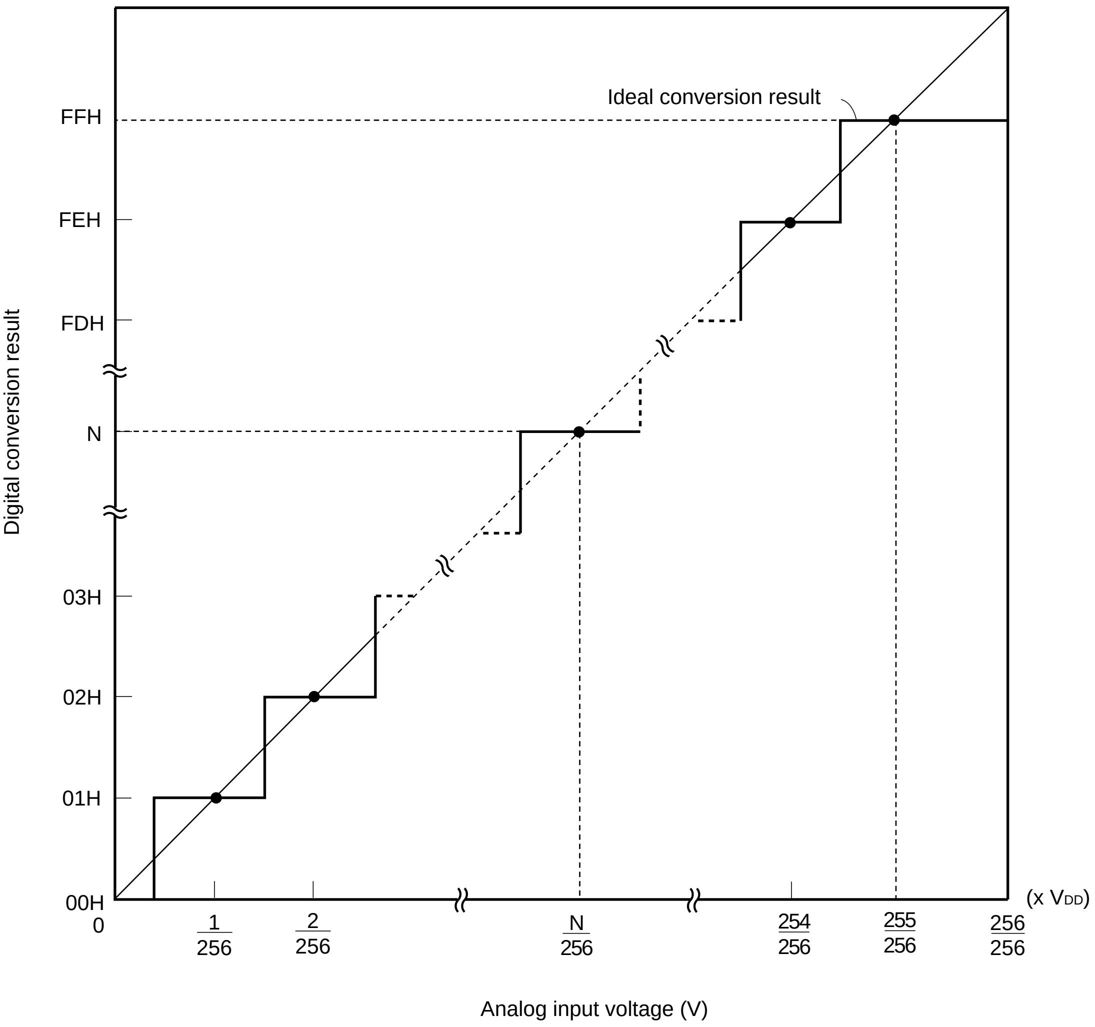
ADCR (Peripheral address 04H)							
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
1	1	1	0	0	0	1	0

13.3.5 A/D Converter Operation

The A/D converter operates in two modes: continuous mode and single mode. The mode can be switched by setting the ADCSOFT flag.

ADCSOFT	A/D converter operation mode
0	Continuous mode (A/D conversion)
1	Single mode (compare operation)

Figure 13-17. Relationship between the Analog Input Voltage and Digital Conversion Result



(1) Continuous mode

(a) Overview of continuous mode

8-bit A/D conversion is done based on the successive approximation method and the conversion result is automatically stored in the 8-bit data register (ADCR).

The internal comparator compares an analog input voltage with the voltage output from the D/A converter. Conversion data is sequentially obtained starting from the most significant bit of the eight bits. In this mode, it takes the time required to execute 25 instructions to complete 8-bit A/D conversion. Completion of 8-bit A/D conversion can be confirmed by checking that the ADCEND flag has been set to 1.

(b) Explanation of continuous mode operation

If ADCSOFT is 0, the A/D converter enters the continuous mode.

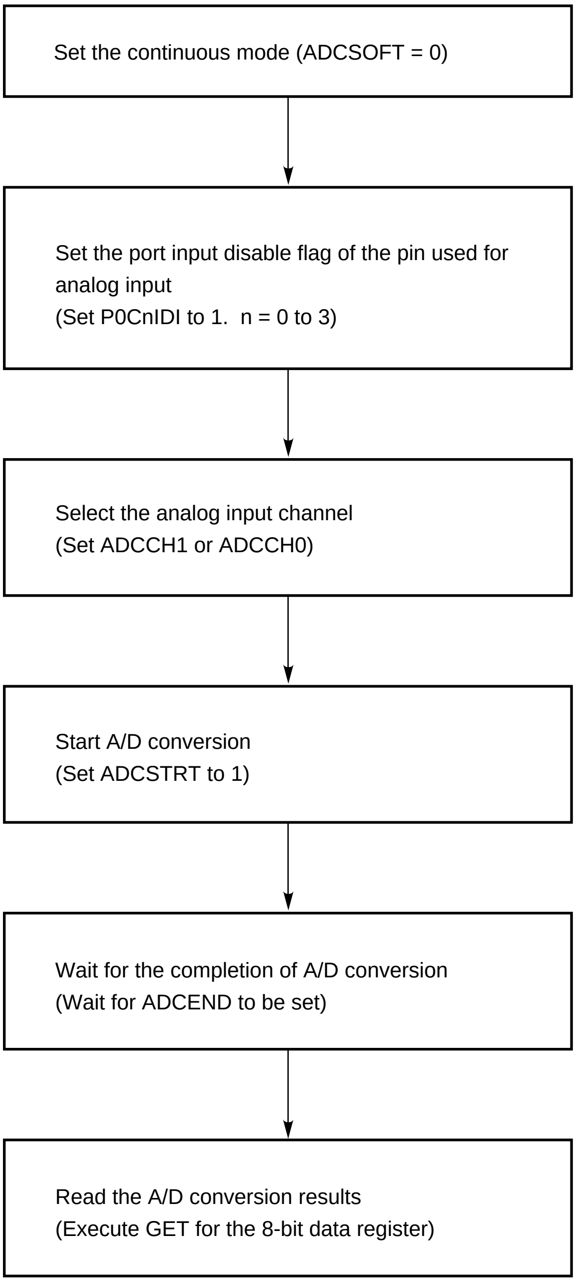
Before A/D conversion starts, port input is disabled for the pin used for analog input by setting P0CnIDI to 1. This is done to prevent the increase in through current of the port input buffer when the level of voltage of the pin specified for analog input is intermediate.

Then an analog input signal is selected by ADCCH1 or ADCCH0. A/D conversion is started by setting the ADCSTRT flag to 1. This flag is cleared to 0 immediately after A/D conversion starts.

During A/D conversion, internal hardware performs successive approximation starting with the most significant bit of the eight bits. The conversion results are sequentially stored bit-by-bit in the 8-bit data register, starting with the most significant bit. The conversion time required for each bit is equivalent to the execution time of three instructions. If 8-bit resolution is not required, data being A/D converted can be fetched by calculating the number of instructions before the ADCEND flag is set.

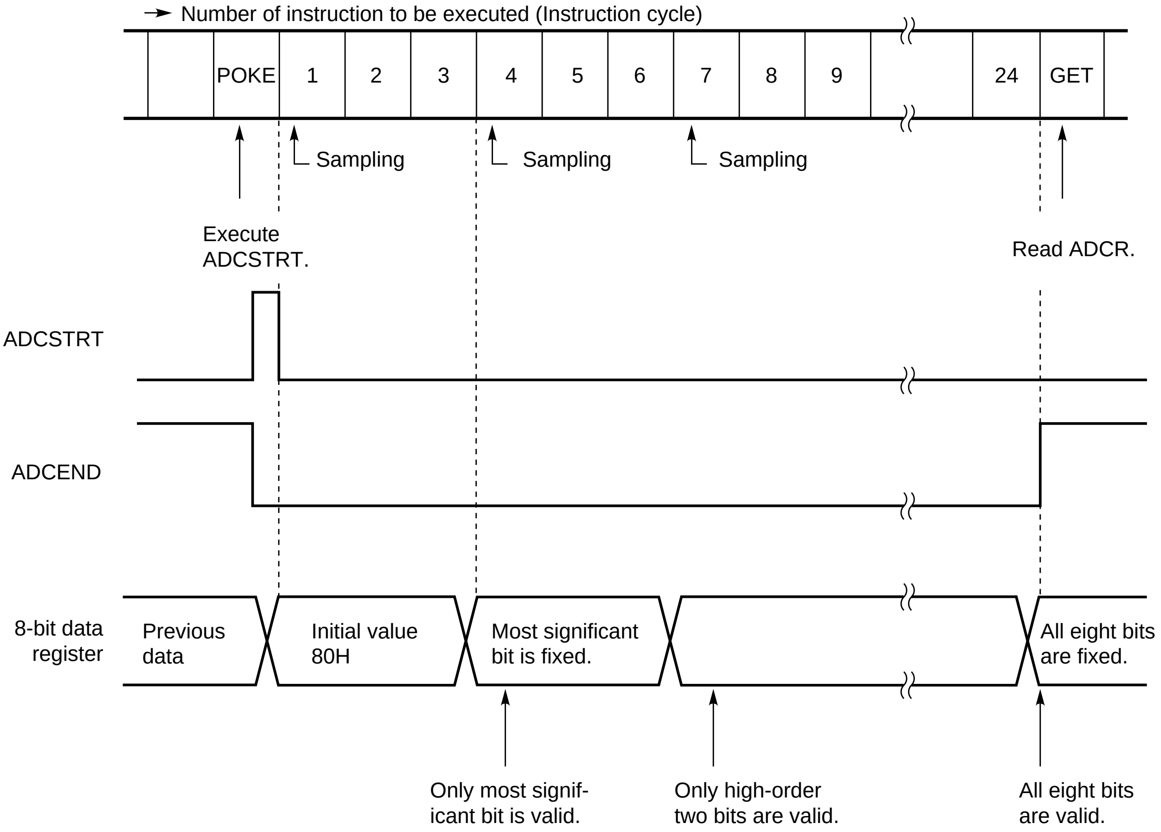
Completion of A/D conversion can be confirmed by checking that the ADCEND flag has been set to 1 which indicate that data has been stored in the least significant bit of the 8-bit data register.

Figure 13-18. Using the Continuous Mode for the A/D Converter


Set the continuous mode (ADCSOFT = 0)
Set the port input disable flag of the pin used for analog input
(Set P0CnIDI to 1. n = 0 to 3)
Select the analog input channel
(Set ADCCH1 or ADCCH0)
Start A/D conversion
(Set ADCSTRT to 1)
Wait for the completion of A/D conversion
(Wait for ADCEND to be set)
Read the A/D conversion results
(Execute GET for the 8-bit data register)

(c) Continuous mode (A/D conversion) timing

Figure 13-19. Continuous Mode (A/D Conversion) Timing


Number of instruction to be executed (Instruction cycle)
POKE
1
2
3
4
5
6
7
8
9
24
GET
Sampling
Sampling
Sampling
Execute ADCSTRT.
Read ADCR.
ADCSTRT
ADCEND
8-bit data register
Previous data
Initial value 80H
Most significant bit is fixed.
All eight bits are fixed.
Only most significant bit is valid.
Only high-order two bits are valid.
All eight bits are valid.

Caution Sampling is performed eight times for each A/D conversion. If the analog input voltage changes considerably during A/D conversion, accurate A/D conversion cannot be performed. To obtain accurate conversion, minimize any change in the analog input voltage during A/D conversion.

Time required for one sampling operation = $14/f_X$ (1.75 µs, at f_X = 8 MHz)
Sampling cycle period = $48/f_X$ (6 µs, at f_X = 8 MHz)

Table 13-1. Data Conversion Time for the A/D Converter

Number of instructions executed after ADCSTRT is set to 1[Note]	Bits for which A/D conversion is completed\ (valid bits when ADCR is read)
4 instructions	Most significant bit
7 instructions	High-order 2 bits
10 instructions	High-order 3 bits
13 instructions	High-order 4 bits
16 instructions	High-order 5 bits
19 instructions	High-order 6 bits
22 instructions	High-order 7 bits
25 instructions	All 8 bits

Note Including a GET instruction to read data from ADCR

(2) Single mode

(a) Overview of single mode

In the single mode, data in the 8-bit data register (ADCR) is compared with voltage subjected to D/ A conversion and with an analog input voltage.

The comparison result appears in the ADCCMP flag.

(b) Explanation of single mode operation

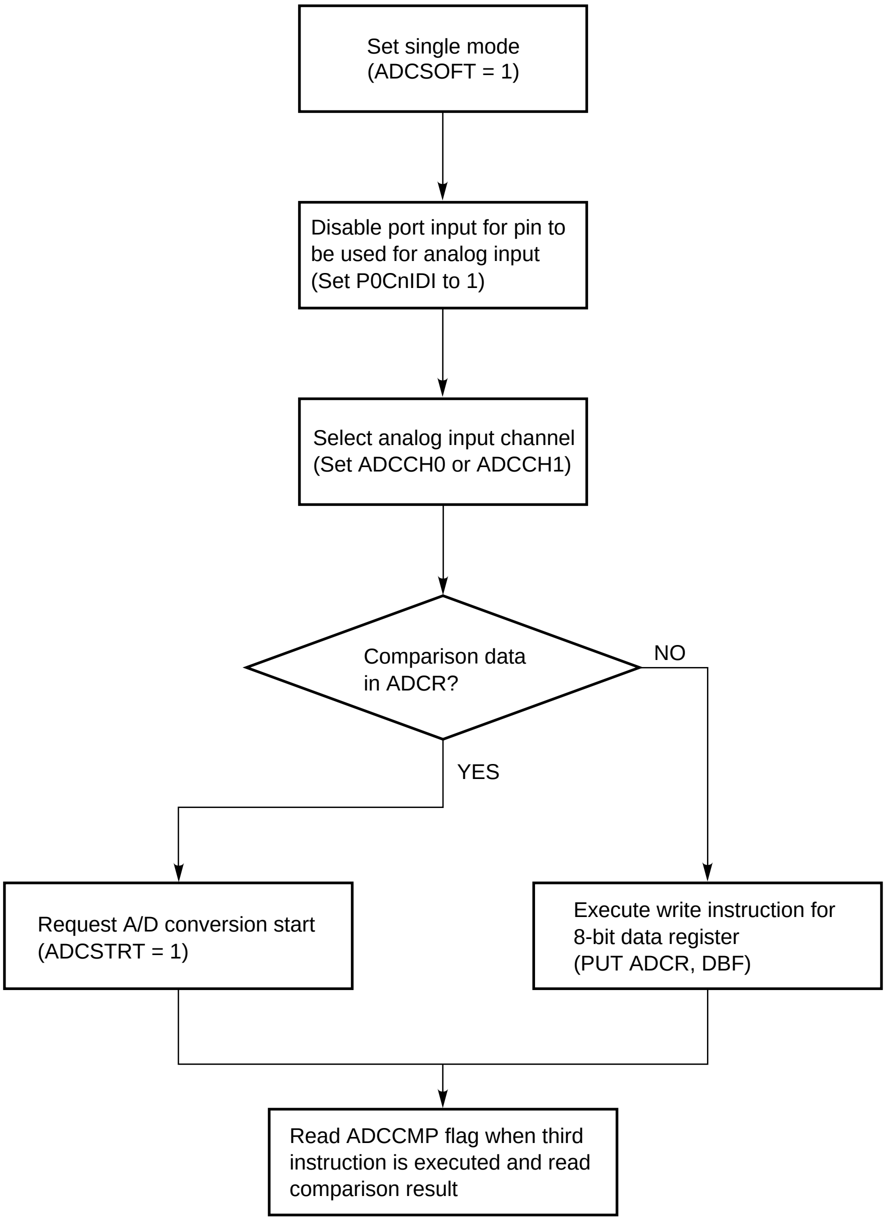
If ADCSOFT is 1, the AD converter enters the single mode.

Before single mode operation starts, port input is disabled for the pin to be used for analog input by setting P0CnIDI to 1. This is done for the same reason as in the continuous mode. Then an analog input signal is selected by ADCCH1 or ADCCH0.

To start single mode operation, execute a write instruction (PUT ADCR, DBF) for the 8-bit data register (ADCR) or set ADCSTRT to 1 when ADCSOFT is 1. When starting the operation by setting ADCSTRT to 1, store the data to be converted in the 8-bit data register (ADCR) before setting ADCSTRT.

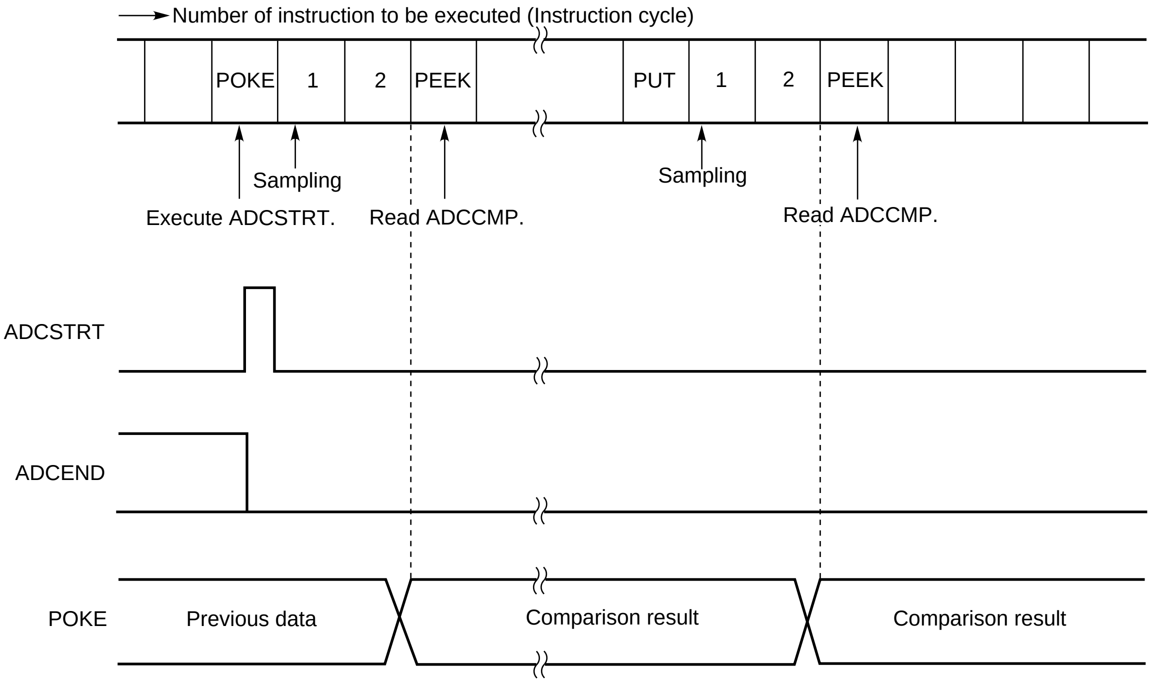
The comparison result in single mode appears in ADCCMP at the execution of the third instruction after a PUT instruction is executed to write to the 8-bit data register (ADCR). At this time, the ADCEND flag becomes invalid.

Figure 13-20. Using the Single Mode for the A/D Converter


Set single mode
(ADCSOFT = 1)
Disable port input for pin to be used for analog input
(Set P0CnIDI to 1)
Select analog input channel
(Set ADCCH0 or ADCCH1)
Comparison data in ADCR?
NO
YES
Request A/D conversion start
(ADCSTRT = 1)
Execute write instruction for 8-bit data register
(PUT ADCR, DBF)
Read ADCCMP flag when third instruction is executed and read comparison result

(c) Single mode (compare operation) timing

Figure 13-21. Single Mode (Compare Operation) Timing



The timing in the single mode is as follows: After 1 is written to ADCSTRT (by executing the POKE instruction), a value is stored in ADCCMP and the comparison result is read by the PEEK instruction at the execution of the third instruction. Setting a value in ADCR (by executing the PUT instruction) also starts the comparison and the result is read at the third instruction after the setting.

ADCCMP is cleared to 0 by reset or by the execution of a write instruction to ADCR.

Caution Before setting a value in ADCR, always set ADCSOFT to 1. If ADCSOFT = 0, no value can be set in ADCR. (The PUT ADCR,DBF instruction is ineffective.)

Time required for one sampling operation = 14/f_X (1.75 μs, at f_X = 8 MHz)

13.4 SERIAL INTERFACE (SIO)

The serial interface of the µPD17149 consists of an 8-bit shift register (SIOSFR), 4-bit serial mode register, and serial clock counter. It is used for serial data input/output.

13.4.1 Functions of the Serial Interface

This serial interface provides three signal lines: serial clock input pin ($\overline{\text{SCK}}$), serial data output pin (SO), and serial data input pin (SI). It allows 8 bits to be sent or received in synchronization with clocks. It can be connected to peripheral input/output devices using any method with a mode compatible to that used by the µPD7500 or 75X series.

(1) Serial clock

Three types of internal clocks and one type of external clock are able to be selected. If an internal clock is selected as a serial clock, it is automatically output to the P0D$_0$/$\overline{\text{SCK}}$ pin.

Table 13-2. Serial Clocks *

SIOCK1	SIOCK0	Serial clock to be selected
0	0	External clock from the $\overline{\text{SCK}}$ pin
0	1	f$_X$/16
1	0	f$_X$/128
1	1	f$_X$/1024

f$_X$: System clock oscillation frequency

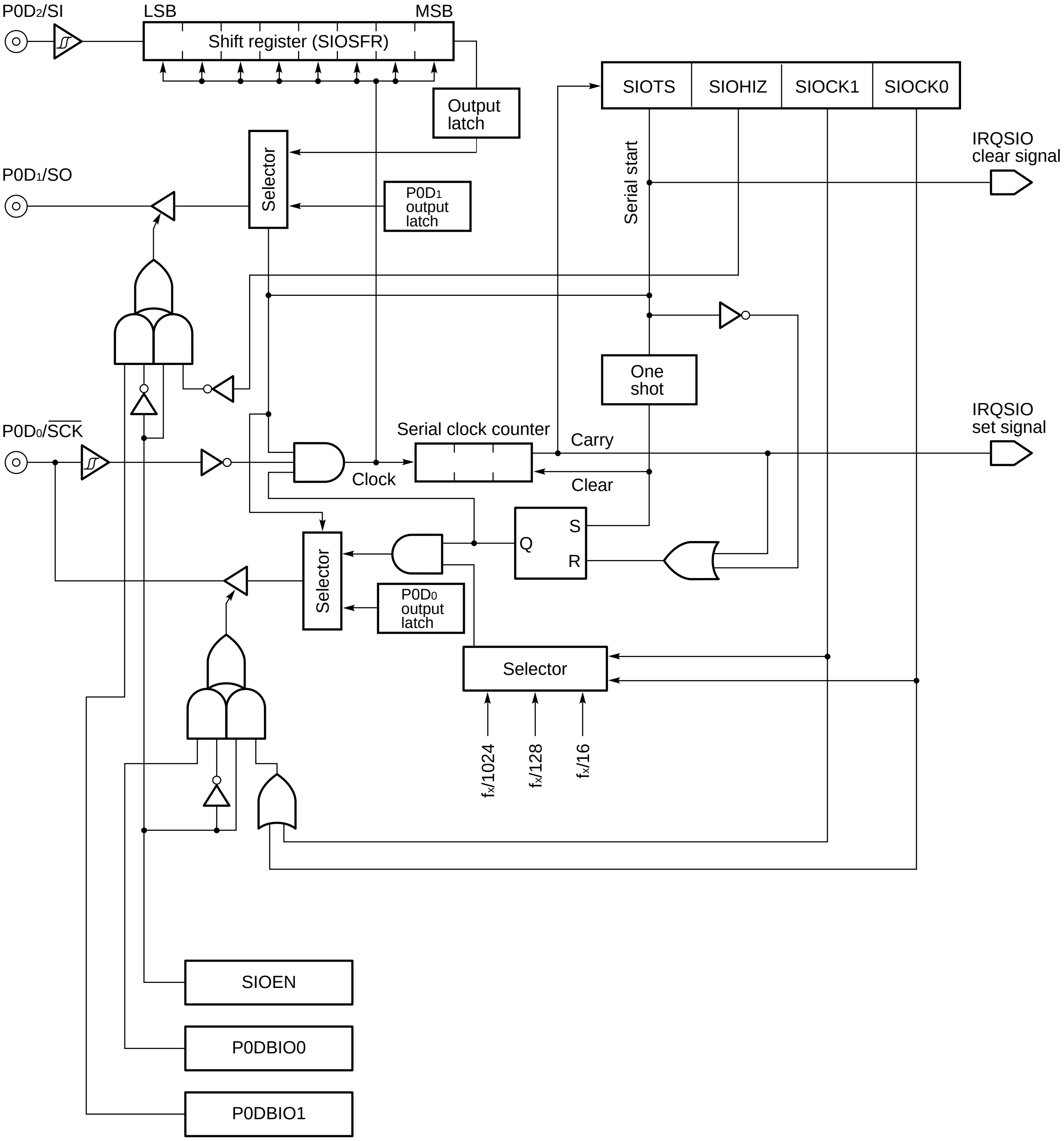
(2) Transmission operation

When SIOEN is set to 1, the pins of port 0D (P0D$_0$/$\overline{\text{SCK}}$, P0D$_1$/SO, P0D$_2$/SI) function as the pins of the serial interface. The serial interface operates in synchronization with the falling edge of the external or internal clock by setting SIOTS to 1. When SIOTS is set, IRQSIO is automatically cleared.
Transmission starts from the most significant bit of the shift register in synchronization with the falling edge of the serial clock. SI pin information is stored in the shift register starting at the least significant bit in synchronization with the rising edge of the serial clock.

When the 8-bits data transmission is terminated, SIOTS is automatically cleared and IRQSIO is set.

Remark Serial transmission starts only from the most significant bit of the shift register contents. It is not possible to transmit from the least significant bit. SI pin status is always stored in the shift register in synchronization with the rising edge of the serial clock.

Figure 13-22. Block Diagram of the Serial Interface



Caution The output latch of the shift register is independent of that of the P0D$_1$ pin. Therefore, even if an output instruction is executed for the P0D$_1$ pin, the output latch status of the shift register does not change. The output latch of the shift register is cleared to 0 by a reset. After that, the latch retains the LSB of the data transmitted previously.

13.4.2 3-Wire Serial Interface Operation Modes

Two modes can be used for the serial interface. If the serial interface function is selected, the $P0D_2$/SI pin always takes in data in synchronization with the serial clock.

- 8-bit transmission and reception mode (simultaneous transmission and reception)
- 8-bit reception mode (SO pin: high impedance status)

Table 13-3. Serial Interface Operation Mode

SIOEN	SIOHIZ	$P0D_0$/SI pin	$P0D_1$/SO pin	Serial interface operation mode
1	0	SI	SO	8-bit transmission and reception mode
1	1	SI	$P0D_1$ (input)	8-bit reception mode
0	x	$P0D_0$ (I/O)	$P0D_1$ (I/O)	General port mode

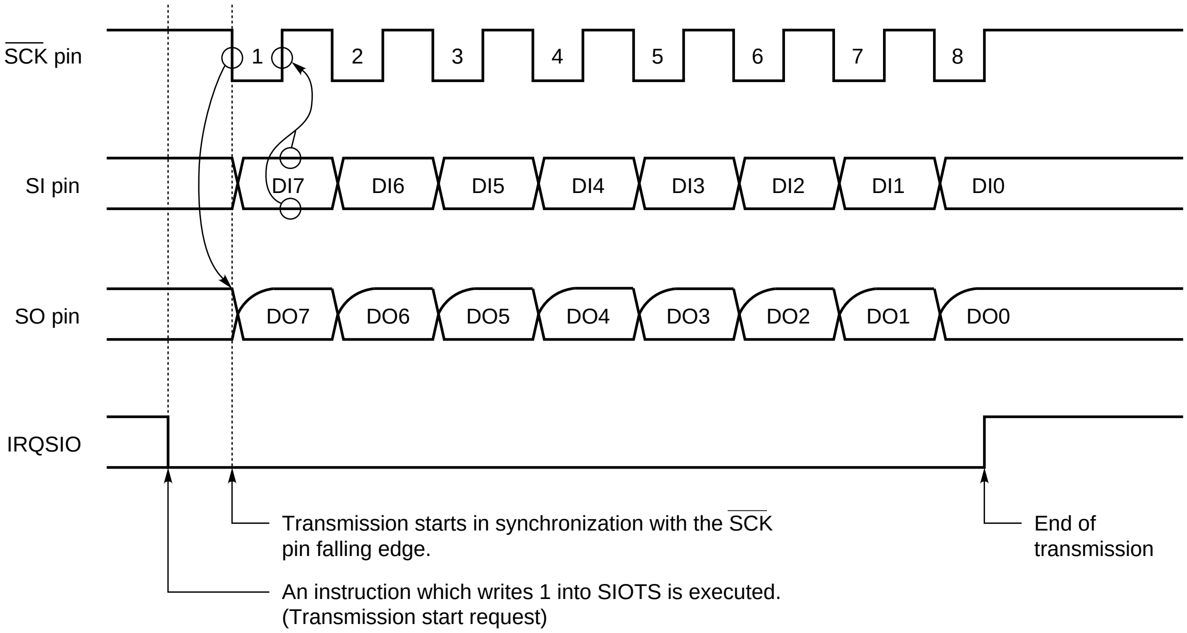
x: Don't care

(1) 8-bit transmission and reception mode (simultaneous transmission and reception)

Serial data input/output is controlled by a serial clock. The most significant bit of the shift register is output from the SO line with a falling edge of the serial clock ($\overline{\text{SCK}}$ pin signal). The contents of the shift register is shifted one bit and at the same time, data on the SI line is loaded into the least significant bit of the shift register.

The serial clock counter (3-bit counter) counts serial clock pulses. Every time it counts eight clocks, the internal interrupt request flag (IRQSIO) is set to 1.

Figure 13-23. Timing of 8-Bit Transmission and Reception Mode (Simultaneous Transmission and Reception)



SCK pin
1
2
3
4
5
6
7
8
SI pin
DI7
DI6
DI5
DI4
DI3
DI2
DI1
DI0
SO pin
DO7
DO6
DO5
DO4
DO3
DO2
DO1
DO0
IRQSIO
Transmission starts in synchronization with the SCK pin falling edge.
End of transmission
An instruction which writes 1 into SIOTS is executed. (Transmission start request)

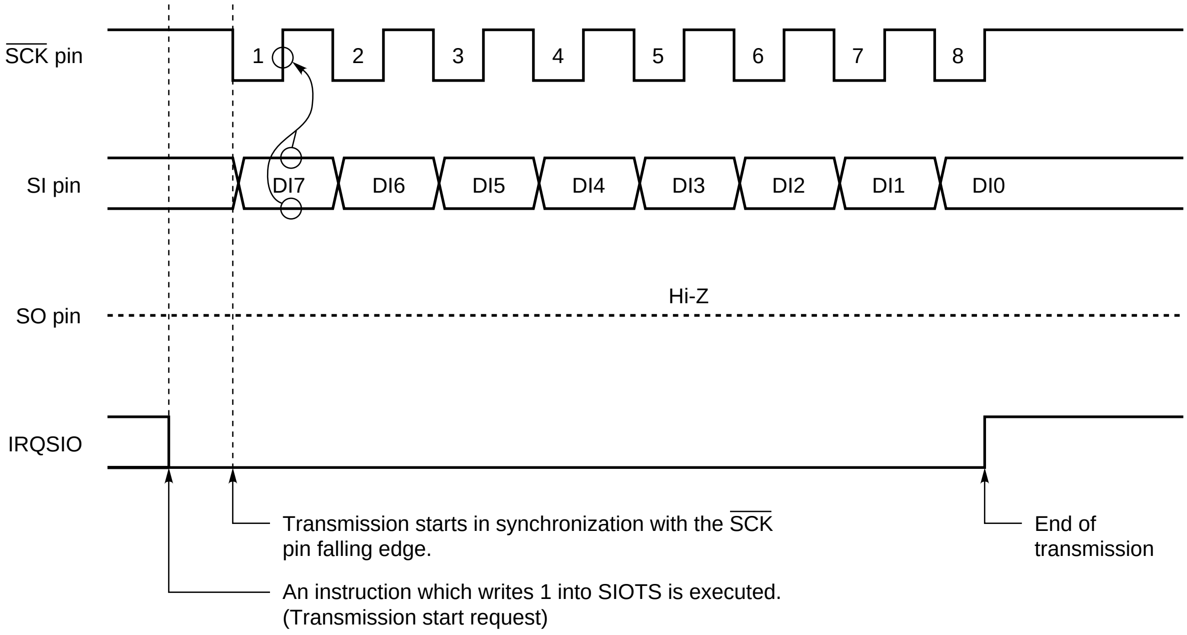
Remark DI : Input serial data
DO: Output serial data

(2) Clock synchronization 8-bit transmission and reception mode (SO pin: high impedance status)

When SIOHIZ is 1, the P0D$_1$/SO pin is in the high impedance status. If serial clock supply starts by writing 1 in SIOTS, only the reception function of the serial interface operates.

The P0D$_1$/SO pin is in the high impedance status and can be used for input port (P0D$_1$).

Figure 13-24. Timing of the 8-Bit Reception Mode


SCK pin
1
2
3
4
5
6
7
8
SI pin
DI7
DI6
DI5
DI4
DI3
DI2
DI1
DI0
SO pin
Hi-Z
IRQSIO
Transmission starts in synchronization with the SCK pin falling edge.
End of transmission
An instruction which writes 1 into SIOTS is executed. (Transmission start request)

Remark DI: Input serial data

(3) Operation stop mode

If the value in SIOTS (RF: address 02H, bit 3) is 0, the serial interface enters operation stop mode. In this mode, no serial transfer occurs.

In this mode, the shift register does not perform shifting and can be used as an ordinary 8-bit register.

Figure 13-25. Serial Interface Control Register (1/2)

RF: 02H

	Bit 3	Bit 2	Bit 1	Bit 0
	SIOTS	SIOHIZ	SIOCK1	SIOCK0
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

SIOCK1	SIOCK0	Serial clock selection
0	0	External clock ($\overline{\text{SCK}}$ pin)
0	1	f_X/16
1	0	f_X/128
1	1	f_X/1024

SIOHIZ	Function selection of the P0D$_1$/SO pin
0	Serial data output (SO pin)
1	Input port (P0D$_1$ pin)

SIOTS	Start and stop of serial transfer
0	Forced termination of the serial transfer. (Disables intermediate restart.)
1	Start of serial transfer operation • At internal clock selection Starts operation using the internally divided system-clock signal as a serial clock. • At external clock selection Starts operation at the falling edge of the $\overline{\text{SCK}}$ signal.

Remark SIOTS is automatically cleared to 0 when serial transmission is completed.

Figure 13-25. Serial Interface Control Register (2/2)

RF: 0BH

	Bit 3	Bit 2	Bit 1	Bit 0
	TM1OSEL	0	0	SIOEN
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

SIOEN	Enabling SIO operation
0	$P0D_0$/$\overline{SCK}$, $P0D_1$/SO, and $P0D_2$/SI function as ports.
1	$P0D_0$/$\overline{SCK}$, $P0D_1$/SO, and $P0D_2$/SI function as the serial interface.

Remark See also **Chapter 12**.

TM1OSEL	Function selection of the $P0D_3$/$\overline{TM1OUT}$
0	The $P0D_3$/$\overline{TM1OUT}$ pin is used as a port.
1	The $P0D_3$/$\overline{TM1OUT}$ pin is used for timer 1 output.

Caution The TM1OSEL bit has no relationship with the serial interface.

13.4.3 Setting Values in the Shift Register

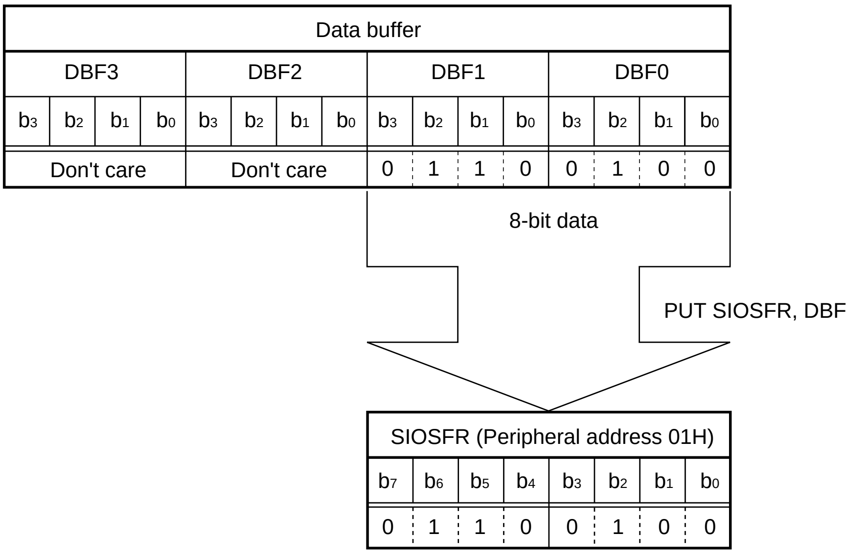
Values are set in the shift register via the data buffer (DBF) using the PUT instruction.

The peripheral address of the shift register is 01H. When sending a value to the shift register using the PUT instruction, only the low-order eight bits (DBF1, DBF0) of DBF are valid. The DBF3 and DBF2 values do not affect the shift register.

Figure 13-26. Setting a Value in the Shift Register

Example of setting value 64H in the shift register

```
SIODATL  DAT  4H              ; SIODATL is assigned to 4H using symbol definition.
SIODATH  DAT  6H              ; SIODATH is assigned to 6H using symbol definition.
         MOV  DBF0, #SIODATL ;
         MOV  DBF1, #SIODATH;
         PUT  SIOSFR, DBF     ; Value is transmitted using reserved word SIOSFR.
```



Data buffer															
DBF3				DBF2				DBF1				DBF0			
b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Don't care				Don't care				0	1	1	0	0	1	0	0

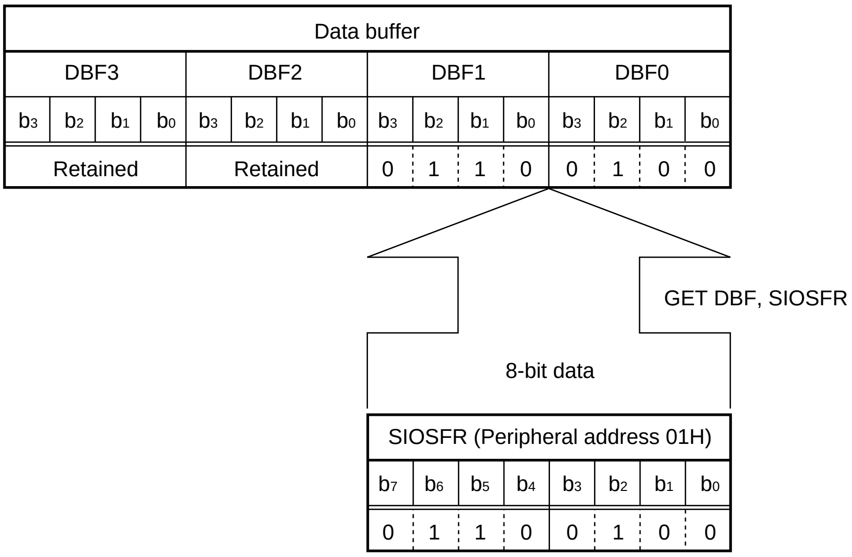
SIOSFR (Peripheral address 01H)							
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	1	1	0	0	1	0	0

13.4.4 Reading Values from the Shift Register

A value is read from the shift register via the data buffer (DBF) using the GET instruction. The shift register has peripheral address 01H and only the eight low-order bits (DBF1, DBF0) are valid. Executing the GET instruction does not affect the eight high-order bits of DBF.

Figure 13-27. Reading a Value from the Shift Register

GET DBF, SIOSFR; Example of using reserved words DBF and SIOSFR



Data buffer															
DBF3				DBF2				DBF1				DBF0			
b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Retained				Retained				0	1	1	0	0	1	0	0

SIOSFR (Peripheral address 01H)							
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	1	1	0	0	1	0	0

[MEMO]

CHAPTER 14 INTERRUPT FUNCTIONS

The µPD17149 has four internal interrupt functions and one external interrupt function. It can be used in various applications.

The interrupt control circuit of the µPD17149 has the features listed below. This circuit enables very high-speed interrupt handling.

(a) Used to determine whether an interrupt can be accepted with the interrupt mask enable flag (INTE) and interrupt enable flag (IPxxx).

(b) The interrupt request flag (IRQxxx) can be tested or cleared. (Interrupt generation can be checked by software.)

(c) Multiple interrupts are possible (up to three levels).

(d) Standby mode (STOP, HALT) can be released by an interrupt request. (Release condition can be selected by the interrupt enable flag.)

Caution In interrupt handling, only the BCD, CMP, CY, Z, and IXE flags are saved in the stack automatically by the hardware for up to three levels of multiple interrupts. The DBF and WR are not saved by the hardware when peripheral hardware such as the timers or A/D converter is accessed in interrupt handling. It is recommended that the DBF and WR be saved in RAM by the software at the beginning of interrupt handling. Saved data can be loaded back into the DBF and WR immediately before the end of interrupt handling.

14.1 INTERRUPT SOURCES AND VECTOR ADDRESSES

For every interrupt in the µPD17149, when the interrupt is accepted, a branch occurs to the vector address associated with the interrupt source. This method is called the vectored interrupt method. Table 14-1 lists the interrupt sources and vector addresses.

If two or more interrupt requests occur or multiple suspended interrupt requests are enabled at the same time, they are handled according to priorities shown in Table 14-1.

Table 14-1. Interrupt Source Types

Interrupt source	Priority	Vector address	IRQ flag	IP flag	IEG flag	Internal/ external	Remarks
INT pin (RF: 0FH, bit 0)	1	0005H	IRQ RF: 3FH, bit 0	IP RF: 2FH, bit 0	IEGMD0,1 RF: 1FH	External	Rising edge or falling edge can be selected.
Timer 0	2	0004H	IRQTM0 RF: 3EH, bit 0	IPTM0 RF: 2FH, bit 1	–	Internal	
Timer 1	3	0003H	IRQTM1 RF: 3DH, bit 0	IPTM1 RF: 2FH, bit 2	–	Internal	
Basic interval timer	4	0002H	IRQBTM RF: 3CH, bit 0	IPBTM RF: 2FH, bit 3	–	Internal	
Serial interface	5	0001H	IRQSIO RF: 3BH, bit 0	IPSIO RF: 2EH, bit 0	–	Internal	

14.2 HARDWARE COMPONENTS OF THE INTERRUPT CONTROL CIRCUIT

The flags of the interrupt control circuit are explained below.

(1) Interrupt request flag and the interrupt enable flag

The interrupt request flag (IRQxxx) is set to 1 when an interrupt request occurs. When interrupt handling is executed, the flag is automatically cleared to 0.

An interrupt enable flag (IPxxx) is provided for each interrupt request flag. If the flag is 1, an interrupt is enabled. If it is 0, the interrupt is disabled.

(2) EI/DI instruction

The EI/DI instruction is used to determine whether an accepted interrupt is to be executed.

If the EI instruction is executed, the interrupt enable flag (INTE) for enabling interrupt reception is set to 1 (when an interrupt is accepted, INTE is cleared to 0). Since the INTE flag is not registered in the register file, flag status cannot be checked by instructions.

The DI instruction clears the INTE flag to 0 and disables all interrupts.

At reset the INTE flag is cleared to 0 and all interrupts are disabled.

Table 14-2. Interrupt Request Flag and Interrupt Enable Flag

Interrupt request flag	Signal for setting the interrupt request flag	Interrupt enable flag
IRQ	Set by edge detection of an INT pin input signal. A detection edge is selected by IEGMD0 or IEGMD1.	IP
IRQTM0	Set by a match signal from timer 0.	IPTM0
IRQTM1	Set by a match signal from timer 1.	IPTM1
IRQBTM	Set by an overflow (reference time interval signal) from the basic interval timer.	IPBTM
IRQSIO	Set by a serial data transmission end signal from the serial interface.	IPSIO

Figure 14-1. Interrupt Control Register (1/7)

RF: 0FH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	0	INT
Read/write	R			
* Initial value when reset	0	0	0	Note

Read = R, write = W

INT	INT pin status
* 0	The logical status of the INT signal that has passed through the noise eliminator is 0 during PEEK instruction execution.
* 1	The logical status of the INT signal that has passed through the noise eliminator is 1 during PEEK instruction execution.

Note Since the INT flag is not latched, it changes according to the logical status of the pin. Once the IRQ flag is set, however, the flag remains set until an interrupt is accepted.

The POKE instruction cannot be used with address 0FH.

RF: 1FH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	IEGMD1	IEGMD0
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

IEGMD1	IEGMD0	Selection of the interrupt detection edge of the INT pin
0	0	Interrupt at the rising edge
0	1	Interrupt at the falling edge
1	0	Interrupt at both edges
1	1	

Figure 14-1. Interrupt Control Register (2/7)

RF: 3FH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	0	IRQ
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

At reading

IRQ	INT pin interrupt request
0	No interrupt request has been issued from the INT pin or an INT pin interrupt is being handled.
1	An interrupt request from the INT pin occurs or an INT pin interrupt is being held.

At writing

IRQ	INT pin interrupt request
0	An in upt request from the INT pin is forcibly released.
1	An interrupt request from the INT pin is forced to occur.

RF: 3EH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	0	IRQTM0
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

At reading

IRQTM0	TM0 interrupt request
0	No interrupt request has been issued from timer 0 or a timer 0 interrupt is being handled.
1	The contents of the timer 0 count register matches that of the timer 0 modulo register and an interrupt request occurs. Or a timer 0 interrupt request is being held.

At writing

IRQTM0	TM0 interrupt request
0	An interrupt request from timer 0 is forcibly released.
1	An interrupt request from timer 0 is forced to occur.

Remark If TMORES is set to 1, IRQTMO is cleared to 0.

Figure 14-1. Interrupt Control Register (3/7)

RF: 3DH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	0	IRQTM1
Read/write	R/W			
Initial value when reset	0	0	0	1

Read = R, write = W

At reading

IRQTM1	TM1 interrupt request
0	No interrupt request has been issued from timer 1 or a timer 1 interrupt is being handled.
1	The contents of the timer 1 count register matches that of the timer 1 modulo register and an interrupt request occurs. Or a timer 1 interrupt request is being held.

At writing

IRQTM1	TM1 interrupt request
0	An interrupt request from timer 1 is forcibly released.
1	An interrupt request from timer 1 is forced to occur.

Remark If TM1RES is set to 1, IRQTM1 is cleared to 0. IRQTM1 is cleared to 0 also immediately after the execution of the STOP instruction.

Figure 14-1. Interrupt Control Register (4/7)

RF: 3CH

*

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	0	IRQBTM
Read/write	R/W			
Initial value when reset	0	0	0	1

Read = R, write = W

At reading

IRQBTM	BTM interrupt request
0	No interrupt request has been issued from the basic interval timer or a basic interval timer interrupt is being handled.
1	The basic interval timer overflows and an interrupt request occurs. Or a basic interval timer interrupt request is being held.

At writing

IRQBTM	BTM interrupt request
0	An interrupt request from the basic interval timer is forcibly released.
1	An interrupt request from the basic interval timer is forced to occur.

Remark If BTMRES is set to 1, IRQBTM is cleared to 0.

Figure 14-1. Interrupt Control Register (5/7)

RF: 3BH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	0	IRQSIO
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

At reading

IRQSIO	SIO interrutp request
0	No interrupt request has been issued from the serial interface or a serial interface interrupt isbeing handled.
1	Serial interface transmission is completed and an interrupt request occurs. Or, a serial interface interrupt request is being held.

At writing

IRQSIO	SIO interrutp request
0	An interrupt request from the serial interface is forcibly released.
1	An interrupt request from the serial interface is forced to occur.

Figure 14-1. Interrupt Control Register (6/7)

RF: 2FH

	Bit 3	Bit 2	Bit 1	Bit 0
	IPBTM	IPTM1	IPTM0	IP
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

IP	Enabling INT pin interrupt
0	Disables an interrupt from the INT pin. Holds interrupt handling even when the IRQ flag is set to 1.
1	Enables an interrupt from the INT pin. If the IRQ flag is set to 1 in the EI status, executes interrupt handling.

IPTM0	Enabling TM0 interrupt
0	Disables an interrupt from timer 0. Holds an interrupt even when the IRQTM0 flag is set to 1.
1	Enables an interrupt from timer 0. If the IRQTM0 flag is set to 1 in the EI status, executes interrupt handling.

IPTM1	Enabling TM1 interrupt
0	Disables an interrupt from timer 1. Holds an interrupt even when the IRQTM1 flag is set to 1.
1	Enables an interrupt from timer 1. If the IRQTM1 flag is set to 1 in the EI status, executes interrupt handling.

IPBTM	Enabling BTM interrupt
0	Disables an interrupt from the basic interval timer. Holds an interrupt even when the IRQBTM flag is set to 1.
1	Enables an interrupt from the basic interval timer. If the IRQBTM flag is set to 1 in the EI status, executes interrupt handling.

Figure 14-1. Interrupt Control Register (7/7)

RF: 2EH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	0	IPSIO
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

IPSIO	Enabling SIO interrupt
0	Disables an interrupt from the serial interface. Holds an interrupt even when the IRQSIO flag is set to 1.
1	Enables an interrupt from the serial interface. If the IRQSIO flag is set to 1 in the EI status, executes interrupt handling.

14.3 INTERRUPT SEQUENCE

14.3.1 Receiving an Interrupt

When an interrupt is accepted, interrupt handling starts after the instruction cycle of the instruction being executed is completed. The program flow is transferred to a vector address. Note that an interrupt during the execution of the MOVT instruction, EI instruction, or an instruction which satisfies the skip condition starts after two instruction cycles are completed.

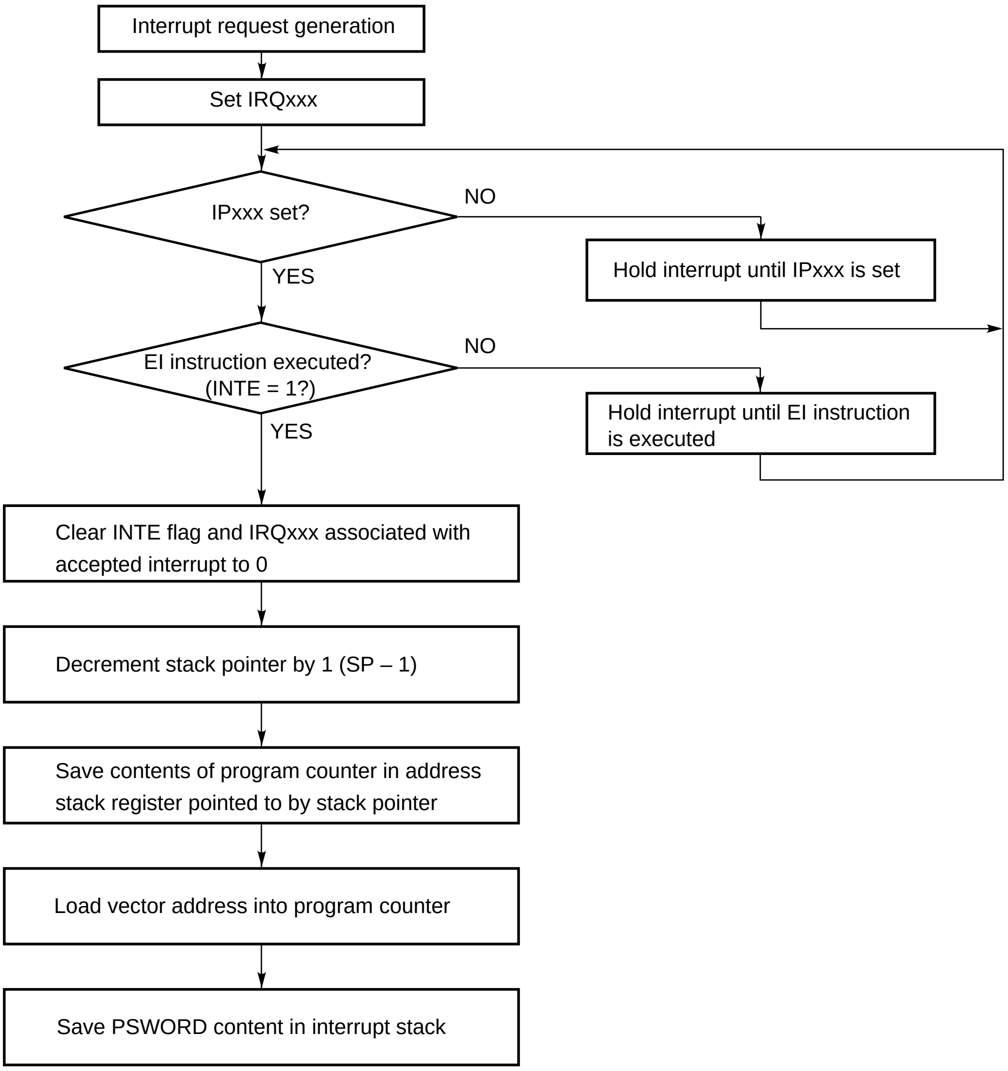
When an interrupt is accepted, the INTE flag is cleared to 0. One level of the address stack register is consumed to store the program return address, and one level of the interrupt stack register is consumed to save PSWORD in the system register.

If two or more interrupts occur or are enabled, interrupt handling is executed in descending order of priority. A lower-priority interrupt is held until a higher-priority interrupt is handled.

See priorities shown in **Table 14-1**.

Caution PSWORD is automatically reset to 00000B after it is saved in the interrupt stack register.

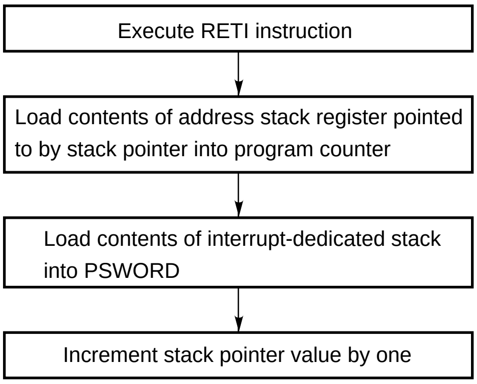
Figure 14-2. Interrupt Handling Procedure


Interrupt request generation
Set IRQxxx
IPxxx set?
NO
Hold interrupt until IPxxx is set
YES
EI instruction executed?
(INTE = 1?)
NO
Hold interrupt until EI instruction is executed
YES
Clear INTE flag and IRQxxx associated with accepted interrupt to 0
Decrement stack pointer by 1 (SP – 1)
Save contents of program counter in address stack register pointed to by stack pointer
Load vector address into program counter
Save PSWORD content in interrupt stack

14.3.2 Return from the Interrupt Routine

Execute the RETI instruction to return from the interrupt handling routine. During the RETI instruction cycle, processing in the figure below occurs.

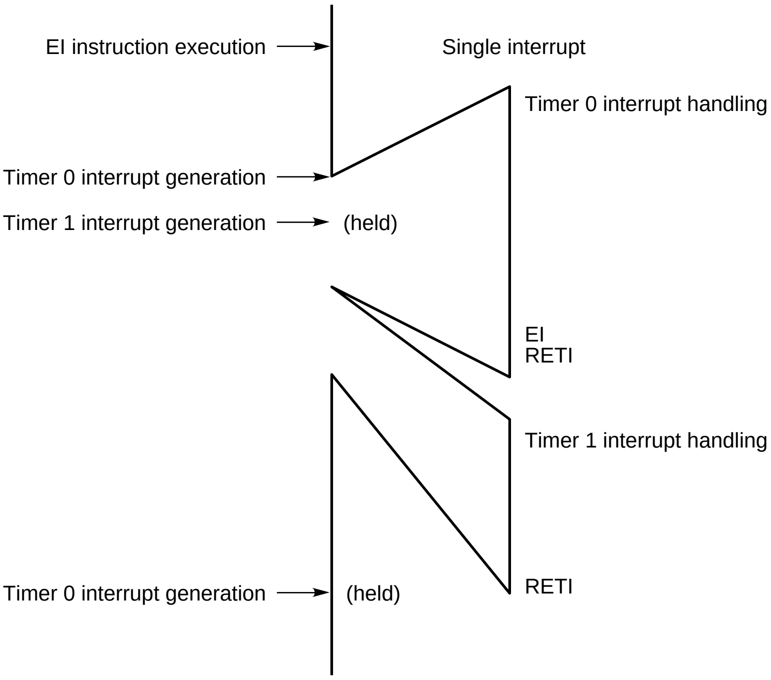
Figure 14-3. Return from Interrupt Handling


Execute RETI instruction
Load contents of address stack register pointed to by stack pointer into program counter
Load contents of interrupt-dedicated stack into PSWORD
Increment stack pointer value by one

Cautions 1. The INTE flag is not set for the RETI instruction.
Interrupt handling is completed. To handle a pending interrupt successively, execute the EI instruction immediately before the RETI instruction and set the INTE flag to 1.

2. To execute the RETI instruction following the EI instruction, no interrupt is accepted between EI instruction execution and RETI instruction execution. This is because the EI instruction sets the INTE flag to 1 after the execution of the subsequent instruction is completed.

Example


EI instruction execution
Single interrupt
Timer 0 interrupt handling
Timer 0 interrupt generation
Timer 1 interrupt generation
(held)
EI
RETI
Timer 1 interrupt handling
RETI
Timer 0 interrupt generation
(held)

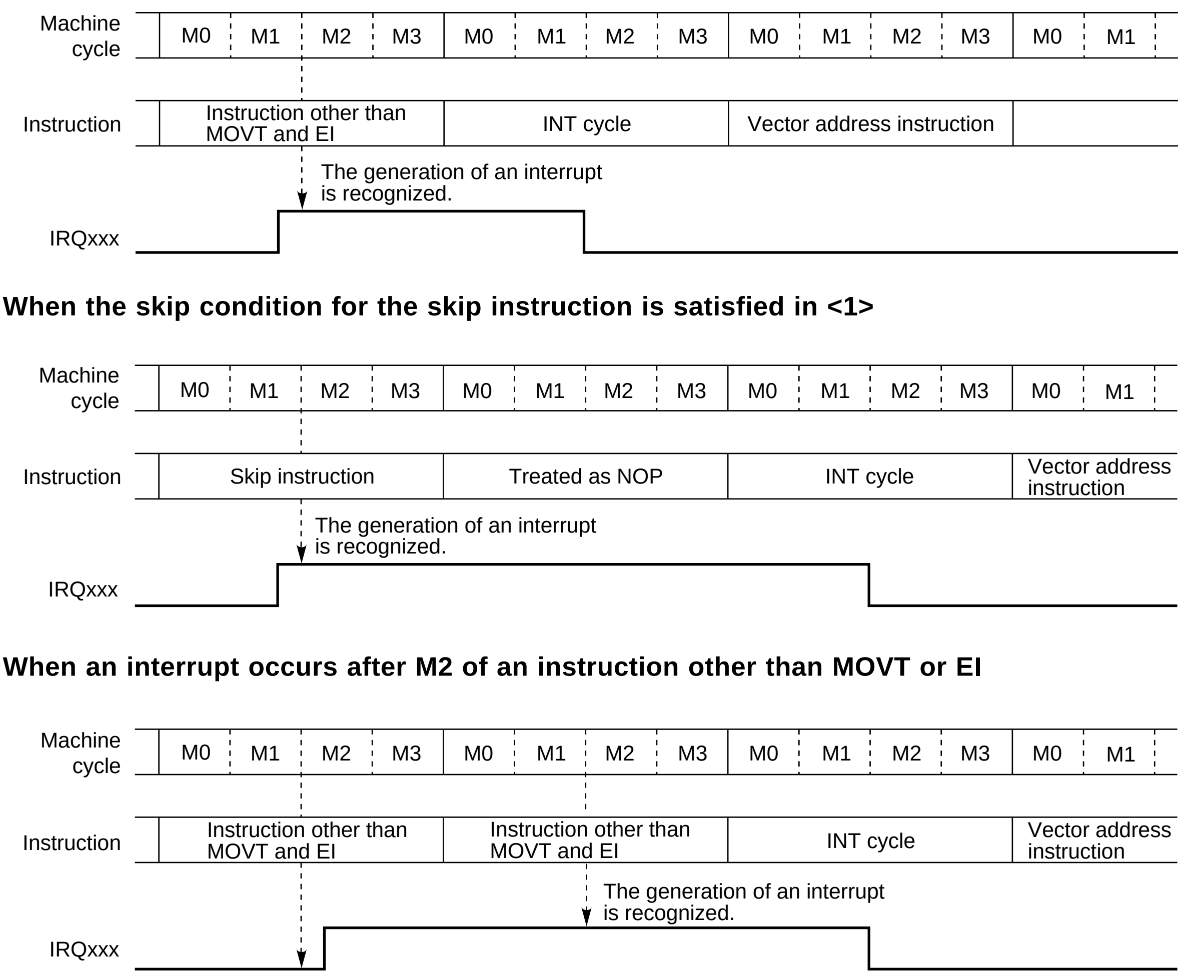
14.3.3 Timing for the Acceptance of an Interrupt

Figure 14-4 shows the timing diagram for the acceptance of an interrupt.

The μPD17149 takes 16 clock pulses to execute a single instruction. A period equal to these 16 clock pulses is called one instruction cycle. One instruction cycle is divided, into blocks of four clock pulses, called M0 to M3.

The program recognizes the generation of an interrupt on an edge signal occurring before M2.

Figure 14-4. Timing Diagram for the Acceptance of an Interrupt (When INTE = 1, IPxxx = 1) (1/3)



<1> When an interrupt occurs before M2 of an instruction other than MOVT or EI

Machine cycle
M0 M1 M2 M3 M0 M1 M2 M3 M0 M1 M2 M3 M0 M1
Instruction
Instruction other than MOVT and EI
INT cycle
Vector address instruction
The generation of an interrupt is recognized.
IRQxxx

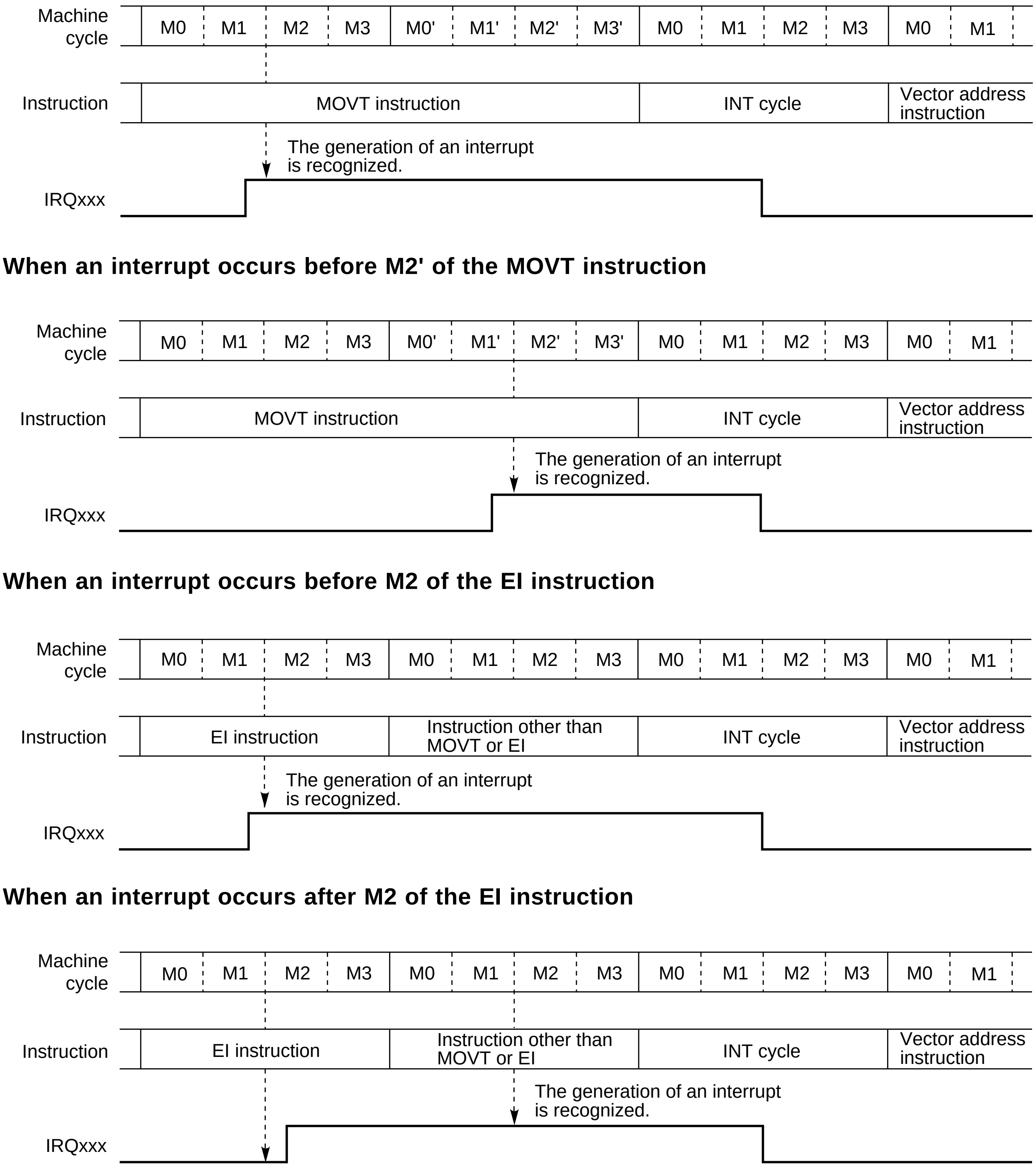
<2> When the skip condition for the skip instruction is satisfied in <1>

Machine cycle
M0 M1 M2 M3 M0 M1 M2 M3 M0 M1 M2 M3 M0 M1
Instruction
Skip instruction
Treated as NOP
INT cycle
Vector address instruction
The generation of an interrupt is recognized.
IRQxxx

<3> When an interrupt occurs after M2 of an instruction other than MOVT or EI

Machine cycle
M0 M1 M2 M3 M0 M1 M2 M3 M0 M1 M2 M3 M0 M1
Instruction
Instruction other than MOVT and EI
Instruction other than MOVT and EI
INT cycle
Vector address instruction
The generation of an interrupt is recognized.
IRQxxx

Figure 14-4. Timing Diagram for the Acceptance of an Interrupt (When INTE = 1, IPxxx = 1) (2/3)



<4> When an interrupt occurs before M2 of the MOVT instruction

Machine cycle
M0 M1 M2 M3 M0' M1' M2' M3' M0 M1 M2 M3 M0 M1
Instruction
MOVT instruction
INT cycle
Vector address instruction
The generation of an interrupt is recognized.
IRQxxx

<5> When an interrupt occurs before M2' of the MOVT instruction

Machine cycle
M0 M1 M2 M3 M0' M1' M2' M3' M0 M1 M2 M3 M0 M1
Instruction
MOVT instruction
INT cycle
Vector address instruction
The generation of an interrupt is recognized.
IRQxxx

<6> When an interrupt occurs before M2 of the EI instruction

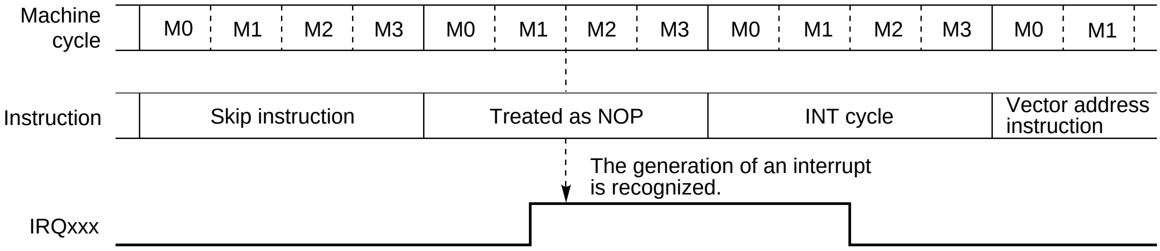
Machine cycle
M0 M1 M2 M3 M0 M1 M2 M3 M0 M1 M2 M3 M0 M1
Instruction
EI instruction
Instruction other than MOVT or EI
INT cycle
Vector address instruction
The generation of an interrupt is recognized.
IRQxxx

<7> When an interrupt occurs after M2 of the EI instruction

Machine cycle
M0 M1 M2 M3 M0 M1 M2 M3 M0 M1 M2 M3 M0 M1
Instruction
EI instruction
Instruction other than MOVT or EI
INT cycle
Vector address instruction
The generation of an interrupt is recognized.
IRQxxx

Figure 14-4. Timing Diagram for the Acceptance of an Interrupt (When INTE = 1, IPxxx = 1) (3/3)

<8> When an interrupt occurs during a skip operation (treated as NOP), instigated by a skip instruction



Remarks 1. The INT cycle is a preparatory cycle for an interrupt. During this cycle, the contents of PC and PSWORD are saved, and IRQxxx is cleared.

2. Exceptionally, the execution of the MOVT instruction requires two instruction cycles.

3. The EI instruction is designed to prevent multiple interrupts from occurring when control is returned from interrupt handling.

CHAPTER 15 STANDBY FUNCTION

15.1 OVERVIEW OF THE STANDBY FUNCTION

The µPD17149 can reduce its current by using the standby function. The standby function supports STOP and HALT modes.

In the STOP mode, the system clock is stopped and the CPU current is reduced to almost only a leak current. This mode is useful in retaining data memory contents without operating the CPU.

In the HALT mode, the oscillation of the system clock continues. However, the system clock is not supplied to the CPU, stopping CPU operation. In this mode, current reduction is less than that in the STOP mode. However, since the system clock is oscillating, operation can be started immediately after the HALT mode is released. In both STOP and HALT modes, the statuses of the data memory, registers, and output latches of the output port used immediately before the standby mode is set are maintained (except STOP 0000B). Therefore, in order to lower consumption current for the entire system, input/output port statuses should be set beforehand.

Table 15-1. Standby Mode Status

		STOP mode	HALT mode
Programmed instruction		STOP instruction	HALT instruction
System clock oscillator		Oscillation stopped	Oscillation continued
Operation status	CPU	• Operation stopped	
	RAM	• The contents held immediately before setting standby mode are retained.	
	Port	• The status existing immediately before setting standby mode is retained.Note	
	TM0	• Operable only when the INT input is selected as the count pulse. • Stopped when the system clock is selected. (The count is retained.)	• Operable
	TM1	• Operation stopped. (The count is reset to 0.) (Count-up is also inhibited.)	• Operable
	BTM	• Operation stopped. (The count is retained.)	• Operable
	SIO	• Operable only when the external clock is selected as the serial clock.Note	• Operable
	A/D	• Operation stoppedNote (ADCR <– 00H)	• Operable
	INT	• Operable	• Operable

Note When STOP 0000B is executed, all pins are switched to input port pins. This is also true for pins which are also used for other purposes other than port pins. *

Cautions 1. Always specify a NOP instruction immediately before STOP and HALT instructions.

2. When an interrupt request flag and the corresponding interrupt enable flag are both set, and the associated interrupt is specified as the standby mode release condition, standby mode is not set. *

15.2 HALT MODE

15.2.1 Setting HALT Mode

Executing a HALT instruction sets HALT mode.

Operand $b_3b_2b_1b_0$ of the HALT instruction indicates the HALT mode release conditions.

Table 15-2. HALT Mode Release Conditions

Format: HALT $b_3b_2b_1b_0$B

Bit	HALT mode release conditions**Note 1**
b_3	When this bit is 1, release by IRQxxx is permitted.**Notes 2, 4**
b_2	Always 0
b_1	When this bit is 1, forced release by IRQTM1 is permitted.**Notes 3, 4**
b_0	When this bit is 1, release by $\overline{\text{RLS}}$ input is permitted.**Note 4**

* **Notes 1.** When HALT 0000B is specified, HALT mode can be released only by reset ($\overline{\text{RESET}}$ input or POC).

2. IPxxx must be 1.

3. HALT mode is released regardless of the IPTM1 status.

4. If a HALT instruction is executed when IRQxxx = 1 or when the $\overline{\text{RLS}}$ input is low, the HALT instruction is ignored (treated as a NOP instruction), and HALT mode is not set.

15.2.2 Starting Address After HALT Mode is Released

The starting address depends on the release conditions and interrupt enable conditions.

Table 15-3. Starting Address After HALT Mode is Released

Release condition	Starting address after release
RESET**Note1**	Address 0
$\overline{\text{RLS}}$	Address subsequent to the HALT instruction
IRQxxx**Note2**	For DI, address subsequent to the HALT instruction
	For EI, interrupt vector (When more than one IRQxxx is set, the interrupt vector having the highest priority)

* **Notes 1.** $\overline{\text{RESET}}$ input or POC is valid as reset.

2. Except when forced release is made with IRQTM1, IPxxx must be 1.

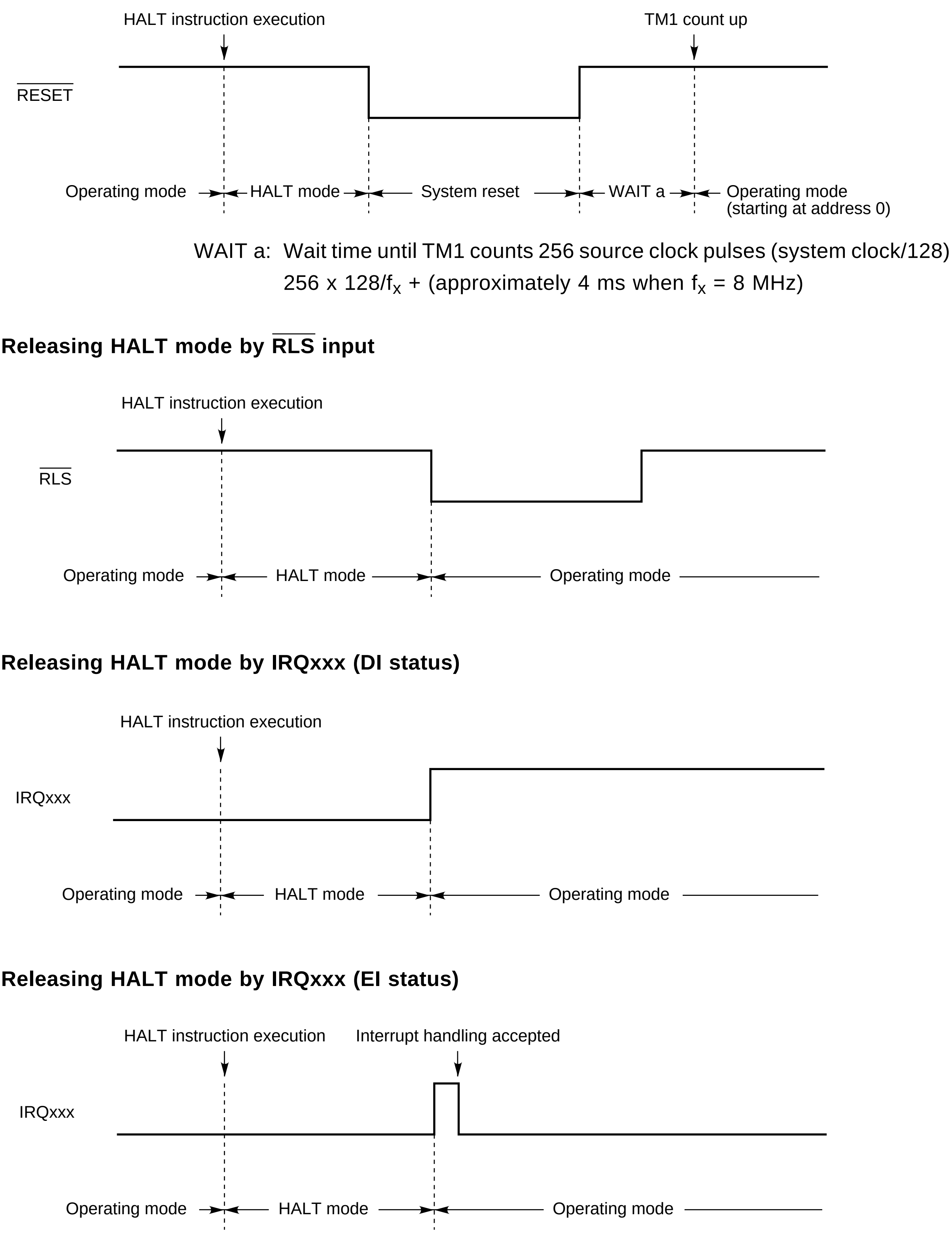


Figure 15-1. Releasing HALT Mode

(a) Releasing HALT mode by $\overline{\text{RESET}}$ input

WAIT a: Wait time until TM1 counts 256 source clock pulses (system clock/128)
256 x 128/f_X + (approximately 4 ms when f_X = 8 MHz)

(b) Releasing HALT mode by $\overline{\text{RLS}}$ input

(c) Releasing HALT mode by IRQxxx (DI status)

(d) Releasing HALT mode by IRQxxx (EI status)

15.2.3 HALT Setting Conditions

(1) Release by $\overline{\text{RLS}}$ input

No special register setting is needed.

Caution If the P0F$_0$/$\overline{\text{RLS}}$ pin is low when a HALT instruction is executed, the HALT instruction is ignored (regarded as a NOP instruction), such that HALT mode is not set.

(2) Forced release by IRQTM1

	Setting conditions
When the external clock is used for release	• Timer 0 and timer 1 are set as a 16-bit timer. (TM0CK1 = 1, TM0CK0 = 1, TM1CK1 = 1, TM1CK0 = 1) • Timer 0 and timer 1 are operable. (TM0EN = 1, TM1EN = 1) • The timer 1 interrupt request flag is cleared to 0. (IRQTM1 = 0)
When the internal clock is used for release	• Timer 1 is operable. • The timer 1 interrupt request flag is cleared to 0. (IRQTM1 = 0)

(3) Release by an interrupt request flag (IRQxxx)

- Ready the peripheral hardware to be used to release HALT mode.

Timer 0	Operable. (TM0EN = 1)
* Timer 1	Operable. (TM1EN = 1)
Timer 0 + timer 1	For timer 1, the count up signal output by timer 0 is used. (TM1CK1 = 1), TM1CK0 = 1) Timer 0 and timer 1 are operable. (TM0EN = 1, TM1EN = 1)
Basic interval timer	Always operable.
Serial interface	The serial interface circuit is operable. (SIOTS = 1, SIOEN = 1)
INT pin	Edge selection setting

- Clear the interrupt request flag (IRQxxx) of the peripheral hardware used for releasing HALT mode to 0.
- Set the interrupt enable flag (IPxxx) of the peripheral hardware to be used to release HALT mode to 1.

Caution Always specify a NOP instruction immediately before a HALT instruction. When a HALT instruction is preceded by a NOP instruction, the time needed to execute one instruction is generated between the IRQxxx manipulation instruction and the HALT instruction. Therefore, when the CLR1 IRQxxx instruction is specified, for example, the result of the IRQ333 clear operation is reflected in the execution of the HALT instruction. (See Example 1.) If a NOP instruction is not specified immediately before a HALT instruction, however, the result of executing the CLR1 IRQxxx instruction is not reflected in the HALT instruction, hence HALT mode is not set. (See Example 2.)

Example 1. Correct program

```
            :
(Sets IRQxxx.)
            :

CLR1      IRQxxx
NOP                ; Place a NOP instruction before the HALT instruction.
                   ; (Clearing IRQxxx correctly affects the HALT instruction.)
HALT      1000B    ; Executes the HALT instruction correctly (enters the HALT mode).
            :
```

Example 2. Incorrect program

```
            :
(Sets IRQxxx.)
            :

CLR1      IRQxxx   ; Clearing IRQxxx does not affect the HALT instruction.
                   ; (It affects the instruction after the HALT instruction.)
HALT      1000B    ; Ignores the HALT instruction (does not enter the HALT mode).
            :
```

15.3 STOP MODE

15.3.1 Setting STOP Mode

Executing a STOP instruction results in STOP mode being set.

Operand $b_3b_2b_1b_0$ of the STOP instruction indicates the STOP mode release conditions.

Table 15-4. STOP Mode Release Conditions

Format: STOP $b_3b_2b_1b_0$B

Bit	STOP mode release condition[Note 1]
b_3	When this bit is 1, release by IRQxxx is permitted.[Notes 2, 4]
b_2	Always 0
b_1	Always 0
b_0	When this bit is 1, release by $\overline{\text{RLS}}$ input is permitted.[Note 3, 4]

* **Notes 1.** When STOP 0000B is specified, STOP mode can be released only with reset ($\overline{\text{RESET}}$ input or POC). When STOP 0000B is executed, the microcomputer is initialized to the state existing immediately after the reset.

2. IPxxx must be 1. STOP mode cannot be released with IRQTM1.

3. Setting only b_0 to 1 is not allowed. (STOP 0001B is inhibited.) To set b_0 to 1, b_3 must also be set to 1.

* **4.** Even when the STOP instruction is executed when IRQxxx = 1 or when the $\overline{\text{RLS}}$ input is low, STOP mode is ignored (regarded as a NOP instruction), such that STOP mode is not set.

15.3.2 Starting Address After STOP Mode is Released

The starting address depends on the release conditions and interrupt enable conditions.

Table 15-5. Starting Address After STOP Mode is Released

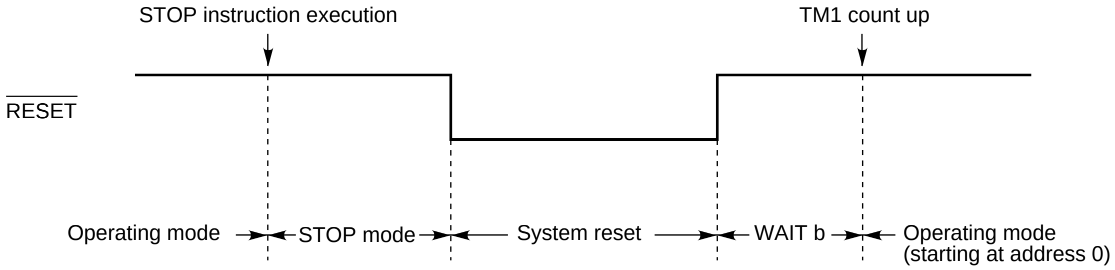
Release condition	Starting address after release
RESET[Note 1]	Address 0
$\overline{\text{RLS}}$	Address subsequent to the STOP instruction
IRQxxx[Note 2]	For DI, address subsequent to the STOP instruction
	For EI, interrupt vector (When more than one IRQxxx is set, the interrupt vector having the highest priority)

* **Notes 1.** Only $\overline{\text{RESET}}$ input and POC are valid as reset.

2. IPxxx must be 1. STOP mode cannot be released with IRQTM1.

Figure 15-2. Releasing STOP Mode (1/2)

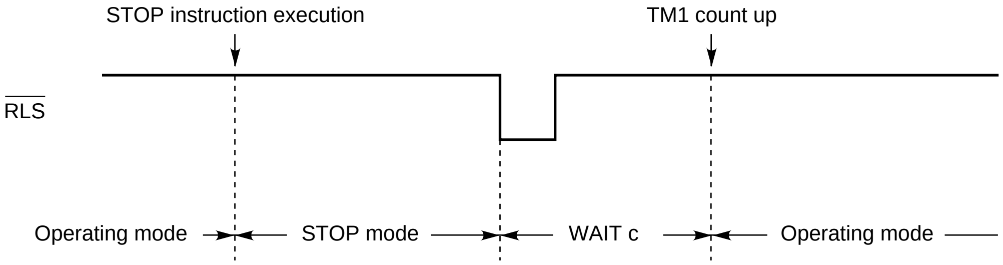
(a) Releasing STOP mode $\overline{\text{RESET}}$ input



STOP instruction execution
TM1 count up
RESET
Operating mode
STOP mode
System reset
WAIT b
Operating mode (starting at address 0)

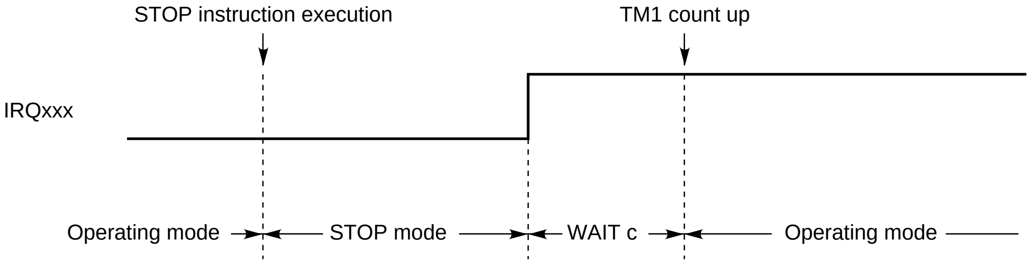
WAIT b: Wait time until TM1 counts 256 source clock pulses(system clock/128)
256 x 128/f_X + α (approximately 4 ms when f_X = 8 MHz)
α: Oscillation development time (which depends on the resonator)

(b) Releasing STOP mode by $\overline{\text{RLS}}$ input



WAIT c: Wait time until TM1 counts n + 1 source clock pulses (system clock/m) *
(n + 1) x m/f_X + α (n and m are the values used immediately before STOP mode is set)
α: Oscillation development time (which depends on the resonator)

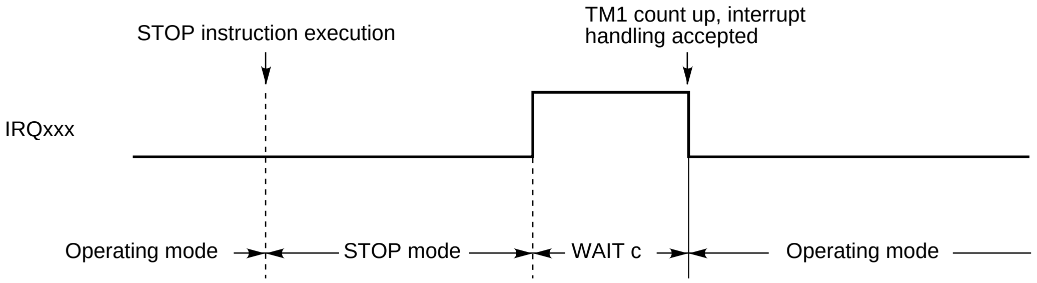
(c) Releasing STOP mode by IRQxxx (DI status)



WAIT c: Wait time until TM1 counts n + 1 source clock pulses (system clock/m) *
(n + 1) x m/f_X + α (n and m are the values used immediately before STOP mode is set)
α: Oscillation development time (which depends on the resonator)

Figure 15-2. Releasing STOP Mode (2/2)

(d) Releasing STOP mode by IRQxxx (EI status)


STOP instruction execution
TM1 count up, interrupt handling accepted
IRQxxx
Operating mode
STOP mode
WAIT c
Operating mode

* WAIT c: Wait time until TM1 counts n + 1 source clock pulses (system clock/m)
(n + 1) x m/f_X + α (n and m are the values used immediately before STOP mode is set)

α: Oscillation development time (which depends on the resonator)

15.3.3 STOP Setting Conditions

(1) Release by $\overline{\text{RLS}}$ input

* • Set the modulo register value for timer 1 (generation of an oscillation settling time).

* **Caution When the $\text{POF}_0/\overline{\text{RLS}}$ pin is low during the STOP instruction execution, the STOP instruction is ignored (regarded as a NOP instruction) and STOP mode is not set.**

(2) Release by IRQxxx

Release by IRQ pin	• Select an edge (IEGMD1, IEGMD0) for the signal input to the INT pin. • Set the modulo register value for timer 1 (generation of an oscillation settling time). • Clear the INT pin interrupt request flag (IRQ) to 0. • Set the INT pin interrupt enable flag (IP) to 1.
Release by IRQSIO	• Set the external clock input on the $\overline{\text{SCK}}$ pin as the source clock (SIOCK1 = 0, SIOCK0 = 0). • Make the serial interface operable (SIOTS = 1). • Set the modulo register value for timer 1 (generation of an oscillation settling time). • Clear the serial interface interrupt request flag (IRQSIO) to 0. • Set the serial interface interrupt enable flag (IPSIO) to 1.
Release by IRQTM	• Set the external clock input on the INT pin as the source clock of timer 0 (TM0CK1 = 1,TM0CK0 = 1). • Set the modulo register value for timer 0. • Set the modulo register value and source clock for timer 1 (generation of an oscillation settling time). • Make timer 0 operable (TM0EN = 1). • Clear the timer 0 interrupt request flag (IRQTM0) to 0. • Set the timer 0 interrupt enable flag (IPTM0) to 1.

Caution Always specify a NOP instruction immediately before a STOP instruction. When a STOP instruction is preceded by a NOP instruction, the time needed to execute one instruction is generated between the IRQxxx manipulation instruction and the STOP instruction. Therefore, when the CLR1 IRQxxx instruction is specified, for example, the result of the IRQxxx clear operation is reflected in the execution of the STOP instruction. (See Example 1.) If a NOP instruction is not specified immediately before a STOP instruction, however, the result of executing the CLR1 IRQxxx instruction is not reflected in the STOP instruction, hence STOP mode is not set. (See Example 2.)

Example 1. Correct program

```
                 :
                 :
        (Sets IRQxxx.)
                 :
                 :

        CLR1     IRQxxx
        NOP               ; Place a NOP instruction before the STOP instruction.
                          ; (Clearing IRQxxx correctly affects the STOP instruction.)
        STOP     1000B    ; Executes the STOP instruction correctly (enters the STOP mode).
                 :
                 :
```

Example 2. Incorrect program

```
                 :
                 :

        (Sets IRQxxx.)
                 :
                 :
        CLR1     IRQxxx   ; Clearing IRQxxx does not affect the STOP instruction.
                          ; (It affects the instruction after the STOP instruction.)
        STOP     1000B    ; Ignores the STOP instruction (does not enter the STOP mode).
                 :
                 :
```

CHAPTER 16 RESET

The µPD17149 is reset when a reset signal is applied to the $\overline{\text{RESET}}$ pin, when the incorporated POC circuit detects a supply voltage drop, when the watchdog timer function detects a program crush, or when the address stack overflows or underflows. The incorporated POC circuit is specified with the mask option.

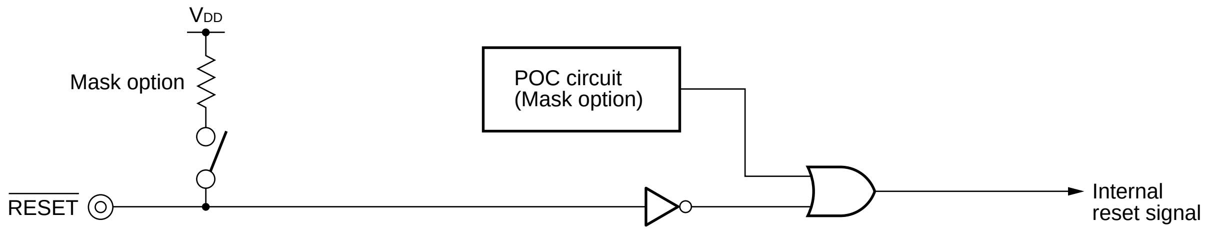
16.1 RESET FUNCTIONS

The reset functions are used to initialize device operations. The operations initialized depend on the reset type.

Table 16-1. Hardware Statuses after Reset

Reset type Hardware		• $\overline{\text{RESET}}$ input during operation • Reset by the incorporated POC circuit	• $\overline{\text{RESET}}$ input in the standby mode • Reset by the incorporated POC circuit in the standby mode	• Watchdog timer overflow • Stack overflow or underflow
Program counter		0000H	0000H	0000H
Port	Input/output	Input	Input	Input
	Output latch content	0	0	Not defined
General-purpose data memory	General- purpose data memory (excluding DBF)	Not defined	Status before reset is retained.	Not defined *
	DBF	Not defined	Not defined	Not defined
	System register (excluding WR)	0	0	0
	WR	Not defined	Status before reset is retained.	Not defined *
Control register		SP = 5H, IRQTM1 = 1, TM1EN = 1, IRQBTM = 1, and INT indicate the current status of the INT pin. The others are 0. See **Chapter 9**.		SP = 5H and INT indicate the current status of the INT pin. The others retain their statuses before reset.
Timer 0 and timer 1	Count register	00H	00H	Timer 0: 00H Timer 1: Not defined
	Modulo register	FFH	FFH	FFH
Basic interval timer binary counter		Not defined	Not defined	Not defined. 40H for watchdog timer overflow *
Serial interface	Shift register (SIOSFR)	Not defined	Status before reset is retained.	Not defined
	Serial output latch	0	0	Not defined
A/D converter data register (ADCR)		00H	00H	00H

Figure 16-1. Reset Block Configuration



16.2 RESETTING

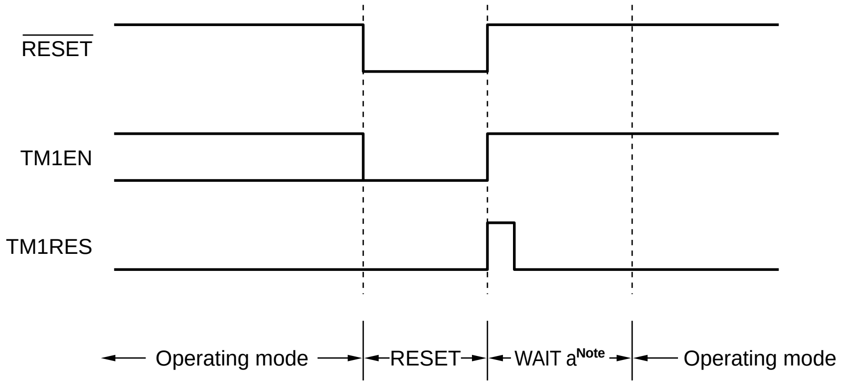
Operation when reset is caused by $\overline{RESET}$ input is shown in Figure 16-2.

If the $\overline{RESET}$ pin is set from low to high, system clock generation starts and the timer 1 generates an oscillation settling time. Program execution starts from address 0000H.

The controller also operates in the same way when the POC circuit causes a reset.

At watchdog timer overflow reset or overflow and underflow reset of the address stack register, an oscillation settling time (WAIT a) is not generated. Operation starts from address 0000H after initial statuses are internally set.

Figure 16-2. Resetting



Note This is an oscillation settling time. Operating mode is set when timer 1 counts system clocks (f_X) 128 x 256 times (approximately 4 ms when f_X = 8 MHz).

CHAPTER 17   POC CIRCUIT (MASK OPTION)

The POC circuit monitors the power supply voltage. It resets the microcomputer when the power is turned on/off. It can be used in application circuits using clock frequencies (f_X) of between 400 kHz and 4 MHz.

The POC circuit can be included in the µPD17149 by specifying a mask option. The µPD17P149, however, cannot include the POC circuit.

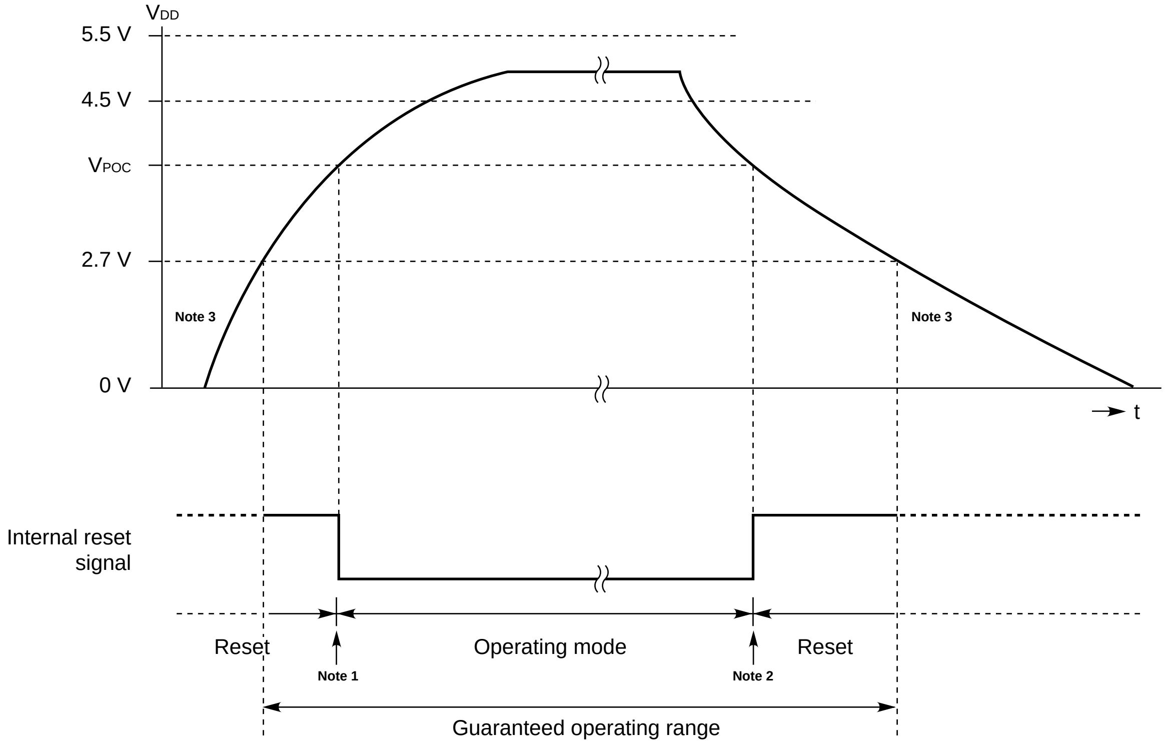
17.1 FUNCTIONS OF THE POC CIRCUIT

The POC circuit operates as follows:

- When V_{DD} - V_{POC}, an internal reset signal is generated.
- When V_{DD} > V_{POC}, the internal reset signal is released.

  (V_{DD}: power voltage, V_{POC}: POC-detected voltage)

Figure 17-1. Operation of the POC Circuit



Notes 1. In actual operation, an oscillation settling time, controlled by timer 1, is inserted before the operating mode is set. The oscillation settling time is equal to the time needed to execute approximately 2048 instructions (approximately 8 ms at 4 MHz).

2. A power supply voltage fall can cause a reset only when V_{POC} or a lower voltage level is maintained for a duration equal to at least the reset detection pulse width t_{SAMP}. Thus, there is a delay of up to t_{SAMP} before reset.

3. When the supply voltage (V_{DD}) is less than 2.7 V, operations of all functions of the μPD17149 are not guaranteed. The POC circuit, however, is designed to generate an internal reset signal whenever possible, regardless of whether an oscillation is received. So, an internal reset should occur when the voltage reaches a level at which the internal circuit can operate.

Remark For the values of V_{POC} and t_{SAMP}, refer to "ELECTRICAL CHARACTERISTICS" of the Data Sheet.

17.2 CONDITIONS UNDER WHICH THE POC CIRCUIT MAY BE USED

The POC circuit can be used when the application circuit satisfies the following conditions:

- The application circuit is not required to provide high reliability[Note].
- The supply voltage (V_{DD}) of the application circuit is between 4.5 V and 5.5 V.
- The system clock frequency (f_X) of the application circuit is between 400 kHz and 4 MHz.
- The supply voltage (V_{DD}) characteristics satisfy the POC circuit specifications (see **Section 17.4**).

Note When the POC circuit is used with an application circuit requiring high reliability, be sure to design the POC circuit so that the $\overline{\text{RESET}}$ signal is input from the outside. *

Caution When the POC circuit is used, the current drawn in standby mode will be slightly higher than when the circuit is not used.

Remark POC circuit operation is guaranteed at 2.7 V to 5.5 V.

17.3 CAUTIONS FOR USING THE POC CIRCUIT

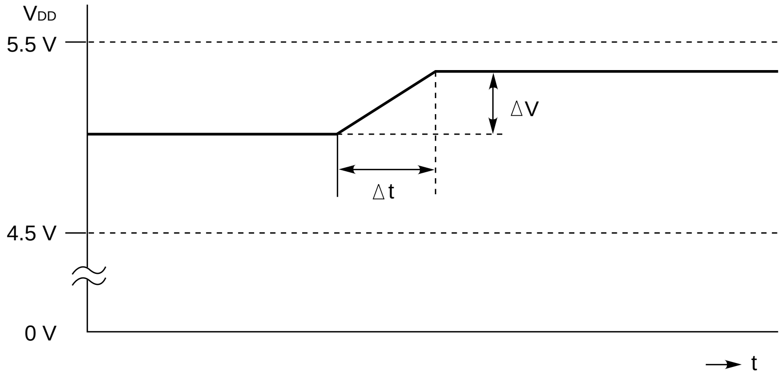
The POC circuit is designed in a fail-safe configuration. It has an auxiliary function by which if the supply voltage changes abruptly even within the rated range (V_{DD} = 4.5 to 5.5 V), a reset signal is issued as much as possible to avoid program crashes.[(Note)]

Note that if the supply voltage does not meet the conditions listed below, the POC circuit may cause a reset.

- **Standard and special (A) products**
  - The variation (ĐV) of the supply voltage must be within 100 mV.
  - If the variation of the supply voltage is not within 100 mV, the variation slant (ĐV/Đt) must be within 3 mV/µs.

- **Special (A1) products**
  - The variation (ĐV) of the supply voltage must be within 80 mV.
  - If the variation of the supply voltage is not within 80 mV, the variation slant (ĐV/Đt) must be within 2 mV/µs.

Figure 17-2. Supply Voltage Variation



Note The reset function based on supply voltage variation is of auxiliary nature. It does not assure that a reset occurs without failure. There is no condition to assure that a reset occurs based on this auxiliary function. Do not count on this function when designing.

17.4 SUPPLY VOLTAGE CHARACTERISTIC CONSIDERATIONS AND POC CIRCUIT SPECIFICATIONS

17.4.1 Supply Voltage Fall Speed t_{POCS}

Consider an application circuit operating at supply voltage V (4.5 - V - 5.5 V). For simplicity, suppose that the power supply voltage fall shows a simple attenuation characteristic, and that the characteristic v(t) can be expressed as follows:

$v(t) = V\ \varepsilon^{-(t/T)}$ (T: Time constant for the voltage fall)

Immediately after the voltage starts falling (t = 0), the fall speed (inclination of the fall) is maximized. At this point, the specification of the supply voltage fall speed t_{POCS} must be satisfied.

Therefore, the power supply must have the following voltage fall time constant:

$T = V/t_{POCS}$

Substituting actual values into the expression, we have:

T = 5.5[V]/0.08[V/ms] = 68.75 [ms]

Therefore, a minimum time constant of approximately 70 ms is required for the power supply voltage fall.

Caution The above example assumes that the supply voltage fall characteristic is expressed as v(t) = V $\varepsilon^{-(t/T)}$. This example should be used only as a guideline for application circuit design.

17.4.2 Reset Detection Pulse Width t_{SAMP}

A power supply voltage fall can cause a reset only when V_{POC} or a lower voltage level is maintained for a period of at least the reset detection pulse width t_{SAMP}.

Even when a momentary power failure occurs, reset by the POC circuit prevents a runaway status if the period of t_{SAMP} is assured.

17.5 CHECKING THE POC CIRCUIT OPERATION STATUS EXTERNALLY

Perform programming so that a low signal is output from an externally pulled-up I/O port.

When the POC circuit functions and causes a reset, the I/O port is set to input mode. So, the port goes high.

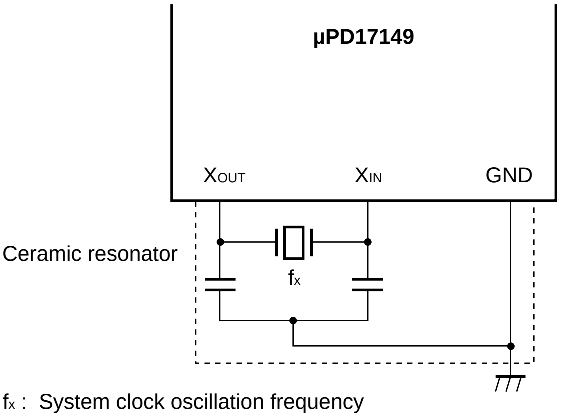
By observing the port status, POC circuit operation within the µPD17149 operating supply voltage range (V_{DD} = 2.7 V to 5.5 V) can be checked externally.

[MEMO]

CHAPTER 18   NOTES ON SYSTEM CLOCK OSCILLATOR CONFIGURATION

The system clock oscillator uses a ceramic resonator connected to pins X_{IN} and X_{OUT}. *
Figure 18-1 shows the external circuitry used to configure the system clock oscillator.

Figure 18-1. External Circuit of the System Clock Oscillator

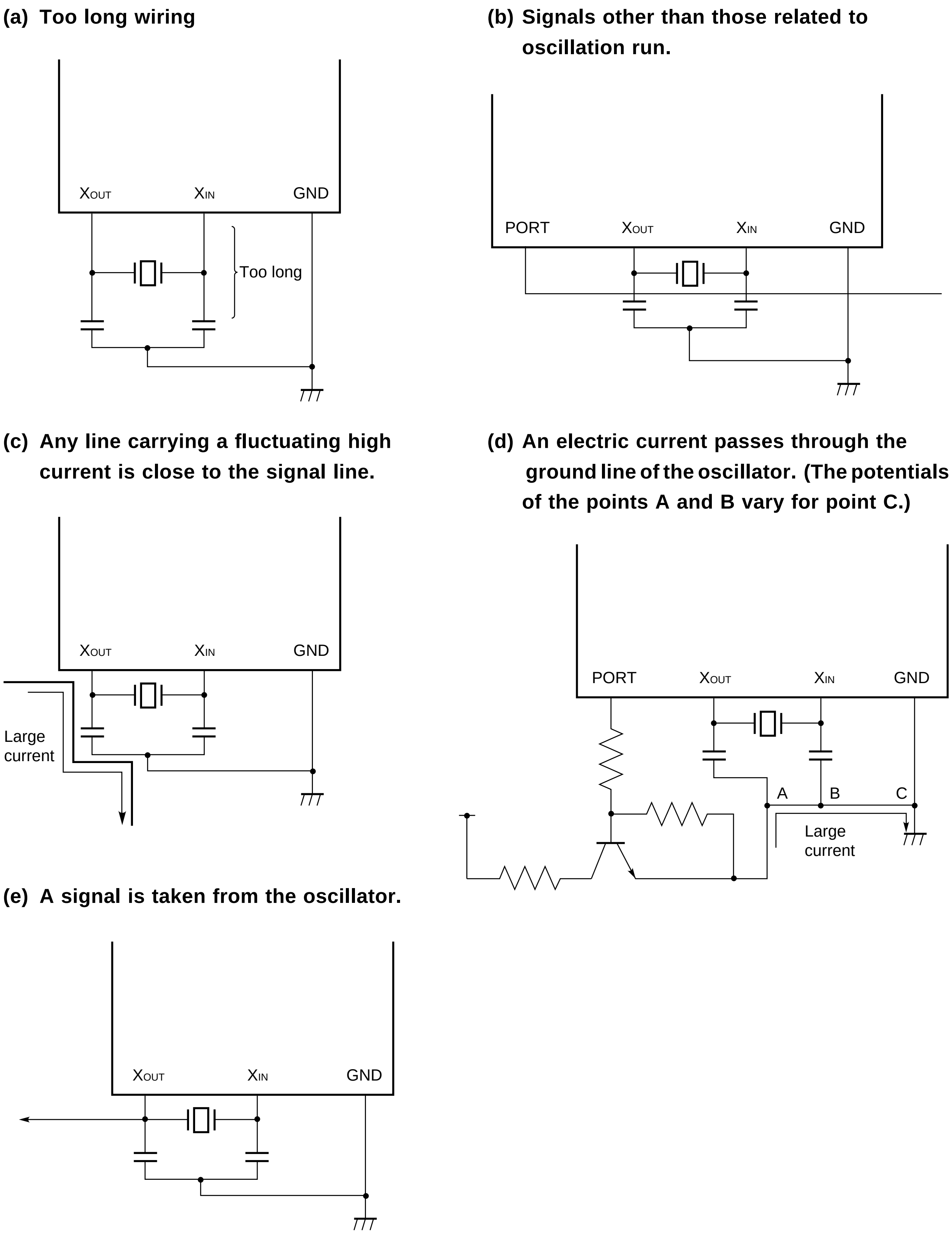


Caution Design a system clock oscillator so that the resistance and inductance of ground patterns are minimized. Conform to the following guidelines when wiring at the portions surrounded by dotted lines in Figure 18-1 to eliminate the influence of the wiring capacity.

- **The wiring must be as short as possible.**
- **Signals other than those related to oscillation must not run in these areas. Any line carrying a fluctuating high current must be kept away as far as possible.**
- **The grounding point of the capacitor of the oscillator must have the same potential as that of V_{SS}. It must not be grounded to ground patterns carrying a large current.**
- **No signal must be taken from the oscillator.**

Figure 18-2 shows bad examples of a system clock oscillator.

Figure 18-2. Bad Examples of a System Clock Oscillator



(a) Too long wiring

(b) Signals other than those related to oscillation run.

(c) Any line carrying a fluctuating high current is close to the signal line.

(d) An electric current passes through the ground line of the oscillator. (The potentials of the points A and B vary for point C.)

(e) A signal is taken from the oscillator.

CHAPTER 19 WRITING TO AND VERIFYING ONE-TIME PROM

The µPD17P149's internal program memory consists of a 4096 x 16 bit one-time PROM.

Writing to the one-time PROM or verifying the contents of the PROM is accomplished using the pins shown in Table 19-1. Note that address inputs are not used; instead, the address is updated using the clock input from the CLK pin.

Caution The $P0F_0/\overline{RLS}/V_{PP}$ pin is used as the V_{PP} pin when writing to program memory or verifying its contents. If an voltage equal to or more than V_{DD} + 0.3 V is applied to the $P0F_0/\overline{RLS}$ pin in normal operation mode, the microcontroller may cause a system crash. Protect the pins from high voltages.

Table 19-1. Pins Used When Writing to Program Memory or Verifying Its Contents

Pin	Function
V_{PP}	Pin for applying programming supply voltage. Voltage (+12.5 V) is applied to this pin.
V_{DD}	Positive power supply pin. +6 V is applied to this pin.
CLK	Input pin for address update clocks. Input of four pulses to this pin updates the address of the program memory.
MD_0 - MD_3	Input pins that select an operation mode
D_0 - D_7	Input/output pins for 8-bit data

19.1 DIFFERENCES BETWEEN MASK ROM PRODUCTS AND A ONE-TIME PROM PRODUCT

The µPD17P149 is a one-time PROM version of the µPD17149. The program memory of the µPD17149, which has a mask ROM, is replaced with a one-time PROM.

Table19-2 shows differences between mask ROM products and a one-time PROM product.

The differences are only the mask ROM size and whether the mask option can be specified. The CPU functions and built-in peripheral hardware are the same in each product. Therefore, the µPD17P149 is suited for evaluating a program in developing a system which uses the µPD17145, µPD17147, or µPD17149.

Table 19-2. Differences between Mask ROM Products and a One-Time PROM Product

Item	µPD17145	µPD17147	µPD17149	µPD17P149
ROM	Mask ROM			One-time PROM
	1024 x 16 bits (0000H–03FFH)	2048 x 16 bits (0000H–07FFH)	4096 x 16 bits (0000H–0FFFH)	
Program counter (PC)	10 bits	11 bits	12 bits	
* Address register (AR)				
Address stack register				
Pull-up resistors of the P0F, $\overline{\text{RESET}}$, and INT pins	Mask option			None
POC circuit	Mask option			None
V_{PP} pin and operating mode selection pin	None			Provided
* Quality grade	Standard Special [(A),(A1)]			Standard

* **Caution Although a PROM product is highly compatible with a mask ROM product in respect of functions, they differ in internal ROM circuits and part of electrical characteristics. Before changing the PROM product to the mask ROM product in an application system, evaluate the system carefully using the mask ROM product.**

19.2 PROGRAM MEMORY WRITE/VERIFY MODES

If +6 V is applied to the V_{DD} pin and +12.5 V is applied to the V_{PP} pin after a certain duration of reset status (V_{DD} = 5 V, $\overline{\text{RESET}}$ = 0 V), the µPD17P149 enters program memory write/verify mode. A specific operating mode is then selected by setting the MD_0 through MD_3 pins as follows. Leave the X_{OUT} pin open. Connect all pins other than those listed in Table 19-1 (including the $\overline{\text{RESET}}$ pin) to GND through pull-down resistors.

Table 19-3. Specification of Operating Modes

Operating mode specification						Operating mode
V_{PP}	V_{DD}	MD_0	MD_1	MD_2	MD_3	
+12.5 V	+6 V	H	L	H	L	Program memory address clear mode
		L	H	H	H	Write mode
		L	L	H	H	Verify mode
		H	x	H	H	Program inhibit mode

Remark x: Don’t care. L (low) or H (high)

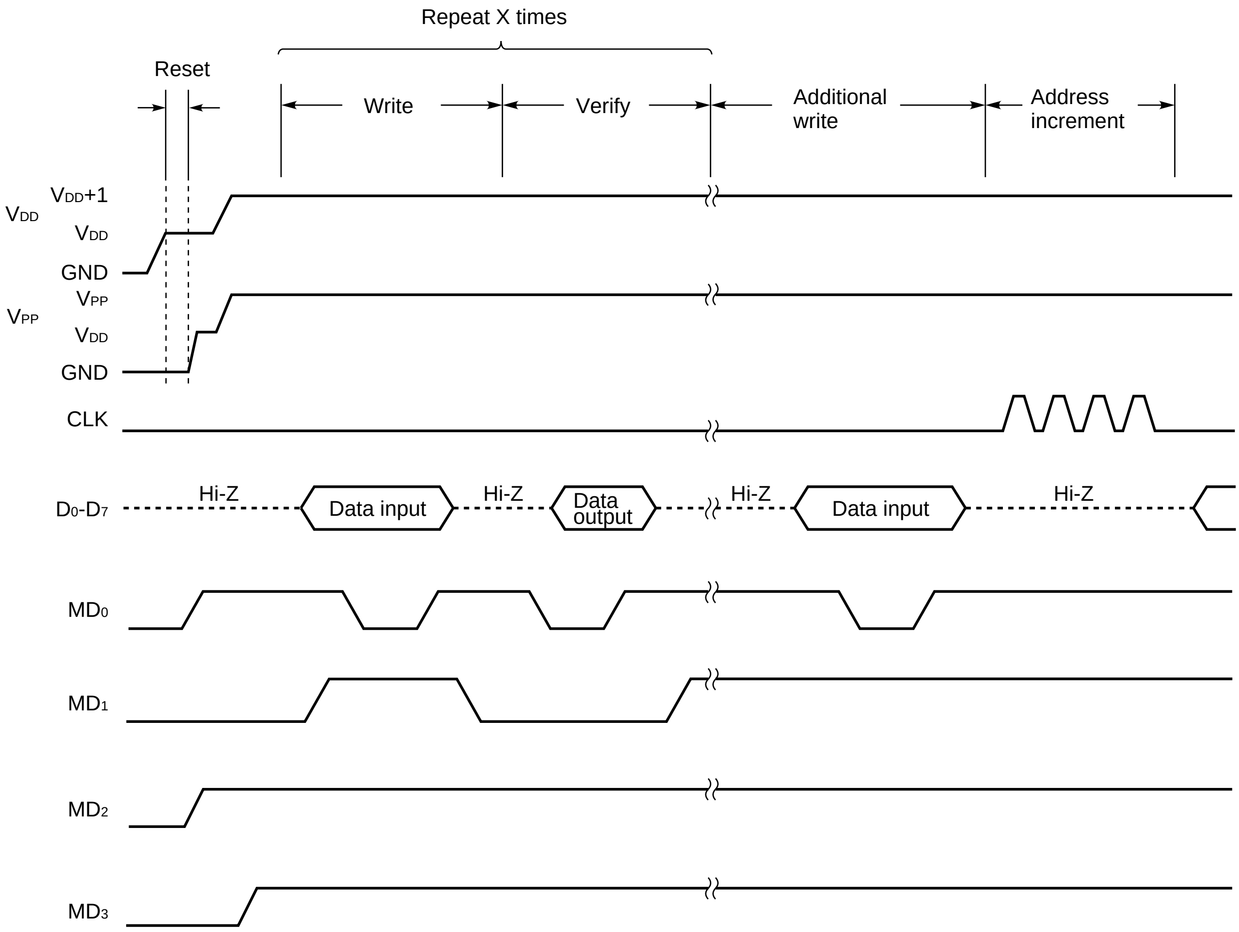
19.3 WRITING TO PROGRAM MEMORY

The procedure for writing to program memory is described below.

(1) Connect all unused pins to GND through resistors (the X_{OUT} pin is left open). Apply a low-level signal to the CLK pin.
(2) Apply 5 V to V_{DD} and apply a low-level signal to the V_{PP} pin.
(3) Wait 10 μs. Then apply 5 V to V_{PP}.
(4) Set the mode selection pins to program memory address clear mode.
(5) Apply 6 V to V_{DD} and 12.5 V to V_{PP}.
(6) Select program inhibit mode.
(7) Write data in 1-ms write mode.
(8) Select program inhibit mode.
(9) Select verify mode. If the write operation is found successful, proceed to step (10). If the operation is found unsuccessful, repeat steps (7) to (9).
(10) Perform additional write for X (number of repetitions of steps (7) to (9)) x 1 ms.
(11) Select program inhibit mode.
(12) Increment the program memory address by one on reception of four pulses on the CLK pin.
(13) Repeat steps (7) to (12) until the last address is reached.
(14) Select program memory address clear mode.
(15) Apply 5 V to the V_{DD} and V_{PP} pins.
(16) Turn power off.

A timing chart for program memory writing steps (2) to (12) is shown in Figure 19-1.

Figure 19-1. Timing Chart for Program Memory Writing Steps



19.4 READING PROGRAM MEMORY

(1) Connect all unused pins to GND through resistors (the X_{OUT} pin is left open). Apply a low-level signal to the CLK pin.

(2) Apply 5 V to V_{DD} and apply a low-level signal to the V_{PP} pin.

(3) Wait 10 μs. Then apply 5 V to V_{PP}.

(4) Set the mode selection pins to program memory address clear mode.

(5) Apply 6 V to V_{DD} and 12.5 V to V_{PP}.

(6) Select program inhibit mode.

(7) Select verify mode. Data is output sequentially one address at a time for every four input clock pulses on the CLK.

(8) Select program inhibit mode.

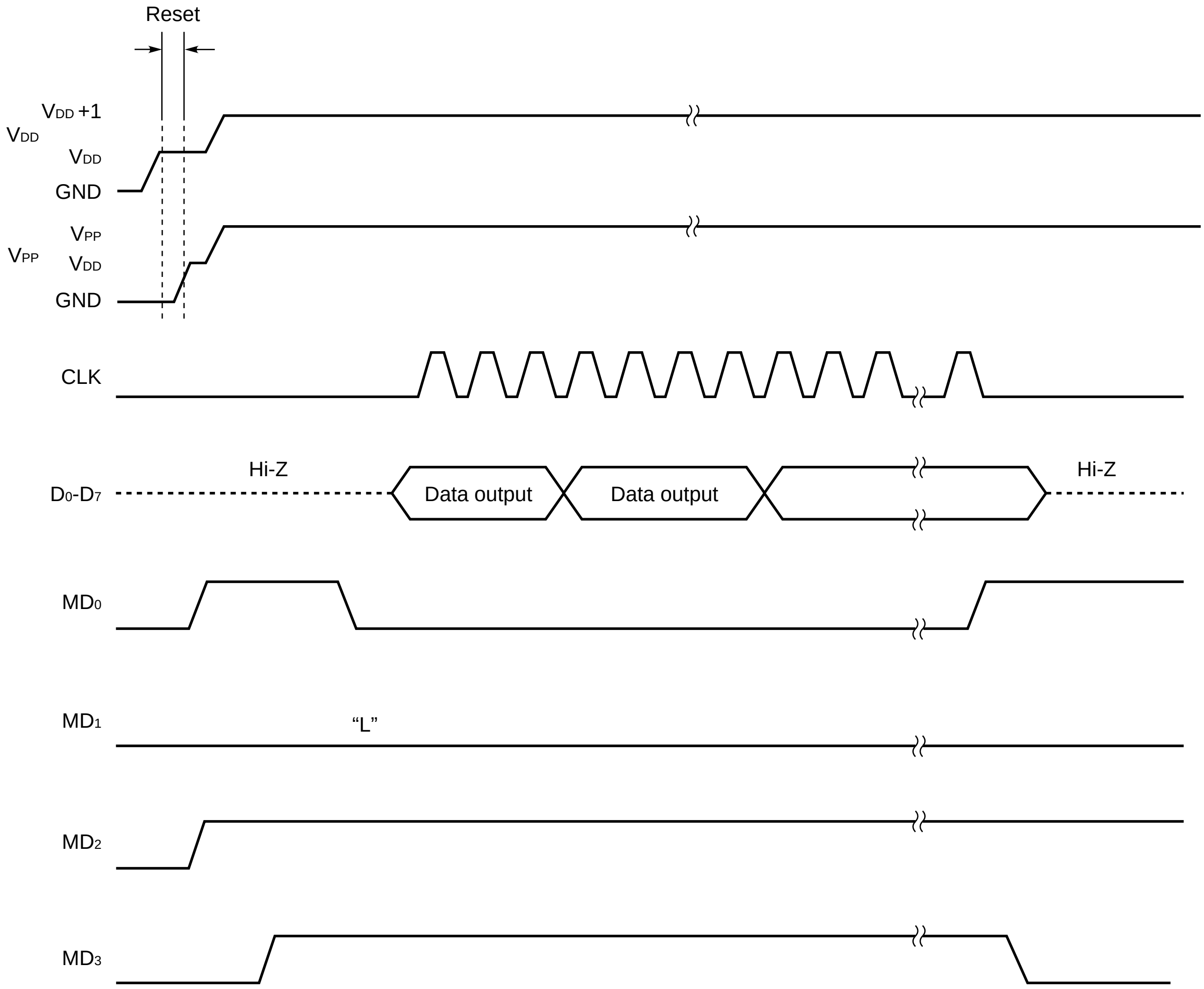
(9) Select program memory address clear mode.

(10) Apply 5 V to the V_{DD} and V_{PP} pins.

(11) Turn power off.

A timing chart for program memory reading steps (2) to (9) is shown below.

Figure 19-2. Timing Chart for Program Memory Reading Steps



CHAPTER 20 INSTRUCTION SET

20.1 OVERVIEW OF THE INSTRUCTION SET

(1/2)

b_{14}-b_{11} \ b_{15}		0		1	
BIN	HEX				
0000	0	ADD	r, m	ADD	m, #n4
0001	1	SUB	r, m	SUB	m, #n4
0010	2	ADDC	r, m	ADDC	m, #n4
0011	3	SUBC	r, m	SUBC	m, #n4
0100	4	AND	r, m	AND	m, #n4
0101	5	XOR	r, m	XOR	m, #n4
0110	6	OR	r, m	OR	m, #n4
0111	7	INC	AR		
		INC	IX		
		MOVT	DBF, @AR		
		BR	@AR		
		CALL	@AR		
		RET			
		RETSK			
		EI			
		DI			
		RETI			
		PUSH	AR		
		POP	AR		
		GET	DBF, p		
		PUT	p, DBF		
		PEEK	WR, rf		
		POKE	rf, WR		
		RORC	r		
		STOP	s		
		HALT	h		
		NOP			

(2/2)

b_{15} b_{14}-b_{11}		0			1	
BIN	HEX					
1000	8	LD	r, m		ST	m, r
1001	9	SKE	m, #n4		SKGE	m, #n4
1010	A	MOV	@r, m		MOV	m, @r
1011	B	SKNE	m, #n4		SKLT	m, #n4
1100	C	BR	addr	(Page 0)	CALL	addr
1101	D	BR	addr	(Page 1)	MOV	m, #n4
1110	E				SKT	m, #n
1111	F				SKF	m, #n

20.2 LEGEND

AR	Address register
ASR	Address stack register pointed to by the stack pointer
addr	Program memory address (11 low-order bits)
BANK	Bank register
CMP	Compare flag
CY	Carry flag
DBF	Data buffer
h	Halt release condition
INTEF	Interrupt enable flag
INTR	Register automatically saved in the stack when an interrupt occurs
INTSK	Interrupt stack register
IX	Index register
MP	Data memory row address pointer
MPE	Memory pointer enable flag
m	Data memory address specified by m_R and m_C
m_R	Data memory row address (high-order)
m_C	Data memory column address (low-order)
n	Bit position (four bits)
n4	Immediate data (four bits)
PAGE	Page (bit 11 of the program counter)
PC	Program counter
p	Peripheral address
p_H	Peripheral address (three high-order bits)
p_L	Peripheral address (four low-order bits)
r	General register column address
rf	Register file address
rf_R	Register file row address (three high-order bits)
rf_C	Register file column address (four low-order bits)
SP	Stack pointer
s	Stop release condition
WR	Window register
(x)	Contents of x

20.3 LIST OF THE INSTRUCTION

(1/2)

Instruction set	Mnemonic	Operand	Operation	Instruction code			
				Op code	Operand		
Add	ADD	r, m	(r) <– (r) + (m)	00000	m_R	m_C	r
		m, #n4	(m) <– (m) + n4	10000	m_R	m_C	n4
	ADDC	r, m	(r) <– (r) + (m) + CY	00010	m_R	m_C	r
		m, #n4	(m) <– (m) + n4 + CY	10010	m_R	m_C	n4
	INC	AR	AR <– AR + 1	00111	000	1001	0000
		IX	IX <– IX + 1	00111	000	1000	0000
Subtract	SUB	r, m	(r) <– (r) – (m)	00001	m_R	m_C	r
		m, #n4	(m) <– (m) – n4	10001	m_R	m_C	n4
	SUBC	r, m	(r) <– (r) – (m) – CY	00011	m_R	m_C	r
		m, #n4	(m) <– (m) – n4 – CY	10011	m_R	m_C	n4
Logical operation	OR	r, m	(r) <– (r) ∨ (m)	00110	m_R	m_C	r
		m, #n4	(m) <– (m) ∨ n4	10110	m_R	m_C	n4
	AND	r, m	(r) <– (r) ∧ (m)	00100	m_R	m_C	r
		m, #n4	(m) <– (m) ∧ n4	10100	m_R	m_C	n4
	XOR	r, m	(r) <– (r) ⊻ (m)	00101	m_R	m_C	r
		m, #n4	(m) <– (m) ⊻ n4	10101	m_R	m_C	n4
Test	SKT	m, #n	CMP <– 0, if (m) ∧ n = n, then skip	11110	m_R	m_C	n
	SKF	m, #n	CMP <– 0, if (m) ∧ n = 0, then skip	11111	m_R	m_C	n
Compare	SKE	m, #n4	(m) – n4, skip if zero	01001	m_R	m_C	n4
	SKNE	m, #n4	(m) – n4, skip if not zero	01011	m_R	m_C	n4
	SKGE	m, #n4	(m) – n4, skip if not borrow	11001	m_R	m_C	n4
	SKLT	m, #n4	(m) – n4, skip if borrow	11011	m_R	m_C	n4
Rotation	RORC	r	→ CY → $(r)b_3$ → $(r)b_2$ → $(r)b_1$ → $(r)b_0$ ⮌	00111	000	0111	r
Transfer	LD	r, m	(r) <– (m)	01000	m_R	m_C	r
	ST	m, r	(m) <– (r)	11000	m_R	m_C	r

	MOV	@r, m	if MPE = 1: (MP, (r)) <– (m)
			if MPE = 0: (BANK, m_R, (r)) <– (m)
				01010	m_R	m_C	r

		m, @r	if MPE = 1: (m) <– (MP, (r))
			if MPE = 0: (m) <– (BANK, m_R, (r))
				11010	m_R	m_C	r

		m, #n4	(m) <– n4	11101	m_R	m_C	n4
*	MOVT[Note]	DBF, @AR	SP <– SP – 1, ASR <– PC, PC <– AR, DBF <– (PC), PC <– ASR, SP <– SP + 1	00111	000	0001	0000
	PUSH	AR	SP <– SP – 1, ASR <– AR	00111	000	1101	0000
	POP	AR	AR <– ASR, SP <– SP + 1	00111	000	1100	0000

* **Note** Exceptionally, execution of a MOVT instruction requires two instruction cycles.

(2/2)

Instruction set	Mnemonic	Operand	Operation	Instruction code			
				Op code	Operand		
Transfer	PEEK	WR, rf	WR <– (rf)	00111	rf_R	0011	rf_C
	POKE	rf, WR	(rf) <– WR	00111	rf_R	0010	rf_C
	GET	DBF, p	DBF <– (p)	00111	p_H	1011	p_L
	PUT	p, DBF	(p) <– DBF	00111	p_H	1010	p_L
Branch	BR	addr	if 0000H - (PC) - 07FFH PC <– addr, PAGE <– 0	01100	addr		
			if 0800H - (PC) - 0FFFH[Note] PC <– addr, PAGE <– 1	01101			
		@AR	PC <– AR	00111	000	0100	0000
Subroutine	CALL	addr	SP <– SP – 1, ASR <– PC, PC <– addr	11100	addr		
		@AR	SP <– SP – 1, ASR <– PC, PC <– AR	00111	000	0101	0000
	RET		PC <– ASR, SP <– SP + 1	00111	000	1110	0000
	RETSK		PC <– ASR, SP <– SP + 1 and skip	00111	001	1110	0000
	RETI		PC <– ASR, INTR <– INTSK, SP <– SP + 1	00111	100	1110	0000
Interrupt	EI		INTEF <– 1	00111	000	1111	0000
	DI		INTEF <– 0	00111	001	1111	0000
Others	STOP	s	STOP	00111	010	1111	s
	HALT	h	HALT	00111	011	1111	h
	NOP		No operation	00111	100	1111	0000

Note The µPD17145 and µPD17147 don't have page 1 (0800H to 0FFFH).

20.4 ASSEMBLER (AS17K) BUILT-IN MACRO INSTRUCTIONS

Legend

flag n	:	Flag type symbol
< >	:	Data < > is omissible.

	Mnemonic	Operand	Operation	n
Built-in macro	SKTn	flag 1, ···flag n	if (flag 1) to (flag n) = all "1", then skip	1 - n - 4
	SKFn	flag 1, ···flag n	if (flag 1) to (flag n) = all "0", then skip	1 - n - 4
	SETn	flag 1, ···flag n	(flag 1) to (flag n) <–1	1 - n - 4
	CLRn	flag 1, ···flag n	(flag 1) to (flag n) <– 0	1 - n - 4

	NOTn	flag 1, ···flag n	if (flag n) = "0", then (flag n) <– 1
			if (flag n) = "1", then (flag n) <– 0
				1 - n - 4

	INITFLG	<NOT> flag 1,
		···<<NOT> flag n>
			if description = NOT flag n, then (flag n) <– 0
			if description = flag n, then (flag n) <– 1
				1 - n - 4

	BANKn		(BANK) <– n	n = 0

20.5 EXPLANATION OF THE MACRO INSTRUCTIONS

20.5.1 Add Instructions

(1) ADD r, m **Add data memory to a general register**

<1> Instruction code

	10 8	7 4	3 0
00000	m_R	m_C	r

<2> Function

When CMP = 0 (r) <– (r) + (m)

The value of a data memory location is added to a general register, after which the result of the addition is loaded into the general register.

When CMP = 1 (r) + (m)

The result of addition is not loaded into a register, but the states of the carry flag CY and zero flag Z change according to the result of the addition.

If the addition produces a carry, the carry flag CY is set; if the addition does not produce a carry, the carry flag CY is reset.

If the result of addition is other than 0, the zero flag Z is reset, regardless of the state of the compare flag CMP.

If the result of addition is 0 when the compare flag is not set (CMP = 0), the zero flag Z is set.

If the result of addition is 0 when the compare flag is set (CMP = 1), the state of the zero flag Z remains as is.

Two types of addition are supported: one type is a binary 4-bit operation, while the other is a BCD operation. The BCD flag of PSWORD is used to specify the operation to be performed.

<3> Example 1:

When row address 0 (0.00H-0.0FH) of bank 0 is specified as a general register (RPH = 0, RPL = 0), the value of address 0.2FH is added to the value of address 0.03H, after which the result of the addition is stored at address 0.03H.

(0.03H) <– (0.03H) + (0.2FH)

```
MEM003    MEM    0.03H
MEM02F    MEM    0.2FH
          MOV    BANK, #00H         ; Selects bank 0 of the data memory.
          MOV    RPH, #00H          ; Selects bank 0 to specify a general register.
          MOV    RPL, #00H          ; Selects row address 0 to specify a general register.
          ADD    MEM003, MEM02F
```

Example 2:

When row address 2 (0.20H-0.2FH) of bank 0 is specified as a general register (RPH = 0, RPL = 4), the value of address 0.2FH is added to the value of address 0.23H, after which the result of the addition is stored at address 0.23H.

(0.23H) <– (0.23H) + (0.2FH)

```
MEM023    MEM    0.23H
MEM02F    MEM    0.2FH
          MOV    BANK, #00H         ; Selects bank 0 of the data memory.
          MOV    RPH, #00H          ; Selects bank 0 to specify a general register.Note
          MOV    RPL, #04H          ; Selects row address 2 to specify a general register.
          ADD    MEM023, MEM02F
```

Note

<table>
<tr><th rowspan="2">Register</th><th colspan="8">RP</th></tr>
<tr><th colspan="4">RPH</th><th colspan="4">RPL</th></tr>
<tr><td>Bit</td><td>b_3</td><td>b_2</td><td>b_1</td><td>b_0</td><td>b_3</td><td>b_2</td><td>b_1</td><td>b_0</td></tr>
<tr><td rowspan="3">Data</td><td colspan="4">Bank</td><td colspan="3" rowspan="3">Row address</td><td>B</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>C</td></tr>
<tr><td colspan="4"></td><td>D</td></tr>
</table>

The general register pointer (RP) is allocated in the system register as indicated in the table above. Accordingly, to set bank 0 and row address 2 as a general register, 00H must be stored at RPH, and 04H must be stored at RPL.

In this case, the BCD flag is reset, so that binary 4-bit operations are performed for subsequent arithmetic operations.

Example 3:

The value of address 0.6FH is added to the value of address 0.03H, after which the result of the addition is stored at address 0.03H. At this time, data memory address 0.6FH can be specified by specifying data memory address 2FH when IXE = 1, IXH = 0, IXM = 4, and IXL = 0, that is, when IX = 0.40H.

(0.03H) <– (0.03H) + (0.6FH)

Address obtained by ORing the value (0.04H) of the index register with data memory address 0.2FH

```
MEM003    MEM   0.03H
MEM02F    MEM   0.2FH
          MOV   RPH, #00H         ; Selects bank 0 to specify a general register.
          MOV   RPL, #00H         ; Selects row address 0 to specify a general register.
          MOV   IXH, #00H         ; IX <– 00001000000B
          MOV   IXM, #04H         ;
          MOV   IXL, #00H         ;
          SET1  IXE               ; IXE flag <– 1
          ADD   MEM003, MEM02F    ; IX                   00001000000B  (0.40H)
                                  ; Bank operand     OR) 00000101111B  (0.2FH)
                                  ; Specified address    00001101111B  (0.6FH)
```

Example 4:

The value of address 0.3FH is added to the value of address 0.03H, after which the result of addition is stored at address 0.03H. At this time, data memory address 0.3FH can be specified by specifying data memory address 2FH when IXE = 1, IXH = 0, IXM = 1, and IXL = 0, that is, when IX = 0.10H.

(0.03H) <– (0.03H) + (0.3FH)

Address obtained by ORing the value (0.10H) of the index register with data memory address 0.2FH

```
MEM003    MEM   0.03H
MEM02F    MEM   0.2FH
          MOV   BANK, #00H
          MOV   RPH, #00H         ; Selects bank 0 to specify a general register.
          MOV   RPL, #00H         ; Selects row address 0 to specify a general register.
          MOV   IXH, #00H         ; IX <– 00000010000B (0.10H)Note
          MOV   IXM, #01H
          MOV   IXL, #00H
          SET1  IXE               ; IXE flag <– 1
          ADD   MEM003, MEM02F    ; IX                   00000010000B  (0.10H)
                                  ; Bank operand     OR) 00000101111B  (0.2FH)
                                  ; Specified address    00100111111B (0.3FH)
```

Note

Register	IX											
	IXH				IXM				IXL			
Bit	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Data	M P E	Bank 0	0	0	0	Row address			Column address			

The index register (IX) is allocated in the system register as indicated in the table above. Accordingly, to set IX = 0.10H, 00H must be stored at IXH, 01H must be stored at IXM, and 00H must be stored at IXL.

In this case, the memory pointer enable flag (MPE) is reset. This means that the memory pointer (MP) is disabled for transfer between general registers.

<4> **Caution**

The first operand of the ADD r,m instruction specifies the column address of a general register. When the following add instruction is written, the column address of the general register is 03H:

```
MEM013    MEM    0.13H
MEM02F    MEM    0.2FH
          MOV    RPH, #00H       ; Selects bank 0 to specify a general register.          *
          MOV    RPL, #00H       ; Selects row address 0 to specify a general register.   *
          ADD    MEM013, MEM02F
```

(2) ADD m, #n4 — Add immediate data to data memory

<1> **Instruction code**

	10 8	7 4	3 0
10000	m_R	m_C	n4

<2> **Function**

When CMP = 0 (m) <– (m) + n4

Immediate data is added to the value of a data memory location, after which the result of the addition is stored at the data memory location.

When CMP = 1 (m) + n4

The result of addition is not stored at a data memory location, but the states of the carry flag CY and zero flag Z change according to the result of the addition.

If the addition produces a carry, the carry flag CY is set; if the addition does not produce a carry, the carry flag CY is reset.

If the result of the addition is other than 0, the zero flag Z is reset, regardless of the state of the compare flag CMP.

If the result of the addition is 0 when the compare flag is not set (CMP = 0), the zero flag Z is set.

If the result of addition is 0 when the compare flag is set (CMP = 1), the state of the zero flag Z remains as is.

Two types of addition are supported: one type is a binary 4-bit operation, while the other is a BCD operation. The BCD flag of PSWORD is used to specify which is to be performed.

<3> Example 1:

The immediate data 5 is added to the value of address 0.2FH, after which the result of the addition is stored at address 0.2FH.

(0.2FH) <– (0.2FH) + 5

```
MEM02F     MEM    0.2FH
           ADD    MEM02F, #05H
```

Example 2:

The immediate data 5 is added to the value of address 0.6FH, after which the result of the addition is stored at address 0.6FH. At this time, data memory address 0.6FH can be specified by specifying data memory address 2FH when IXE = 1, IXH = 0, IXM = 4, and IXL = 0, that is, when IX = 0.40H.

(0.6FH) <– (0.6FH) + 05H

Address obtained by ORing the value (0.40H) of the index register with data memory address 0.2FH

```
MEM02F     MEM    0.2FH
           MOV    BANK, #00H        ; Selects bank 0 of the data memory.
           MOV    IXH, #00H         ; IX <– 00001000000B (0.40H)
           MOV    IXM, #04H
           MOV    IXL, #00H
           SET1   IXE               ; IXE flag <– 1
           ADD    MEM02F, #05H      ; IX                    00001000000B  (0.40H)
                                    ; Bank operand      OR) 00000101111B  (0.2FH)
                                    ; Specified address     00001101111B  (0.6FH)
```

Example 3:

The immediate data 5 is added to the value of address 0.2FH, after which the result of the addition is stored at address 0.2FH. At this time, data memory address 0.2FH can be specified by specifying data memory address 2FH when IXE = 1, IXH = 0, IXM = 0, and IXL = 0, that is, when IX = 0.00H.

(0.2FH) <– (0.2FH) + 05H

Address obtained by ORing the value (0.00H) of the index register with data memory address 0.2FH

```
MEM02F      MEM     0.2FH
            MOV     BANK, #00H         ; Selects bank 0 of the data memory.
            MOV     IXH, #00H          ; IX <– 00000000000B
            MOV     IXM, #00H
            MOV     IXL, #00H
            SET1    IXE                ; IXE flag <– 1
            ADD     MEM02F, #05H       ; IX                  00000000000B  (0.00H)
                                       ; Bank operand     OR)  00000101111B  (0.2FH)
                                       ; Specified address     00000101111B  (0.2FH)
```

(3) ADDC r, m — Add data memory to general register with carry flag

<1> Instruction code

	10 8	7 4	3 0
00010	m_R	m_C	r

<2> Function

When CMP = 0 (r) <– (r) + (m) + CY

The value of a data memory location and the value of the carry flag CY are added to the value of the general register represented by r, after which the result of the addition is stored in the general register.

When CMP = 1 (r) + (m) + CY

The result of the addition is not loaded into a register, but the states of the carry flag CY and zero flag Z change according to the result of the addition.

By using the ADDC instruction, an add operation involving more than one nibble data can be performed easily. If the addition produces a carry, the carry flag CY is set; if the addition does not produce a carry, the carry flag CY is reset.

If the result of addition is other than 0, the zero flag Z is reset, regardless of the state of the compare flag CMP.

If the result of addition is 0 when the compare flag is not set (CMP = 0), the zero flag Z is set.

If the result of addition is 0 when the compare flag is set (CMP = 1), the state of the zero flag Z remains as is.

Two types of addition are supported: one type is a binary 4-bit operation, while the other is a BCD operation. The BCD flag of the program status word PSWORD is used to specify which is to be performed.

<3> **Example 1:**

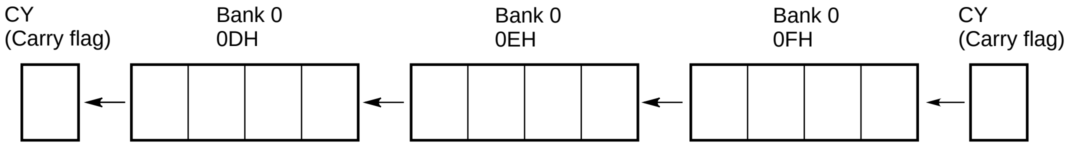
When row address 0 (0.00H-0.0FH) of bank 0 is specified as a general register, the value of the 12 bits from address 0.2DH to address 0.2FH is added to the value of the 12 bits from address 0.0DH to address 0.0FH, after which the result of addition is stored at the 12 bits from address 0.0DH to address 0.0FH.

(0.0FH) <– ($\underline{\text{0.0FH}}$) + (0.2FH)
(0.0EH) <– ($\underline{\text{0.0EH}}$) + (0.2EH) + CY
(0.0DH) <– ($\underline{\text{0.0DH}}$) + (0.2DH) + CY

```
MEM00D   MEM    0.0DH
MEM00E   MEM    0.0EH
MEM00F   MEM    0.0FH
MEM02D   MEM    0.2DH
MEM02E   MEM    0.2EH
MEM02F   MEM    0.2FH
         MOV    BANK, #00H        ; Selects bank 0 of the data memory.
         MOV    RPH, #00H         ; Selects bank 0 to specify a general register.
         MOV    RPL, #00H         ; Selects row address 0 to specify a general register.
         ADD    MEM00F, MEM02F    ; Low-order nibble
         ADDC   MEM00E, MEM02E
         ADDC   MEM00D, MEM02D    ; High-order nibble
```

Example 2:

When row address 2 (0.20H-0.2FH) of bank 0 is specified as a general register, the value of the 12 bits from address 0.2DH to address 0.2FH is shifted one bit to the left, together with the carry flag.



```
MEM00D   MEM    0.0DH
MEM00E   MEM    0.0EH
MEM00F   MEM    0.0FH
MEM02D   MEM    0.2DH
MEM02E   MEM    0.2EH
MEM02F   MEM    0.2FH
         MOV    RPH, #00H         ; Selects bank 0 to specify a general register.
         MOV    RPL, #04H         ; Selects row address 2 to specify a general register.
         MOV    BANK, #00H        ; Selects bank 0 of the data memory.
         ADDC   MEM00F, MEM02F
         ADDC   MEM00E, MEM02E
         ADDC   MEM00D, MEM02D
```

Example 3:

The value of the 12 bits from address 0.40H to address 0.42H is added to the value of the 12 bits from address 0.0DH to address 0.0FH, after which the result of the addition is stored in the 12 bits from address 0.0DH to address 0.0FH.

(0.0DH) <– (0.0DH) + (0.40H)
(0.0EH) <– (0.0EH) + (0.41H) + CY
(0.0FH) <– (0.0FH) + (0.42H) + CY

```
MEM000    MEM     0.00H
MEM001    MEM     0.01H
MEM002    MEM     0.02H
MEM00D    MEM     0.0DH
MEM00E    MEM     0.0EH
MEM00F    MEM     0.0FH
          MOV     BANK, #00H          ; Selects bank 0 of the data memory.
          MOV     RPH, #00H           ; Selects bank 0 to specify a general register.
          MOV     RPL, #00H           ; Selects row address 0 to specify a general register.
          MOV     IXH, #00H           ; IX <– 00001000000B (0.40H)
          MOV     IXM, #04H
          MOV     IXL, #00H
          SET1    IXE                 ; IXE flag <– 1
          ADD     MEM00D, MEM000      ; (0.0DH) <– (0.0DH) + (0.40H) : Low-order nibble
          ADDC    MEM00E, MEM001      ; (0.0EH) <– (0.0EH) + (0.41H)
          ADDC    MEM00F, MEM002      ; (0.0FH) <– (0.0FH) + (0.42H) : High-order nibble
```

(4) ADDC m, #n4 — Add immediate data to data memory with carry flag

<1> Instruction code

10	8 7	4 3	0
10010	m_R	m_C	n4

<2> Function

When CMP = 0   (m) <– (m) + n4 + CY

Immediate data and the value of the carry flag CY are added to the value of a data memory location, after which the result of the addition is stored at the data memory location.

When CMP = 1   (m) + n4 + CY

The result of the addition is not stored at a data memory location, but the states of the carry flag CY and zero flag Z change according to the result of the addition.

If the addition produces a carry, the carry flag CY is set; if the addition does not produce a carry, the carry flag CY is reset.

If the result of the addition is other than 0, the zero flag Z is reset, regardless of the state of the compare flag CMP.

If the result of the addition is 0 when the compare flag is not set (CMP = 0), the zero flag Z is set. If the result of the addition is 0 when the compare flag is set (CMP = 1), the state of the zero flag Z remains as is.

Two types of addition are supported: one type is a binary 4-bit operation, while the other is a BCD operation. The BCD flag of PSWORD is used to specify which is to be performed.

<3> Example 1:

The immediate data 5 is added to the value of the 12 bits from address 0.0DH to address 0.0FH, after which the result of the addition is stored at address 0.0DH to address 0.0FH.

```
(0.0FH) <– (0.0FH) + 05H
(0.0EH) <– (0.0EH) + CY
(0.0DH) <– (0.0DH) + CY
```

```
MEM00D      MEM     0.0DH
MEM00E      MEM     0.0EH
MEM00F      MEM     0.0FH
            MOV     BANK, #00H          ; Selects bank 0 of the data memory.
            ADD     MEM00F, #05H
            ADDC    MEM00E, #00H
            ADDC    MEM00D, #00H
```

Example 2:

The immediate data 5 is added to the value of the 12 bits from address 0.4DH to address 0.4FH, after which the result of the addition is stored at address 0.4DH to address 0.4FH.

```
(0.4FH) <– (0.4FH) + 05H
(0.4EH) <– (0.4EH) + CY
(0.4DH) <– (0.4DH) + CY
```

```
MEM00D      MEM     0.0DH
MEM00E      MEM     0.0EH
MEM00F      MEM     0.0FH
            MOV     BANK, #00H          ; Selects bank 0 of the data memory.
            MOV     IXH, #00H           ; IX <– 00001000000B (0.40H)
            MOV     IXM, #04H
            MOV     IXL, #00H
            SET1    IXE                 ; IXE flag <– 1
            ADD     MEM00F, #5          ; (0.4FH) <– (0.4FH) + 5H
            ADDC    MEM00E, #0          ; (0.4EH) <– (0.4EH) + CY
            ADDC    MEM00D, #0          ; (0.4DH) <– (0.4DH) + CY
```

(5) INC AR — Increment address register

<1> Instruction code

00111	000	1001	0000

<2> Function

AR <– AR + 1

The value of the address register AR is incremented by 1.

<3> Example 1:

The value 1 is added to the value of the 16 bits from AR3 to AR0 (address register) in the system register, after which the result of the addition is stored at AR3 to AR0.

AR0 <– AR0 + 1
AR1 <– AR1 + CY
AR2 <– AR2 + CY
AR3 <– AR3 + CY

```
INC AR
```

This operation can be performed by using add instructions as described below.

```
ADD      AR0, #01H
ADDC     AR1, #00H
ADDC     AR2, #00H
ADDC     AR3, #00H
```

Example 2:

Table data is transferred in blocks of 16 bits (1 address) to the data buffer (DBF) by using table reference instructions. (For details of the table reference instructions, see **Section 10.2.3**.)

```
         ORG      10H
         DW       0F3FFH
         DW       0A123H
         DW       0FFF1H
         DW       0FFF5H
         DW       0FF11H

           ⋮

         MOV      AR3, #0H          ; Table data address
         MOV      AR2, #0H          ; Loads 0010H into the address register.
         MOV      AR1, #1H
         MOV      AR0, #0H
```

```
LOOP :
*                 MOVT    DBF, @AR          ; Reads table data into DBF.
                  ⋮
                  (Processing which refers table data)
                  ⋮
                  INC     AR                ; Increments the address register by 1.
                  BR      LOOP
```

<4> Caution

The number of bits available with the address register AR (AR0-AR3) depends on the product being used. When using the address register, always check the data sheet of the product being used.

(6) INC IX — Increment index register

<1> Instruction code

00111	000	1000	0000

<2> Function

IX <– IX + 1

The value of the index register IX is incremented.

<3> Example 1:

The value 1 is added to the 12 bits of the index register (consisting of IXH, IXM, and IXL) in the system register, after which the result of the addition is stored at IXH, IXM, and IXL.

IXL <– IXL + 1
IXM <– IXM + CY
IXH <– IXH + CY

INC IX

This operation can be performed by using add instruction, as described below.

```
ADD       IXL, #01H
ADDC      IXM, #00H
ADDC      IXH, #00H
```

Example 2:

The values stored in the data memory locations 0.00H to 0.73H are cleared to 0 by using the index register.

```
MEM000    MEM     0.00H
          MOV     IXH, #00H         ; Stores 00H of bank 0 into the index register.
          MOV     IXM, #00H         ;
          MOV     IXL, #00H
```

```
RAM clear:
            SET1    IXE                 ; IXE flag <– 1
            MOV     MEM000, #00H        ; Writes 0 to the data memory location indicated by
                                        ; the index register.
            CLR1    IXE                 ; IXE flag <– 0
            INC     IX
            SET2    CMP, Z              ; CMP flag <– 1, Z flag <– 1
            SUB     IXL, #03H           ; Checks if the value of the index register is 73H of
            SUB     IXM, #07H           ; bank 0.
            SUB     IXH, #00H           ;
            SKT1    Z                   ; Loops until the value of the index register becomes
            BR      RAM clear           ; 73H of bank 0.
```

20.5.2 Subtract Instructions

(1) SUB r, m **Subtract data memory from general register**

<1> Instruction code

	10 8	7 4	3 0
00001	m_R	m_C	r

<2> Function

When CMP = 0 (r) <– (r) – (m)

The value of a data memory location is subtracted from the value stored in a general register, after which the result of the subtraction is loaded into the general register.

When CMP = 1 (r) – (m)

The result of the subtraction is not loaded into a register, but the states of the carry flag CY and zero flag Z change according to the result of the subtraction.

If the subtraction produces a borrow, the carry flag CY is set; if the subtraction does not produce a borrow, the carry flag CY is reset.

If the result of the subtraction is other than 0, the zero flag Z is reset, regardless of the state of the compare flag CMP.

If the result of the subtraction is 0 when the compare flag is not set (CMP = 0), the zero flag Z is set.

If the result of the subtraction is 0 when the compare flag is set (CMP = 1), the state of the zero flag Z remains as is.

Two types of subtraction are supported: one type is a binary 4-bit operation, while the other is a BCD operation. The BCD flag of the program status word PSWORD is used to specify which is to be performed.

<3> **Example 1:**

When row address 0 (0.00H-0.0FH) of bank 0 is specified as a general register (RPH = 0, RPL = 0), the value of address 0.2FH is subtracted from the value of address 0.03H, after which the result of the subtraction is stored at address 0.03H.

```
                (0.03H) <– (0.03H) – (0.2FH)

  MEM003        MEM      0.03H
  MEM02F        MEM      0.2FH
                SUB      MEM003, MEM02F
```

Example 2:

When row address 2 (0.20H-0.2FH) of bank 0 is specified as a general register (RPH = 0, RPL = 4), the value of address 0.2FH is subtracted from the value of address 0.23H, after which the result of the subtraction is stored at address 0.23H.

```
                (0.23H) <– (0.23H) – (0.2FH)

MEM023          MEM      0.23H
MEM02F          MEM      0.2FH
                MOV      BANK, #00H          ; Selects bank 0 of the data memory.
                MOV      RPH, #00H           ; Selects bank 0 to specify a general register.
                MOV      RPL, #04H           ; Selects row address 2 to specify a general register.
                SUB      MEM023, MEM02F
```

Example 3:

The value of address 0.6FH is subtracted from the value of address 0.03H, after which the result of the subtraction is stored at address 0.03H. At this time, data memory address 0.6FH can be specified by specifying data memory address 2FH when IXE = 1, IXH = 0, IXM = 4, and IXL = 0, that is, when IX = 0.40H.

```
                (0.03H) <– (0.03H) – (0.6FH)

MEM003          MEM      0.03H
MEM02F          MEM      0.2FH
                MOV      BANK, #00H          ; Selects bank 0 of the data memory.
                MOV      RPH, #00H           ; Selects bank 0 to specify a general register.
                MOV      RPL, #00H           ; Selects row address 0 to specify a general register.
                MOV      IXH, #00H           ; IX <– 00001000000B (0.40H)
                MOV      IXM, #04H           ;
                MOV      IXL, #00H           ;
                SET1     IXE                 ; IXE flag <– 1
                SUB      MEM003, MEM02F      ; IX                   00001000000B  (0.40H)
                                             ; Bank operand     OR) 00000101111B  (0.2FH)
                                             ; Specified address    00001101111B  (0.6FH)
```

Example 4:

The value of address 0.3FH is subtracted from the value of address 0.03H, after which the result of the subtraction is stored at address 0.03H. At this time, data memory address 0.3FH can be specified by specifying data memory address 2FH when IXE = 1, IXH = 0, IXM = 1, and IXL = 0, that is, when IX = 0.10H.

```
                (0.03H) <– (0.03H) – (0.3FH)

MEM003          MEM     0.03H
MEM02F          MEM     0.2FH
                MOV     BANK, #00H          ; Selects bank 0 of the data memory.
                MOV     RPH, #00H           ; Selects bank 0 to specify a general register.
                MOV     RPL, #00H           ; Selects row address 0 to specify a general register.
                MOV     IXH, #00H           ; IX <– 00000010000B (0.10H)
                MOV     IXM, #01H           ;
                MOV     IXL, #00H           ;
                SET1    IXE                 ; IXE flag <– 1
                SUB     MEM003, MEM02F      ; IX                    00000010000B  (0.10H)
                                            ; Bank operand      OR) 00000101111B  (0.2FH)
                                            ; Specified address     00000111111B  (0.3FH)
```

<4> Caution

The first operand of the SUB r,m instruction must specify the address of a general register. When the following subtract instruction is written, address 03H is specified as a register:

```
MEM013          MEM     0.13H
MEM02F          MEM     0.2FH
                MOV     RPH, #00H           ; Selects bank 0 to specify a general register.         *
                MOV     RPL, #00H           ; Selects row address 0 to specify a general register.  *
                SUB     MEM013, MEM02F
```

(2) SUB m, #n4 Subtract immediate data from data memory

<1> Instruction code

	10 8	7 4	3 0
10001	m_R	m_C	n4

<2> Function

When CMP = 0 (m) <– (m) – n4

Immediate data is subtracted from the value stored at a data memory location, after which the result of the subtraction is stored at the data memory location.

When CMP = 1 (m) – n4

The result of the subtraction is not stored at a data memory location, but the states of the carry flag CY and zero flag Z change according to the result of the subtraction.

If the subtraction produces a borrow, the carry flag CY is set; if the subtraction does not produce a borrow, the carry flag CY is reset.

If the result of the subtraction is other than 0, the zero flag Z is reset, regardless of the state of the compare flag CMP.

If the result of the subtraction is 0 when the compare flag is reset (CMP = 0), the zero flag Z is set.

If the result of the subtraction is 0 when the compare flag is set (CMP = 1), the state of the zero flag Z remains as is.

Two types of subtraction are supported: one type is a binary 4-bit operation, while the other is a BCD operation. The BCD flag of the program status word PSWORD is used to specify which is to be performed.

<3> Example 1:

The immediate data 5 is subtracted from the value stored at address 0.2FH, after which the result of the subtraction is stored at address 0.2FH.

(0.2FH) <– (0.2FH) – 5

```
MEM02F      MEM      0.2FH
            SUB      MEM02F, #05H
```

Example 2:

The immediate data 5 is subtracted from the value stored at address 0.6FH, after which the result of the subtraction is stored at address 0.6FH. At this time, data memory address 0.6FH can be specified by specifying data memory address 2FH when IXE = 1, IXH = 0, IXM = 4, and IXL = 0, that is, when IX = 0.40H.

(0.6FH) <– (0.6FH) – 5

Address obtained by ORing the value (0.40H) of the index register with data memory address 0.2FH

```
MEM02F     MEM     0.2FH
           MOV     BANK, #00H          ; Selects bank 0 of the data memory.
           MOV     IXH, #00H           ; IX ¨ 00001000000B (0.40H)
           MOV     IXM, #04H           ;
           MOV     IXL, #00H           ;
           SET1    IXE                 ; IXE flag <– 1
           SUB     MEM02F, #05H        ; IX                    00001000000B  (0.40H)
                                       ; Bank operand       OR) 00000101111B  (0.2FH)
                                       ; Specified address     00001101111B  (0.6FH)
```

Example 3:

The immediate data 5 is subtracted from the value stored at address 0.2FH, after which the result of the subtraction is stored at address 0.2FH. At this time, data memory address 0.2FH can be specified by specifying data memory address 2FH when IXE = 1, IXH = 0, IXM = 0, and IXL = 0, that is, when IX = 0.00H.

(0.2FH) <– (0.2FH) – 5

Address obtained by ORing the value (0.00H) of the index register with data memory address 0.2FH

```
MEM02F    MEM     0.2FH
          MOV     BANK0, #00H          ; Selects bank 0 of the data memory.
          MOV     IXH, #00H            ; IX <– 00000000000B (0.00H)
          MOV     IXM, #00H            ;
          MOV     IXL, #00H            ;
          SET1    IXE                  ; IXE flag <– 1
          SUB     MEM02F, #05H         ; IX                    00000000000B  (0.00H)
                                       ; Bank operand       OR) 00000101111B  (0.2FH)
                                       ; Specified address     00000101111B  (0.2FH)
```

(3) SUBC r, m — Subtract data memory from general register with carry flag

<1> Instruction code

	10 8	7 4	3 0
00011	m_R	m_C	r

<2> Function

When CMP = 0 (r) <– (r) – (m) – CY

The value of a data memory location and the value of the carry flag CY are subtracted from the value stored at a general register, after which the result of the subtraction is stored in the general register.
* By using the SUBC instruction, a subtract operation involving more than one nibble data can be performed easily.

When CMP = 1 (r) – (m) – CY

The result of the subtraction is not loaded into a register, but the states of the carry flag CY and zero flag Z change according to the result of the subtraction.

If the subtraction produces a borrow, the carry flag CY is set; if the subtraction does not produce a borrow, the carry flag CY is reset.

If the result of the subtraction is other than 0, the zero flag Z is not set, regardless of the state of the compare flag CMP.

If the result of the subtraction is 0 when the compare flag is not set (CMP = 0), the zero flag Z is set.

If the result of the subtraction is 0 when the compare flag is set (CMP = 1), the state of the zero flag Z remains as is.

Two types of subtraction are supported: one type is a binary 4-bit operation, while the other is a BCD operation. The BCD flag of the program status word PSWORD is used to specify which is to be performed.

<3> Example 1:

When row address 0 (0.00H-0.0FH) of bank 0 is specified as a general register, the value of the 12 bits from address 0.2DH to address 0.2FH is subtracted from the value of the 12 bits from address 0.0DH to address 0.0FH, after which the result of the subtraction is stored in the 12 bits from address 0.0DH to address 0.0FH.

```
               (0.0FH) <– (0.0FH) – (0.2FH)
               (0.0EH) <– (0.0EH) – (0.2EH) – CY
               (0.0DH) <– (0.0DH) + (0.2DH) – CY
```

```
MEM00D         MEM    0.0DH
MEM00E         MEM    0.0EH
MEM00F         MEM    0.0FH
MEM02D         MEM    0.2DH
MEM02E         MEM    0.2EH
MEM02F         MEM    0.2FH
               SUB    MEM00F, MEM02F  ; Low–order nibble
               SUBC   MEM00E, MEM02E
               SUBC   MEM00D, MEM02D  ; High–order nibble
```

Example 2:

The value of the 12 bits from address 0.40H to address 0.42H is subtracted from the value of the 12 bits from address 0.0DH to address 0.0FH, after which the result of the subtraction is stored in the 12 bits from address 0.0DH to address 0.0FH.

```
               (0.0DH) <– (0.0DH) – (0.40H)
               (0.0EH) <– (0.0EH) – (0.41H) – CY
               (0.0FH) <– (0.0FH) + (0.42H) – CY
```

```
MEM000         MEM    0.00H
MEM001         MEM    0.01H
MEM002         MEM    0.02H
MEM00D         MEM    0.0DH
MEM00E         MEM    0.0EH
MEM00F         MEM    0.0FH
               MOV    BANK, #00H      ; Selects bank 0 of the data memory.
               MOV    RPH, #00H       ; Selects bank 0 to specify a general register.
               MOV    RPL, #00H       ; Selects row address 0 to specify a general register.
               MOV    IXH, #00H       ; IX <– 00001000000B (0.40H)
               MOV    IXM, #04H       ;
               MOV    IXL, #00H       ;
               SET1   IXE             ; IXE flag <– 1
               SUB    MEM00D, MEM000  ; (0.0DH) <– (0.0DH) – (0.40H)
               SUBC   MEM00E, MEM001  ; (0.0EH) <– (0.0EH) – (0.41H)
               SUBC   MEM00F, MEM002  ; (0.0FH) <– (0.0FH) – (0.42H)
```

(4) SUBC m, #n4 Subtract immediate data from data memory with carry flag

<1> Instruction code

	10 8	7 4	3 0
10011	m_R	m_C	n4

<2> Function

When CMP = 0 (m) <– (m) – n4 – CY

Immediate data and the value of the carry flag CY are subtracted from the value stored at a data memory location, after which the result of the subtraction is stored at the data memory location.

When CMP = 1 (m) – n4 – CY

The result of the subtraction is not stored at a data memory location, but the states of the carry flag CY and zero flag Z change according to the result of the subtraction.

If the subtraction produces a borrow, the carry flag CY is set; if the subtraction does not produce a borrow, the carry flag CY is reset.

If the result of the subtraction is other than 0, the zero flag Z is reset, regardless of the state of the compare flag CMP.

If the result of the subtraction is 0 when the compare flag is reset (CMP = 0), the zero flag Z is set.

If the result of the subtraction is 0 when the compare flag is set (CMP = 1), the state of the zero flag Z remains as is.

Two types of subtraction are supported: one type is a binary 4-bit operation, while the other is a BCD operation. The BCD flag of the program status word PSWORD is used to specify which is to be performed.

<3> Example 1:

The immediate data 5 is subtracted from the value of the 12 bits from address 0.0DH to address 0.0FH, after which the result of the subtraction is stored at address 0.0DH to address 0.0FH.

(0.0FH) <– (0.0FH) – 05H
(0.0EH) <– (0.0EH) – CY
(0.0DH) <– (0.0DH) – CY

```
MEM00D    MEM     0.0DH
MEM00E    MEM     0.0EH
MEM00F    MEM     0.0FH
          SUB     MEM00F, #05H
          SUBC    MEM00E, #00H
          SUBC    MEM00D, #00H
```

Example 2:

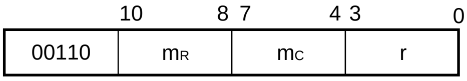
The immediate data 5 is subtracted from the value of the 12 bits from address 0.4DH to address 0.4FH, after which the result of the subtraction is stored at address 0.4DH to address 0.4FH.

```
                (0.4FH) <– (0.4FH) – 05H
                (0.4EH) <– (0.4EH) – CY
                (0.4DH) <– (0.4DH) – CY

MEM00D          MEM     0.0DH
MEM00E          MEM     0.0EH
MEM00F          MEM     0.0FH
                MOV     BANK, #00H          ; Selects bank 0 of the data memory.
                MOV     IXH, #00H           ; IX <– 00001000000B (0.40H)
                MOV     IXM, #04H           ;
                MOV     IXL, #00H           ;
                SET1    IXE                 ; IXE flag <– 1
                SUB     MEM00F, #5          ; (0.4FH) <– (0.4FH) – 5
                SUBC    MEM00E, #0          ; (0.4EH) <– (0.4EH) – CY
                SUBC    MEM00D, #0          ; (0.4DH) <– (0.4DH) – CY
```

20.5.3 Logical Instructions

(1) OR r, m **OR between a general register and data memory**

<1> Instruction code



<2> Function

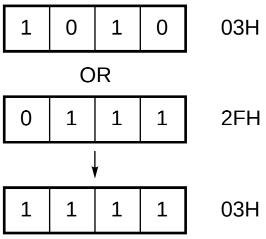
(r) <– (r) ∨ (m)

The value stored in a general register is ORed with the value of a data memory location, after which the result of the OR operation is stored in the general register.

<3> Example

The value (1010B) of address 0.03H is ORed with the value (0111B) stored at address 0.2FH, after which the result (1111B) of the OR operation is stored at 0.03H.

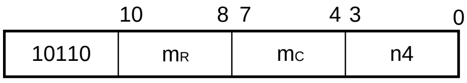
(0.03H) <– (0.03H) ∨ (0.2FH)



```
MEM003    MEM     0.03H
MEM02F    MEM     0.2FH
          MOV     MEM003, #1010B
          MOV     MEM02F, #0111B
          OR      MEM003, MEM02F
```

(2) OR m, #n4 **OR between data memory and immediate data**

<1> Instruction code



<2> Function

(m) <– (m) ∨ n4

The value stored at a data memory location is ORed with the immediate data, after which the result of the OR operation is stored at the data memory location.

<3> Example 1:

Bit 3 (MSB) of address 0.03H is set.

(0.03H) <– (0.03H) ¦ 1000B

0.03H

1	x	x	x

x: Don't care

```
MEM003    MEM    0.03H
          OR     MEM003, #1000B
```

Example 2:

All bits of address 0.03H are set.

```
MEM003    MEM    0.03H
          OR     MEM003, #1111B
              or
MEM003    MEM    0.03H
          MOV    MEM003, #0FH
```

(3) AND r, m — AND between general register and data memory

<1> Instruction code

10 8 7 4 3 0

00100	m_R	m_C	r

<2> Function

(r) <– (r) ∧ (m)

The value stored in a general register is ANDed with the value of a data memory location, after which the result of the AND operation is stored in the general register.

<3> Example

The value (1010B) stored at address 0.03H is ANDed with the value (0110B) stored at address 0.2FH, after which the result (0010B) of the AND operation is stored at address 0.03H.

(0.03H) <– (0.03H) ∧ (0.2FH)

1	0	1	0	03H

AND

0	1	1	0	2FH

↓

0	0	1	0	03H

```
MEM003     MEM     0.03H
MEM02F     MEM     0.2FH
           MOV     MEM003, #1010B
           MOV     MEM02F, #0110B
           AND     MEM003, MEM02F
```

(4) AND m, #n4 AND between data memory and immediate data

<1> Instruction code

	10 8	7 4	3 0
10100	m_R	m_C	n4

<2> Function

(m) <– (m) ∧ n4

The value stored at a data memory location is ANDed with the immediate data, after which the result of the AND operation is stored at the data memory location.

<3> Example 1:

Bit 3 (MSB) of address 0.03H is reset.

(0.03H) <– (0.03H) ∧ 0111B

0.03H

0	x	x	x

x: Don't care

```
MEM003     MEM     0.03H
           AND     MEM003, #0111B
```

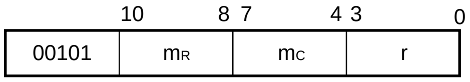
Example 2:

All bits of address 0.03H are reset.

```
MEM003     MEM     0.03H
           AND     MEM003, #0000B
                or
MEM003     MEM     0.03H
           MOV     MEM003, #00H
```

(5) XOR r, m — Exclusive OR between a general register and data memory

<1> Instruction code



<2> Function

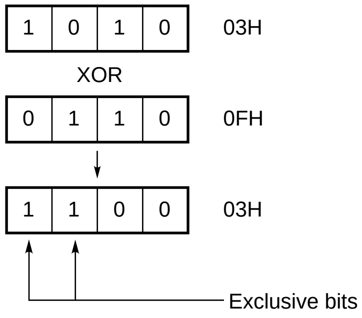
(r) <– (r) ∀ (m)

The value stored in a general register is exclusive-ORed with the value stored in a data memory location, after which the result of the XOR operation is stored in the general register.

<3> Example 1:

The value stored at address 0.03H is compared with the value stored at address 0.0FH. Any bits for which the comparison reveals a mismatch are set, after which the values of the four bits, including those set bits, are stored at address 0.03H. If all the bits of address 0.03H are reset (that is, the value of address 0.03H is the same as the value of address 0.0FH), a jump to LBL1 is performed. In other cases, a jump to LBL2 is performed.

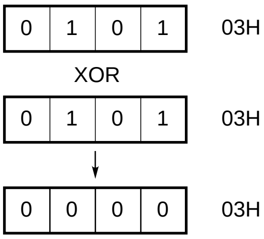
This example can find an application where the states of alternate switches (the value of address 0.03H) are compared with an internal state (value of address 0.0FH), causing a branch to the processing for the changed switches.



```
MEM003    MEM     0.03H
MEM00F    MEM     0.0FH
          XOR     MEM003, MEM00F
          SKNE    MEM003, #00H
          BR      LBL1
          BR      LBL2
```

Example 2:

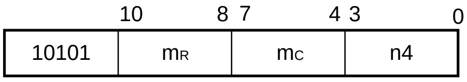
The value of address 0.03H is cleared.



```
MEM003     MEM     0.03H
           XOR     MEM003, MEM003
```

(6) XOR m, #n4 Exclusive OR between data memory and immediate data

<1> Instruction code



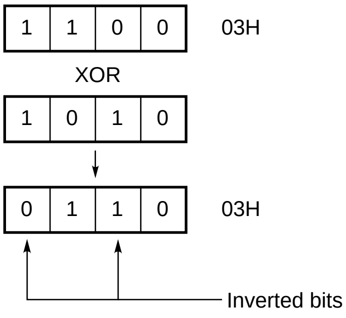
<2> Function

(m) <– (m) ∀ n4

The value of a data memory location is exclusive-ORed with the immediate data, after which the result of the XOR operation is stored at the data memory location.

<3> Example

The values of bits 1 and 3 of address 0.03H are inverted, after which the values of the four bits are stored at address 0.03H.



```
MEM003     MEM     0.03H
           XOR     MEM003, #1010B
```

20.5.4 Evaluation Instructions

(1) SKT m, #n **Skip the next instruction if data memory bits are true**

<1> Instruction code

	10 8	7 4	3 0
11110	m_R	m_C	n

<2> Function

CMP <– 0, if (m) ∧ n=n, then skip

The value stored at a data memory location is ANDed with the immediate data n. If the result of an AND operation is equal to n, the next instruction is skipped (executed as a NOP instruction). *

<3> Example 1:

If bit 0 of address 03H is 1, a jump to AAA is performed. If the bit is 0, a jump to BBB is performed.

```
SKT        03H, #0001B
BR         BBB
BR         AAA
```

Example 2:

If bits 0 and 1 of address 03H are 1, the next instruction is skipped.

```
SKT        03H, #0011B
```

	b_3	b_2	b_1	b_0
Skip condition 03H	x	x	1	1

x: Don't care

Example 3:

The two instructions below produce the same result.

```
SKT        13H, #1111B
SKE        13H, #0FH
```

(2) SKF m, #n **Skip the next instruction if data memory bits are false**

<1> Instruction code

	10 8	7 4	3 0
11111	m_R	m_C	n

<2> Function

CMP <– 0, if (m) ∧ n=0, then skip

The value stored at a data memory location is ANDed with the immediate data n. If the result of the AND operation is 0, the next instruction is skipped (executed as a NOP instruction). *

<3> Example 1:

If bit 2 of address 13H is 0, the immediate data 00H is stored at data memory address 0FH. If the bit is 1, a jump to ABC is performed.

```
MEM013     MEM     0.13H
MEM00F     MEM     0.0FH
           SKF     MEM013, #0100B
           BR      ABC
           MOV     MEM00F, #00H
```

Example 2:

If bits 3 and 0 of address 29H are 0, the next instruction is skipped.

```
SKF        29H, #1001B
```

		b_3	b_2	b_1	b_0	
Skip condition	29H	0	x	x	0	x: Don't care

Example 3:

The two instructions below produce the same result.

```
SKF        34H, #1111B
SKE        34H, #00H
```

20.5.5 Compare Instructions

(1) SKE m, #n4 Skip if the contents of data memory are equal to the immediate data

<1> Instruction code

10	8 7	4 3	0
01001	m_R	m_C	n4

<2> Function

(m) – n4, skip if zero

If the value stored at a data memory location is equal to the immediate data, the next instruction is skipped (executed as a NOP instruction). *

<3> Example

If the value of address 24H is 0, 0FH is transferred to address 24H. If the value is other than 0, a jump to OPE1 is performed.

```
MEM024     MEM     0.24H
           SKE     MEM024, #00H
           BR      OPE1
           MOV     MEM024, #0FH
OPE1:
```

(2) SKNE m, #n4 Skip if the contents of data memory are not equal to the immediate data

<1> Instruction code

```
         10        8 7        4 3        0
| 01011  |   mR    |   mC    |   n4    |
```

<2> Function

(m) – n4, skip if not zero

If the value stored at a data memory location is not equal to immediate data, the next instruction is skipped (executed as a NOP instruction). *

<3> Example

If the value stored at address 1FH is 1, and that stored at address 1EH is 3, a jump to XYZ is performed. In other cases, a jump to ABC is performed.

When an 8-bit comparison is made, the following combination is used:

```
         3                      1
1EH | 0011 |          1FH | 0001 |
```

```
MEM01E      MEM     0.1EH
MEM01F      MEM     0.1FH
            SKNE    MEM01F, #01H
            SKE     MEM01E, #03H
            BR      ABC
            BR      XYZ
```

The same operation as that above can be performed by using the compare flag and zero flag, as follows:

```
MEM01E      MEM     0.1EH
MEM01F      MEM     0.1FH
            SET2    CMP, Z              ; CMP flag <– 1, Z flag <– 1
            SUB     MEM01F, #01H
            SUBC    MEM01E, #03H
            SKT1    Z
            BR      ABC
            BR      XYZ
```

(3) SKGE m, #n4 Skip if the contents of data memory are greater than or equal to the immediate data

<1> Instruction code

```
         10        8 7        4 3        0
| 11001  |   mR    |   mC    |   n4    |
```

<2> Function

(m) – n4, skip if not borrow

If the value stored at a data memory location is greater than or equal to the immediate data, the next instruction is skipped (executed as a NOP instruction). *

<3> Example

If the 8-bit data stored at address 1FH (higher address) and 2FH (lower address) is greater than or equal to immediate data 17H, RET is executed; if not, RETSK is executed.

```
MEM01F     MEM     0.1FH
MEM02F     MEM     0.2FH
           SKGE    MEM01F, #1
           RETSK
           SKNE    MEM01F, #1
           SKLT    MEM02F, #8      ; 7 + 1
           RET
           RETSK
```

(4) SKLT m, #n4 — Skip if the contents of data memory are less than the immediate data

<1> Instruction code

	10 8	7 4	3 0
11011	m_R	m_C	n4

<2> Function

(m) – n4, skip if borrow

If the value stored at a data memory location is less than the immediate data, the next instruction is skipped (executed as a NOP instruction). *

<3> Example

If the value stored at address 10H is greater than or equal to immediate data 6, 01H is stored at address 0FH. If the value of address 10H is less than immediate data 6, 02H is stored at address 0FH.

```
MEM00F     MEM     0.0FH
MEM010     MEM     0.10H
           MOV     MEM00F, #02H
           SKLT    MEM010, #06H
           MOV     MEM00F, #01H
```

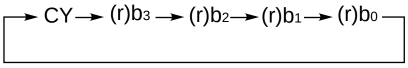
20.5.6 Rotate Instructions

(1) RORC r — Rotate right general register with carry flag

<1> Instruction code

			3 0
00111	000	0111	r

<2> Function



The value stored by the general register represented by r is rotated one bit to the right, together with the carry flag.

<3> Example 1:

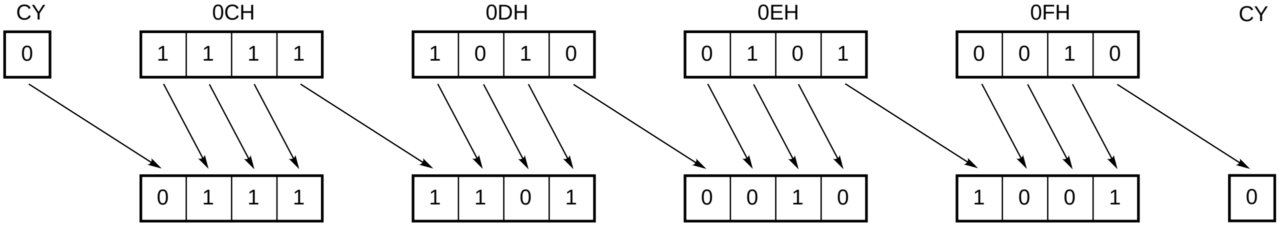
When row address 0 (0.00H-0.0FH) of bank 0 is specified as a general register (RPH = 0, RPL = 0), the value (1000B) of address 0.00H is rotated one bit to the right to produce the value 0100B.

(0.00H) <– (0.00H) ÷ 2

```
MEM000      MEM     0.00H
            MOV     RPH, #00H          ; Selects bank 0 to specify a general register.
            MOV     RPL, #00H          ; Selects row address 0 to specify a general register.
            CLR1    CY                 ; CY flag <– 0
            RORC    MEM000
```

Example 2:

When row address 0 (0.00H-0.0FH) of bank 0 is specified as a general register (RPH = 0, RPL = 0), the value (0FA52H) of the data buffer DBF is rotated one bit to the right to change the value of DBF to 7D29H.

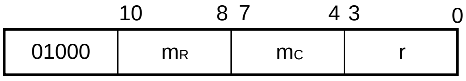


```
MEM00C      MEM     0.0CH
MEM00D      MEM     0.0DH
MEM00E      MEM     0.0EH
MEM00F      MEM     0.0FH
            MOV     RPH, #00H          ; Selects bank 0 to specify a general register.
            MOV     RPL, #00H          ; Selects row address 0 to specify a general register.
            CLR1    CY                 ; CY flag <– 0
            RORC    MEM00C
            RORC    MEM00D
            RORC    MEM00E
            RORC    MEM00F
```

20.5.7 Transfer Instructions

(1) LD r, m **Load data memory to general register**

<1> Instruction code



<2> Function

(r) <– (m)

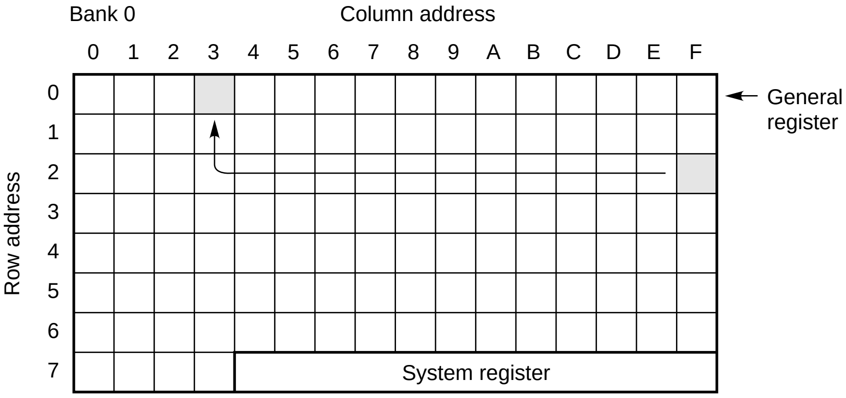
The value stored at a data memory location is loaded into a general register.

<3> Example 1:

The value stored at address 0.2FH is stored at address 0.03H.

(0.03H) <– (0.2FH)

```
  MEM003    MEM    0.03H
  MEM02F    MEM    0.2FH
*           MOV    RPH, #00H          ; Selects bank 0 to specify a general register.
*           MOV    RPL, #00H          ; Selects row address 0 to specify a general register.
            LD     MEM003, MEM02F
```



Example 2:

The value stored at address 0.6FH is stored at address 0.03H. At this time, data memory address 0.6FH can be specified by specifying data memory address 2FH when IXE = 1, IXH = 0, IXM = 4, and IXL = 0, that is, when IX = 0.40H.

IXH <– 00H

IXM <– 04H

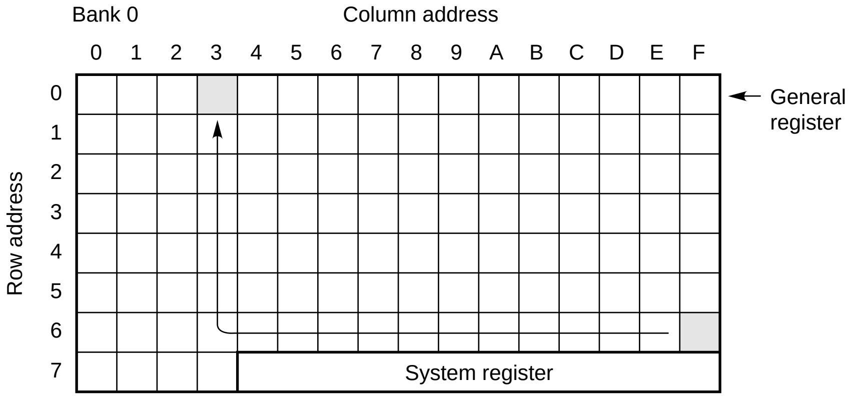
IXL <– 00H

IXE flag <– 1

(0.03H) <– (0.6FH)

Address obtained by ORing the value (040H) of the index register with data memory address 0.2FH

```
MEM003    MEM    0.03H
MEM02F    MEM    0.2FH
          MOV    IXH, #00H          ; IX <– 00001000000B (0.40H)
          MOV    IXM, #04H
          MOV    IXL, #00H
          SET1   IXE                ; IXE flag <– 1
          LD     MEM003, MEM02F
```



(2) ST m, r — Store general register to data memory

<1> Instruction code

```
            10         8 7        4 3        0
| 11000    | mR        | mC        | r        |
```

<2> Function

(m) <– (r)

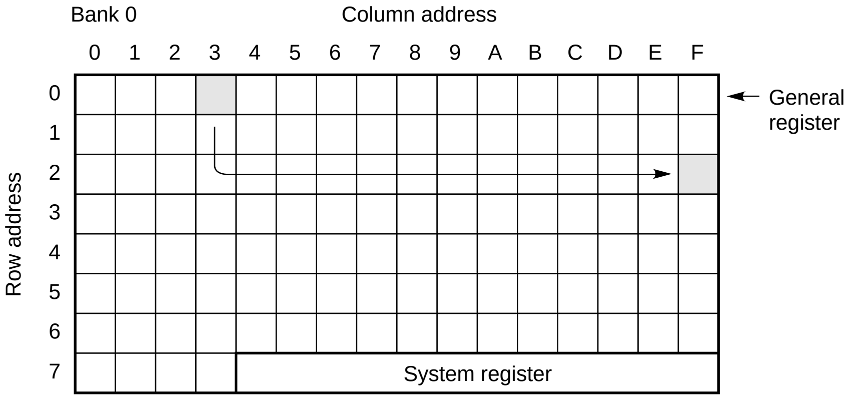
The value stored in a general register is stored at a data memory location.

<3> Example 1:

The value of address 0.03H is stored at address 0.2FH.

(0.2FH) <– (0.03H)

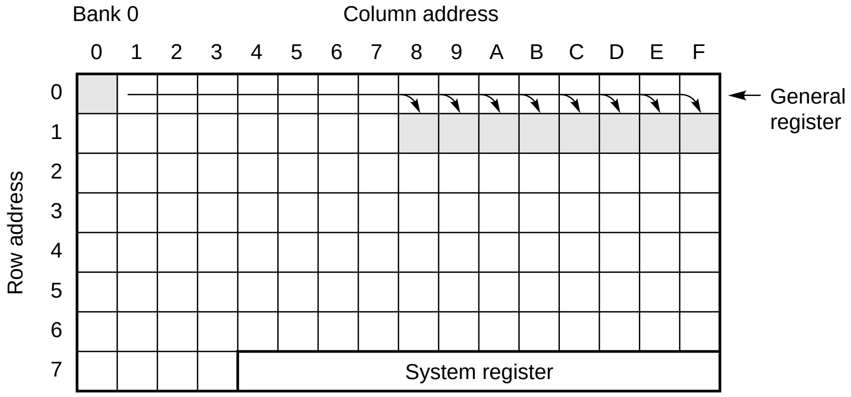
```
MOV     RPH, #00H        ; Selects bank 0 to specify a general register.         *
MOV     RPL, #00H        ; Selects row address 0 to specify a general register.  *
ST      2FH, 03H         ; Transfers the value of the general register to the
                         ; data memory location.
```



Example 2:

The value stored at address 0.00H is stored at address 0.18H to address 0.1FH. The data memory locations (18H-1FH) are specified with the index register.

```
(0.18H) <– (0.00H)
(0.19H) <– (0.00H)
   :
   :
(0.1FH) <– (0.00H)

        MOV     IXH, #00H           ; IX <– 00000000000B (0.00H)
        MOV     IXM, #00H
        MOV     IXL, #00H           ; Specifies address 0.00H as a data memory
                                    ;location.
MEM018  MEM     0.18H
MEM000  MEM     0.00H
LOOP1 :
        SET1    IXE                 ; IXE flag <– 1
        ST      MEM018, MEM000      ; (0.1 x H) <– (0.00H)
        CLR1    IXE                 ; IXE flag <– 0
        INC     IX                  ; IX <– IX + 1
        SKGE    IXL, #08H
        BR      LOOP1
```



(3) MOV @r, m — Move data memory to destination indirect

<1> Instruction code

10 8 7 4 3 0

01010	m_R	m_C	r

<2> Function

When MPE = 1

(MP,(r)) <– (m)

When MPE = 0

(BANK, m_R ,(r)) <– (m)

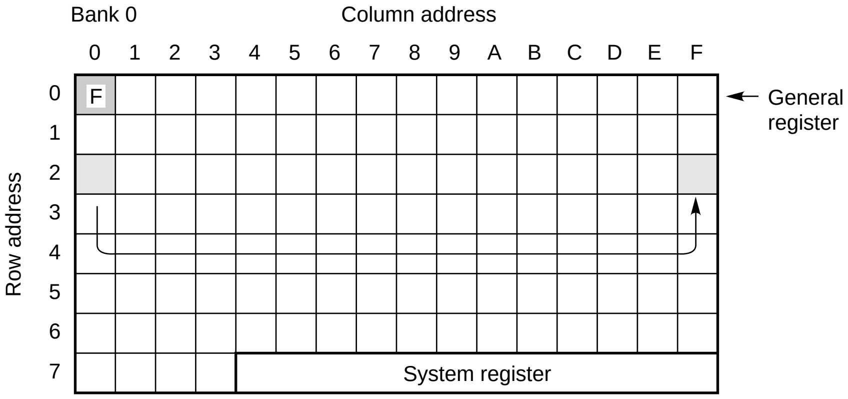
The value of a data memory location is stored at the data memory location addressed by the value of a general register. When MPE = 0, a transfer takes place within the same row address of the same bank.

<3> Example 1:

The MPE flag is set to 0, and the value of address 0.20H is stored at address 0.2FH. The transfer destination data memory location takes the same row address as the transfer source, and the value of address 0.00H of the general register as its column address.

(0.2FH) <– (0.20H)

```
MEM000    MEM     0.00H
MEM020    MEM     0.20H
          CLR1    MPE                 ; MPE flag <– 0
          MOV     MEM000, #0FH        ; Stores a column address in the general register.
          MOV     @MEM000, MEM020; Performs a move operation.
```

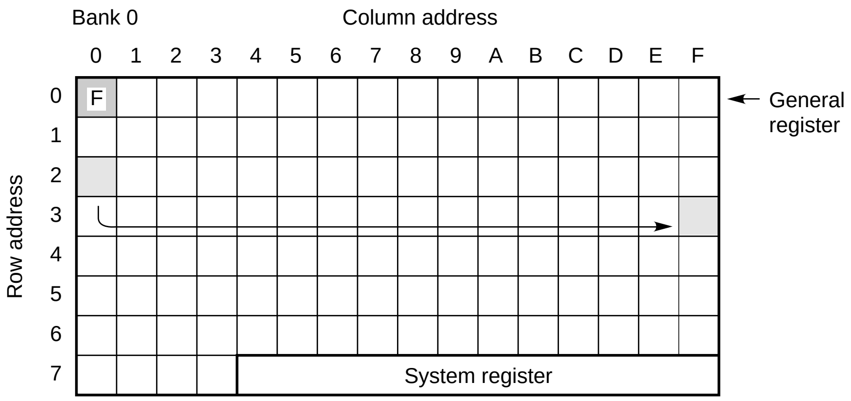


Example 2:

The MPE flag is set to 1, and the value of address 0.20H is stored at address 0.3FH. The transfer destination data memory location takes the value of the memory pointer MP as its row address, and the value of address 0.00H of the general register as its column address.

(0.3FH) <– (0.20H)

```
MEM000    MEM     0.00H
MEM020    MEM     0.20H
          MOV     RPH, #00H          ; Selects bank 0 to specify a general register.
          MOV     RPL, #00H          ; Selects row address 0 to specify a general register.
          MOV     00H, #0FH          ; Stores a column address into the general register.
          MOV     MPH, #00H          ; Stores a row address into the memory pointer.
          MOV     MPL, #03H          ;
          SET1    MPE                ; MPE flag <– 1
          MOV     @MEM000, MEM020; Performs a move operation.
```



(4) MOV m, @r — Move data memory to destination indirect

<1> Instruction code

	10 8	7 4	3 0
11010	m_R	m_C	r

<2> Function

When MPE = 1

(m) <– (MP,(r))

When MPE = 0

(m) <– (BANK, m_R ,(r))

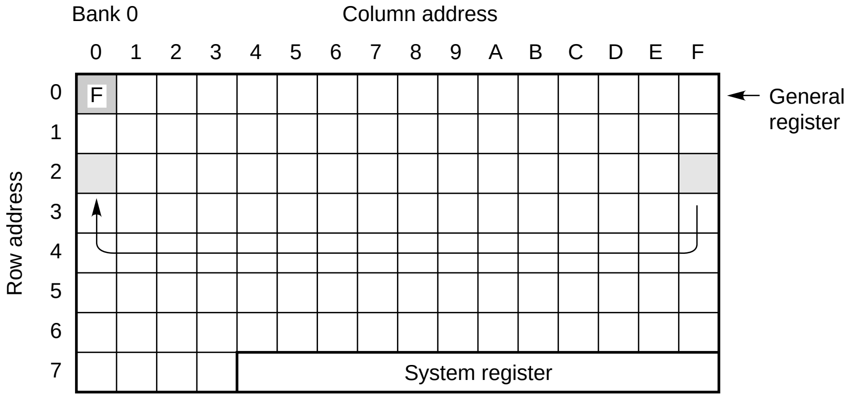
The value stored at the data memory location addressed by the value of a general register is stored at another data memory location. When MPE = 0, a transfer is performed within the same row address of the same bank.

<3> Example 1:

The MPE flag is set to 0, and the value of address 0.2FH is stored at address 0.20H. The transfer source data memory location takes the same row address as the transfer destination, and the value of address 0.00H of the general register as its column address.

(0.20H) <– (0.2FH)

```
MEM000    MEM    0.00H
MEM020    MEM    0.20H
          CLR1   MPE                 ; MPE flag <– 0
          MOV    MEM000, #0FH        ; Stores a column address into the general register.
          MOV    MEM020, @MEM000     ; Performs a move operation.
```

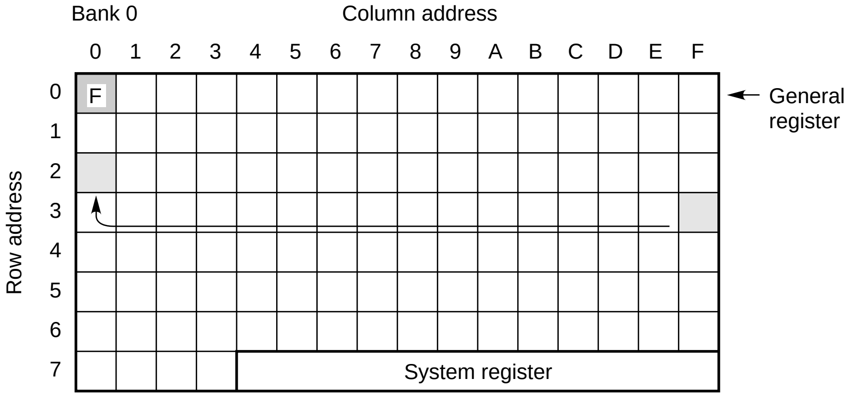


Example 2:

The MPE flag is set to 1, and the value stored at address 0.3FH is stored at address 0.20H. The transfer source data memory location takes the value of the memory pointer MP as its row address, and the value of address 0.00H of the general register as its column address.

(0.20H) <– (0.3FH)

```
MEM000    MEM    0.00H
MEM020    MEM    0.20H
          MOV    MEM000, #0FH        ; Stores a column address into the general register.
          MOV    MPH, #00H           ; Stores a row address into the memory pointer.
          MOV    MPL, #03H           ;
          SET1   MPE                 ; MPE flag <–1
          MOV    MEM020, @MEM000     ; Performs a move operation.
```



(5) MOV m, #n4 — Move immediate data to data memory

<1> Instruction code

	10 8	7 4	3 0
11101	m_R	m_C	n4

<2> Function

(m) <– n4

Immediate data is stored at a data memory location.

<3> Example 1:

The immediate data 0AH is stored at data memory location 0.50H.

(0.50H) <– 0AH

```
MEM050      MEM     0.50H
            MOV     MEM050, #0AH
```

Example 2:

When address 0.00H is specified as a data memory location, and IXH = 0, IXM = 3, IXL = 2, and IXE flag = 1, immediate data 07H is stored at address 0.32H.

(0.32H) <– 07H

```
MEM000      MEM     0.00H
            MOV     IXH, #00H           ; IX <– 00000110010B (0.32H)
            MOV     IXM, #03H
            MOV     IXL, #02H
            SET1    IXE                 ; IXE flag <–1
            MOV     MEM000, #07H
```

(6) MOVT DBF, @AR — Move program memory data specified by AR to DBF

<1> Instruction code

00111	000	0001	0000

<2> Function

SP <– SP – 1, ASR <– PC, PC <– AR,

DBF <– (PC), PC <– ASR, SP <– SP + 1

The value of the program memory location addressed by the address register AR is stored in data buffer DBF.

This instruction temporarily uses one level of the stack. So, be careful when nesting subroutines and interrupts.

<3> Example

Sixteen-bit table data is transferred to the data buffer consisting of DBF3, DBF2, DBF1, and DBF0 according to the value of the address register consisting of AR3, AR2, AR1, and AR0.

```
; *
; * *     Table data
; *
ORG       10H
DW        0000000000000000B    ; (0000H)
DW        1010101111001101B    ; (0ABCDH)

                    ⋮

; *
; * *  Table reference program
; *
MOV       AR3, #00H         ; AR3 <– 00H   Stores 0011H into the address register.
MOV       AR2, #00H         ; AR2 <– 00H
MOV       AR1, #01H         ; AR1 <– 01H
MOV       AR0, #01H         ; AR0 <– 01H
MOVT      DBF, @AR          ; <–  Transfers data held at addresses 0011H to DBF.
```

In this case, the data is stored in DBF as follows:
DBF3 = 0AH
DBF2 = 0BH
DBF1 = 0CH
DBF0 = 0DH

(7) PUSH AR — Push address register

<1> Instruction code

00111	000	1101	0000

<2> Function

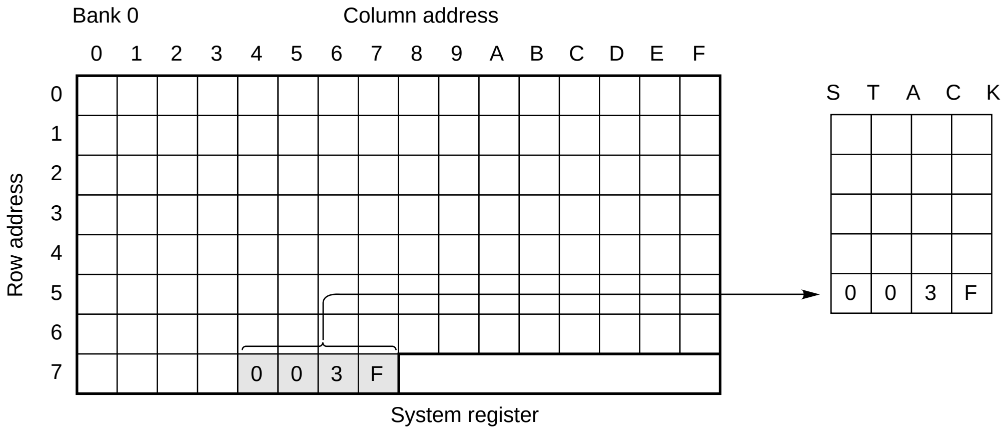
SP <– SP – 1,
ASR <– AR

The stack pointer SP is decremented by 1, after which the value of the address register AR is stored to the address stack register pointed to by the stack pointer.

<3> **Example 1:**

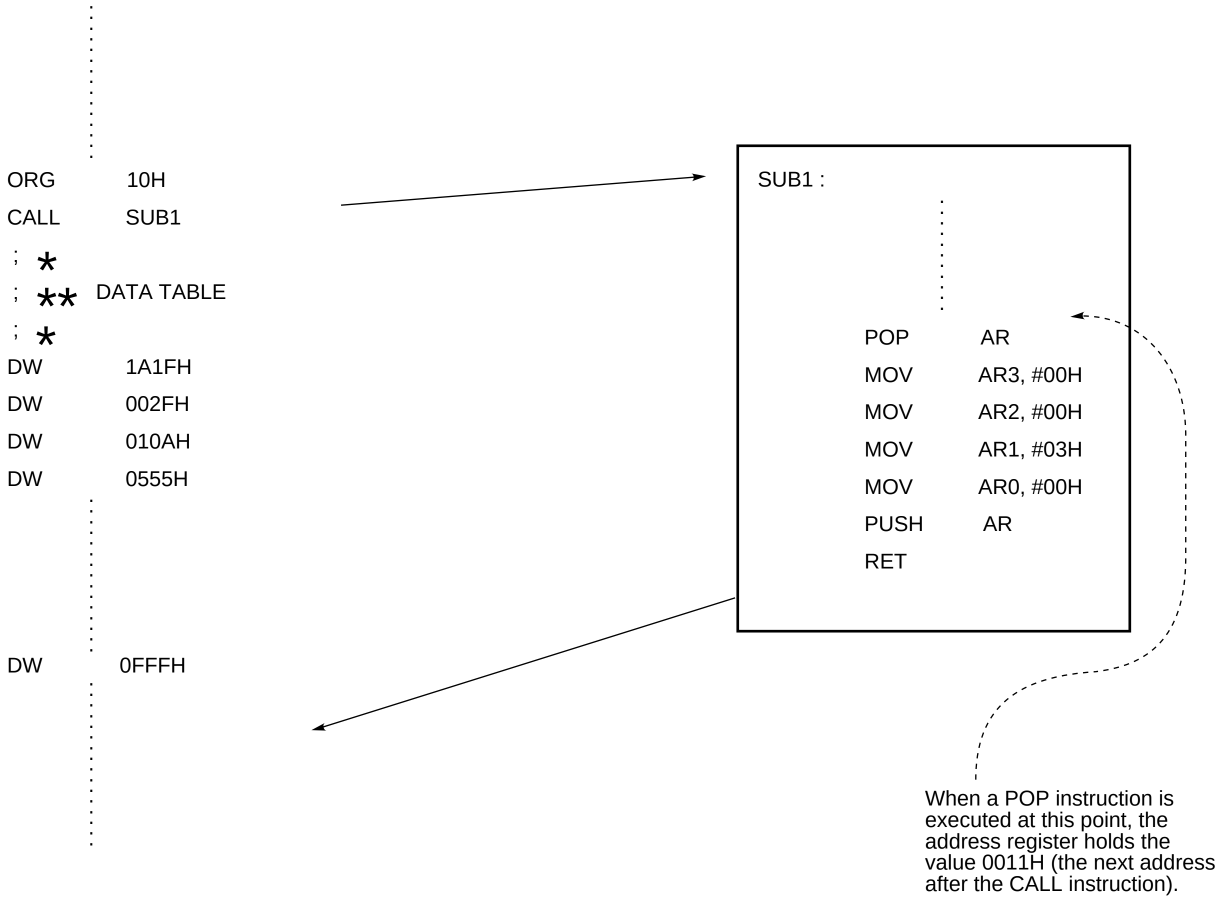
The value 003FH is loaded into the address register, then is transferred to the stack.

```
MOV        AR3, #00H
MOV        AR2, #00H
MOV        AR1, #03H
MOV        AR0, #0FH
PUSH       AR
```



Example 2:

When a CALL instruction is followed by a data table, the return address (the next address after the data table) of the subroutine is stored in the address register to return control to the return address.



```
ORG        10H
CALL       SUB1
;  *
;  **  DATA TABLE
;  *
DW         1A1FH
DW         002FH
DW         010AH
DW         0555H

DW         0FFFH
```

```
SUB1 :

          POP        AR
          MOV        AR3, #00H
          MOV        AR2, #00H
          MOV        AR1, #03H
          MOV        AR0, #00H
          PUSH       AR
          RET
```

When a POP instruction is executed at this point, the address register holds the value 0011H (the next address after the CALL instruction).

(8) POP AR　　　　Pop address register

<1> Instruction code

00111	000	1100	0000

<2> Function

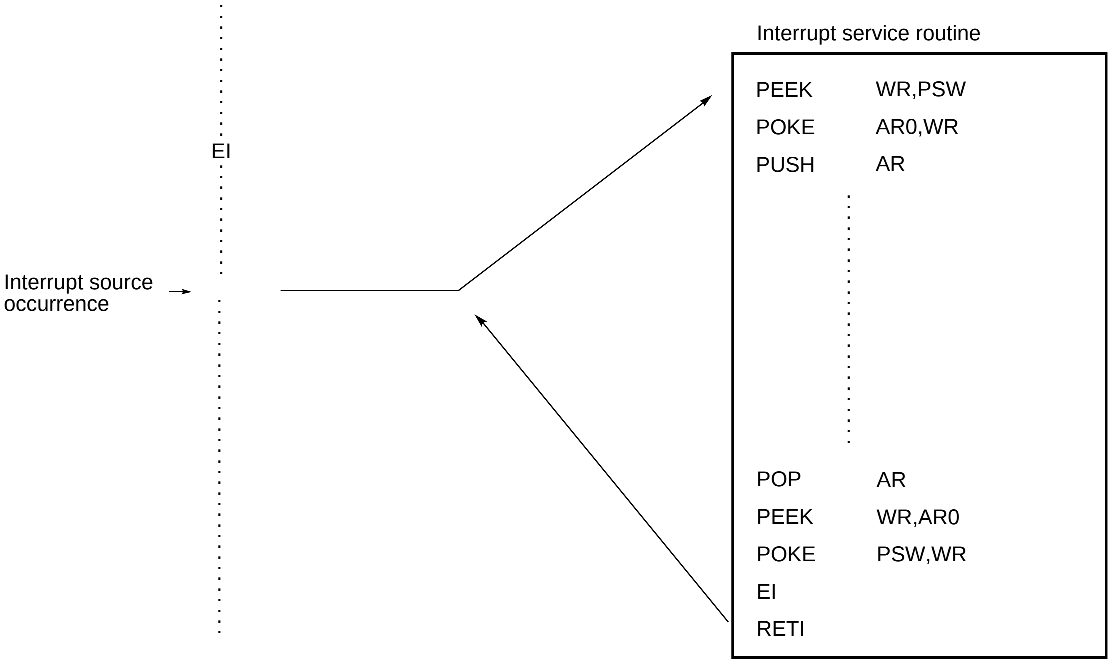
AR <– ASR,

SP <– SP + 1

The value stored in the address stack register pointed to by the stack pointer SP is transferred to address register AR, after which the stack pointer SP is incremented by one.

<3> Example

In interrupt handling, the interrupt service routine may change the contents of PSW. In such a case, the contents of PSW are transferred via WR to the address register at the start of interrupt handling, then are saved to the address stack register with the PUSH instruction. Then, before a return, the contents of PSW are returned to the address register with the POP instruction, then to PSW via WR.



```
PEEK    WR,PSW
POKE    AR0,WR
PUSH    AR
   :
POP     AR
PEEK    WR,AR0
POKE    PSW,WR
EI
RETI
```

(9) PEEK WR, rf — Peek register file to window register

<1> Instruction code

	10 8	7 4	3 0
00111	rf$_R$	0011	rf$_C$

<2> Function

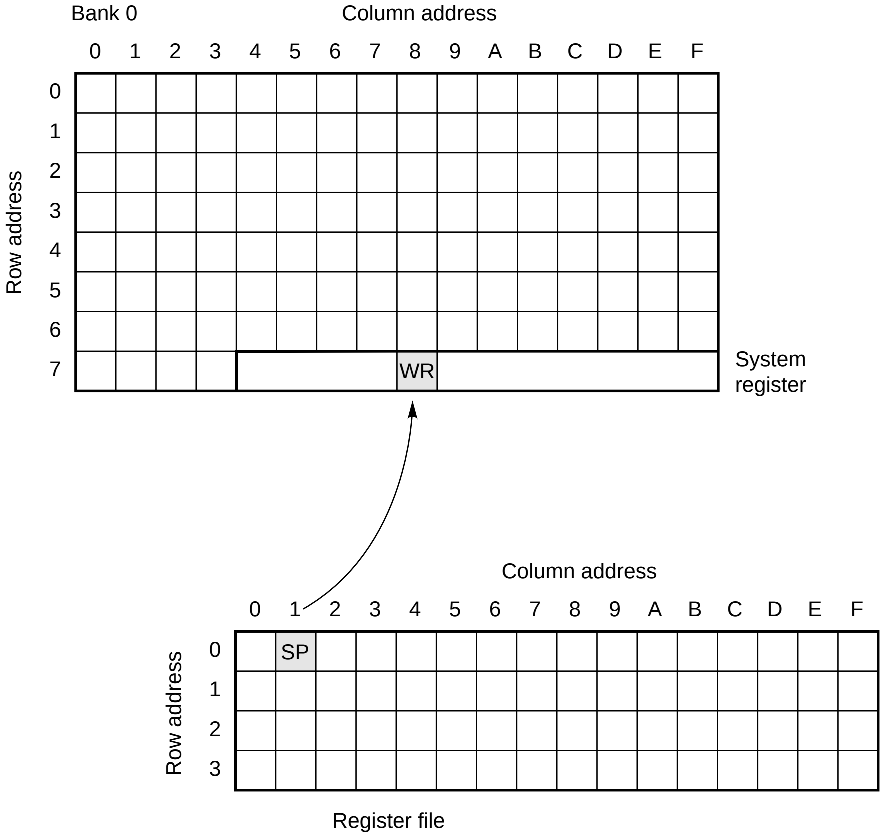
WR <– (rf)

Register file data is stored to the window register WR.

<3> Example

The value of the stack pointer SP at address 01H in the register file is stored to the window register.

```
PEEK        WR, SP
```



(10) POKE rf, WR   Poke window register to register file

<1> Instruction code

	10 8	7 4	3 0
00111	rf_R	0010	rf_C

<2> Function

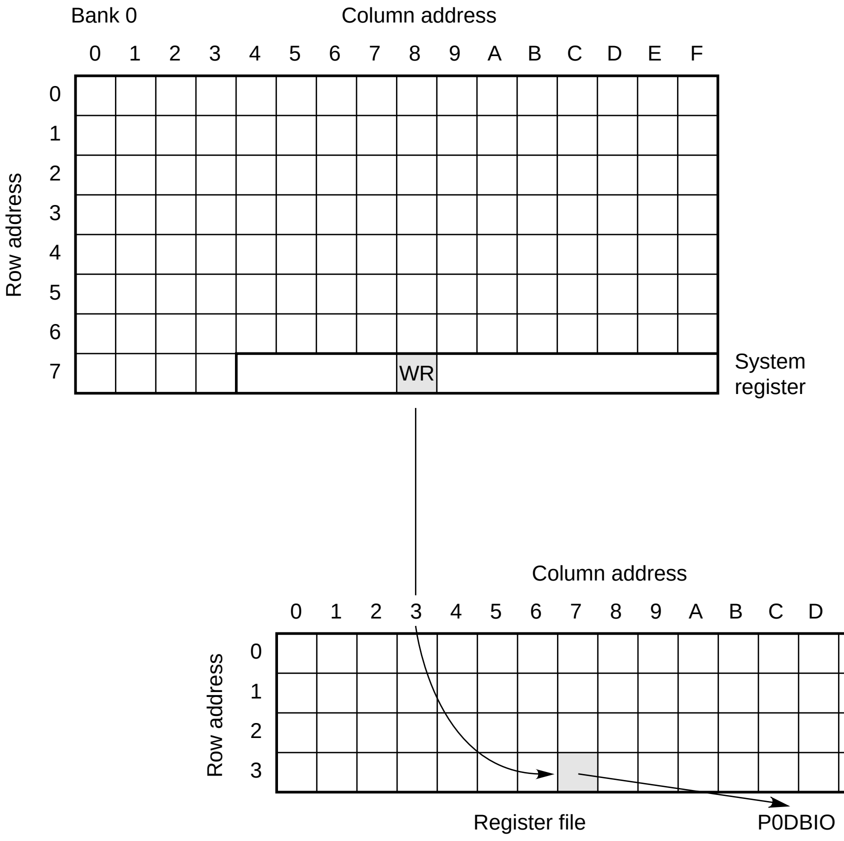
(rf) <– WR

The value of the window register WR is stored to the register file.

<3> Example

The immediate data 0FH is stored via the window register to P0DBIO of the register file.

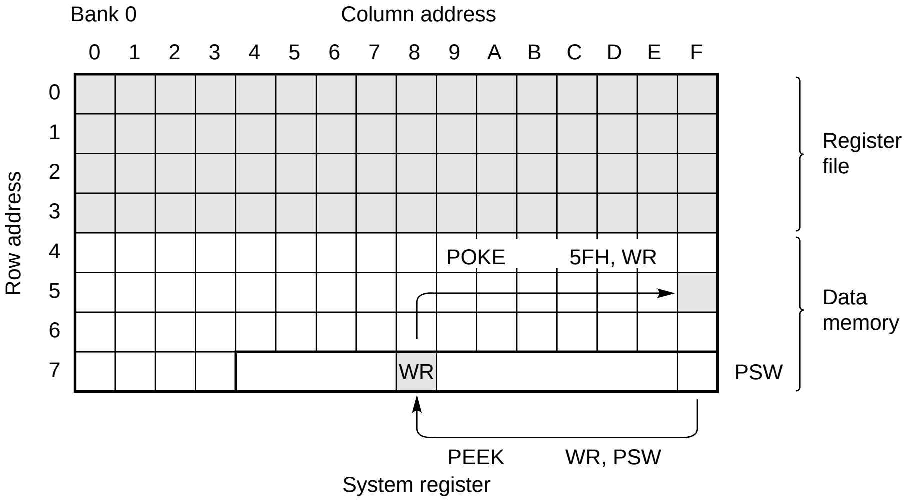
```
MOV     WR, #0FH
POKE    P0DBIO, WR          ; Places P0D0, P0D1, P0D2, and P0D3, in output
                            ; mode.
```



<4> Caution

When viewed from the program, memory at addresses 40H to 7FH in the register file seems to be the same as that at addresses 40H to 7FH in the data memory. Therefore, the PEEK instruction and POKE instruction can access address 40H to address 7FH of each bank of data memory as well as the register file. For example, these instructions can be used as follows:

```
MEM05F      MEM     0.5FH
            PEEK    WR, PSW           ; Stores the contents of PSW (7FH) in the system
                                      ; register to WR.
            POKE    MEM05F, WR        ; Stores the contents of WR to data memory
                                      ; address 5FH.
```



(11) GET DBF, p — Get peripheral data to data buffer

<1> Instruction code

	10 8	7 4	3 0
00111	p_H	1011	p_L

<2> Function

DBF <– (p)

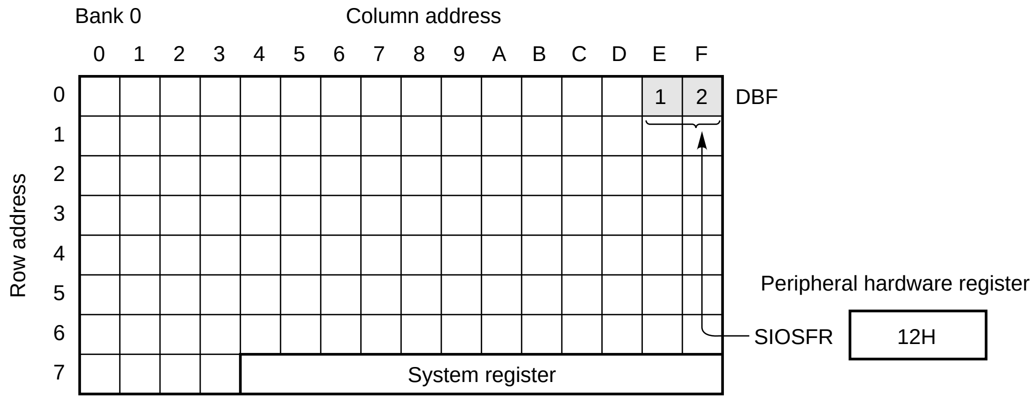
The value of a peripheral hardware register is stored to data buffer DBF.

The DBF area consists of 16 bits at addresses 0CH to 0FH in the data memory BANK0, regardless of the value of the bank register.

<3> Example

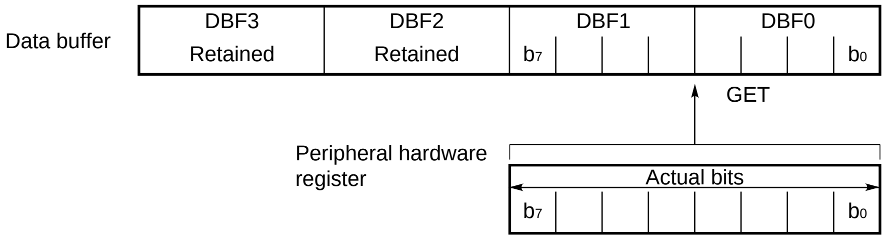
The value stored in the serial interface shift register SIOSFR (8 bits) is stored to DBF0 and DBF1 in the data buffer.

```
GET         DBF, SIOSFR
```



<4> Caution

The data buffer consists of 16 bits. However, the number of bits to be accessed depends on the peripheral hardware. For example, when the GET instruction is executed for the peripheral hardware register whose actual bit length is 8 bits, the contents of the peripheral hardware register are stored in the low-order 8 bits (DBF1 and DBF0) of data buffer DBF.



(12) PUT p, DBF — Put data buffer to peripheral

<1> Instruction code

	10 8	7 4	3 0
00111	p_H	1010	p_L

<2> Function

(p) <– DBF

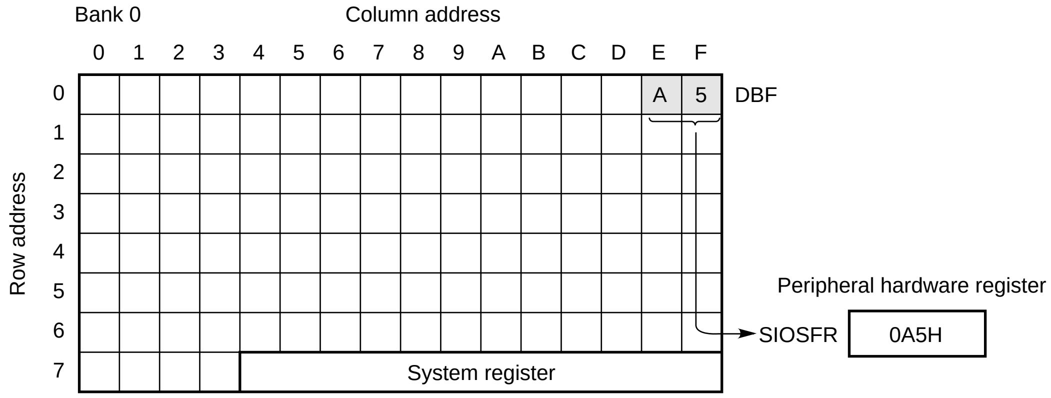
The value of data buffer DBF is stored to a peripheral hardware register.

The DBF area consists of 16 bits at addresses 0CH to 0FH in the data memory BANK0, regardless of the value of the bank register.

<3> Example

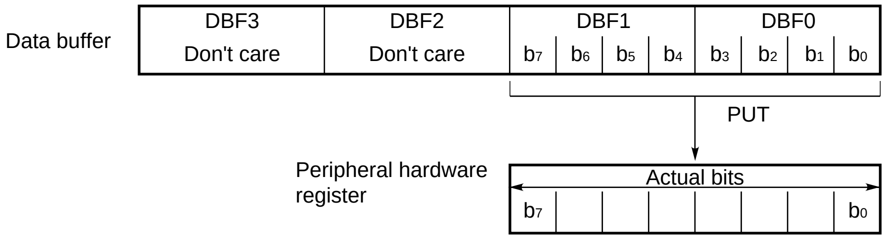
The values 0AH and 05H are stored at DBF1 and DBF0 of the data buffer, respectively. Then, the values are transferred to the serial interface shift register (SIOSFR), which is a peripheral register.

```
MOV         BANK, #00H              ; Selects bank 0 of the data memory.
MOV         DBF0, #05H
MOV         DBF1, #0AH
PUT         SIOSFR, DBF
```



<4> Caution

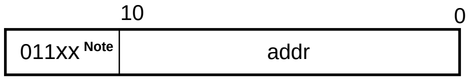
The data buffer consists of 16 bits. However, the number of bits to be accessed depends on the peripheral hardware. For example, when the PUT instruction is executed for the peripheral hardware register whose actual bit length is 8 bits, the contents of the low-order 8 bits (DBF1 and DBF0) of data buffer DBF are stored in the peripheral hardware register (the contents of DBF2 and DBF3 are invalid).



20.5.8 Branch Instructions

(1) BR addr **Branch to the address**

<1> Instruction code



Note See **Item <4> (Caution)** below.

<2> Function

PC <– addr

A branch to the address specified by addr is caused.

<3> Example

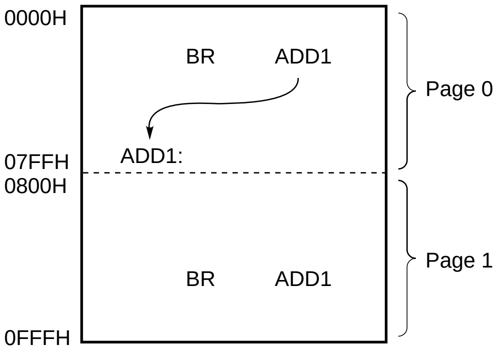
```
FLY         LAB     0FH             ; Defines FLY = 0FH.
             ⋮
            BR      FLY             ; Jumps to address 0FH.
             ⋮
            BR      LOOP1           ; Jumps to LOOP1.
             ⋮
            BR      $ + 2           ; Jumps to the address two addresses below the
             ⋮      ;               current address.
            BR      $ – 3           ; Jumps to the address three addresses above the
             ⋮                      ; current address.

LOOP1 :
```

<4> Caution

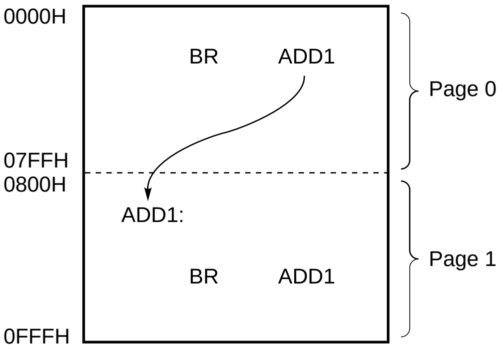
The concept of page is not involved in the writing of a BR instruction by the assembly language; the same coding as that of the BR instruction can be used between ROM addresses 0000H to 1FFFH. However, a different operation code is used for the BR instruction to cause a branch to page 0 (addresses 0000H to 07FFH) and for the BR instruction to cause a branch to page 1 (addresses 07FFH to 0FFFH).

The operation code used for a branch to page 0 is 0C. The operation code used for a branch to page 1 is 0D. When the assembler of the 17K series is used for assembly, these codes are automatically converted by referencing branch destinations.

When operation code 0C is to be used (when the address of a jump destination is in page 0)

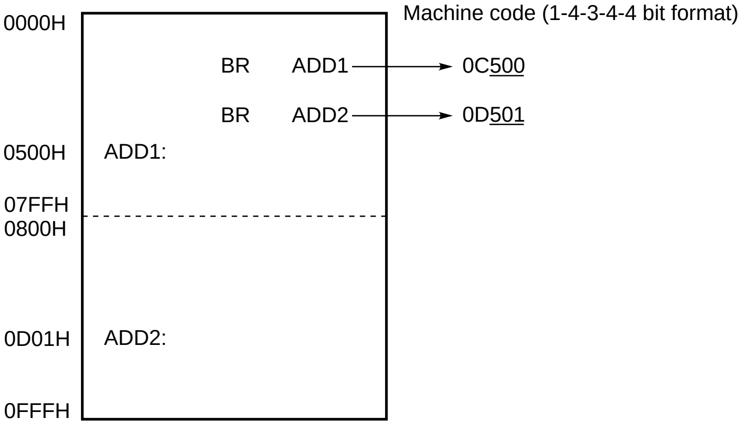


When operation code 0D is to be used (when the address of a jump destination is in page 1)



In debugging, take care of the operation code when patching is performed using machine code directly rather than the assembler (Both 0C and 0D must be converted).

When a BR instruction specifies a branch to an address between 0800H and 0FFFH, address conversion is needed. This means that a converted address is an address incremented by one, starting from 000H for address 0800H.



Caution The number of pages depends on which µPD17145 sub-series product is being used.

(2) BR @AR — Branch to the address specified by address register

<1> Instruction code

00111	000	0100	0000

<2> Function

PC <– AR

A branch to the program address specified by address register AR is caused.

<3> Example 1:

The value 003FH is stored in address register AR (AR0-AR3), after which the BR @AR instruction causes a jump to address 003FH.

```
MOV         AR3, #00H                  ; AR3 <– 00H
MOV         AR2, #00H                  ; AR2 <– 00H
MOV         AR1, #03H                  ; AR1 <– 03H
MOV         AR0, #0FH                  ; AR0 <– 0FH
BR          @AR                        ; Jumps to address 003FH.
```

Example 2:

The branch destination depends on the value stored at data memory address 0.10H, as follows:

Value of address 0.10H		Branch destination label
00H	–>	AAA
01H	–>	BBB
02H	–>	CCC
03H	–>	DDD
04H	–>	EEE
05H	–>	FFF
06H	–>	GGG
07H	–>	HHH
08H - 0FH	–>	ZZZ

```
            ; *
            ; * * Jump table
            ; *
            ORG     10H
            BR      AAA
            BR      BBB
            BR      CCC
            BR      DDD
            BR      EEE
            BR      FFF
            BR      GGG
            BR      HHH
            BR      ZZZ

                            ⋮

MEM010      MEM     0.10H
            MOV     AR3, #00H          ; AR3 <– 00H  Stores 001xH in AR.
            MOV     AR2, #00H          ; AR2 <– 00H
            MOV     AR1, #01H          ; AR1 <– 01H
            MOV     RPH, #00H          ; Selects bank 0 to specify a general register.
            MOV     RPL, #02H          ; Selects row address 1 to specify a general register.
```

```
   ST      AR0, MEM010      ; AR0 <– 0.10H
*  SKLT    AR0, #08H
*  MOV     AR0, #08H        ; Changes the value of AR0 to 08H if the value of AR0
                            ; is greater than 08H.
   BR      @AR
```

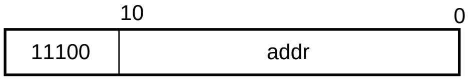
<4> Caution

The number of bits available with address register AR (AR0-AR3) depends on the product being. When using the address register, always check the data sheet of the product.

20.5.9 Subroutine Instructions

(1) CALL addr — Call subroutine

<1> Instruction code



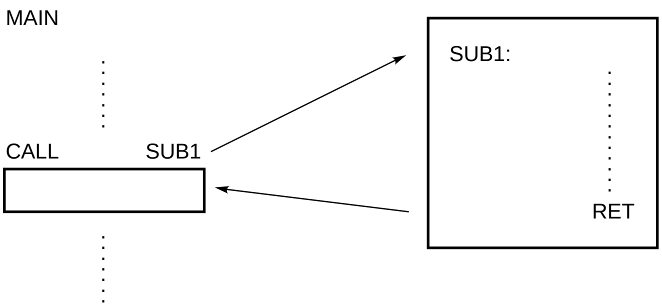
<2> Function

SP <– SP – 1, ASR <– PC

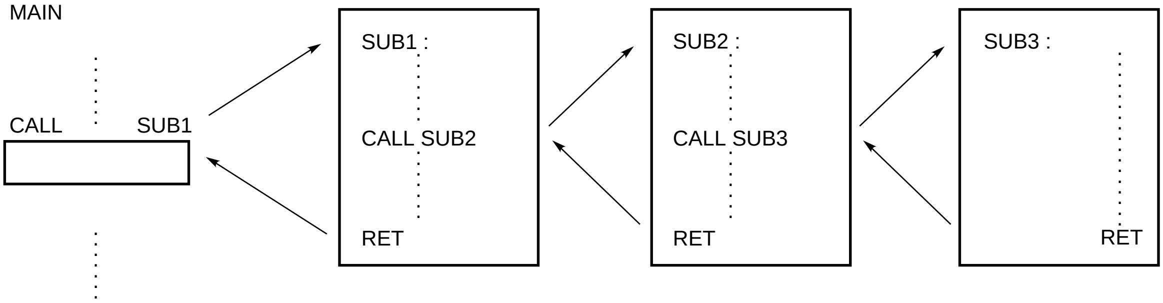
PC <– addr, PAGE <– 0

The value of the program counter PC is incremented and stored in the stack, after which a branch to the subroutine specified by addr is caused.

<3> Example 1:



Example 2:



(2) CALL @AR Call subroutine specified by address register

<1> Instruction code

00111	000	0101	0000

<2> Function

SP <– SP – 1,
ASR <– PC,
PC <– AR

The value of the program counter PC is incremented and stored in the stack, after which a branch to a subroutine, starting at the address specified by address register AR, is caused.

<3> Example 1:

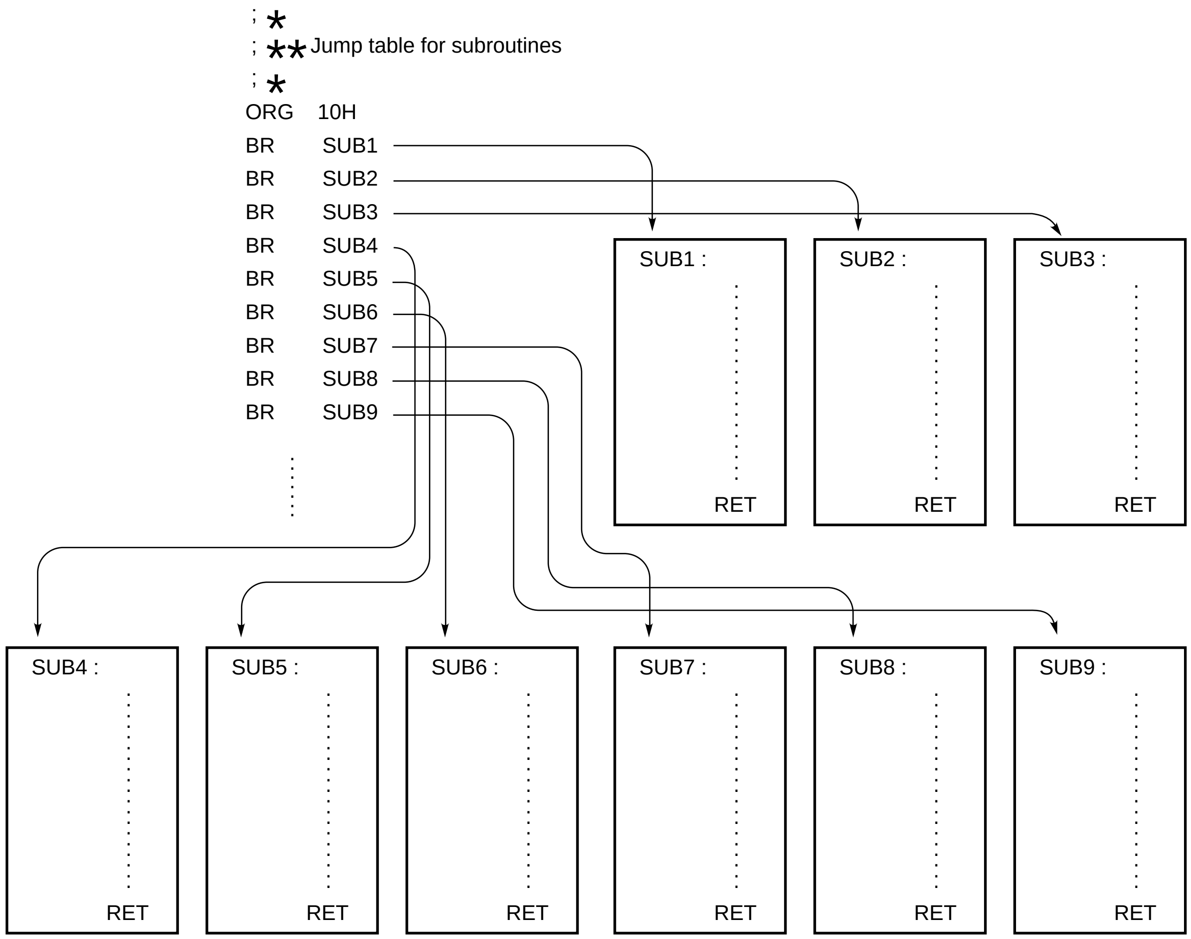
The value 0020H is stored in address register AR (AR0-AR3), after which the CALL @AR instruction calls the subroutine stored at address 0020H.

```
MOV     AR3, #00H          ; AR3 <– 00H
MOV     AR2, #00H          ; AR2 <– 00H
MOV     AR1, #02H          ; AR1 <– 02H
MOV     AR0, #00H          ; AR0 <– 00H
CALL    @AR                ; Calls the subroutine stored at address 0020H.
```

Example 2:

A subroutine is called according to the value stored at data memory address 0.10H, as described below.

Value of address 0.10H		Subroutine name
00H	–>	SUB1
01H	–>	SUB2
02H	–>	SUB3
03H	–>	SUB4
04H	–>	SUB5
05H	–>	SUB6
06H	–>	SUB7
07H	–>	SUB8
08H - 0FH	–>	SUB9



```
;*
;** Jump table for subroutines
;*
ORG   10H
BR    SUB1
BR    SUB2
BR    SUB3
BR    SUB4
BR    SUB5
BR    SUB6
BR    SUB7
BR    SUB8
BR    SUB9
  :
```

```
      :
   MOV    AR3,   #00H   ; AR3 <– 00H  Stores 001xH in the address register.
   MOV    AR2,   #00H   ; AR2 <– 00H
   MOV    AR1,   #01H   ; AR1 <– 01H
   MOV    RPH,   #00H   ; Selects bank 0 to specify a general register.
   MOV    RPL,   #02H   ; Selects row address 1 to specify a general register.
   ST     AR0,   10H    ; AR0 <– 0.10H
*  SKLT   AR0,   #08H   ; Changes the value of AR0 to 08H if the value of
*  MOV    AR0,   #08H   ; AR0 is greater than 08H.
   CALL   @AR           ————→ To the jump table
      :
```

To the jump table

Control returns to this point when the RET instruction is executed by each subroutine.

<4> **Caution**

The number of bits available with address register AR (AR0-AR3) depends on the product being used. When using the address register, always check the data sheet of the product.

(3) RET Return to the main program from a subroutine

<1> Instruction code

00111	000	1110	0000

<2> Function

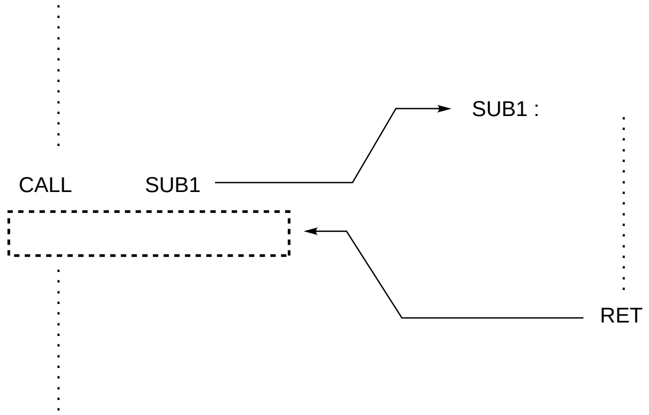
PC <– ASR,

SP <– SP + 1

The RET instruction is used to return from a subroutine to the main program.

A return address, saved to the stack by the CALL instruction, is loaded back into the program counter.

<3> Example



(4) RETSK Return to the main program then skip the next instruction

<1> Instruction code

00111	001	1110	0000

<2> Function

PC <– ASR, SP <– SP + 1 and skip

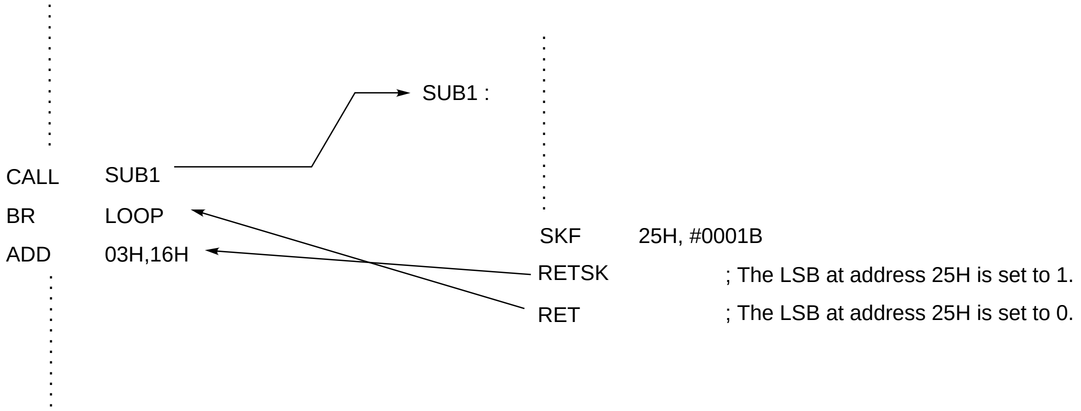
The RETSK instruction is used to return from a subroutine to the main program.

The instruction immediately after the CALL instruction is skipped (that is, executed as an NOP instruction).

This means that a return address, saved to the stack by the CALL instruction, is loaded back into the program counter PC, after which the program counter is incremented.

<3> Example

When the least significant bit (LSB) of data memory (RAM) address 25H is set to 0, the RET instruction is executed, after which control returns to the instruction immediately subsequent to the CALL instruction. When the LSB is set to 1, the RETSK instruction is executed, after which control returns to the instruction after the instruction that is immediately subsequent to the CALL instruction (ADD 03H,16H in this example).



```
                                   SUB1 :
CALL    SUB1
BR      LOOP
ADD     03H,16H                    SKF      25H, #0001B
                                   RETSK             ; The LSB at address 25H is set to 1.
                                   RET               ; The LSB at address 25H is set to 0.
```

(5) RETI Return to the main program from an interrupt service routine

<1> Instruction code

00111	100	1110	0000

<2> Function

PC <– ASR, INTR <– INTSK, SP <– SP + 1

The RETI instruction is used to return from the interrupt service routine to the main program.

A return address, saved to the stack by a vectored interrupt, is loaded back into the program counter.

In addition, part of the system register (PSWORD) also returns to the state existing prior to the generation of a vectored interrupt.

20.5.10 Interrupt Instructions

(1) EI Enable interrupt

<1> Instruction code

00111	000	1111	0000

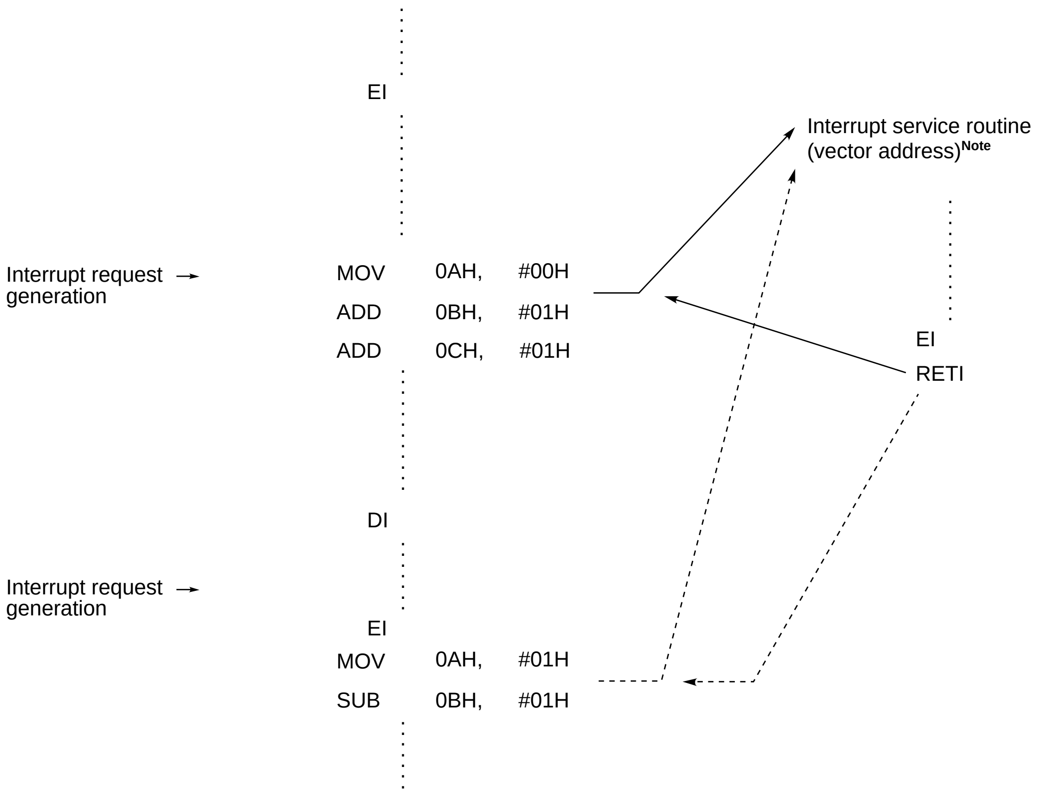
<2> Function

INTEF <– 1

The EI instruction enables a vectored interrupt.

<3> Example

An example of the interrupt handling is shown below.



```
EI

MOV   0AH,  #00H
ADD   0BH,  #01H
ADD   0CH,  #01H

DI

EI
MOV   0AH,  #01H
SUB   0BH,  #01H
```

```
EI
RETI
```

Interrupt request generation
Interrupt request generation
Interrupt service routine (vector address)Note

Note Before an interrupt can be accepted (an interrupt request is generated after execution of the EI instruction, then control is transferred to the interrupt service routine), the interrupt enable flag (IPxxx) for the interrupt must be set. When an interrupt request is generated after the completion of execution of the EI instruction, the flow of program control does not change (the interrupt is not accepted) if the corresponding interrupt enable flag is not set. Note, however, that the interrupt request flag (IRQxxx) is set, so that the interrupt is accepted when the interrupt enable flag is set.

(2) DI — Disable interrupt

<1> Instruction code

00111	001	1111	0000

<2> Function

INTEF <– 0

The DI instruction disables a vectored interrupt.

<3> Example

See Example of (1) (for EI) above.

20.5.11 Other Instructions

(1) STOP s **Stop CPU and release by condition s**

<1> Instruction code

			3 0
00111	010	1111	s

<2> Function

This instruction stops the system clock, then places the device in STOP mode.
By placing the device in STOP mode, the supply current can be minimized.
A condition for cancelling STOP mode must be specified using operand s.
For information about the STOP mode cancelling condition s, see **Section 15.3**.

(2) HALT h **Halt CPU and release by condition h**

<1> Instruction code

			3 0
00111	011	1111	h

<2> Function

This instruction places the device in HALT mode.
By placing the device in HALT mode, the supply current can be reduced.
A condition for cancelling HALT mode is to be specified using operand h.
For information about the HALT mode cancelling condition h, see **Section 15.2**.

(3) NOP **No operation**

<1> Instruction code

00111	100	1111	0000

<2> Function

This instruction occupies one machine cycle without performing any operation.

CHAPTER 21   ASSEMBLER RESERVED WORDS

21.1   MASK OPTION PSEUDO INSTRUCTIONS

The µPD17145, µPD17147,and µPD17149 has the following mask options.

- Built-in pull-up resistor for pin $\overline{\text{RESET}}$
- Built-in pull-up resistors for pins P0F_1 and P0F_0
- Built-in pull-up resistor for pin INT
- Incorporated POC circuit

Specify whether these mask options are used in the source program using mask-option-definition pseudo instructions.

21.1.1 Specifying Mask Options

Mask options are coded in the assembler source program with the following pseudo instructions.

- OPTION and ENDOP pseudo instructions
- Mask-option-definition pseudo instructions

(1) OPTION and ENDOP pseudo instructions

These pseudo instructions specify the block in which mask options are coded (mask option definition block).

To specify a mask option, code the corresponding mask-option-definition pseudo instruction in the block enclosed with the OPTION pseudo instruction and ENDOP pseudo instruction.

Format:

Symbol	Mnemonic	Operand	Comment
[label:]	OPTION		[;comment]
	⋮		
	ENDOP		

(2) Mask option definition pseudo instructions

Table 21-1. Mask Option Definition Pseudo Instructions

Option	Definition pseudo instruction and format	Operand	Meaning
Built-in pull-up resistor for pin $\overline{\text{RESET}}$	OPTRES <operand>	OPEN	Without pull-up resistor
		PULLUP	With pull-up resistor
Built-in pull-up resistors for pins P0F$_1$ and P0F$_0$	OPTP0F <operand-1>, <operand-2>Note	OPEN	Without pull-up resistor
		PULLUP	With pull-up resistor
Built-in pull-up resistor for pin INT	OPTINT <operand>	OPEN	Without pull-up resistor
		PULLUP	With pull-up resistor
Incorporated POC circuit	OPTPOC <operand>	NOUSE	Without POC circuit
		USE	With POC circuit

Note <operand-1> and <operand-2> specify the mask options for the P0F$_1$ and P0F$_0$ pins, respectively.

(3) Example of specifying mask options

```
; Example of specifying mask options in µPD17149

MASK_OPTION:

OPTION                    ; Beginning of mask option definition block
OPTRES    PULLUP          ; RESET pin has the built-in pull-up resistor.
OPTP0F    PULLUP, OPEN;   P0F1 pin has the built-in pull-up resistor.
                          ; P0F0 pin leaves open (externa pull-up).
OPTINT    PULLUP          ; INT pin has the built-in pull-up resistor.
OPTPOC    NOUSE           ; Internal POC circuit is not used.
ENDOP                     ; End of mask option definition block
```

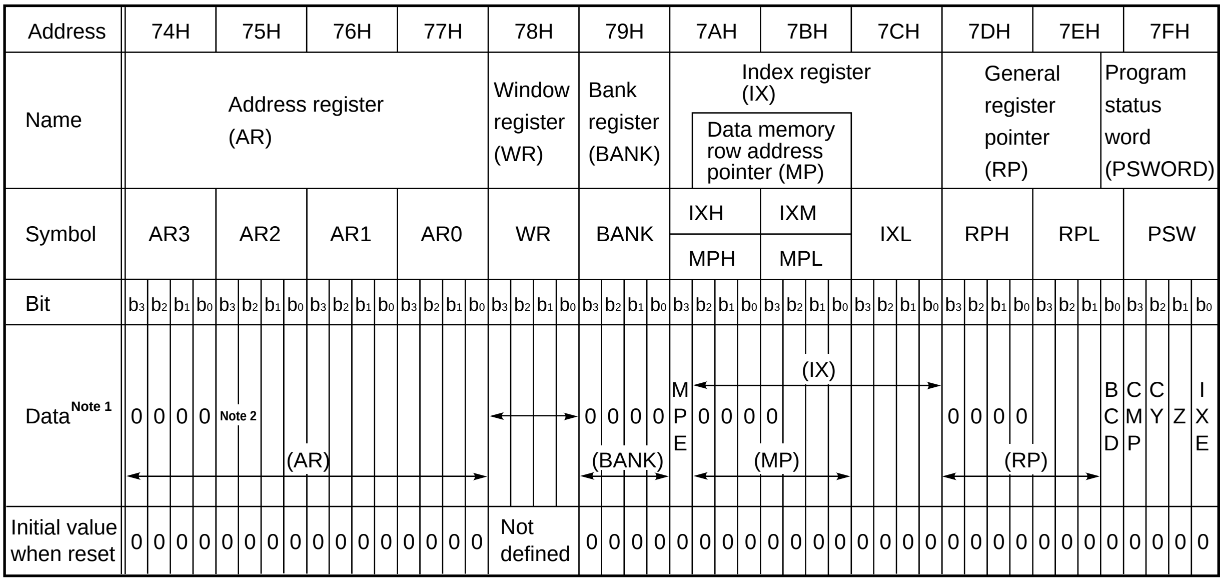
21.2 RESERVED SYMBOLS

The reserved symbols defined in the µPD17149 device file (AS17149) are listed below.

System register (SYSREG)

Symbolic name	Attribute	Value	Read/write	Description
AR3	MEM	0.74H	R	Bits 15 to 12 of the address register
AR2	MEM	0.75H	R/W	Bits 11 to 8 of the address register
AR1	MEM	0.76H	R/W	Bits 7 to 4 of the address register
AR0	MEM	0.77H	R/W	Bits 3 to 0 of the address register
WR	MEM	0.78H	R/W	Window register
BANK	MEM	0.79H	R/W	Bank register
IXH	MEM	0.7AH	R/W	Index register high
MPH	MEM	0.7AH	R/W	Data memory row address pointer high
MPE	FLG	0.7AH.3	R/W	Memory pointer enable flag
IXM	MEM	0.7BH	R/W	Index register middle
MPL	MEM	0.7BH	R/W	Data memory row address pointer low
IXL	MEM	0.7CH	R/W	Index register low
RPH	MEM	0.7DH	R/W	General register pointer high
RPL	MEM	0.7EH	R/W	General register pointer low
PSW	MEM	0.7FH	R/W	Program status word
BCD	FLG	0.7EH.0	R/W	BCD flag
CMP	FLG	0.7FH.3	R/W	Compare flag
CY	FLG	0.7FH.2	R/W	Carry flag
Z	FLG	0.7FH.1	R/W	Zero flag
IXE	FLG	0.7FH.0	R/W	Index enable flag

Figure 21-1. System Register Configuration



Address	74H	75H	76H	77H	78H	79H	7AH	7BH	7CH	7DH	7EH	7FH
Name	Address register (AR)				Window register (WR)	Bank register (BANK)	Index register (IX) Data memory row address pointer (MP)			General register pointer (RP)		Program status word (PSWORD)
Symbol	AR3	AR2	AR1	AR0	WR	BANK	IXH MPH	IXM MPL	IXL	RPH	RPL	PSW
Bit	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0
Data Note 1	0 0 0 0	Note 2	(AR)		←→	0 0 0 0 (BANK)	MPE 0 0 0	0 (IX) (MP)		0 0 0 0	(RP) BCD	CMP CY Z IXE
Initial value when reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	Not defined	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Notes 1. A bit indicating zero is fixed to 0.

2. Bit b_3 and b_2 of AR2 are always 0 for the μPD17145. Bit b_3 of AR2 is always 0 also for the μPD17147.

Data buffer (DBF)

Symbolic name	Attribute	Value	Read/write	Description
DBF3	MEM	0.0CH	R/W	DBF bits 15 to 12
DBF2	MEM	0.0DH	R/W	DBF bits 11 to 8
DBF1	MEM	0.0EH	R/W	DBF bits 7 to 4
DBF0	MEM	0.0FH	R/W	DBF bits 3 to 0

Port register

Symbolic name	Attribute	Value	Read/write	Description
P0A3	FLG	0.70H.3	R/W	Port 0A bit 3
P0A2	FLG	0.70H.2	R/W	Port 0A bit 2
P0A1	FLG	0.70H.1	R/W	Port 0A bit 1
P0A0	FLG	0.70H.0	R/W	Port 0A bit 0
P0B3	FLG	0.71H.3	R/W	Port 0B bit 3
P0B2	FLG	0.71H.2	R/W	Port 0B bit 2
P0B1	FLG	0.71H.1	R/W	Port 0B bit 1
P0B0	FLG	0.71H.0	R/W	Port 0B bit 0
P0C3	FLG	0.72H.3	R/W	Port 0C bit 3
P0C2	FLG	0.72H.2	R/W	Port 0C bit 2
P0C1	FLG	0.72H.1	R/W	Port 0C bit 1
P0C0	FLG	0.72H.0	R/W	Port 0C bit 0
P0D3	FLG	0.73H.3	R/W	Port 0D bit 3
P0D2	FLG	0.73H.2	R/W	Port 0D bit 2
P0D1	FLG	0.73H.1	R/W	Port 0D bit 1
P0D0	FLG	0.73H.0	R/W	Port 0D bit 0
P0E3	FLG	0.6EH.3	R/W	Port 0E bit 3
P0E2	FLG	0.6EH.2	R/W	Port 0E bit 2
P0E1	FLG	0.6EH.1	R/W	Port 0E bit 1
P0E0	FLG	0.6EH.0	R/W	Port 0E bit 0
P0F1	FLG	0.6FH.1	R	Port 0F bit 1
P0F0	FLG	0.6FH.0	R	Port 0F bit 0

Register file (control register)

(1/2)

Symbolic name	Attribute	Value	Read/write	Description
SP	MEM	0.81H	R/W	Stack pointer
SIOTS	FLG	0.82H.3	R/W	Serial interface start flag
SIOHIZ	FLG	0.82H.2	R/W	Function selection flag for the P0D$_1$/SO pin
SIOCK1	FLG	0.82H.1	R/W	Bit 1 of serial clock selection flag
SIOCK0	FLG	0.82H.0	R/W	Bit 0 of serial clock selection flag
WDTRES	FLG	0.83H.3	R/W	Watchdog timer reset flag
WDTEN	FLG	0.83H.0	R/W	Watchdog timer enable flag
TM1OSEL	FLG	0.8BH.3	R/W	Function selection flag for the P0D$_3$/$\overline{\text{TM1OUT}}$ pin
SIOEN	FLG	0.8BH.0	R/W	Serial interface enable flag
P0EGPU	FLG	0.8CH.2	R/W	P0E group pull-up selection flag (1: Pull-up)
P0BGPU	FLG	0.8CH.1	R/W	P0B group pull-up selection flag (1: Pull-up)
P0AGPU	FLG	0.8CH.0	R/W	P0A group pull-up selection flag (1: Pull-up)
P0DBPU3	FLG	0.8DH.3	R/W	P0D$_3$ pull-up selection flag (1: Pull-up)
P0DBPU2	FLG	0.8DH.2	R/W	P0D$_2$ pull-up selection flag (1: Pull-up)
P0DBPU1	FLG	0.8DH.1	R/W	P0D$_1$ pull-up selection flag (1: Pull-up)
P0DBPU0	FLG	0.8DH.0	R/W	P0D$_0$ pull-up selection flag (1: Pull-up)
INT	FLG	0.8FH.0	R	INT pin status flag
TM0EN	FLG	0.91H.3	R/W	Timer 0 enable flag
TM0RES	FLG	0.91H.2	R/W	Timer 0 reset flag
TM0CK1	FLG	0.91H.1	R/W	Bit 1 of count pulse selection flag for timer 0
TM0CK0	FLG	0.91H.0	R/W	Bit 0 of count pulse selection flag for timer 0
TM1EN	FLG	0.92H.3	R/W	Timer 1 enable flag
TM1RES	FLG	0.92H.2	R/W	Timer 1 reset flag
TM1CK1	FLG	0.92H.1	R/W	Bit 1 of count pulse selection flag for timer 1
TM1CK0	FLG	0.92H.0	R/W	Bit 0 of count pulse selection flag for timer 1
BTMISEL	FLG	0.93H.3	R/W	Selection flag for the BTM interval
BTMRES	FLG	0.93H.2	R/W	Basic interval timer reset flag
BTMCK1	FLG	0.93H.1	R/W	Bit 1 of count pulse selection flag for the basic interval timer
BTMCK0	FLG	0.93H.0	R/W	Bit 0 of count pulse selection flag for the basic interval timer
P0C3IDI	FLG	0.9BH.3	R/W	ADC$_3$ and P0C$_3$ pin function selection flag
P0C2IDI	FLG	0.9BH.2	R/W	ADC$_2$ and P0C$_2$ pin function selection flag
P0C1IDI	FLG	0.9BH.1	R/W	ADC$_1$ and P0C$_1$ pin function selection flag
P0C0IDI	FLG	0.9BH.0	R/W	ADC$_0$ and P0C$_0$ pin function selection flag

Register file (control register)

(2/2)

Symbolic name	Attribute	Value	Read/write	Description
P0CBIO3	FLG	0.9CH.3	R/W	Input/output selection flag for $P0C_3$ (1: Output port)
P0CBIO2	FLG	0.9CH.2	R/W	Input/output selection flag for $P0C_2$ (1: Output port)
P0CBIO1	FLG	0.9CH.1	R/W	Input/output selection flag for $P0C_1$ (1: Output port)
P0CBIO0	FLG	0.9CH.0	R/W	Input/output selection flag for $P0C_0$ (1: Output port)
IEGMD1	FLG	0.9FH.1	R/W	Bit 1 of detection edge selection flag for the INT pin
IEGMD0	FLG	0.9FH.0	R/W	Bit 0 of detection edge selection flag for the INT pin
ADCSTRT	FLG	0.0A0H.0	R/W	A/D converter start flag (always 0 when read)
ADCSOFT	FLG	0.0A1H.3	R/W	A/D converter operating mode selection flag (1: Single mode)
ADCCMP	FLG	0.0A1H.1	R	A/D converter comparison result flag (valid only in the single mode)
ADCEND	FLG	0.0A1H.0	R	A/D converter conversion end flag
ADCCH3	FLG	0.0A2H.3	R/W	Dummy flag
ADCCH2	FLG	0.0A2H.2	R/W	Dummy flag
ADCCH1	FLG	0.0A2H.1	R/W	Bit 1 of channel selection flag for the A/D converter
ADCCH0	FLG	0.0A2H.0	R/W	Bit 0 of channel selection flag for the A/D converter
P0DBIO3	FLG	0.0ABH.3	R/W	Input/output selection flag for $P0D_3$ (1: Output port)
P0DBIO2	FLG	0.0ABH.2	R/W	Input/output selection flag for $P0D_2$ (1: Output port)
P0DBIO1	FLG	0.0ABH.1	R/W	Input/output selection flag for $P0D_1$ (1: Output port)
P0DBIO0	FLG	0.0ABH.0	R/W	Input/output selection flag for $P0D_0$ (1: Output port)
P0EGIO	FLG	0.0ACH.2	R/W	Group input/output selection flag for P0E (1: All P0E pins are output ports.)
P0BGIO	FLG	0.0ACH.1	R/W	Group input/output selection flag for P0B (1: All P0B pins are output ports.)
P0AGIO	FLG	0.0ACH.0	R/W	Group input/output selection flag for P0A (1: All P0A pins are output ports.)
IPSIO	FLG	0.0AEH.0	R/W	Interrupt enable flag for the serial interface
IPBTM	FLG	0.0AFH.3	R/W	Interrupt enable flag for the basic interval timer
IPTM1	FLG	0.0AFH.2	R/W	Interrupt enable flag for timer 1
IPTM0	FLG	0.0AFH.1	R/W	Interrupt enable flag for timer 0
IP	FLG	0.0AFH.0	R/W	Interrupt enable flag for the INT pin
IRQSIO	FLG	0.0BBH.0	R/W	Interrupt request flag for the serial interface
IRQBTM	FLG	0.0BCH.0	R/W	Interrupt request flag for the basic interval timer
IRQTM1	FLG	0.0BDH.0	R/W	Interrupt request flag for timer 1
IRQTM0	FLG	0.0BEH.0	R/W	Interrupt request flag for timer 0
IRQ	FLG	0.0BFH.0	R/W	Interrupt request flag for the INT pin

Peripheral hardware register

Symbolic name	Attribute	Value	Read/write	Description
SIOSFR	DAT	01H	R/W	Peripheral address of the shift register
TM0M	DAT	02H	W	Peripheral address of the timer 0 modulo register
TM1M	DAT	03H	W	Peripheral address of the timer 1 modulo register
ADCR	DAT	04H	R/W	Peripheral address of A/D converter data register
TM0TM1C	DAT	45H	R	Peripheral address of timer 0 timer 1 count register
AR	DAT	40H	R/W	Peripheral address of the address register for GET, PUT, PUSH, CALL, BR, MOVT, and INC instructions

Others

Symbolic name	Attribute	Value	Description
DBF	DAT	0FH	Fixed operand value for a GET/PUT/MOVT instruction
IX	DAT	01H	Fixed operand value for an INC instruction

[MEMO]

Figure 21-2. Control Register Configuration (1/2)

Column address Row address	Item	0				1				2				3				4				5				6				7			
0 (8)	Symbol					0	SP			SIOTS	SIOHIZ	SIOCK1	SIOCK0	WDTRES	0	0	WDTEN																
	When reset					0	1	0	1	0	0	0	0	0	0	0	0																
	Read/ Write					R/W				R/W				R/W																			
1 (9)	Symbol					TM0EN	TM0RES	TM0CK1	TM0CK0	TM1EN	TM1RES	TM1CK1	TM1CK0	BTMISEL	BTMRES	BTMCK1	BTMCK0																
	When reset					0	0	0	0	1	0	0	0	0	0	0	0																
	Read/ Write					R/W				R/W				R/W																			
2 (A)	Symbol	0	0	0	ADCSTRT	ADCSOFT	0	ADCCMP	ADCEND	ADCCH3	ADCCH2	ADCCH1	ADCCH0																				
	When reset	0	0	0	0	0	0	0	0	0	0	0	0																				
	Read/ Write	R/W				R/W		R		R/W																							
3 (B)	Symbol																																
	When reset																																
	Read/ Write																																

Remark The address enclosed in parentheses apply when the assembler is used.
The names of all the flags in the control registers are assembler reserved words saved in the device file. Using these reserved words is useful in programming.

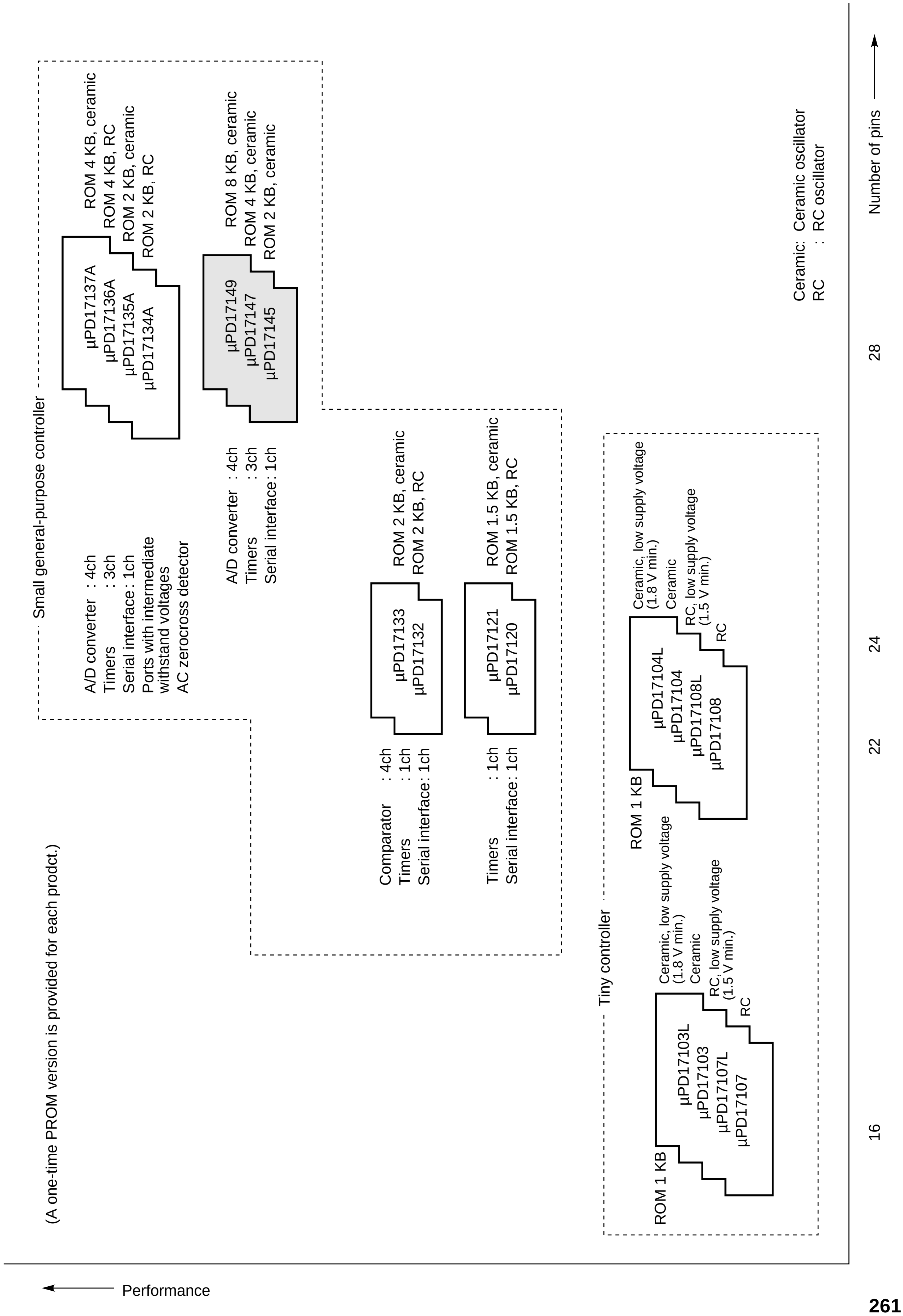
Figure 21-2. Control Register Configuration (2/2)

8				9				A				B				C				D				E				F			
												TM1OSEL	0	0	SIOEN	0	P0EGPU	P0BGPU	P0AGPU	P0DBPU3	P0DBPU2	P0DBPU1	P0DBPU0					0	0	0	INT
												0	0	0	0	0	0	0	0	0	0	0	0					0	0	0	Note
												R/W				R/W				R/W								R			
												P0C3IDI	P0C2IDI	P0C1IDI	P0C0IDI	P0CBIO3	P0CBIO2	P0CBIO1	P0CBIO0									0	0	IEGMD1	IEGMD0
												0	0	0	0	0	0	0	0									0	0	0	0
												R/W				R/W												R/W			
												P0DBIO3	P0DBIO2	P0DBIO1	P0DBIO0	0	P0EGIO	P0BGIO	P0AGIO					0	0	0	IPSIO	IPBTM	IPTM1	IPTM0	IP
												0	0	0	0	0	0	0	0					0	0	0	0	0	0	0	0
												R/W				R/W								R/W				R/W			
												0	0	0	IRQSIO	0	0	0	IRQBTM	0	0	0	IRQTM1	0	0	0	IRQTM0	0	0	0	IRQ
												0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
												R/W				R/W				R/W				R/W				R/W			

Note The INT flag depends on the status of the INT pin. *

[MEMO]

APPENDIX A   DEVELOPMENT OF THE µPD171xx SUB-SERIES



[MEMO]

APPENDIX B   COMPARISON OF FUNCTIONS OF µPD17145 SUB-SERIES, µPD17135A, AND µPD17137A

(1/2)

		µPD17145	µPD17147	µPD17149	µPD17135A	µPD17137A	
ROM		2K bytes	4K bytes	8K bytes	2K bytes	4K bytes	
RAM		110 x 4 bits			112 x 4 bits		
Stack		Five levels of address stack Three levels of interrupt stack					
Instruction execution time (clock, supply voltage)		2 µs (f_x = 8 MHz, V_{DD} = 4.5 to 5.5 V) 4 µs (f_x = 4 MHz, V_{DD} = 3.6 to 5.5 V) 8 µs (f_x = 2 MHz, V_{DD} = 2.7 to 5.5 V)			2 µs (f_x = 8 MHz, V_{DD} = 4.5 to 5.5 V) 4 µs (f_x = 4 MHz, V_{DD} = 2.7 to 5.5 V)		*
I/O	CMOS I/O	12 (P0A, P0B, P0C)					
	Dedicated to input	2 ($P0F_0$, $P0F_1$)			2 ($P1B_0$)		
	Sense input	1 (INT) Pull-up by mask option enabled			1 (INT)		
	N-ch open-drain I/O	8 (P0D, P0E Withstand voltage : V_{DD}) P0D pull-up : Software P0E pull-up : Software			8 (P0D, P1A Withstand voltage : 9 V) P0D pull-up : Mask option P1A pull-up : Mask option		
Built-in pull-up resistance		100 kΩ TYP. (other than P0D) 10 kΩ TYP. (P0D)			100 kΩ TYP.		
A/D converter (supply voltage)		Four 8-bit channels (V_{DD} = 4.0 to 5.5 V)			Four 8-bit channels (V_{DD} = 4.5 to 5.5 V)		
	Reference voltage pin	V_{REF} (V_{REF} = 2.5 V to V_{DD})			None ($V_{REF} = V_{ADC} = V_{DD}$)		
Timer	8-bit (TM0, TM1)	2 (Timer output : $\overline{\text{TM1OUT}}$) TM0 clock : f_x/512 f_x/64 f_x/16 INT TM1 clock : f_x/8192 f_x/128 f_x/16 TM1 count-up			2 (Timer output : $\overline{\text{TM0OUT}}$) TM0 clock : f_x/256 f_x/64 f_x/16 INT TM1 clock : f_x/1024 f_x/512 f_x/256 TM1 count-up		*
	Basic interval (BTM)	1 (Also used as a watchdog timer) Count pulse : f_x/16384 f_x/4096 f_x/512 f_x/16			1 (Also used as a watchdog timer) Count pulse : f_x/8192 f_x/4096 TM0 count-up INT		

(2/2)

		µPD17145	µPD17147	µPD17149	µPD17135A	µPD17137A
Inter-rupt	External	1			1 (AC zero cross detection is performed.)	
	Internal	4 (TM0, TM1, BTM, SIO)				
* SIO		1 (Clock synchronous, three-wire)				
	Internal	Independent of $P0D_1$ latch			Also used for $P0D_1$ latch	
Stanby function		HALT, STOP (Input pin $\overline{\text{RLS}}$ is provided for cancellation.)			HALT, STOP	
Oscillation settling time		128 x 256 count			512 x 256 count	
POC function		Mask option			Built-in	
Package		28-pin plastic SDIP (400 mil) 28-pin plastic SOP (375 mil)				
One-time PROM		µPD17P149			µPD17P137A	

Caution µPD17145 sub-series, µPD17135A, and µPD17137A are not pin-compatible. Furthermore, the µPD17145 sub-series does not feature units that are equivalent to the µPD17134A and µPD17136A (RC oscillation type). For the electrical characteristics of these units, refer to the data sheet for each unit.

Remark f_X : System clock oscillation frequency

APPENDIX C   DEVELOPMENT TOOLS

The following support tools are available for developing programs for the µPD17145 sub–series.

Hardware

Name	Description
In-circuit emulator 17K IE-17K IE-17K-ET Note 1 EMU-17K Note 2	The IE-17K, IE-17K-ET, and EMU-17K are in-circuit emulators applicable to the series. The IE-17K and IE-17K-ET are connected to the PC-9800 series (host machine) or IBM PC/AT™ through the RS-232-C interface. The EMU-17K is inserted into the extension slot of the PC-9800 series (host machine). Use the system evaluation board (SE board) corresponding to each product together with one of these in-circuit emulators. *SIMPLEHOST*®, a man machine interface, implements an advanced debug environment. The EMU-17K also enables user to check the contents of the data memory in real time.
SE board (SE-17145)	The SE-17145 is an SE board for the µPD17145 sub-series. It is used solely for evaluating the system. It is also used for debugging in combination with the in-circuit emulator.
Emulation probe (EP-17K28CT)	The EP-17K28CT is an emulation probe for the 17K series 28-pin shrink DIP (400 mil).
Emulation probe (EP-17K28GT)	The EP-17K28GT is an emulation probe for the 17K series 28-pin SOP (375 mil). Use this probe together with the conversion adapter EV-9500GT-28 Note 3, to check the target system with the corresponding SE board.
Conversion adapter (EV-9500GT-28 Note 3)	The EV-9500GT-28 is an adapter for the 28-pin SOP (375 mil). Use this conversion adapter to connect the emulation probe, EP-17K28GT, to the target system.
PROM Programmer Note 4 AF-9703, AF-9704, AF-9705 or AF-9706	The AF-9703, AF-9704, AF-9705 and AF-9706 are PROM writers for the µPD17P149. Use one of these PROM writers with the program adapter, AF-9808M, to program the µPD17P149.
Programmer adapter Note 4 (AF-9808M)	The AF-9808M is a socket unit for the µPD17P149. It is used with the AF-9703, AF-9704, AF-9705 or AF-9706.

Notes 1. Low-end model, operating on an external power supply

2. The EMU-17K is a product of IC Co., Ltd. Contact IC Co., Ltd. (Tokyo, 03-3447-3793) for details.

3. An EP-17K28GT is supplied together with two EV-9500GT-28s. A set of five EV-9500GT-28s is also available.

4. The AF-9703, AF-9704, AF-9705, AF-9706, and AF9808-M are products of Ando Electric Co., Ltd. Contact Ando Electric Co., Ltd. (Tokyo 03-3733-1151) for details.

Software

Name	Description machine	Host	OS	Distribution media	Part number
17K series assembler (AS17K)	AS17K is an assembler applicable to the 17K series. In developing µPD17145, µPD17147, or µPD17149 programs, AS17K is used in combination with a device file (AS17145, AS17147, or AS17149).	PC-9800 series	MS-DOS$^{\text{TM}}$	5.25-inch, 2HD	µS5A10AS17K
				3.5-inch, 2HD	µS5A13AS17K
		IBM PC/AT$^{\text{TM}}$	PC DOS$^{\text{TM}}$	5.25-inch, 2HC	µS7B10AS17K
				3.5-inch, 2HC	µS7B13AS17K
Device file (AS17145 AS17147 AS17149)	AS17145, AS17147, and AS17149 are device files for the µPD17145, µPD17147, µPD17149, and µPD17P149. They are used together with the assembler (AS17K) which is applicable to the 17K series.	PC-9800 series	MS-DOS	5.25-inch, 2HD	µS5A10AS17145[Note]
				3.5-inch, 2HD	µS5A13AS17145[Note]
		IBM PC/AT	PC DOS	5.25-inch, 2HC	µS7B10AS17145[Note]
				3.5-inch, 2HC	µS7B13AS17145[Note]
Support software (*SIMPLEHOST*)	*SIMPLEHOST*, running on the Windows$^{\text{TM}}$, provides man machine-interface in developing programs by using a personal computer and the in-circuit emulator.	PC-9800 series	MS-DOS Windows	5.25-inch, 2HD	µS5A10IE17K
				3.5-inch, 2HD	µS5A13IE17K
		IBM PC/AT	PC DOS	5.25-inch, 2HC	µS7B10IE17K
				3.5-inch, 2HC	µS7B13IE17K

Note µSxxxxAS17145 contains AS17145, AS17147, and AS17149.

Remark The following table lists the versions of the operating systems described in the above table.

OS	Versions
MS-DOS	Ver. 3.30 to Ver. 5.00A[Note]
PC DOS	Ver. 3.1 to Ver. 5.0[Note]
Windows	Ver. 3.0 to Ver. 3.1

Note MS-DOS versions 5.00 and 5.00A and PC DOS Ver. 5.0 are provided with a task swap function. This function, however, cannot be used in these software packages.

APPENDIX D   MASK ROM ORDERING PROCEDURE

After program development is completed, the mask ROM is ordered by the following procedure:

(1) Order booking for mask ROM

Give an advance notice of mask ROM ordering to a NEC's special agent or Sales Department, otherwise the ordered products may be delivered with delay.

(2) Preparation of media for ordering

A UV-EPROM is used as media for ordering a mask ROM.

Add a /PROM to an assemble option for the assembler (AS17K) to create a hexadecimal file for ordering a mask ROM. An extension of its file must be .PRO**Note**.

Then, write the hexadecimal file for ordering a mask ROM to a UV-EPROM. Prepare three UV-EPROMs having the same contents in ordering a mask ROM.

Note An hexadecimal file having .ICE cannot be used for ordering a mask ROM.

(3) Preparation of the required documents

Prepare the following documents when ordering a mask ROM:

- Mask format ROM order sheet
- Mask format ROM order check sheet

(4) Ordering

Send the media created in (2) and the documents created in (3) to a NEC's special agent or Sales Department by the date of order booking.

[MEMO]

APPENDIX E   INSTRUCTION INDEX

E.1   INSTRUCTION INDEX (BY FUNCTION)

E.2 INSTRUCTION INDEX (ALPHABETICAL ORDER)

APPENDIX F REVISION HISTORY

*

A revision history is shown below. Chapters appearing in the revised-chapter column indicate those in the new edition.

Edition	Major changes	Revised chapter
Second	**Figure 13-1** has been modified. **Section 13.1.6** has been added. **Section 13.1.7** has been added. **Figure 13-21** has been changed.	Chapter 13
	Caution 2 for **Table 15-1** has been modified. **Section 15.3.3** has been changed.	Chapter 15
	Section 17.3 has been added.	Chapter 17
	Note 1 has been added to **Section 20.3**.	Chapter 20
	Appendix B has been modified.	Appendix B

[MEMO]