USER'S MANUAL

NEC

μPD17120 SUBSERIES

4-BIT SINGLE-CHIP MICROCONTROLLER

μPD17120
μPD17121
μPD17132
μPD17133
μPD17P132
μPD17P133

Document No. IEU-1367A
(O. D. No. IEU-835A)
Date Published July 1995 P
Printed in Japan

NOTES FOR CMOS DEVICES

① PRECAUTION AGAINST ESD FOR SEMICONDUCTORS

Note:

Strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it once, when it has occurred. Environmental control must be adequate. When it is dry, humidifier should be used. It is recommended to avoid using insulators that easily build static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work bench and floor should be grounded. The operator should be grounded using wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with semiconductor devices on it.

② HANDLING OF UNUSED INPUT PINS FOR CMOS

Note:

No connection for CMOS device inputs can be cause of malfunction. If no connection is provided to the input pins, it is possible that an internal input level may be generated due to noise, etc., hence causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using a pull-up or pull-down circuitry. Each unused pin should be connected to V_{DD} or GND with a resistor, if it is considered to have a possibility of being an output pin. All handling related to the unused pins must be judged device by device and related specifications governing the devices.

③ STATUS BEFORE INITIALIZATION OF MOS DEVICES

Note:

Power-on does not necessarily define initial status of MOS device. Production process of MOS does not define the initial operation status of the device. Immediately after the power source is turned ON, the devices with reset function have not yet been initialized. Hence, power-on does not guarantee out-pin levels, I/O settings or contents of registers. Device is not initialized until the reset signal is received. Reset operation must be executed immediately after power-on for devices having reset function.

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.

INTRODUCTION

Targeted Reader

This manual is intended for the user engineers who understand functions of the μPD17120 subseries and design their application systems using the μPD17120 subseries

Purpose

The purpose of this manual is for the user to understand the hardware functions of the μPD17120 subseries.

Use

The manual assumes that the reader has a general knowledge of electricity, logic circuits, microcontrollers.

- **To understand the functions of the μPD17120 subseries in a general way;**
  → Read the manual from **CHAPTER 1.**

- **To look up instruction functions in detail when you know the mnemonic of an instruction;**
  → Use **APPENDIX D INSTRUCTION LIST.**

- **To look up an instruction when you do not know its mnemonic but know outlines of the function;**
  → Refer to **18.3 LIST OF THE INSTRUCTION SET** for search for the mnemonic of the instruction, then see **18.5 INSTRUCTIONS** for the function.

- **To look up electrical characteristics of the μPD17120 subseries;**
  → Refer to DATA SHEET.

Legend

Data representation weight	: High-order and low-order digits are indicated from left to right.
Active low representation	: $\overline{\text{xxx}}$ (pin or signal name is overlined)
Address of memory map	: Top: low, Bottom: high
Note	: Explanation of **Note** in the text
Caution	: Caution to which you should pay attention
Remark	: Supplementary explanation to the text
Number representation	: Binary number ... xxxx or xxxxB Decimal number ... xxxx or xxxxD Hexadecimal number ... xxxxH

Relevant Documents The following documents are provided for the μPD17120 subseries.
The numbers listed in the table are the document numbers.
Some related documents are preliminary versions. This document, however, is not indicated as "Preliminary".

Part Number / Document Name	μPD17120	μPD17121	μPD17132	μPD17133	μPD17P132	μPD17P133
Data sheet	IC-8407 [IC-2972]	IC-8399 [IC-2976]	IC-8412 [IC-2973]	IC-8411 [IC-2974]	ID-8419 [ID-2971]	ID-8426 [ID-2983]
User's manual	This manual [IEU-1367]					
IE-17K CLICE Ver.1.6 User's manual	EEU-929 [EEU-1467]					
IE-17K-ET CLICE-ET Ver.1.6 User's manual	EEU-931 [EEU-1466]					
SE board User's manual	EEU-847 [EEU-1412]					
SIMPLEHOST™ User's manual	EEU-723 [EEU-1336] (Introduction) EEU-724 [EEU-1337] (Reference)					
AS17K (Ver.1.11) User's manual	EEU-603 [EEU-1287]					
Device file User's manual	EEU-907 [EEU-1464]					

Remark The numbers inside [ ] indicate English document number.

The μPD17120 subseries has different pin names and signal names depending on the system clock type, as shown in the table below.

System Clock / Pin/Signal Names	RC Oscillation (μPD17120, μPD17132, μPD17P132)	Ceramic Oscillation (μPD17121, μPD17133, μPD17P133)
System Clock Oscillation Pin	OSC_1	X_{IN}
	OSC_0	X_{OUT}
System Clock Frequency	f_{CC}	f_X

Unless otherwise specified, this manual uses X_{IN}, X_{OUT}, and f_X for descriptions. When using the μPD17120, 17132, and 17P132, please change the readings to OSC_1, OSC_0 and f_{CC}.

TABLE OF CONTENTS

LIST OF FIGURES (1/2)

LIST OF FIGURES (2/2)

LIST OF TABLES (1/1)

[MEMO]

CHAPTER 1 GENERAL

The μPD17120, 17121, 17132 and 17133 are 4-bit single-chip microcontrollers employing the 17K architecture and containing 8-bit timer (1 channel), 3-wire serial interface, and power-on/power-down reset circuit.

The μPD17P132 and 17P133 are the one-time PROM version of the μPD17132 and 17133, respectively, and are suitable for program evaluation at system development and for small-scale production.

The following are features of the μPD17120 subseries.

- Comparator input (μPD17132, 17133, 17P132, 17P133 only)
  - Comparison function with external reference voltage (V_{ref})
  - Can be used as 4-bit A/D converter by using 15 types of internal reference voltage (1/16 to 15/16 V_{DD}) depending on the software
- 3-wire serial interface: 1 channel
- Power-on/power-down reset circuit (reducing external circuits)
- μPD17P132 and 17P133 can operate in the same way as mask ROM version
  - V_{DD} = 2.7 to 5.5 V

These features of the μPD17120 subseries are suitable for use as a controller or a sub-microcomputer device in the following application fields;

- Electric fan
- Hot plate
- Audio equipment
- Mouse
- Printer
- Plain paper copier

1.1 FUNCTION LIST

Item / Product Name	μPD17120	μPD17132	μPD17P132	μPD17121	μPD17133	μPD17P133
ROM Capacity	Masked ROM		One-time PROM	Masked ROM		One-time PROM
	1.5K bytes (768 × 16 bits)	2K bytes (1024 × 16 bits)		1.5K bytes (768 × 16 bits)	2K bytes (1024 × 16 bits)	
RAM Capacity	64 × 4 bits	111 × 4 bits		64 × 4 bits	111 × 4 bits	
Stack	5 address stacks; 1 interrupt stack					
Input/output port count	19 ports { • 18 input/output ports; • 1 sense input (INT pin[Note]) }					
Comparator (Supply voltage)	None	4-channel (V_{DD} = 2.7 to 5.5 V)		None	4-channel (V_{DD} = 2.7 to 5.5 V)	
Timer	1-channel (8-bit timer)					
Serial Interface	1-channel (3-wire)					
Interrupt	• 1 external interrupt (INT): { Detection of the rising edge; Detection of the trailing edge; Detection of both rising and trailing edges } Selectable • 2 internal interrupts { • Timer (TM); • Serial interface (SIO) }					
System clock	RC oscillation			Ceramic oscillation		
Instruction Execution Time	8 μs (when f_{CC} = 2 MHz)			2 μs (when f_X = 8 MHz)		
Standby Function	HALT, STOP					
Power-on/Power-down Reset Circuit	Incorporated (Can be used on an applied circuit of V_{DD}=5 V±10%)			Incorporated (Can be used on an applied circuit of V_{DD}=5 V±10%; f_X = 400 kHz to 4 MHz)		
Operating Supply Voltage	• 2.7 to 5.5 V • 4.5 to 5.5 V (When using the power-on power/down reset function)					
Package	• 24-pin plastic shrink DIP (300 mil) • 24-pin plastic SOP (375 mil)					
One-time PROM Product	μPD17P132		–	μPD17P133		–

Note When not using the external interrupt function, the INT pin can be used as an input-only pin (sense input). As a sense input, the pin status is read not by the port register but by the control register's INT flag.

Caution Despite a high level of functional compatibility with the masked ROM product, the PROM product is different in terms of the internal ROM circuit and some electric features. When switching from a PROM to a masked ROM product, be sure to sufficiently evaluate the application of the masked ROM product based on its sample.

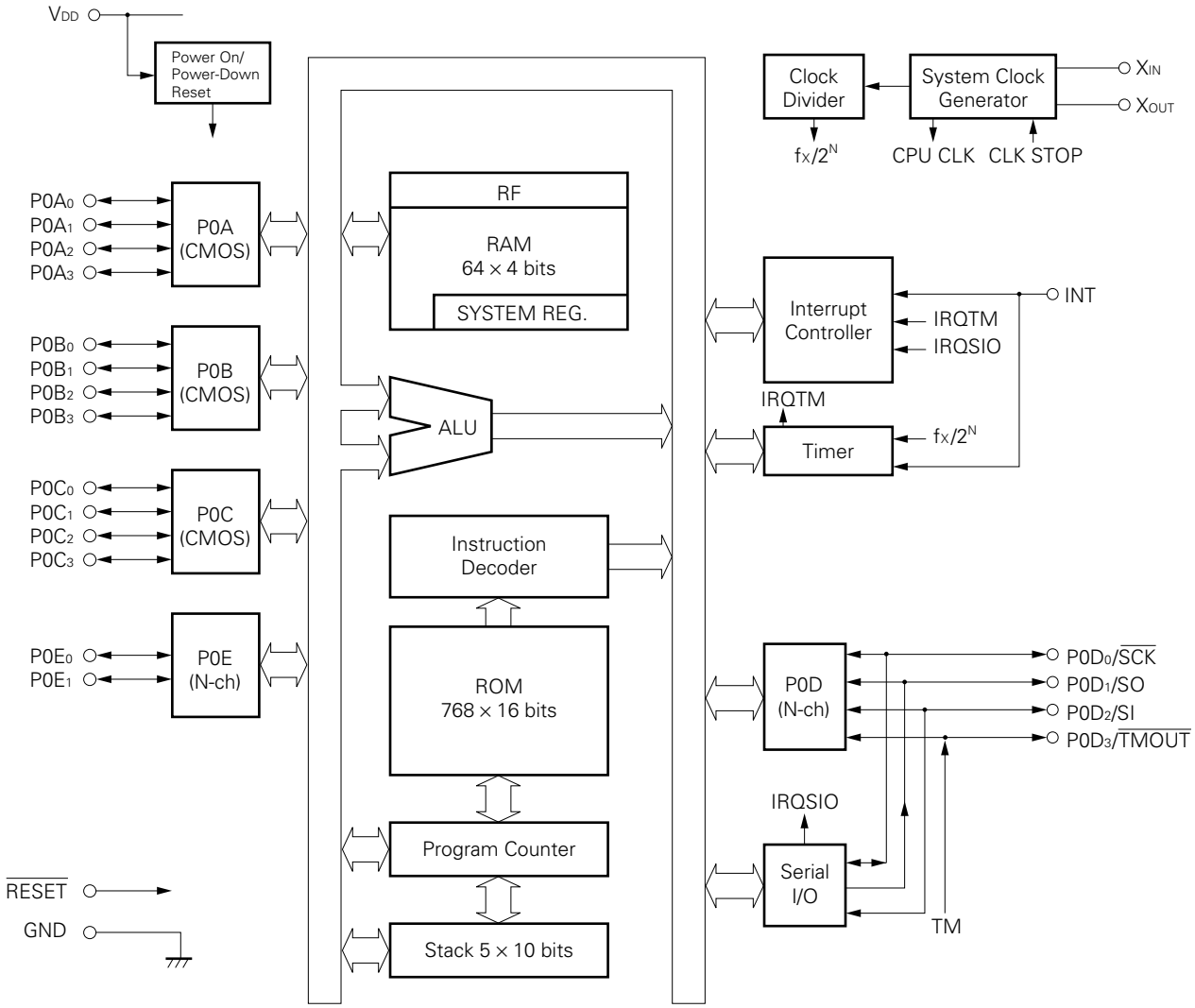
1.2 ORDERING INFORMATION

Part Number	Package	Internal ROM
μPD17120CS-xxx	24-pin plastic shrink DIP (300 mil)	Mask ROM
μPD17120GT-xxx	24-pin plastic SOP (375 mil)	Mask ROM
μPD17121CS-xxx	24-pin plastic shrink DIP (300 mil)	Mask ROM
μPD17121GT-xxx	24-pin plastic SOP (375 mil)	Mask ROM
μPD17132CS-xxx	24-pin plastic shrink DIP (300 mil)	Mask ROM
μPD17132GT-xxx	24-pin plastic SOP (375 mil)	Mask ROM
μPD17133CS-xxx	24-pin plastic shrink DIP (300 mil)	Mask ROM
μPD17133GT-xxx	24-pin plastic SOP (375 mil)	Mask ROM
μPD17P132CS	24-pin plastic shrink DIP (300 mil)	One-time PROM
μPD17P132GT	24-pin plastic SOP (375 mil)	One-time PROM
μPD17P133CS	24-pin plastic shrink DIP (300 mil)	One-time PROM
μPD17P133GT	24-pin plastic SOP (375 mil)	One-time PROM

Remark xxx: ROM code number

1.3 BLOCK DIAGRAM

- **Block diagram of the μPD17120 and 17121**

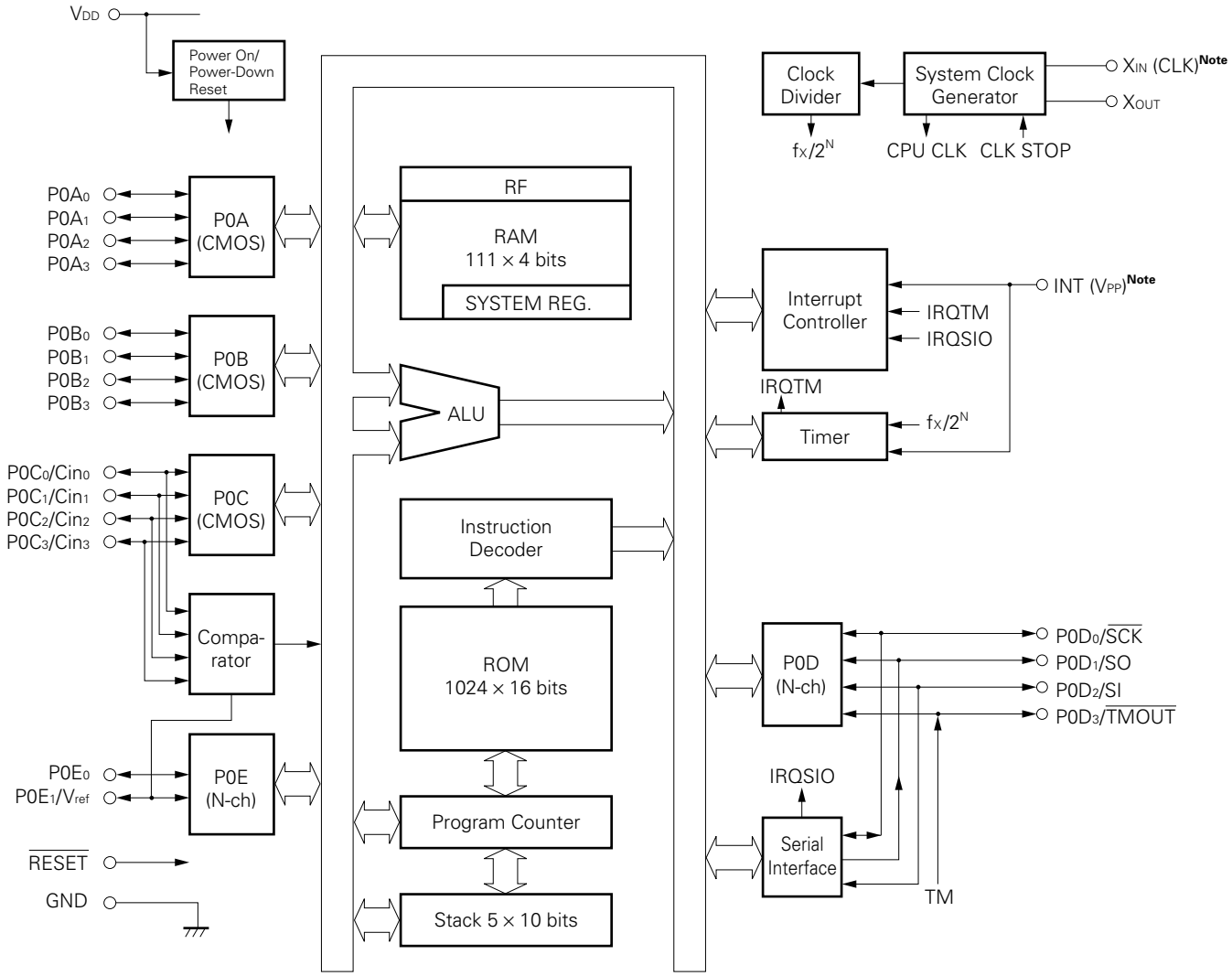


Remark The terms CMOS and N-ch in parentheses indicate the output form of the port.

CMOS: CMOS push-pull output

N-ch: N-channel open-drain output (Each pin can contain pull-up resistor as specified using a mask option.)

- **Block diagram of μPD17132, 17133, 17P132, and 17P133**



Remark The terms CMOS and N-ch in parentheses indicate the output form of the port.

CMOS: CMOS push-pull output

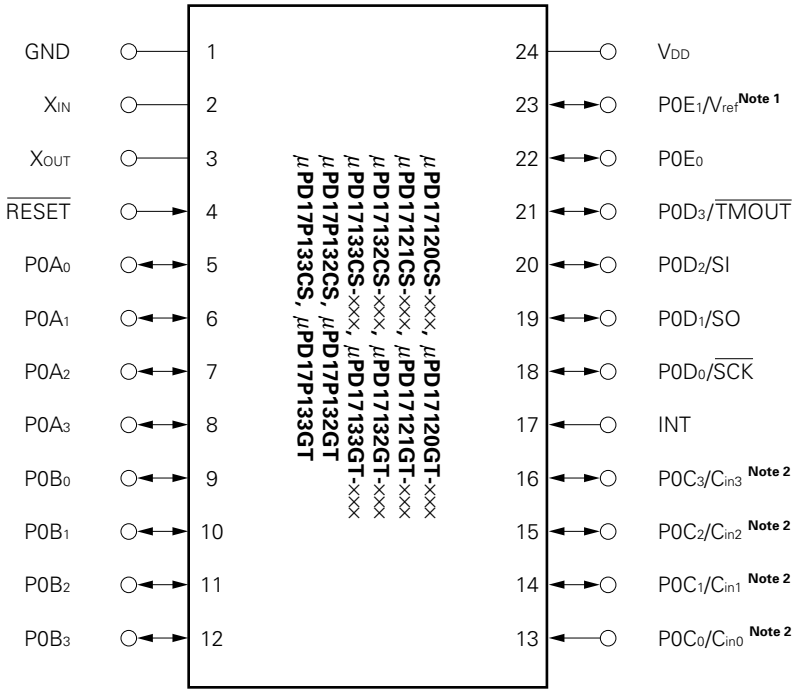
N-ch: N-channel open-drain output (Each pin can contain pull-up resistor as specified using a mask option.)

Note The devices in parentheses are effective only in the case of program memory write/verify mode of the μPD17P132 and μPD17P133.

1.4 PIN CONFIGURATION (Top View)

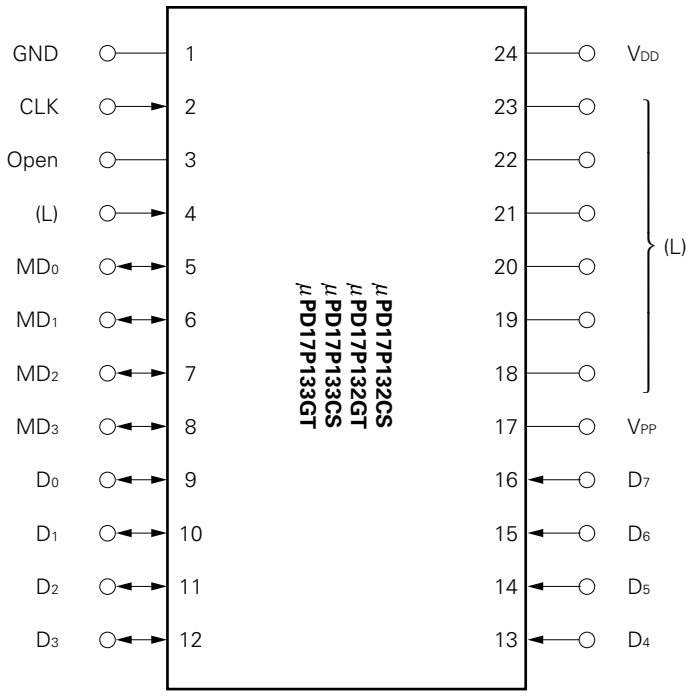
(1) Normal operating mode

24-pin plastic shrink DIP
24-pin plastic SOP



Notes **1.** There is no V_{ref} pin for the μPD17120 and 17121.
2. Pins C_{in0} to C_{in3} do not exist in the μPD17120 and 17121.

(2) Program memory write/verify mode



Caution ( ) represents processing of the pins which are not used in program memory write/verify mode.
L : Connect to GND via pull-down resistor one by one.
Open : This pin should not be connected.

(3) Pin name

Cin_0 to Cin_3	:	Comparator input
CLK	:	Clock input for address verification
D_0 to D_7	:	Data input/output
GND	:	Ground
INT	:	External interrupt input
MD_0 to MD_3	:	Operating mode selection
$P0A_0$ to $P0A_3$	:	Port 0A
$P0B_0$ to $P0B_3$	:	Port 0B
$P0C_0$ to $P0C_3$	:	Port 0C
$P0D_0$ to $P0D_3$	:	Port 0D
$P0E_0$ to $P0E_3$	:	Port 0E
$\overline{RESET}$	:	Reset input
$\overline{SCK}$	:	Serial clock input/output
SI	:	Serial data input
SO	:	Serial data output
$\overline{TMOUT}$	:	Timer output
V_{DD}	:	Power supply
V_{PP}	:	Programming voltage supply
V_{ref}	:	External reference voltage
X_{IN}, X_{OUT}	:	System clock oscillation

[MEMO]

CHAPTER 2 PIN FUNCTIONS

2.1 PIN FUNCTIONS

2.1.1 Pins in Normal Operation Mode

Pin No.	Symbol	Function	Output Format	At Power-on/Reset
1	GND	Grounded	–	–
2 3	X_{IN} X_{OUT}	**μPD17121, 17133, 17P133** • X_{IN}, X_{OUT} · Pins for system clock resonator oscillation · Connected to ceramic resonator	–	–
2 3	OSC_1 OSC_0	**μPD17120, 17132, 17P132** • OSC_0, OSC_1 · Pins for system clock oscillation · Resistor is connected between OSC_0 and OSC_1		
4	$\overline{RESET}$	System reset input Pull-up resistor can be incorporated by mask option**Note**	–	Input
5 I 8	$P0A_0$ I $P0A_3$	Port 0A · 4-bit I/O port · Input/output can be set by each bit	CMOS Push-pull	Input
9 I 12	$P0B_0$ I $P0B_3$	Port 0B · 4-bit I/O port · Input/output can be set by 4-bit unit	CMOS Push-pull	Input
13 I 16	$P0C_0/Cin_0$ I $P0C_3/Cin_3$	Port 0C and analog voltage input of comparator • $P0C_0$ to $P0C_3$ · 4-bit I/O port · Input/output can be set by each bit • Cin_0 to Cin_3 (μPD17132, 17133, 17P132, 17P133 only) · Analog input of comparator	CMOS Push-pull	Input (P0C)
17	INT	External interrupt request signal input and sense input	–	Input

Note The μPD17P132 and 17P133 have no pull-up resistor by mask option.

Pin No.	Symbol	Function	Output Format	At Power-on/Reset
18	$P0D_0/\overline{SCK}$	Port 0D, output of timer, serial data input, serial data output, serial clock input/output • $P0D_0$ to $P0D_3$ · 4-bit I/O port · Input/output can be set per bit · Pull-up resistor can be incorporated by each bit by mask option**Note** • $\overline{SCK}$ · Serial clock input/output	N-ch Open drain	Input (P0D)
19	$P0D_1$/SO	• SO · Serial data output		
20	$P0D_2$/SI	• SI · Serial data input		
21	$P0D_3/\overline{TMOUT}$	• $\overline{TMOUT}$ · Output of timer		
22 23	$P0E_0$ $P0E_1/V_{ref}$	Port 0E and reference voltage input of comparator • $P0E_0$, $P0E_1$ · 2-bit I/O port · Input/output can be set by each bit · Pull-up resistor can be incorporated per bit by mask option**Note** • V_{ref} (μPD17132,17133, 17P132, 17P133 only) · External reference voltage input of comparator	N-ch open drain	Input (P0E)
24	V_{DD}	Positive power supply	–	–

Note The μPD17P132 and 17P133 have no pull-up resistor by mask option.

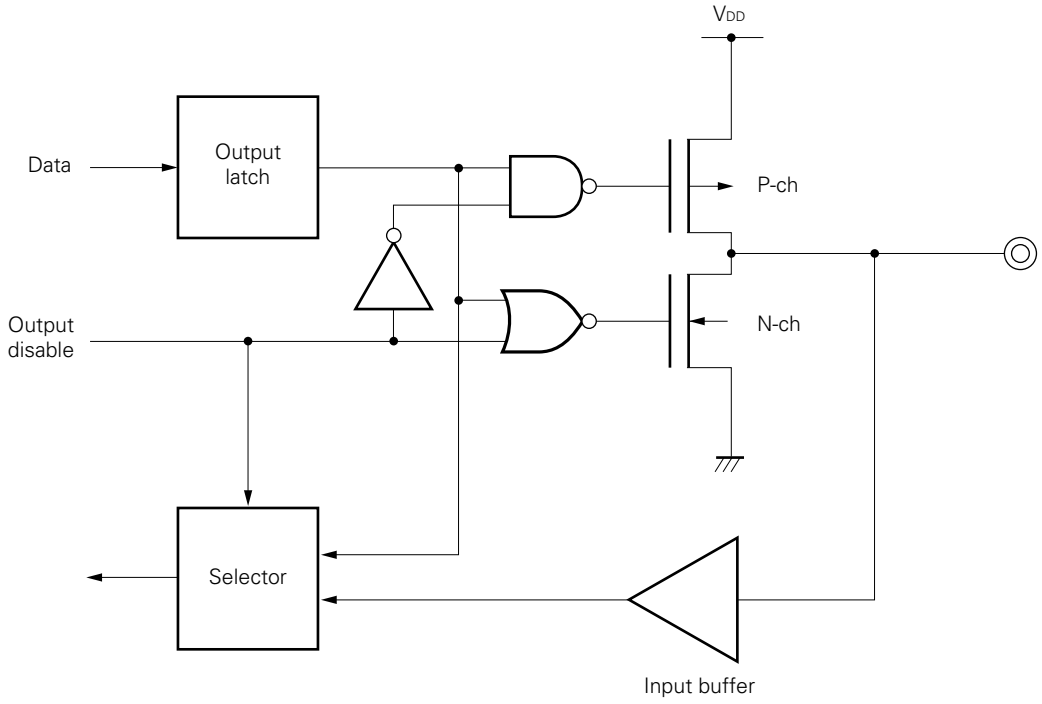
2.1.2 Pins in Program Memory Write/Verify Mode ... μPD17P132, 17P133 only

Pin No.	Symbol	Function	I/O
1	GND	Grounded	–
2	CLK	Clock input for address updating in program memory writing/verifying	Input
5 I 8	MD_0 I MD_3	Input for selecting operation mode in program memory writing/verifying	Input
9 I 12	D_0 I D_7	8-bit data input/output in program memory writing/verifying	Input/Output
17	V_{PP}	Pin for applying programming voltage in program memory writing/verifying Apply +12.5 V	–
24	V_{DD}	Positive power supply Apply +6 V in program memory writing/verifying.	–

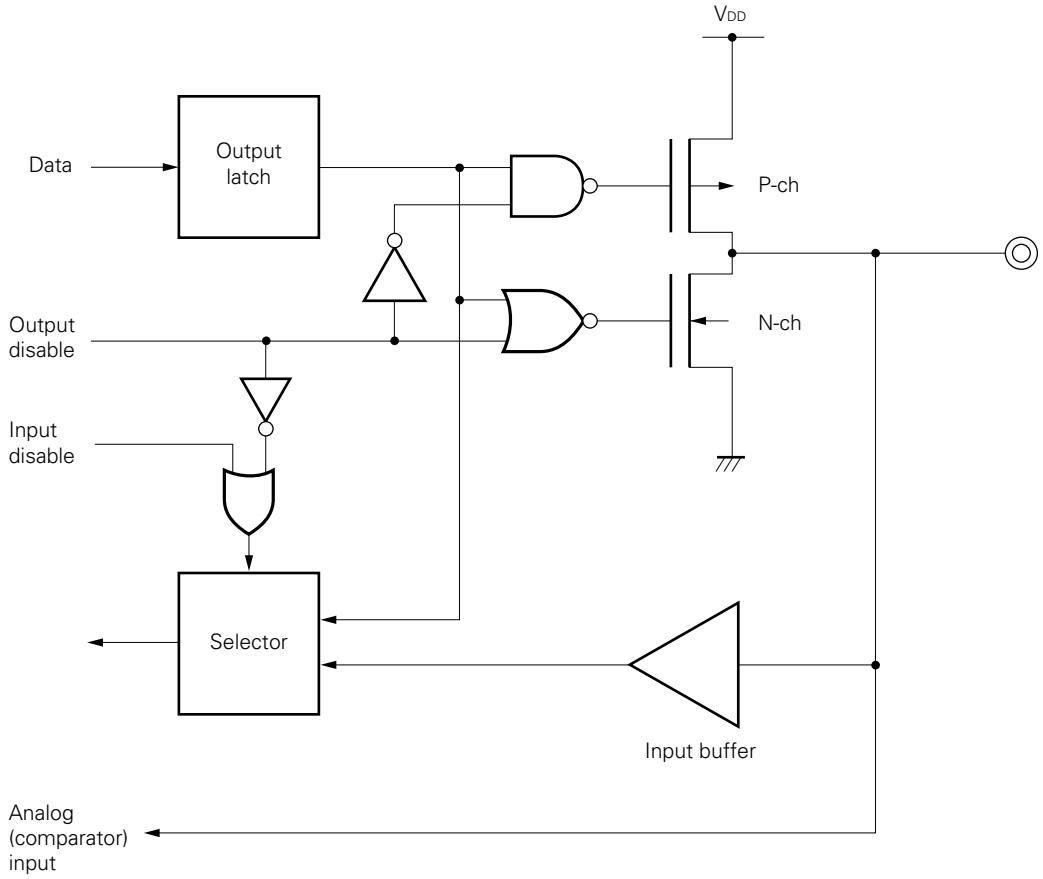
2.2 PIN INPUT/OUTPUT CIRCUIT

Below are simplified diagrams of the input/output circuits for each pin of the μPD17120 subseries.

(1) $P0A_0$-$P0A_3$, $P0B_0$-$P0B_3$

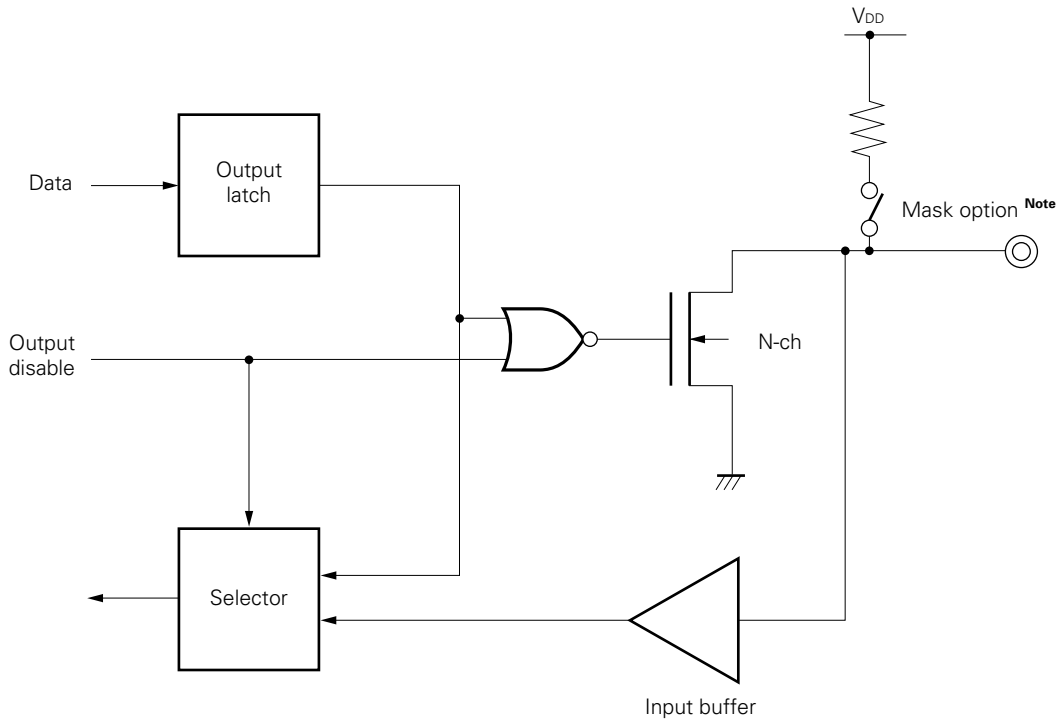


(2) $P0C_0/Cin_0$-$P0C_3/Cin_3$[Note]



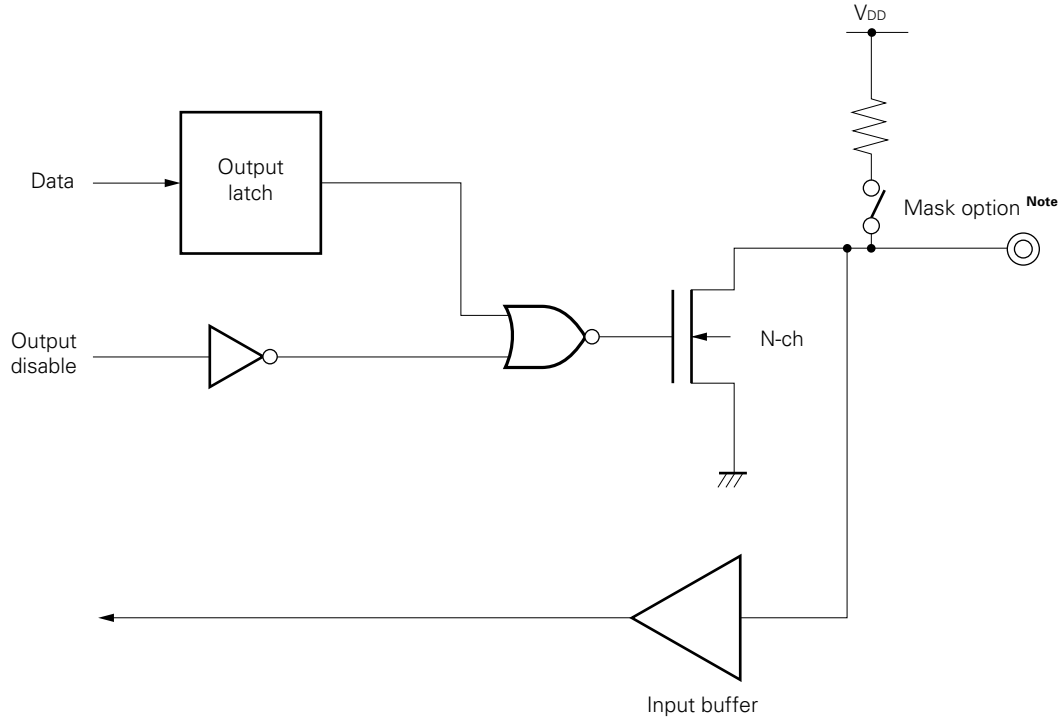
Note Pins Cin_0 to Cin_3 are not included in the μPD17120 and 17121.

(3) $P0D_0$-$P0D_3$



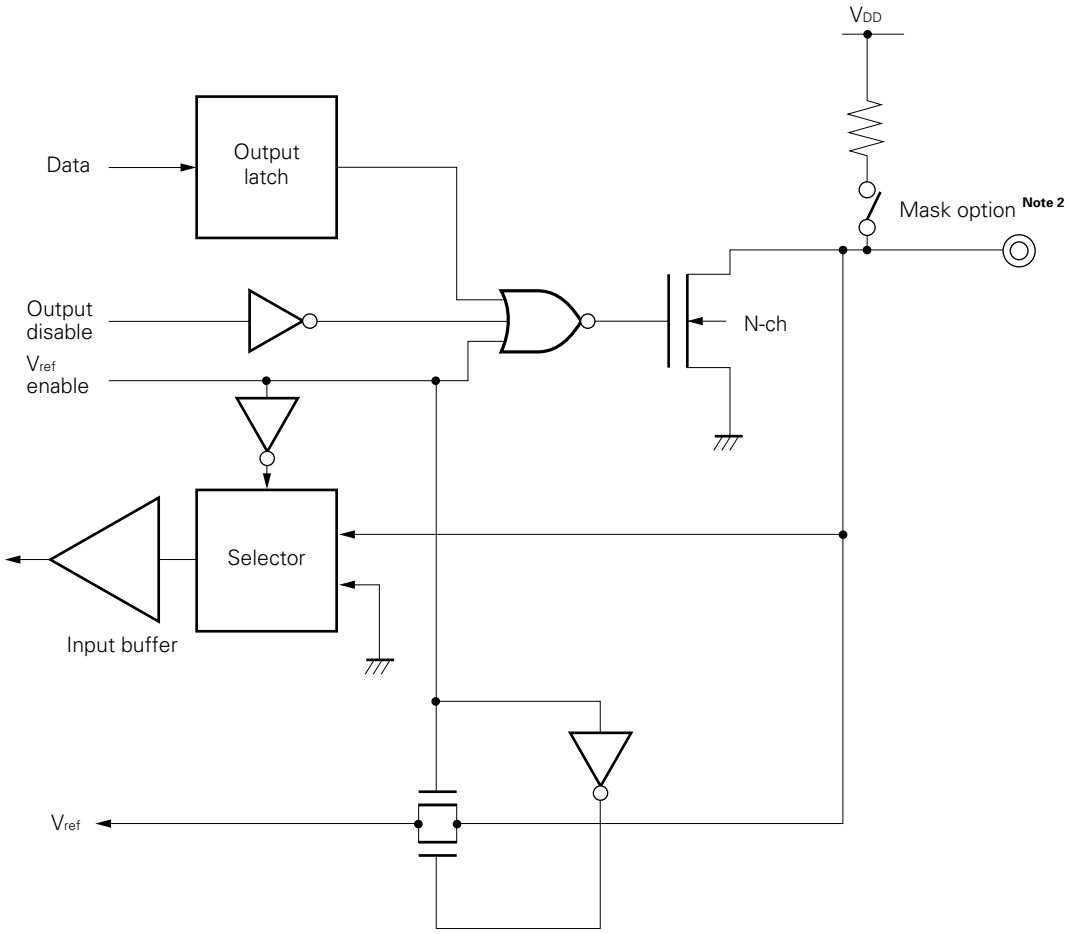
Note The μPD17P132 and 17P133 have no pull-up resistor by mask option, and are always open.

(4) $P0E_0$



Note The μPD17P132 and 17P133 have no pull-up resistor by mask option, and are always open.

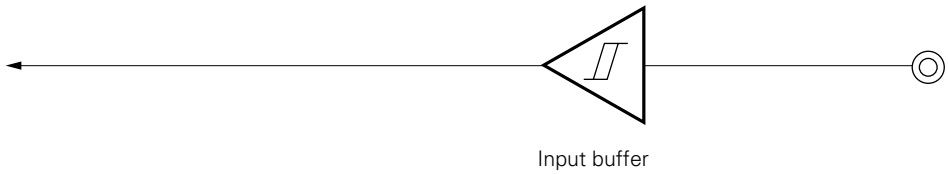
(5) $P0E_1/V_{ref}$ Note1



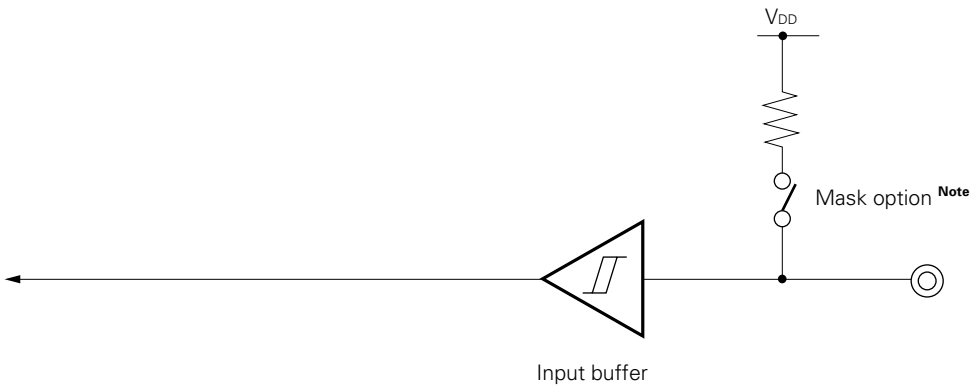
Notes **1.** The μPD17120 and 17121 have no V_{ref} pin function.

2. The μPD17P132 and 17P133 have no pull-up resistor by mask option, and are always open.

(6) INT



(7) $\overline{\text{RESET}}$



Note The μPD17P132 and 17P133 have no pull-up resistor by mask option, and are always open.

2.3 HANDLING UNUSED PINS

In normal operation mode, it is recommended to process the unused pins as follows:

Table 2-1. Handling Unused Pins

Pin Name			Recommended Measures	
			Inside Microcontroller	Outside Microcontroller
Port	Input mode	P0A, P0B, P0C	–	Each pin is connected to V_{DD} or GND through the resistor.**Note1**
		P0D, P0E	Does not incorporate a pull-up resistor by the mask option	
			Incorporates a pull-up resistor by the mask option.	Open
	Output mode	P0A, P0BP0C (CMOS port)	–	Open
		P0D and P0E (N-ch open drain port)	Outputs low level without incorporating pull-up resistor by the mask option.	
			Outputs high level with a pull-up resistor incorporated by the mask option.	
External Interrupt (INT)**Note2**			–	Directly connected to GND
$\overline{\text{RESET}}$**Note3** (when using only the built-in power-ON/power-DOWN reset)			Does not incorporate a pull-up resistor by the mask option	Directly connected to V_{DD}
			Incorporates a pull-up resistor by the mask option.	

Notes 1. When externally pulling up (connecting to V_{DD} through a resistor) or pulling down (connecting to the GND through a resistor), make sure to pay attention to the port's driving ability and current consumption. When pulling up or pulling down at a high resistance value, be careful to ensure that no noise is caused in the relevant pin. Although it depends on the applied circuit as well, it is usual to choose several tens of kΩ as the resistance value for pull-up or pull-down.

2. The INT pin is for the test mode setting function as well; connect it directly to the GND when unused.

3. If the applied circuit requires a high level of reliability, be sure to design it so that the $\overline{\text{RESET}}$ signal is input externally. Also, since the $\overline{\text{RESET}}$ pin is for the test mode setting function as well, connect it directly to the V_{DD} when unused.

Caution The output levels of the input/output mode and pins are recommended to be fixed by being set repeatedly in their respective loops in the program.

Remark The μPD17P132 and 17P133 do not contain pull-up resistors by the mask option.

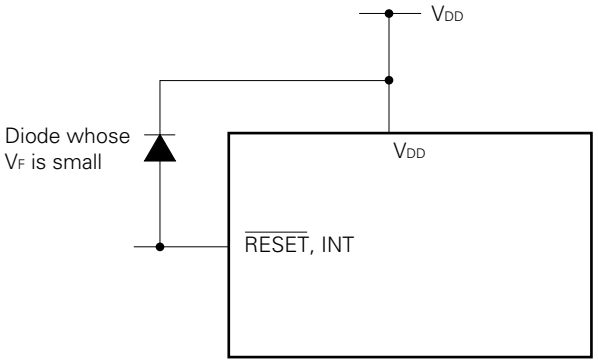
2.4 CAUTIONS ON USE OF THE $\overline{\text{RESET}}$ AND INT PINS (in Normal Operation Mode only)

In addition to the function described in **2.1 PIN FUNCTIONS**, the $\overline{\text{RESET}}$ pin and the INT pin have the function (for IC testing only) of setting test mode for testing the internal operation of the μPD17120 subseries.

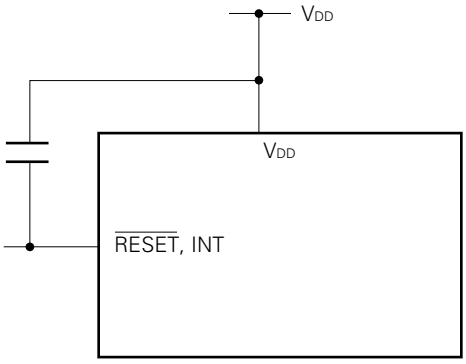
If a voltage exceeding the V_{DD} is applied to either of these pins, test mode is set. Therefore, adding a noise exceeding V_{DD} even in normal operation may result in placing the pin in test mode, thus impeding normal operation.

For example, if the $\overline{\text{RESET}}$ or INT pin wires are laid out too long, wiring noise is added to these pins, thus causing the above problem. Therefore, make sure that the wires are laid down in such a manner that such inter-wire noises are suppressed as much as possible. If noise is still a problem, take noise countermeasures based on external parts as shown in the illustrations below.

- **Connecting a Diode of Small V_F between V_{DD}**



- **Connecting a Capacitor between V_{DD}**



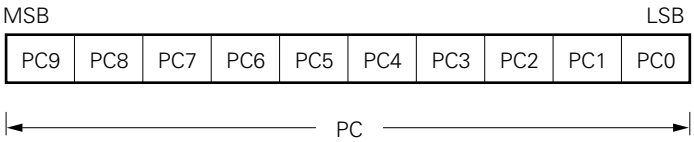
CHAPTER 3 PROGRAM COUNTER (PC)

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

3.1 PROGRAM COUNTER CONFIGURATION

Figure 3-1 shows the configuration of the program counter.
The program counters are 10-bit binary counters.
This program counter is incremented whenever an instruction is executed.

Figure 3-1. Program Counter



3.2 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 **3.2.1** to **3.2.7** explain program counter operating during execution of each instruction.

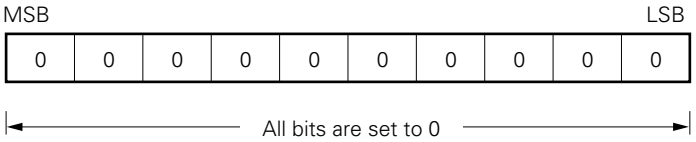
Figure 3-2. Value of the Program Counter after an Instruction Is Executed

Program Counter Bit / Instruction	Program Counter Value									
	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
During reset	0	0	0	0	0	0	0	0	0	0
BR addr CALL addr	Value set by addr									
BR @AR CALL @AR (MOVT DBF, @AR)	Value in the address register (AR)									
RET RETSK RETI	Value in the address stack register location pointed to the stack pointer (return address)									
During interrupt	Each interrupted vector address									

3.2.1 Program Counter at Reset

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

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

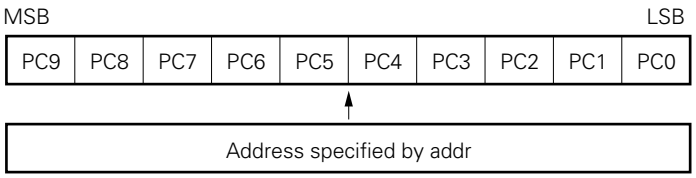


3.2.2 Program Counter during Execution of the Branch Instruction (BR)

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

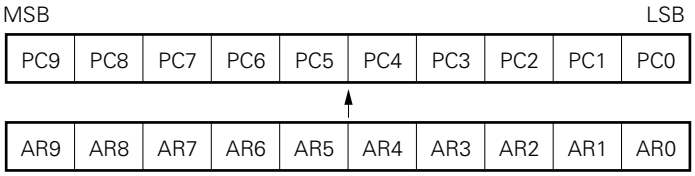
The address specified by a direct branch instruction is placed in the program counter.

Figure 3-4. Value in the Program Counter during Execution of a Direct Branch Instruction



An indirect branch instruction causes the address in the address counter to be placed in the program counter.

Figure 3-5. Value in the Program Counter during Execution of an Indirect Branch Instruction

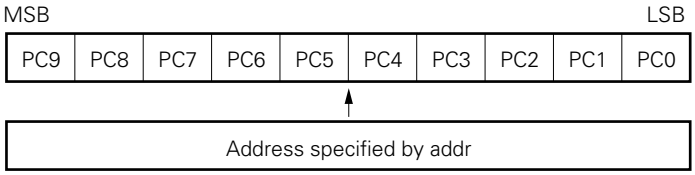


3.2.3 Program Counter during Execution of Subroutine Calls (CALL)

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

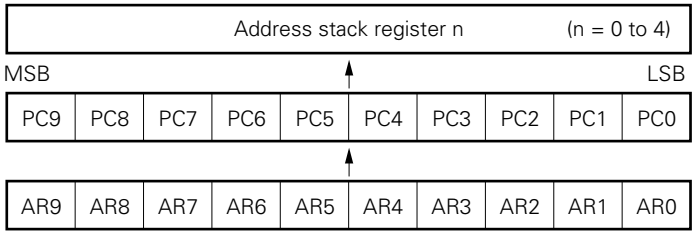
A direct subroutine call causes the value in the program counter to be saved in the stack and then the address specified in the operand to be placed in the program counter. Direct subroutine calls can specify any address in program memory.

Figure 3-6. Value in the Program Counter during Execution of a Direct Subroutine Call



An indirect subroutine call causes the value in the program counter to be saved in the stack and then the value in the address register to be placed in the program counter.

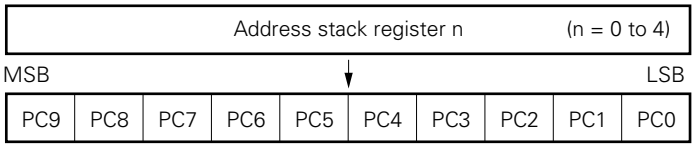
Figure 3-7. Value in the Program Counter during Execution of an Indirect Subroutine Call



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

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

Figure 3-8. Value in the Program Counter during Execution of a Return Instruction



3.2.5 Program Counter during Table Reference (MOVT)

During execution of table reference (MOVT DBF, @AR), 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.

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

When skip conditions are met and a skip instruction (SKE, SKGE, SKLT, SKNE, SKT, SKF) is executed, the instruction immediately following the skip instruction is treated as a no operation instruction (NOP). Therefore, whether skip conditions are met or not, the number of instructions executed and instruction execution time remain the same.

3.2.7 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.

3.3 CAUTIONS ON PROGRAM COUNTER OPERATION

Consisting of 10 bits, the μPD17120/17121's program counter (PC) can specify a program of up to 1024 steps. As opposed to this, the ROM size is only 768 steps (addresses 0000H-02FFH). If the program counter's value exceeds 300H, the contents of the program are equivalent to reading FFFFH and executing the "SKF PSW, #0FH" instruction. Therefore, be careful about the following point:

(1) When the instruction at the 768th step (address 02FFH) is executed, it does not automatically happen that the program counter goes to 0000H. If the instruction up to the 768th step (address 02FFH) is other than a branch (BR) or (RET) instruction, it will result in specifying a program counter not contained in a ROM. Be careful about this.

(2) In the same manner as (1), please avoid using an instruction that will branch to after the 768th step (address 02FFH).

CHAPTER 4 PROGRAM MEMORY (ROM)

The program configuration of the μPD17120 subseries is as follows.

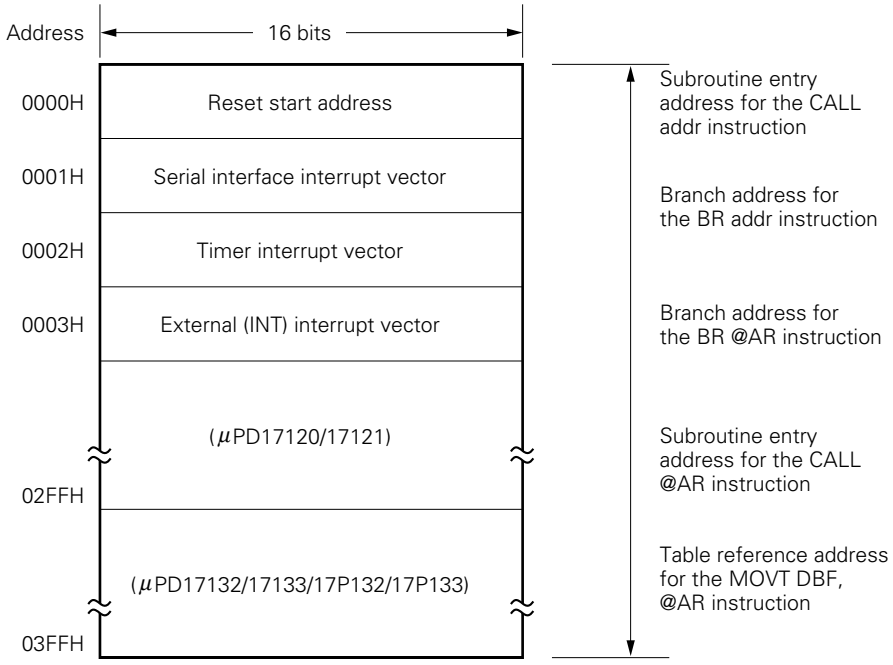
Product Name	Program Memory Capacity	Program Memory Address
μPD17120	1.5K bytes (768 × 16 bits)	0000H-02FFH
μPD17121		
μPD17132	2K bytes (1024 × 16 bits)	0000H-03FFH
μPD17133		
μPD17P132		
μPD17P133		

Program memory stores the program, and the constant data table. The top of the program memory is allocated to the reset start address and the interrupted vector address. The program memory address is specified by the program counter.

4.1 PROGRAM MEMORY CONFIGURATION

Figure 4-1 shows the program memory map. Branch instructions, subroutine calls, and table references can specify any address in program memory (0000H - 07FFH).

Figure 4-1. Program Memory Map for the μPD17120 Subseries



4.2 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 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.

4.2.1 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, by using the CALL instruction, this piece of code can be executed or read as many times as needed within the program. 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 completely 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 (5) explain branching of the program using the interrupt function and CPU instructions.

(1) Vector Address

Table 4-1 shows the address to which the program is branched (vector address) when a reset or interrupt occurs.

Table 4-1. Vector Address for the μPD17120 Subseries

Vector Address	Interrupt Sources
0000H	Reset
0001H	Serial interface interrupt
0002H	Timer interrupt
0003H	External interrupt (INT pin)

(2) Direct branch

When executing a direct branch (BR addr), the 11-bit instruction operand is used to specify an address in program memory. (However, the most significant bit must be 0. If an address is specified outside of this range, an error will occur in the assembler.) A direct branch instruction can be used to branch to any address in program memory.

(3) Indirect branch

When executing an indirect branch (BR @AR), the program branches to the address specified by the value stored in the address register (AR). An indirect branch can be used to branch to any address in program memory.

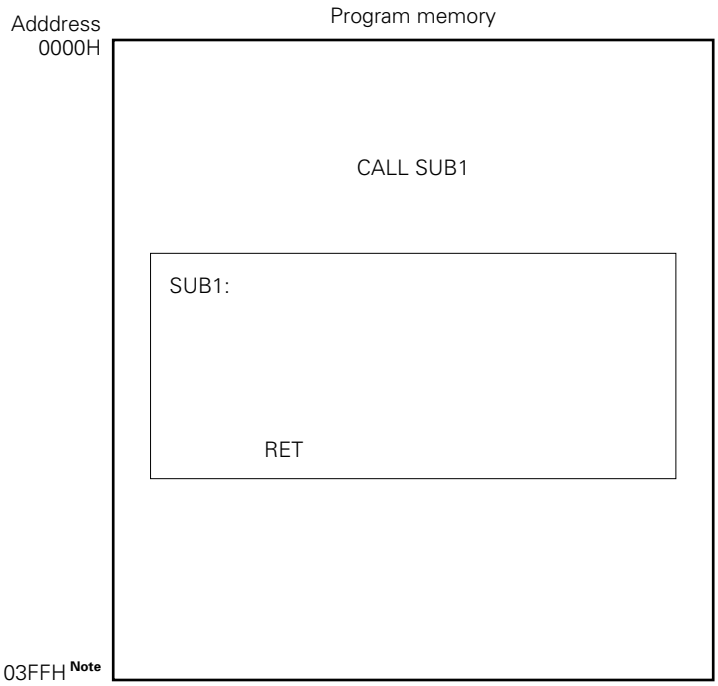
Also refer to **7.2 ADDRESS REGISTER (AR)**.

(4) Direct subroutine call

When using a direct subroutine call (CALL addr), the 11-bit instruction operand is used to specify a program memory address of the called subroutine. (However, the most significant bit must be 0. If an address is specified outside of this range, an error will occur in the assembler).

Example

Figure 4-2. Direct Subroutine Call (CALL addr)



Note The last address of the program memory of the μPD17120 and μPD17121 is 02FFH.

(5) 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 call any address in program memory.

Also refer to **7.2 ADDRESS REGISTER (AR)**.

4.2.2 Table Reference

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

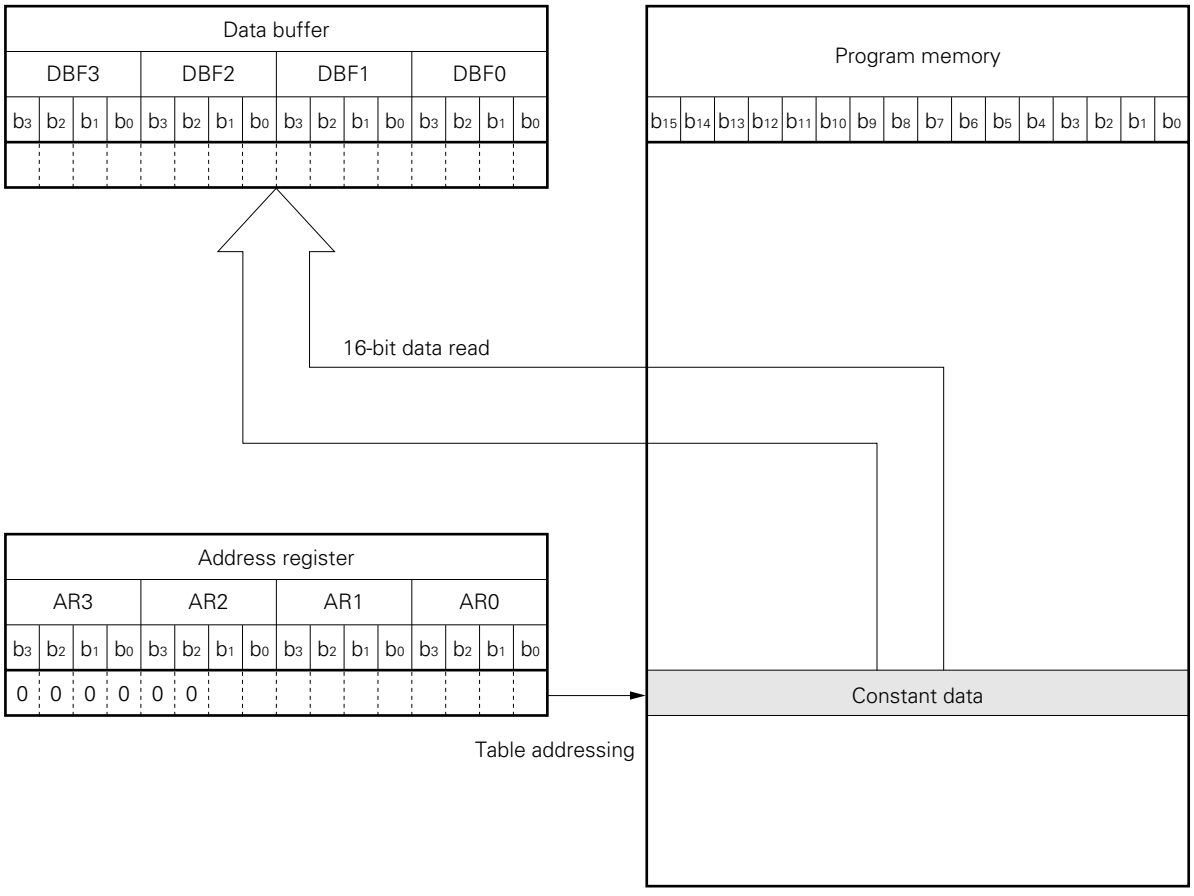
The table reference instruction (MOVT DBF, @AF) is used to store the contents of the program memory address specified by the address register in the data buffer.

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 Note that one level of the stack is temporarily used when performing table reference. Also refer to 7.2 ADDRESS REGISTER (AR) and CHAPTER 10 DATA BUFFER (DBF).

Remark As an exception, execution of table reference instructions requires two instruction cycle.

Figure 4-3. Table Reference (MOVT DBF, @AR)



(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 values recorded in a constant data table. The value specified by an OFFSET value is read.

```
OFFSET    MEM     0.00H           ; Storing area for the offset address.
          ; BANK0
          MOV     RPH,#0          ; Register pointer 7 is used to specify that
          MOV     RPL,#7 SHL 1    ; operation results be stored in row address 7.
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

          ADD     AR0, OFFSET     ; Adds the offset address.
          ADDC    AR1, #0
          ADDC    AR2, #0
          ADDC    AR3, #0
          MOVT    DBF, @AR        ; Executes the table reference instruction.

TABLE:
          DW      0001H
          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

          END
```

(2) Branch table

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

Example 2. Code used for reading the values recorded in a branch table. The value specified by an OFFSET value is read.

```
OFFSET      MEM     0.00H           ; Storing area for the offset address.

            ; BANK0
              MOV     RPH,#0        ; Sets the register pointer to row
              MOV     RPL,#7 SHL 1  ; address 7.
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

              ADD     AR0, OFFSET   ; Adds the offset address.
              ADDC    AR1, #0
              MOVT    DBF, @AR      ; Executes the table reference instruction.
              PUT     AR, DBF
              BR      @AR

TABLE:
            DW      0001H
            DW      0002H
            DW      0004H
            DW      0008H
            DW      0010H
            DW      0020H
            DW      0040H
            DW      0080H
            DW      0100H
            DW      0200H

            END
```

[MEMO]

CHAPTER 5 DATA MEMORY (RAM)

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

5.1 DATA MEMORY CONFIGURATION

Figure 5-1 shows the configuration of data memory.

Data memory is controlled by the concept called banks. The μPD17120 subseries has BANK0 only.

An address is allocated to the data memory for each bank.

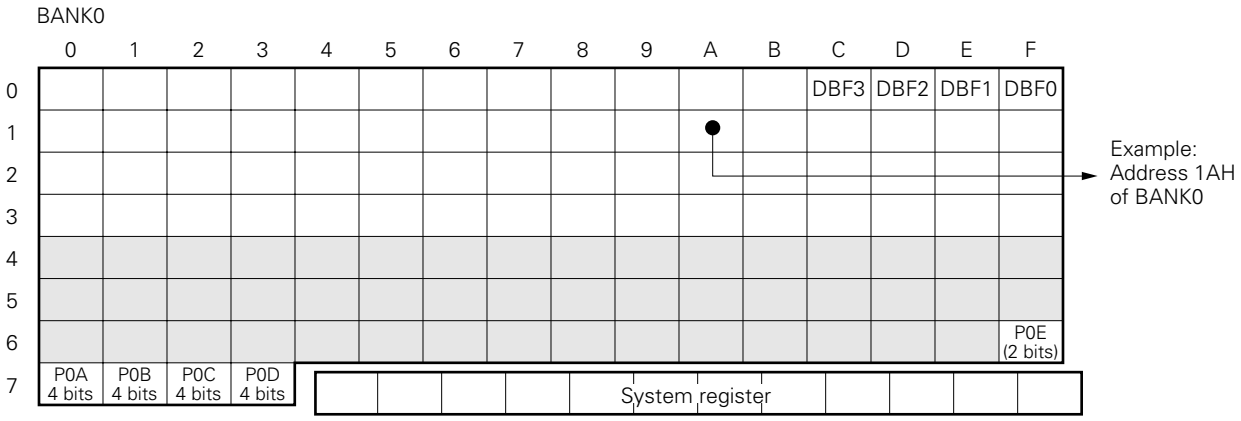
An address consists of four bits of memory called "a nibble".

The address of data memory consists of 7 bits. The three high-order bits are called "the row address", and the four low-order bits are called "the column address". For example, when the address of data memory is 1AH (0011010B), the row address is 1H (001B), and the column address is AH (1010B).

In the case of the μPD17120 and 17121, addresses 40H to 6EH should not be used because they are non-mounted areas.

Sections **5.1.1** to **5.1.6** describe functions of data memory other than its use as address space.

Figure 5-1. Configuration of Data Memory



Remark The shaded parts represent the non-mounted area in the case of the μPD17120 and 17121.

5.1.1 System Register (SYSREG)

The system register (SYSREG) consists of the 12 nibbles allocated at addresses 74H to 7FH in data memory. The system register (SYSREG) is allocated independently of the banks. This means that each bank has the same system register at addresses 74H to 7FH.

Figure 5-2 shows the configuration of the system register.

Figure 5-2. System Register Configuration

System Register (SYSREG)												
Address	74H	75H	76H	77H	78H	79H	7AH	7BH	7CH	7DH	7EH	7FH
Name (Symbol)	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)

5.1.2 Data Buffer (DBF)

The data buffer consists of four nibbles allocated at addresses 0CH to 0FH in BANK0 of data memory. Figure 5-3 shows the configuration of the data buffer.

Figure 5-3. Data Buffer Configuration

Data Buffer (DBF)				
Address	0CH	0DH	0EH	0FH
Symbol	DBF3	DBF2	DBF1	DBF0

5.1.3 General Register (GR)

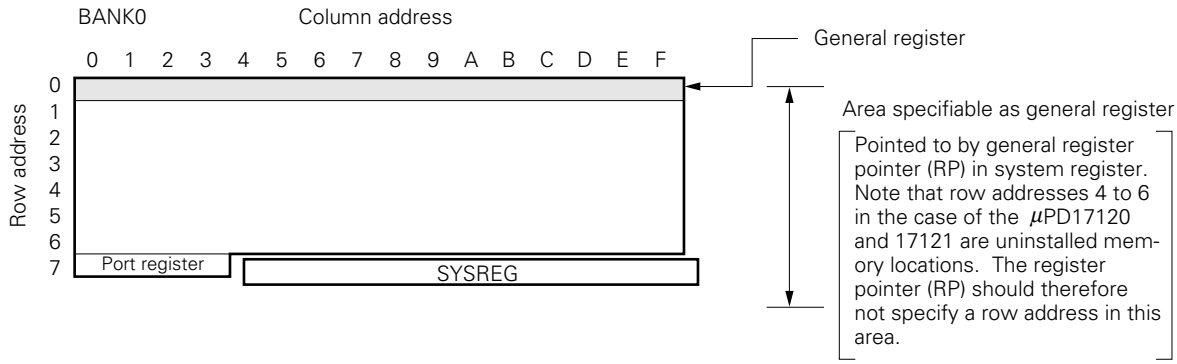
The general register consists of 16 nibbles specified by an arbitrary row address in a bank in data memory.

This arbitrary row address in a bank is pointed to by the register pointer (RP) in the system register (SYSREG).

In the case of the μPD17120 and 17121, addresses 40H to 6EH are non-mounted areas. These areas should not be specified as a general register.

Figure 5-4 shows the configuration of the general register (GR).

Figure 5-4. General Register (GR) Configuration



5.1.4 Port Registers

A port register consists of five nibbles allocated at addresses 6FH to 73H in Bank0 of the data memory. As shown in Figure 5-5, the two high-order bits of address 6FH are always set to 0.

Figure 5-5 shows the configuration of the port registers.

Figure 5-5. Port Register Configuration

Port Register																					
Address		6FH				70H				71H				72H				73H			
Symbol	BANK0	P0E				P0A				P0B				P0C				P0D			
		0	0	P0E1	P0E0	P0A3	P0A2	P0A1	P0A0	P0B3	P0B2	P0B1	P0B0	P0C3	P0C2	P0C1	P0C0	P0D3	P0D2	P0D1	P0D0

5.1.5 General Data Memory

General data memory is all the data memory not used by the port and system registers (SYSREG). In other words, general data memory consists of 64 nibbles (μPD17120 and 17121) or 111 nibbles (μPD17132, 17133, 17P132, and 17P133).

5.1.6 Uninstalled Data Memory

There is no hardware installed at addresses 40H to 6EH of the μPD17120 and 17121. Any attempt to read this area will yield unpredictable results. Writing data to this area is invalid and should therefore not be attempted.

[MEMO]

CHAPTER 6 STACK

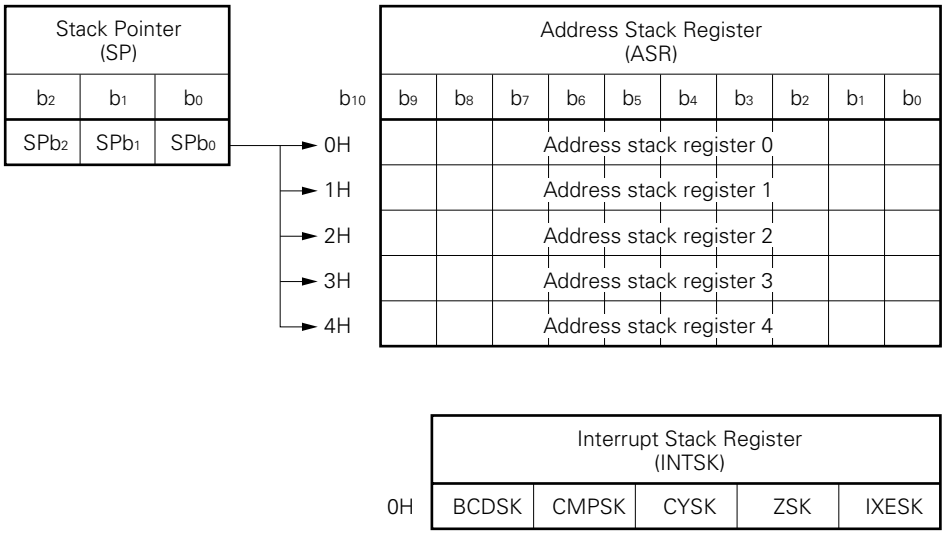
The stack is a register used to save information such as the program return address and the contents of the system register during execution of subroutine calls, interrupts and similar operations.

6.1 STACK CONFIGURATION

Figure 6-1 shows the stack configuration.

The stack consists of the following parts: one 3-bit binary counter stack pointer, five 10-bit address stack registers, and one 5-bit interrupt stack registers.

Figure 6-1. Stack Configuration



6.2 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.

Remark All the 5 bits of PSWORD are automatically cleared to zero after being saved in the interrupt stack register.

6.3 ADDRESS STACK REGISTER

As shown in Figure 6-1, the address stack register consists of five consecutive 10-bit registers.

A value equal to the program counter (PC)+1 (return address) is stored during execution of subroutine calls (CALL addr, CALL @AR), the first cycle of a table reference (MOVT DBF, @AR), and upon receipt of an interrupt in the address stack register. The contents of the address register (AR) is also stored when a stack push (PUSH AR) is executed. The address register holding data is pointed to by the address in the stack pointer at execution time less one (address in stack pointer (SP) – 1).

When a subroutine return (RET, RETSK), an interrupt return (RETI), or the second cycle of a table reference (MOVT DBF, @AR) is executed, the contents of the address pointed to by the stack pointer is restored to the program counter and the stack pointer is incremented. When a stack pop (POP AR) is executed, the value in the address stack register pointed to by the stack pointer is transferred to the address to the address register and the stack pointer is incremented.

If more than five subroutine calls or interrupts are executed, **an internal reset signal is generated, and the address stack register initializes hardware for start at address 0000H** (to prevent a software crash).

6.4 INTERRUPT STACK REGISTER

As shown in Figure 6-1, the interrupt stack register consists of one 5-bit register.

When an interrupt is received five bits in the system register (SYSREG) (mentioned later) that is, each flag (BCD, CMP, CY, Z, IXE) of the program status word (PSWORD), are saved. When the interrupt return (RETI) is executed, the program status word is restored from the interrupt stack register.

In the interrupt stack register, every time an interrupt is received, necessary data is saved.

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

Remark All the 5 bits of PSWORD are automatically cleared to zero after being saved in the interrupt stack register.

6.5 STACK POINTER (SP) AND INTERRUPT STACK REGISTER

As shown in Figure 6-1, the stack pointer (SP) is a 3-bit binary counter used to point to addresses in the five address stack registers. The stack pointer is located at address 01H in the register file. At reset, the stack pointer is set to 5.

As shown in Table 6-1, the stack pointer is decremented when subroutine calls (CALL addr, CALL @AR), the first cycle of a table reference (MOVT DBF, @AR), stack push (PUSH AR), and an interrupt are accepted. The stack pointer is incremented at the following times: subroutine returns (RET, RETSK), the second instruction cycle of a table reference (MOVT DBF, @AR), stack pop (POP AR), and an interrupt return (RETI). The interrupt stack counter as well as the stack pointer is decremented when an interrupt is accepted. The interrupt stack counter is incremented by an interrupt return (RETI) only.

Table 6-1. Operation of the Stack Pointer

Instruction	Stack Pointer Value	Counter of Interrupt Stack Register
CALL addr CALL @AR MOVT DBF, @AR (1st instruction cycle) PUSH AR	–1	Not changed
Interrupt receipt	–1	–1
RET RETSK MOVT DBF, @AR (2nd instruction cycle) POP AR	+1	Not changed
RETI	+1	+1

As mentioned above, the stack pointer 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 value that is greater than five **will cause an internal reset signal to be generated** (to prevent a software crash).

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

6.6 STACK OPERATION DURING SUBROUTINES, TABLE REFERENCES, AND INTERRUPTS

Stack operation during execution of each command is explained in **6.6.1** to **6.6.3**.

6.6.1 Stack Operation during Subroutine Calls (CALL) and Returns (RET, RETSK)

Table 6-2 shows operation of the stack pointer (SP), address stack register, and the program counter (PC) during execution of subroutine calls and returns.

Table 6-2. Operation of the Stack Pointer during Execution

Instruction	Operation
CALL addr	<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) 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 no operation instruction (NOP).

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

Table 6-3 shows stack operation during table reference.

Table 6-3. Stack Operation during Table Reference

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	<1> Contents of the program memory (ROM) pointed to by the program counter (PC) is transferred to the data buffer (DBF). <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.

Remark As an exception, execution of MOVT DBF and @AR instructions require two instruction cycle.

6.6.3 Executing RETI Instruction

Table 6-4 shows stack operation during interrupt receipt and RETI instruction execution.

Table 6-4. Stack Operation during Interrupt Receipt and Return

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. <4> Vector address is transferred to the program counter (PC)
RETI	<1> Values in the interrupt stack register are restored to the PSWORD (BCD, CMP, CY Z, IXE). <2> Values 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.

6.7 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 is 0H and a CALL or MOVT instruction is executed or an interrupt is received, the stack pointer is decremented to 7H. The μPD17120 subseries 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, the PUSH and POP instructions are used as necessary to save/return the address stack register.

Table 6-5 shows stack operation during the PUSH and POP instructions.

Table 6-5. Stack Operation during 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 7 SYSTEM REGISTER (SYSREG)

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

7.1 SYSTEM REGISTER CONFIGURATION

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

Because 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 7-1. Allocation of System Register in Data Memory

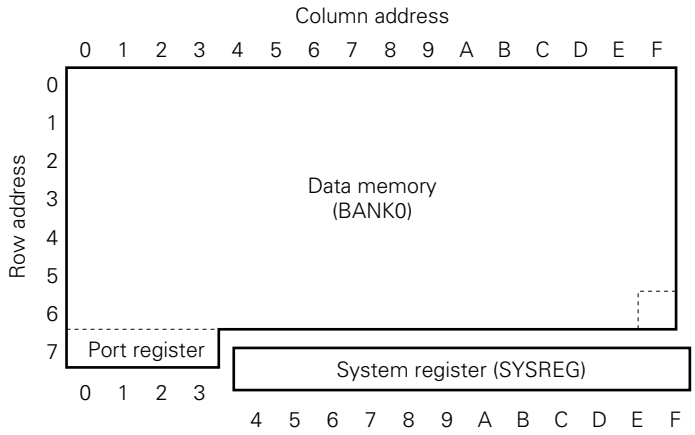


Figure 7-2 shows the configuration of the system register. As shown in Figure 7-2, the system register consists of the following seven registers.

- 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 7-2. 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]	0 0 0 0 (AR)	0 0 0 (AR)	(AR)	(AR)	← →	0 0 0 0 (BANK)	MPE 0 0 0 (MP)	(IX) (MP)	(IX)	0 0 0 0 (RP)	(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 0

Note Zeros in the columns indicates "0 fixed".

7.2 ADDRESS REGISTER (AR)

7.2.1 Address Register Configuration

Figure 7-3 shows the configuration of the address register.

As shown in Figure 7-3, the address register consists of the sixteen bits in address 74H to 77H (AR3 to AR0) of the system register. However, because the six high-order bits are always set to 0, the address register is actually 10 bits. When the system is reset, all sixteen bits of the address register are reset to 0.

Figure 7-3. 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	0	0										
Initial value when reset	0				0				0				0			

7.2.2 Address Register Functions

The address register is used to specify an address in program memory when executing an indirect branch instruction (BR @AR), indirect subroutine call (CALL @AR) or table reference (MOVT DBF, @AR). The address register can also be put on and taken off the stack by using the stack manipulation instructions (PUSH AR, POP AR).

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) Table reference (MOVT DBF, @AR)

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

(2) 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 pointed to by the stack pointer.

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

Also refer to **CHAPTER 6**.

(3) Indirect branch instruction (BR @AR)

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

(4) Indirect subroutine call (CALL @AR)

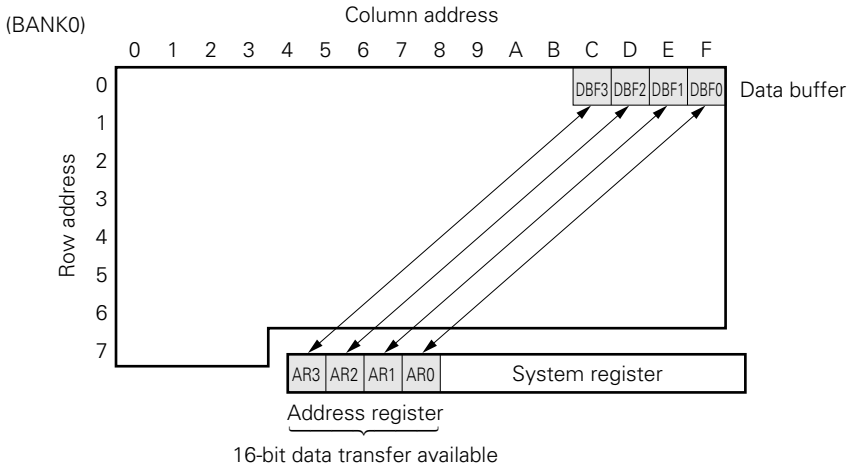
When the CALL @AR instruction is executed, the subroutine located at the address in program memory specified by the value in the address register is called.

(5) Address register used as peripheral register

The address register can be manipulated four bits at a time by using data memory manipulation instructions. The address register can also be used as a peripheral register for transferring 16-bit data to the data buffer. In other words, by using the PUT AR, DBF and GET DBF, AR instructions in addition to the data memory manipulation 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 7-4. Address Register Used as a Peripheral Register



7.3 WINDOW REGISTER (WR)

7.3.1 Window Register Configuration

Figure 7-5 shows the configuration of the window register.

As shown in Figure 7-5, the window register (WR) 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 $\overline{\text{RESET}}$ input is used to release the system from HALT or STOP mode, the previous state of the window register is maintained.

Figure 7-5. 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			

7.3.2 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 the instructions PEEK WR, rf and POKE rf, WR. For details, refer to **9.2.3 Register File Manipulation Instructions**.

7.4 BANK REGISTER (BANK)

Figure 7-6 shows the configuration of the bank register.

The bank register consists of four bits at address 79H (BANK) of the system register.

Bank register is a register for switching the banks of RAM. However, since the μPD17120 subseries has only one bank, every bank register bit is fixed to 0.

Figure 7-6. Bank Register Configuration

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

7.5 INDEX REGISTER (IX) AND DATA MEMORY ROW ADDRESS POINTER (Memory Pointer: MP)

7.5.1 Index Register (IX)

IX is used for address modification to data memory. It differs from MP in that its modification object is an address that is specified as the bank or operand m.

As shown in Figure 7-7, IX is mapped to a total of 12 bits of system registers: 7AH (IXH), 7BH (IXM), and 7CH (IXL). The index register enable flag (IXE) which enables address modification by IX is allocated to the lowest bit of the PSW.

When IXE=1, an address in data memory specified with operand m is not m but the address indicated by the OR of m, IXM and IXL. The bank specified at this time is that indicated by the OR of BANK and IXH.

Remark The IXH of the μPD17120 subseries is "fixed to 0" and therefore the bank is modified even when IXE=1 (thus preventing the bank from becoming other than 0).

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

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

As shown in Figure 7-7, MPH and IXH, and MPL and IXM, are respectively mapped to the same addresses (system registers 7AH and 7BH). It is MPH's lower 3 bits and MPL's full 7 bits that are actually functioning as the MP. To MPH's most significant bit is allocated the memory pointer enable flag (MPE) which enables address modification by the MP.

When MPE=1, the bank and row address of the data memory indirectly specified with operand @r is not BANK and m_R but the address specified by the MP. (The column address is specified with the contents of r regardless of the MPE.) At this time, MPH's lower 3 bits and MPL's most significant 4 bits point to BANK; and MPL's lower 3 bits point to the row address.

Remark The MPH's lower 3 bits and MPL's most significant bit in the μPD17120 subseries are "fixed to 0" and therefore the bank is cleared to 0 even when MPE=1 (thus preventing the bank from becoming other than 0).

Figure 7-7. Index Register and Memory Pointer Configuration

Address	7AH				7BH				7CH				7FH			
Name	Index register (IX) Memory pointer (MP)												Program status word (PSWORD)'S lower 4 bits			
Symbol name	IXH MPH				IXM MPL				IXL				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_1	b_3	b_2	b_1	b_0
Flag name	MPE															IXE
Data		(IX) (MP)														
		0	0	0	0											
Reset-time value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7-8. Data Memory Address Modification by Index Register and Memory Pointer

IXE	MPE	Data Memory Address Specified with m												Indirect Transfer Address Specified with @m											
		Bank				Row address				Column address				Bank				Row address				Column address			
		b_3	b_2	b_1	b_0	b_2	b_1	b_0		b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_2	b_1	b_0		b_3	b_2	b_1	b_0
0	0	BANK								m				BANK				m_R				(r)			
0	1					Same as above								MPH				MPL				(r)			
1	0	BANK				Logical OR				m				BANK				Logical OR				m_R			
		IXH				IXM				IXL				IXH				IXM				(r)			
1	1	Setting disabled																							

BANK : Bank register
IX : Index register
IXE : Index enable flag
IXH : Index register's bits 10-8
IXM : Index register's bits 7-4
IXL : Index register's bits 3-0
m : Data memory address indicated by m_R, m_C
m_R : Data memory row address
m_C : Data memory column address

MP : Memory pointer
MPE : Memory pointer enable flag
MPH : Memory pointer's upper 3 bits
MPL : Memory pointer's lower 4 bits
r : General register column address
RP : General register pointer
(×) : Contents addressed with ×
× : Direct address such as r

Table 7-1. Address-modified Instruction Statements

Arithmetic operation	ADD ADDC	r, m
	SUB SUBC	m, #n4
Logical operation	AND OR XOR	r, m
		m, #n4
Judge-ment	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

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

As shown in Figure 7-8, data memory addresses are not affected by the index register and the data memory row address pointer.

(1) Data memory manipulation instructions

Example 1. General register is in row address 0

```
R003    MEM     0.03H
M061    MEM     0.61H
        ADD     R003, M061
```

As shown in Figure 7-9, when the above instructions are executed, the data in general register address R003 and data memory address M061 are added together and the result is stored in general address R003.

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

Example 2. General register is in row address 0

```
R005    MEM     0.05H
M034    MEM     0.34H
        MOV     R005, #8          ; R005 ← 8
        MOV     @R005, M034       ; Indirect transfer of data in the register
```

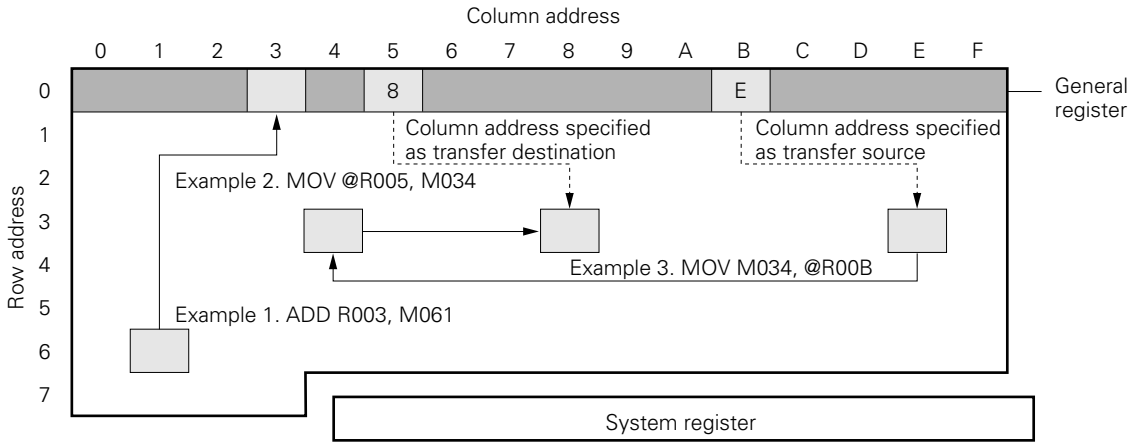
As shown in Figure 7-9, when the above instructions are executed, the data stored in data memory address M034 is transferred to data memory location 38H.
In other words, the MOV @r, m instruction causes the contents in the data memory address specified by m to be transferred to the data memory location specified by @r (which by definition has the same row address as m).
The indirect data transfer address has the same row address as m (example above uses row address 3) and the column address is the value contained in the general register address specified by r (example above uses column address 8). Therefore the address in the above example is 38H.

Example 3. General register is in row address 0

```
R00B    MEM     0.0BH
M034    MEM     0.34H
        MOV     R00B, #0EH      ; R00B ← 0EH
        MOV     M034, @R00B     ; Indirect transfer of data in the register
```

As shown in Figure 7-9, when the above instructions are executed, the contents of data memory stored at address 3EH is transferred to data memory location M034.
In other words, the MOV m, @r instruction causes the contents of the data memory location specified by @r (which by definition has the same row address as m) to be transferred to the data memory location specified by m.
The indirect data transfer address has the same row address as m (example above uses row address is 3) and the column address is the value contained in the general register address specified by r (example above uses column address 0EH). Therefore the address in the above example is 3EH.
The data transfer memory address source and destination in this example are the opposite of those shown in **Example 2** (source and destination are switched).

Figure 7-9. Example of Operation When MPE=0 and IXE=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

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

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

As shown in Figure 7-8, 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. When the general register is in row address 0

```
R005    MEM     0.05H
M034    MEM     0.34H
        MOV     MPL, #0110B     ; MP ← 6
        MOV     R005, #8        ; R005 ← 8
        MOV     MPH, #1000B     ; MPE ← 1
        MOV     @R005, M034     ; Indirect transfer of data in the register
```

As shown in Figure 7-10, when the above instructions are 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 **7.5.3** 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 **7.5.3** the indirect address bank and row address are the same as m.)

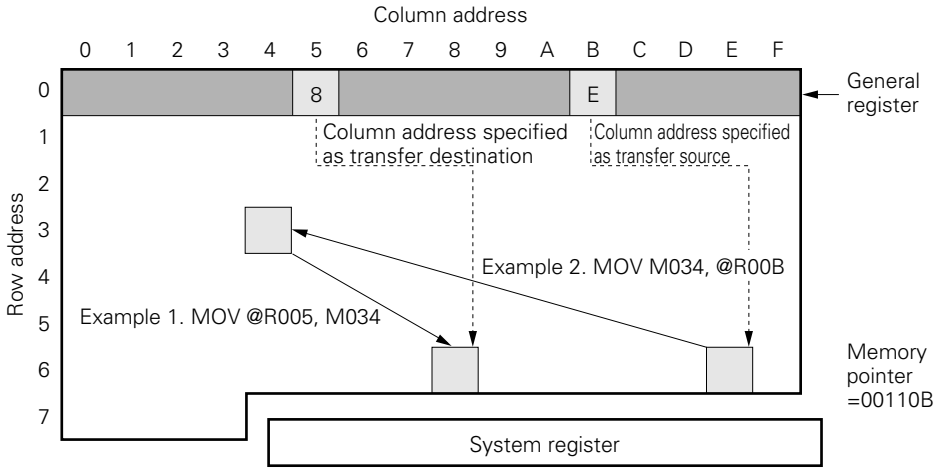
By setting MPE=1, diagonal indirect data transfer can be performed using the general register.

Example 2. General register is in row address 0

```
R00B    MEM     0.0BH
M034    MEM     0.34H
        MOV     MPL, #0110B     ; MP ← 6
        MOV     MPH, #1000B     ; MPE ← 1
        MOV     R00B, #0EH      ; R00B ← 0EH
        MOV     M034, @R00B     ; Indirect transfer of data in the register
```

As shown in Figure 7-10, when the above instructions are executed, the data stored in address 6EH is transferred to data memory location M034.

Figure 7-10. Example of Operation When MPE=1 and IXE=0



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

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

As shown in Figure 7-8, 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. When the general register is in row address 0

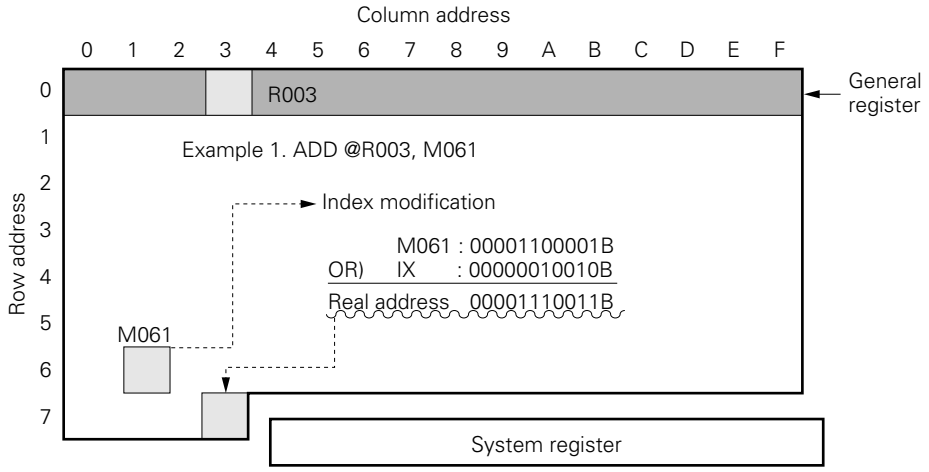
```
R003    MEM     0.03H
M061    MEM     0.61H
        MOV     IXL, #0010B             ; IX ← 00000010010B
        MOV     IXM, #0001B             ;
        MOV     IXH, #0000B             ; MPE ← 0
        OR      PSW, #.DF.IXE AND 0FH   ; IXE← 1
        ADD     R003, M061
```

As shown in Figure 7-11, when the instructions of **Example 1** are 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 **7.5.3**), 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 7-11. Example of Operation When MPE=0 and IXE=1



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

Assume that the general register is row address 0.

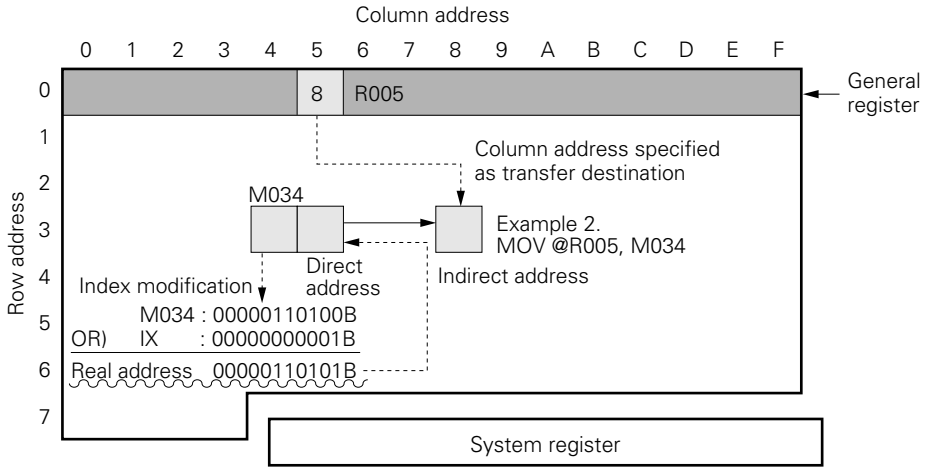
```
R005    MEM     0.05H
M034    MEM     0.34H
        MOV     IXL, #0001B             ;  IX ← 00000000001B
        MOV     IXM, #0000B             ;
        MOV     IXH, #0000B             ;  MPE ← 0
        OR      PSW, #.DF.IXE AND 0FH   ;  IXE  ← 1
        MOV     R005, #8                ;  R005 ← 8
        MOV     @R005, M034             ;  Indirect data transfer using the
                                           register
```

As shown in Figure 7-12, when the above instructions are 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 is 35H and the indirect address is 38H. This example is different from **Example 3** in **7.5.3** 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 **7.5.3** the direct address is not modified.)

Figure 7-12. Example of Operation When MPE=0 and IXE=1



Example 3. Clearing all data memory (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 not modified by IX.
        SKE     IXM, #0111B         ;  Row address 7?
        BR      LOOP                ;  If not 7 then LOOP (row address is
                                    ;  not cleared)
```

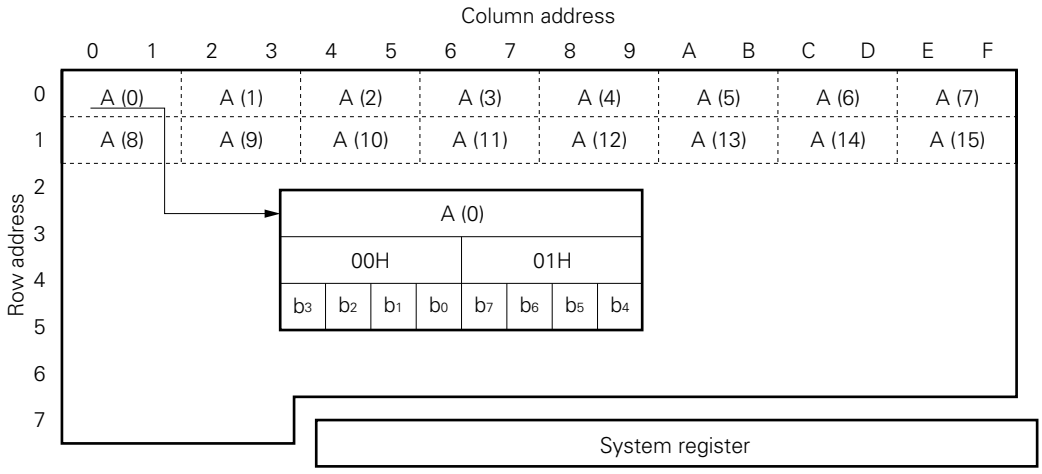
Example 4. Processing an array

As shown in Figure 7-13, to perform the operation:
A(N) = A(N) + 4 (0 ≤ N ≤ 15)
on the element A(N) of a one-dimensional array in which an element is 8 bits, the following instructions are executed:

```
M000    MEM 0.00H
M001    MEM 0.01H
MOV     IXH, #0
MOV     IXM, #N SHR 3             ;  Set the offset of the row address.
MOV     IXL, #N SHL 1 AND 0FH     ;  Set 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 example above, because an element is 8 bits, the value resulting from left-shifting the N's value by 1 bit is set for the index register.

Figure 7-13. Example of Operation When MPE=0 and IXE=1 (Array Processing)



7.6 GENERAL REGISTER POINTER (RP)

7.6.1 General Register Pointer Configuration

Figure 7-14 shows the configuration of the general register pointer.

Figure 7-14. General Register Pointer Configuration

Address	7DH				7EH			
Name	General register pointer (RP)							
Symbol	RPH				RPL			
Bit	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Flag								B C D
Data	0	0	0	0 (RP)				
Initial value when reset	0				0			

As shows in Figure 7-14, the general register pointer consists of seven bits; four bits in system register address 7DH (RPH) and the three high-order bits of system register address 7EH (RPL). However, because the four bits of address 7DH are always set to 0, the register effectively consists of the three high-order bits of address 7EH.

All register bits are cleared to 0 at reset.

7.6.2 Functions of the General Register Pointer

The general register pointer is used to specify the location of the general register in data memory. For a more detailed explanation, refer to **CHAPTER 8 GENERAL REGISTER (GR)**.

The general register consists of sixteen nibbles in any single row of data memory. As shown in Figure 7-15, the general register pointer is used to indicate which row address is being used as the general register.

Since the general register pointer effectively consists of three bits, the data memory row addresses in which the general register can be placed are address locations 0H to 7H of BANK0. In other words, any row in data memory can be specified as the general register.

With the general register allocated in data memory, data can be transferred to and from, and arithmetic/logical operations can be performed on the general register and data memory.

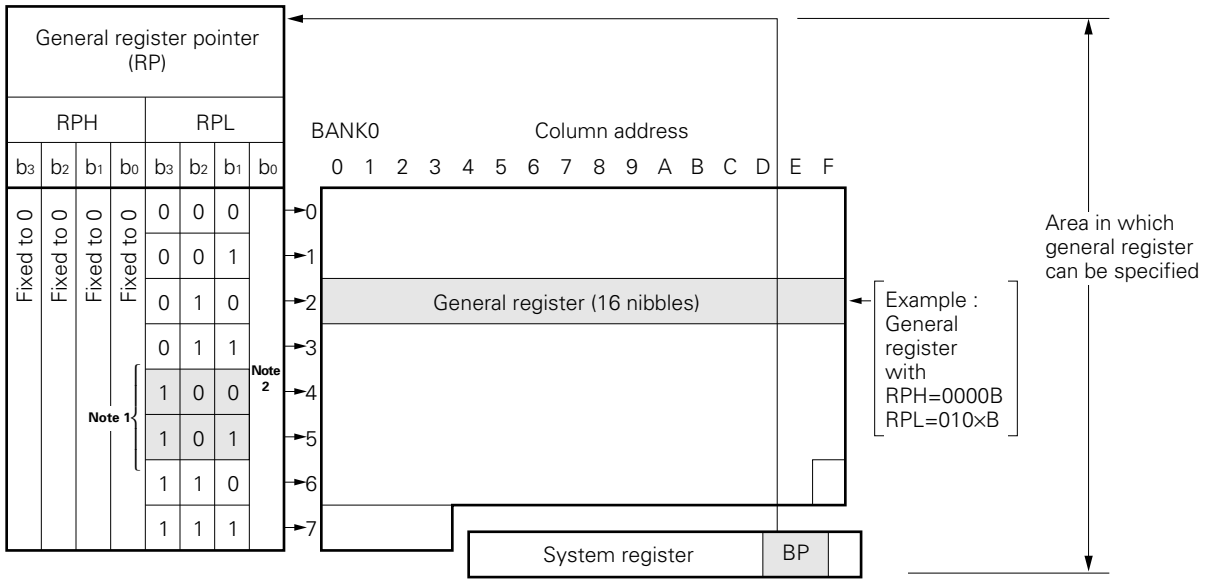
Note that addresses 40H to 6EH are uninstalled memory locations and should therefore not be specified as locations for the general register.

For example, when instructions such as

ADD r,m and LD r,m

are executed, instruction operand r can specify an address in the general register and m specifies an address in data memory. In this way, operations like addition and data transfer can be performed on and between data memory and the general register.

Figure 7-15. General Register Configuration



Notes **1.** These bits should not be specified in the case of the μPD17120 and 17121.

2. This bit is allocated to BCD flag.

7.7 PROGRAM STATUS WORD (PSWORD)

7.7.1 Program Status Word Configuration

Figure 7-16 shows the configuration of the program status word.

Figure 7-16. 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 7-16, 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 register bits are cleared to 0 at reset and at saved at interrupt stack register.

7.7.2 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 7-17 shows an outline of the functions of the program status word.

Figure 7-17. 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.

7.7.3 Index Enable Flag (IXE)

The IXE flag is used to enable to modify index of the data memory address, whether index modification is to be performed on the data memory address used.

For a more detailed explanation, refer to **7.5 INDEX REGISTER (IX) AND DATA MEMORY ROW ADDRESS POINTER (MEMORY POINTER: MP)**.

7.7.4 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 7-2 shows how the CMP flag affects the setting and resetting of the Z flag.

Table 7-2. Zero Flag (Z) and Compare Flag (CMP)

Condition	When CMP=0	When CMP=1
When the result of the arithmatical operation is 0	Z ← 1	Z remains unchanged
When the result of the arithmetic operation is other than 0	Z ← 0	Z ← 0

The Z and CMP flags are used to compare the contents of the general register with those of the data memory. The Z flag does not change other than in arithmetic operations; the CMP flag does not change other than in bit decisions.

Example of 12-bit data comparision

```
;  Is the 12-bit data stored in M001, M002, and M003 equivalent to 456H?
CMP456:
        SET2    CMP, Z
        SUB     M001, #4    ;  Data stored in M001, M002, and M003 are not
        SUB     M002, #5    ;  damaged
        SUB     M003, #6    ;
      ; CLR1    CMP
        SKT     Z           ;  CMP is automatically cleared by the bit decision instruction
        BR      DIFFER      ;  ≠ 456H
        BR      AGREE       ;  =456H
```

7.7.5 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. Also, it is not affected by CMP flag.

7.7.6 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.

7.7.7 Caution on 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.

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.

7.8 CAUTIONS ON USE OF THE SYSTEM REGISTER

7.8.1 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 - AS17K), 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 7-2.

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

For information concerning reserved words, refer to **CHAPTER 19 ASSEMBLER RESERVED WORDS**.

```
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 file with
                                            ;  the pseudo instruction "AR1 MEM 0.76H".
```

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

SETn:	Set a flag to 1
CLRn:	Rest 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 MPE, BCD, CMP, CY, Z and IXE 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
```

Example 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)
```

7.8.2 Handling of System Register Addresses Fixed at 0

In dealing with system register addresses fixed at 0 (refer to **Figure 7-2**), 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) Concerning 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. Below is an example of the type of instructions that will cause the in-circuit emulator to generate an error.

Example 1.
```
MOV   BAMK, #0100B   ;  Attempts to write the value 1 to bit 3 (an address fixed at 0).
```

2.
```
MOV   IXL, #1111B    ;
MOV   IXM, #1111B    ;
MOV   IXH, #0001B    ;
ADD   IXL, #1        ;
ADDC  IXM, #0        ;
ADDC  IXH, #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 a bit fixed at 0 is when the address being written to is in the address register.

(3) When using a 17K series assembler (AS17K)

No error is output when an attempt is made to write the value 1 to a bit 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 (AS17K) does not generate errors because it does not check the correspondence between the symbol (including reserved words) and the data memory address which are the objects of the data memory operation instruction. However, in the following case, the assembler generates an error:

When a value of 1 or more is given to "n" in the incorporation macro instruction "BANKn".

This is so because the assembler determine that no incorporation macro instructions other than BANK0 can be used on the μPD17120 subseries.

CHAPTER 8 GENERAL REGISTER (GR)

The general register (GR) is allocated in data memory. It can therefore be used directly in performing arithmetic/logical operations with and in transferring data to and from general data memory.

8.1 GENERAL REGISTER CONFIGURATION

Figure 8-1 shows the configuration of the general register.

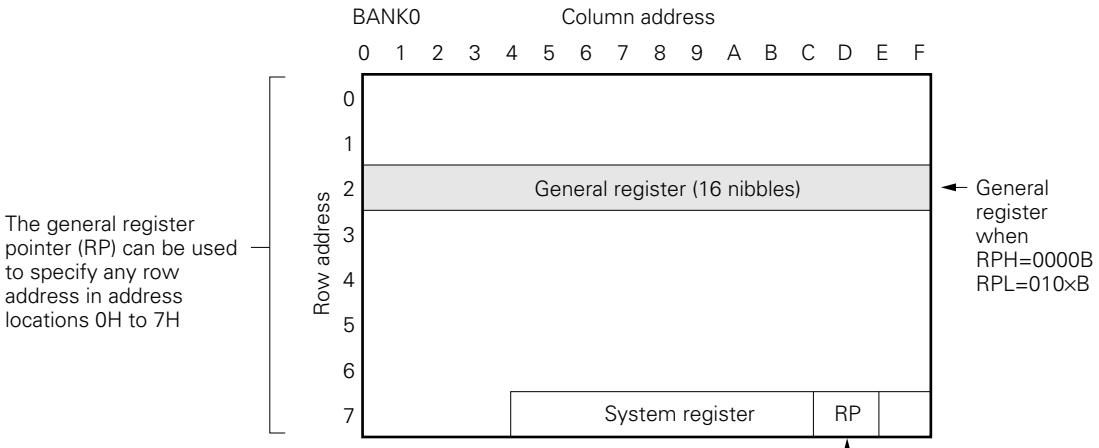
As shown in Figure 8-1, sixteen nibbles in a single row address in data memory (16 × 4 bits) are used as the general register.

The general register pointer (RP) in the system register is used to indicate which row address is to be used as the general register. Because the RP effectively has three valid bits, the data memory row addresses in which the general register can be allocated are address locations 0H to 7H. However, note that addresses 40H to 6EH are uninstalled memory locations and should therefore not be specified as locations for the general register.

8.2 FUNCTIONS OF THE GENERAL REGISTER

The general register can be used in transferring data to and from data memory within an instruction. It can also be used in performing arithmetic/logical operations with data memory within an instruction. In effect, since the general register is data memory, this just means that operations such as arithmetic/logical operations and data transfer can be performed on and between locations in data memory. In addition, because the general register is allocated in data memory, it can be controlled in the same manner as other areas in data memory through the use of data memory manipulation instructions.

Figure 8-1. General Register Configuration



Address	7DH				7EH			
Name	General register pointer (RP)							
Symbol	RPH				RPL			
Bits	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Data	0	0	0	0				BCD
Reset	0	0	0	0	0	0	0	

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

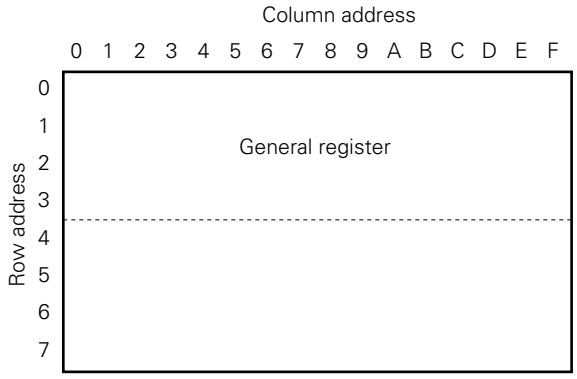
Figure 9-1 shows the configuration of the register file.

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

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

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

Figure 9-1. Register File Configuration



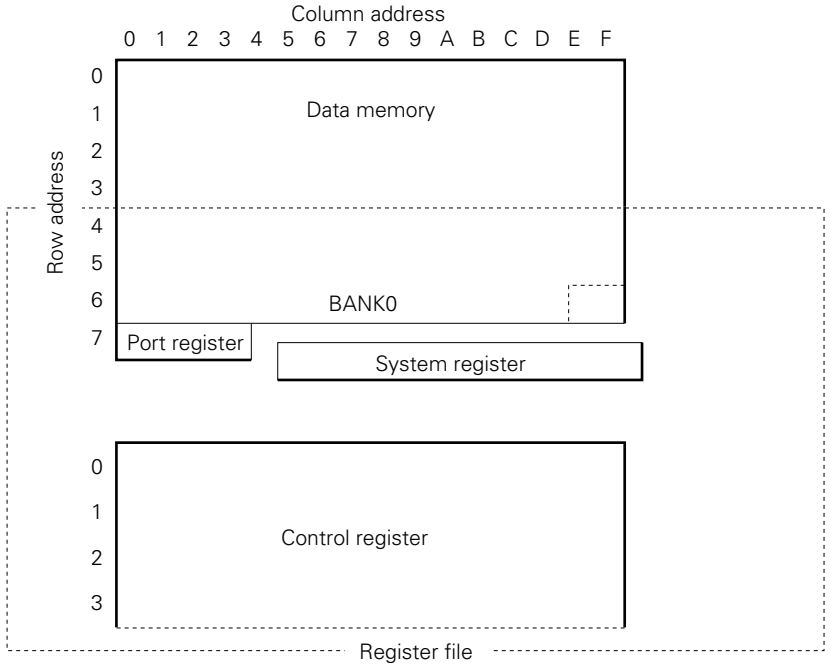
9.1.2 Relationship between the Register File and Data Memory

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 in addresses 40H to 7FH.

This means that the same memory exists in register file addresses 40H to 7FH and in data memory bank 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 mainly used as a control register (a register called the control register is allocated in the register file) for specifying conditions for peripheral hardware.

This control register is allocated within the register file at address location 00H to 3FH.

The rest of the register file (40H to 7FH) overlaps with data memory. As shown in **9.2.3**, because of this overlap, this area of the register file is the same as normal memory with one exception: The register file manipulation instructions PEEK and POKE can be used with this area of memory but not with normal data memory.

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, refer to the section for the peripheral hardware concerned.

- Stack
- Power-on/power-down reset
- Timer
- Serial interface
- INT pin
- Comparator
- General ports
- Interrupts

9.2.3 Register File Manipulation Instructions

Reading and writing data to and from 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 using the PEEK and POKE instructions.

Example

```
M030    MEM     0.30H       ;  Address 30H of the data memory is used as save area of WR.
M032    MEM     0.32H       ;  Address 32H of the data memory is used as operation area of WR.
RF11    MEM     0.91H       ;  Symbol definition
RF33    MEM     0.B3H       ;  Register file addresses 00H to 3FH must be defined with
RF70    MEM     0.70H       ;  symbols as BANK0 address 80H to BFH.
RF73    MEM     0.73H       ;  Refer to 9.4 NOTES ON USING THE REGISTER FILE for details.
        ; BANK0
<1>     PEEK    WR, RF11    ;

        CLR1    MPE         ;  Shows the example of saving WR contents to the general data
        CLR1    IXE         ;  memory (addresses 00H to 3FH). For example, it shows the
        OR      RPL, #0110B ;  case of saving WR contents to address 30H of the data memory
<2>     LD      M030, WR    ;  without address modification.

<3>     POKE    RF73, WR    ;  Data memory of addresses 40H to 7FH and control register can
<4>     PEEK    WR, RF70    ;  transmit/receive data to/from WR directly by PEEK and POKE
<5>     POKE    RF33, WR    ;  instruction.

<6>     ST      WR, M032    ;
```

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

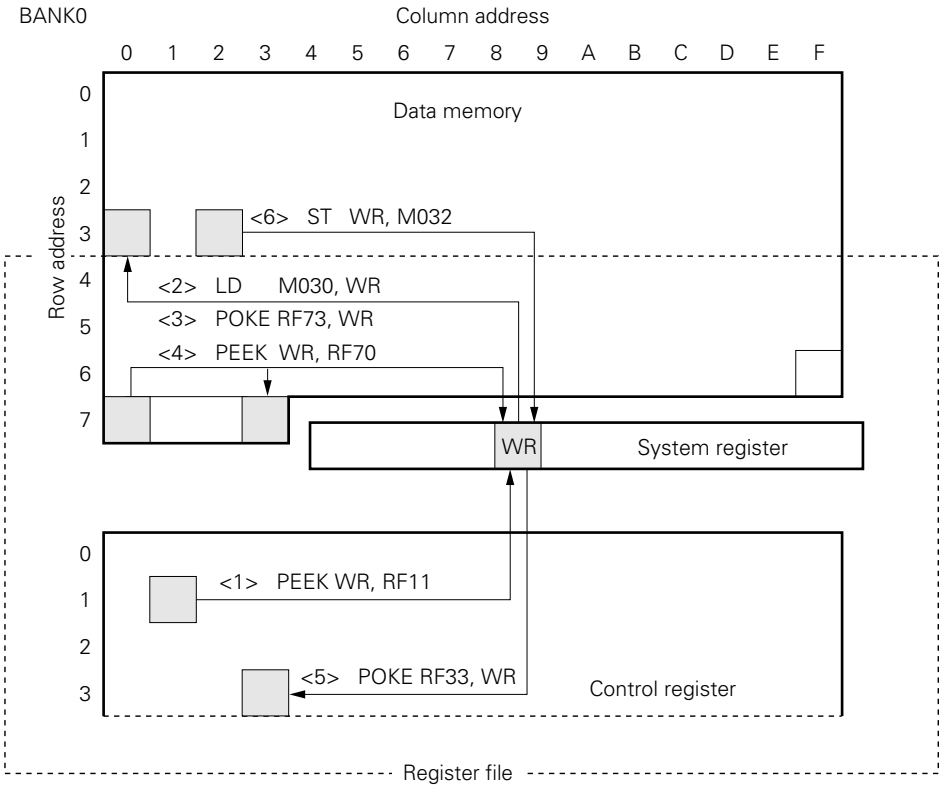
As shown in Figure 9-3, reading and writing of data to and from the control register (address locations 00H to 3FH) is performed using the "PEEK WR, rf" and "POKE rf, WR" instructions. Data within the control register specified using rf can be read from and written to the control register, only by using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

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

Control registers can be manipulated in 1-bit unit by using built-in macro instruction.

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



9.3 CONTROL REGISTER

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

Of these nibbles, only 17 nibbles are actually used in the μPD17120 and 17121, and 20 nibbles are used in the μPD17132, 17133, 17P132, and 17P133.

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:

- TMRES (RF: 11H, bit 2)

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 will yield unpredictable values. In addition, attempting to write to this area has no effect.

Concerning the configuration of control register, refer to **Figures 19-1** and **19-2**.

9.4 CAUTIONS ON USING THE REGISTER FILE

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

It is necessary to take note of the following notes concerning device operation and use of the 17K Series assembler (AS17K) 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 (AS17K)

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 and 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 and no error is generated.

9.4.2 Register File Symbol Definitions and Reserved Words

Attempting to use a numerical value in a 17K Series assembler (AS17K) to specify a register file address in the rf operand of the PEEK WR, rf or POKE rf, WR instructions 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 and 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 (AS17K).

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
        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 (data setting per 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 (AS17K) has reserved words (predefined symbols) for use with these flags.

However, note that there is no reserved word for the stack pointer for its use as a flag. The reserved word used for the stack pointer is "SP", for its use as data memory. For this reason, none of the above flag manipulation instructions using reserved words can be used for 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 is used as a data storage area when data is transferred to/from the CPU peripheral circuit (address register, serial interface, and timer) by the GET and PUT instructions. By using the MOVT DBF, and @AR instructions, 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 BANK0 and consists of a total of 16 bits (4 × 4 bits).

Figure 10-1. Allocation of the Data Buffer

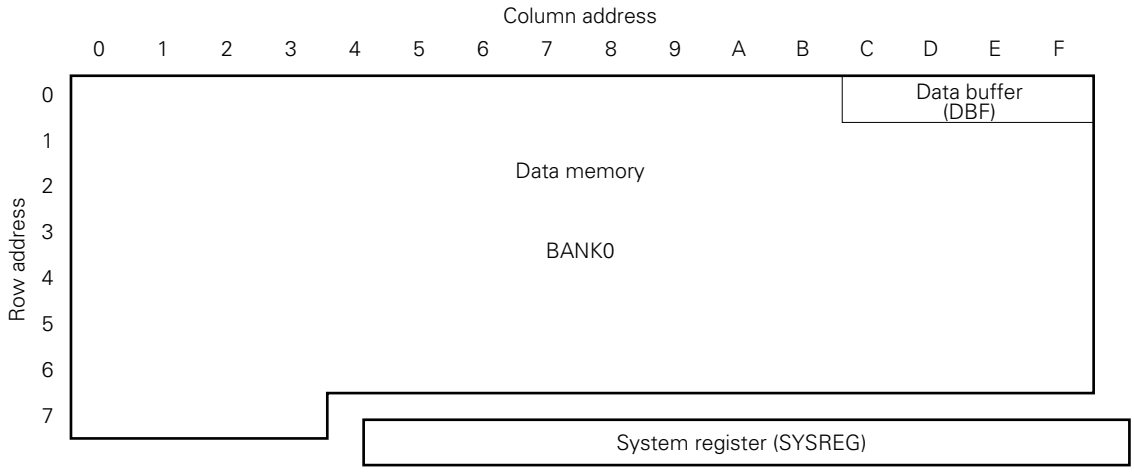


Figure 10-2 shows the configuration of the data buffer. As shown in Figure 10-2, the data buffer is made up of sixteen bits with its least significant bit in bit 0 of address 0FH and its most significant bit in bit 3 of address 0CH.

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	MSB				Data											LSB

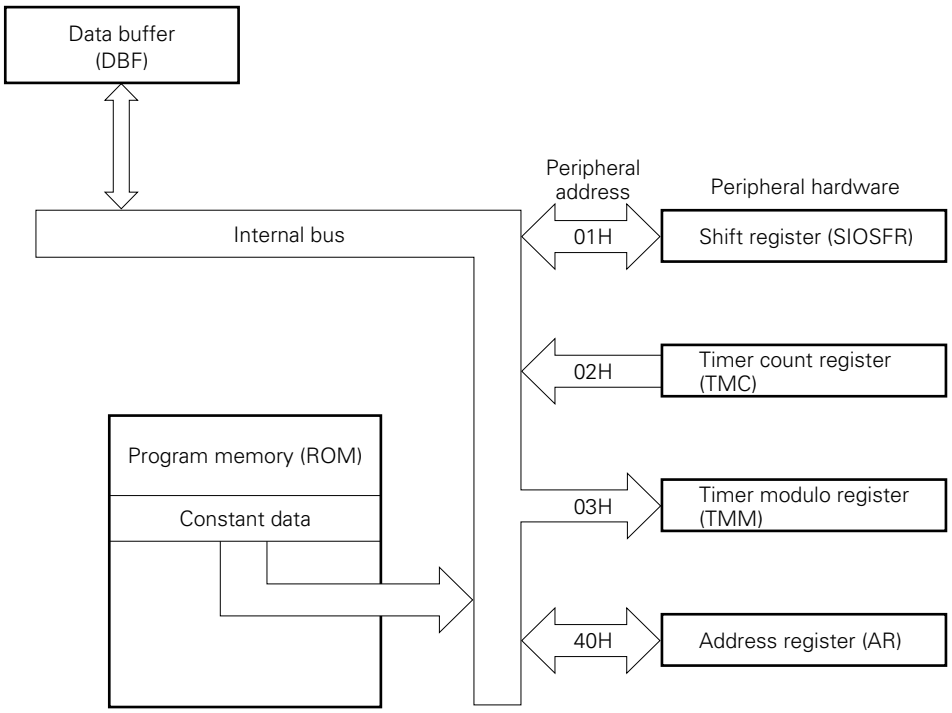
Because the data buffer is allocated in data memory, it can be used in any of the data memory manipulation instructions.

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 the dedicated instructions GET and PUT, data can be transferred between each unit of peripheral hardware and the data buffer.

GET DBF, p : Read the data in the peripheral hardware address specified by p into the data buffer (DBF).

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 (PUT/GET), write-only (PUT) and read-only (GET).

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

- Reading (GET) from write-only (PUT only) peripheral hardware will yield an unpredictable value.
- Writing (PUT) to read-only (GET only) peripheral hardware has no effect (regarded 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		Effective Bit Length
			PUT	GET	
01H	SIOSFR	Shift register	○	○	8 bits
02H	TMC	Timer count register	×	○	8 bits
03H	TMM	Timer modulo register	○	×	8 bits

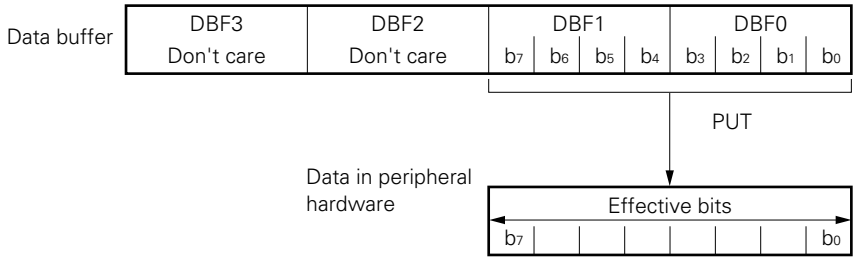
(2) Peripheral hardware with input/output in 16-bit units

Peripheral Address	Name	Peripheral Hardware	Direction of Data		Effective Bit Length
			PUT	GET	
40H	AR	Address register	○	○	10 bits

10.2.2 Data Transfer with Peripheral Hardware

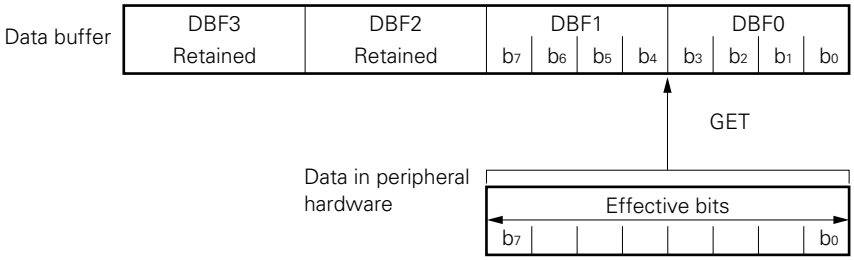
Data can be transferred between the data buffer and peripheral hardware in 8- or 16-bit units. Instruction cycle for a single PUT or GET instruction is the same regardless of whether eight or sixteen bits are being transferred.

Example 1. PUT instruction (when the effective bits in peripheral hardware are the 8 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 effective bits in peripheral hardware are the 8 bits from 7 to 0)

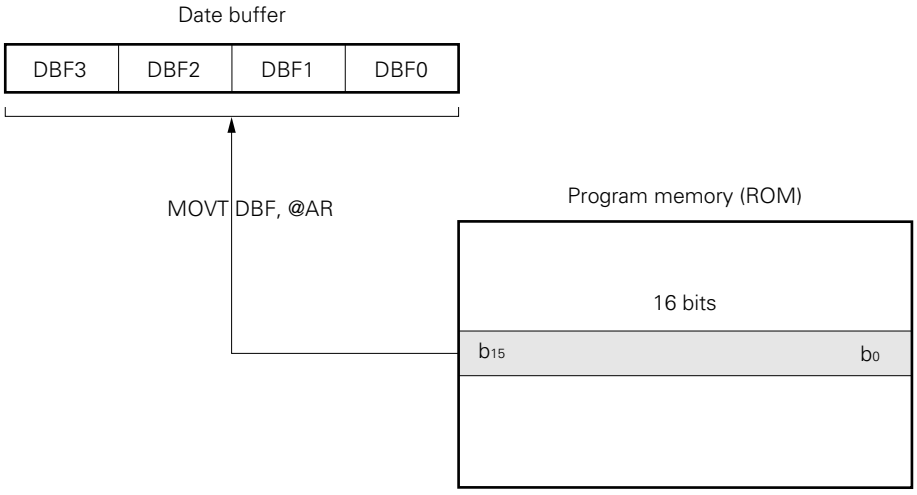


When only eight bits of data are being read into the data buffer, the values in the upper eight bits of the data buffer (DBF3, DBF2) remain unchanged.

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 ARITHMETIC AND LOGIC UNIT

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 FF, carry flag FF, compare flag FF, and the BCD flag FF.

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 consists 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 operations in decimal are performed on one place (BCD operation).

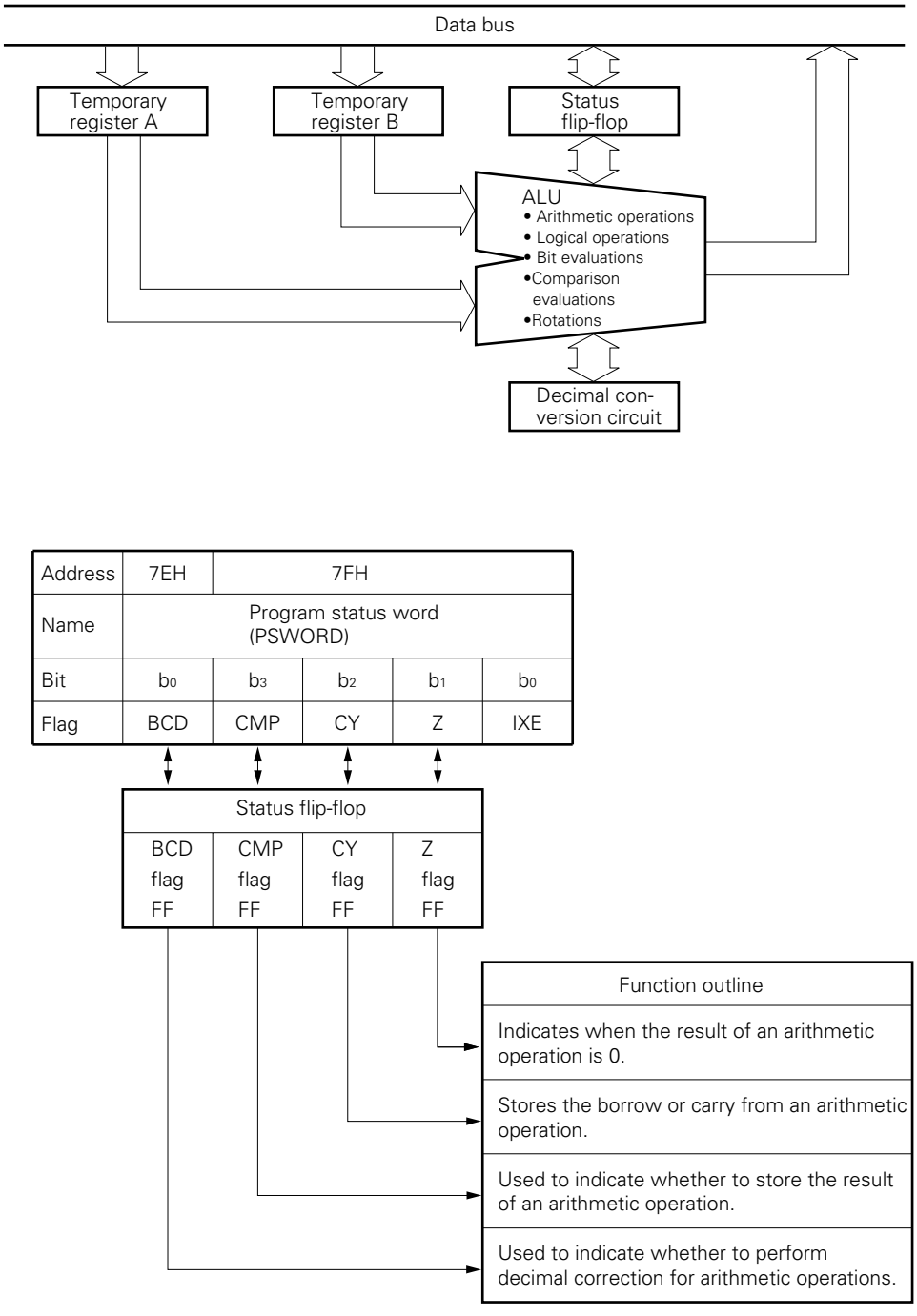
Logical operations include ANDing, ORing, and XORing. Their operands can be general register contents and data memory contents, or data memory contents and immediate data.

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).

Figure 11-1. Configuration of the ALU



[MEMO]

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

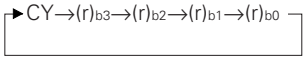
ALU Functions		Instruction	Operation	Explanation
Arithmetic operations	Addition	ADD r, m	(r) ← (r) + (m)	Adds contents of general register and data memory. Result is stored in general register.
		ADD m, #n4	(m) ← (m) + n4	Adds immediate data to contents of data memory. Result is stored in data memory.
		ADDC r, m	(r) ← (r) + (m) + CY	Adds contents of general register, data memory and carry flag. Result is stored in general register.
		ADDC m, #n4	(m) ← (m) + n4 + CY	Adds immediate data, contents of data memory and carry 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 carry flag from contents of general register. Result is stored in general register.
		SUBC m, #n4	(m) ← (m) – n4 – CY	Subtracts immediate data and carry flag from data memory. Result is stored in data memory.
Logical operations	Logical OR	OR r, m	(r) ← (r) $\vee$ (m)	OR operation is performed on contents of general register and data memory. Result is stored in general register.
		OR m, #n4	(m) ← (m) $\vee$ 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) $\wedge$ (m)	AND operation is performed on contents of general register and data memory. Result is stored in general register.
		AND m, #n4	(m) ← (m) $\wedge$ 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) $\veebar$ (m)	XOR operation is performed on contents of general register and data memory. Result is stored in general register.
		XOR m, #n4	(m) ← (m) $\veebar$ n4	XOR operation is performed on immediate data and contents of data memory. Result is stored in data memory.
Bit judgement	TRUE	SKT m, #n	CMP ← 0, if (m) $\wedge$ 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) $\wedge$ n=0, then skip	Skips next instruction if all bits in data memory specified by n are FALSE (0). Result is not stored.
Comparison judgement	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	Operational Variance Depending on Program Status Word (PSWORD)					
Arithmetic Operation	BCD flag's value	CMP flag's value	Operating action	CY flag	Z flag	Modification by IXE=1
	0	0	The binary operation result is stored.	Set if a carry or borrow is generated; reset otherwise.	Set if the operation result is 0000B; reset otherwise.	Yes
	0	1	The binary operation result is not stored.		The status is retained if the operation result is 0000B; reset otherwise.	
	1	0	The BCD operation result is stored.		Set if the operation result is 0000B; reset otherwise.	
	1	1	The BCD operation result is not stored.		The status is retained if the operation result is 0000B; reset otherwise.	
Logical Operation	Don't care (Held)	Don't care (Held)	Unchanged	Don't care (Held)	Don't care (Held)	Yes
Bit Judgement	Don't care (Held)	Is reset	Unchanged	Don't care (Held)	Don't care (Held)	Yes
Comparison Judgement	Don't care (Held)	Don't care (Held)	Unchanged	Don't care (Held)	Don't care (Held)	Yes
Rotation	Don't care (Held)	Don't care (Held)	Unchanged	General register b_0's value	Don't care (Held)	Yes

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), decimal correction is not performed.

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. 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

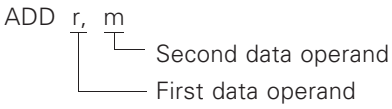
Operation Result	Addition in 4-bit Binary		Addition in BCD	
	CY	Operation Result	CY	Operation Result
0	0	0000	0	0000
1	0	0001	0	0001
2	0	0010	0	0010
3	0	0011	0	0011
4	0	0100	0	0100
5	0	0101	0	0101
6	0	0110	0	0110
7	0	0111	0	0111
8	0	1000	0	1000
9	0	1001	0	1001
10	0	1010	1	0000
11	0	1011	1	0001
12	0	1100	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



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 a no operation instruction: NOP) 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 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.

11.3.1 to **11.3.4** explain the relationship between each command and the program status word.

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 carry 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, refer to **Table 11-2**.

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):

$\overbrace{0}^{\text{CY}}$, 0000B to $\overbrace{1}^{\text{CY}}$, 0011B

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

$\overbrace{1}^{\text{CY}}$, 0110B to $\overbrace{1}^{\text{CY}}$, 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 Cautions on 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 cause either the CY or Z flag in the program status word (PSWORD) to be set. 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 JUDGEMENT

As shown in Table 11-6, there are both TRUE (1) and FALSE (0) bit judgement instructions.

The SKT instruction skips the next instruction when a bit is judged as TRUE (1) and the SKF instruction skips the next instruction when a bit is judged as FALSE (0).

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

Bit judgement are not affected by the BCD flag in the program status word (PSWORD) and bit judgements do not cause either the CY or Z flag 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, refer to **CHAPTER 7 SYSTEM REGISTER (SYS REG)**.

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

Table 11-6. Bit Judgement Instructions

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

11.5.1 TRUE (1) Bit Judgement

The TRUE (1) bit judgement 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, bit 3, 1, and 0 of data memory M1 are judged 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 judged. Since bit 2 of data memory M1 is FALSE (0), the program branches to C.

11.5.2 FALSE (0) Bit Judgement

The FALSE (0) bit judgement 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,   #1011B
SKT   M1,   #0110B   ; <1>
BR    A
BR    B
SKT   M1,   #1101B   ; <2>
BR    C
BR    D
```

In this example, bits 2 and 1 of data memory M1 are judged 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 judged. Since bit 3 of data memory M1 is TRUE (1), the program branches to C.

11.6 COMPARISON JUDGEMENT

As shown in Table 11-7, there are comparison judgement 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, refer to **11.3**.

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

11.6.1 to **11.6.4** explain the "equal to", "not equal to", "greater than or equal to", and "less than" comparison judgements.

Table 11-7. Comparison Judgement Instructions

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

11.6.1 "Equal to" Judgement

The "equal to" judgement 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 to" Judgement

The "not equal to" judgement 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 to" Judgement

The "greater than or equal to" judgement 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. Next, it will branch to D since the value in data memory is equal to that of the immediate data. Lastly it will branch to E since the value in data memory is less than that of the immediate data.

11.6.4 "Less Than" Judgement

The "less than" judgement 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. Next, it will branch to C since the value in data memory is equal to that of the immediate data. Lastly it will branch to E since the value in data memory is greater than that of the immediate data.

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.

11.7.1 and **11.7.2** explain rotation.

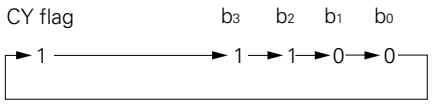
11.7.1 Rotation to the Right

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 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 $\rightarrow b_3$, $b_3 \rightarrow b_2$, $b_2 \rightarrow b_1$, $b_1 \rightarrow b_0$, $b_\infty \rightarrow$ CY flag.

2.
```
MOV  PSW,  #0000B ; Resets CY flag to 0.
MOV  R1,   #1000B
MOV  R2,   #0100B
MOV  R3,   #0010B
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
MOV   R2,   #0100B
MOV   R3,   #0010B
ADDC  R3, R3
ADDC  R2, R2
ADDC  R1, R1
```

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 each bit. Input/output is specified by P0ABIO0 to P0ABIO3 (address 35H) in the register file.

When P0ABIOn is 0 (n=0 to 3), 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 P0ABIOn is 1 (n=0 to 3), 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.

At reset, P0ABIOn is set to 0 and all P0A pins become input ports. The contents of the port output latch are 0.

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

P0ABIOn RF: 35H	Pin Input/Output	BANK0 70H	
		Write	Read
0	Input	Possible Write to the P0A latch	P0A pin status
1	Output		P0A latch contents

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 0 at address 24H) 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.

At reset, P0BGIO is 0 and all P0B pins are input ports. The value of the port 0B output latch is 0.

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

P0BGIO RF: 24H, bit 0	Pin Input/Output	BANK0 71H	
		Write	Read
0	Input	Possible Write to the P0B latch	P0B pin status
1	Output		P0B latch contents

12.3 PORT 0C ($P0C_0$, $P0C_1$, $P0C_2$, $P0C_3$) ... in the case of the μPD17120 and 17121

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 34H) 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.

Table 12-3. Writing/reading to/from Port Register (0.72H) (μPD17120, 17121)

(n=0 to 3)

P0CBIOn RF: 34H	Pin Input/Output	BANK0 72H	
		Write	Read
0	Input	Possible Write to the P0C latch	Status of P0C pin
1	Output		Contents of P0C latch

12.4 PORT 0C ($P0C_0/Cin_0$, $P0C_1/Cin_1$, $P0C_2/Cin_2$, $P0C_3/Cin_3$) ... in the case of the μPD17132, 17133, 17P132, and 17P133

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 is specified with P0CBIO0 to P0CBIO3 (address 34H) in the register file.

If P0CNIOn is 0 (n=0 to 3), the each pin of P0C is used as input port. If a data read instruction is executed for the port register, the pin statuses are read.

If P0CNIOn is 1 (n=0 to 3), the each pin of P0C is 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 0C can also be used as an analog input to the comparator. P0C0IDI to P0C3IDI (address 23H) in the register file are used to switch the port and analog input pin.

If P0CnIDI is 0 (n=0 to 3), the $P0C_n/Cin_n$ pin functions as a port. If P0CnIDI is 1 (n=0 to 3), the $P0C_n/Cin_n$ pin functions as the analog input pin of the comparator.

Therefore, when using pins as analog inputs, 1 should be set to P0CnIDI at the initial setting of the program. Switching of the analog input pins to be compared is executed by CMPCH0 and CMPCH1 (RF: address 1CH). To use the pins as analog input pins of the comparator, set P0CBIOn=0 so that they are set as input ports (Refer to **13.2 COMPARATOR**). At reset, P0CBIOn and P0CnIDI are set to 0 (n=0 to 3) and all of port 0C pins become input ports. The contents of the port output latch become 0.

Table 12-4. Writing into and Reading from the Port Register (0.72H) and Pin Function Selection

(n=0 to 3)

P0CnIDI RF: 23H	P0CBIOn RF: 34H	Function	BANK0 72H	
			Write	Read
0	0	Input port	Write in the P0C latch	Pin state of P0C
	1	Output port		Contents of P0C latch
1	0	Comparator analog input**Note 1**		Pin state of P0C
	1	Analog inputs of comparator and output port**Note 2**		Contents of P0C

Notes 1. This setting is ordinally selected when the pins are used as analog inputs of the comparator.

2. These pins function as an output port. At this time, the analog input voltage is changed by the effect of the output from the port. When using the pin as the analog input, be sure to set it to P0CBIOn=0.

12.5 PORT 0D (P0D0/$\overline{SCK}$, P0D1/SO, P0D2/SI, P0D3/$\overline{TMOUT}$)

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. The mask option can be used to specify that a pin contain a pull-up resistor bit-by-bit **Note**.

Input or output can be specified bit-by bit. Input/output is specified with P0DBIO0 to P0DBIO3 (address 33H) 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.

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 output. SIOEN (0AH bit 0) in the register file is used to switch ports (P0D0 to P0D2) to serial interface input/output ($\overline{SCK}$, SO, SI) and vice versa. TMOSEL (bit 0 at address 12H) in the register file is used to switch a port (P0D3) to timer output ($\overline{TMOUT}$) and vice versa. If TMOSEL=1 is selected, 1 is output at timer reset. This output is inverted every time a timer count value matches the modulo register contents.

Note The μPD17P132 and 17P133 have no pull-up resistor by mask option.

Table 12-5. Register File Contents and Pin Functions

(n=0 to 3)

Register File Value			Pin Function			
TMOSEL RF: 12H Bit 0	SIOEN RF: 0AH Bit 0	P0DBIOn RF: 33H Bit n	$P0D_0/\overline{SCK}$	$P0D_1$/SO	$P0D_2$/SI	$P0D_3/\overline{TMOUT}$
0	0	0	Input port			
		1	Output port			
	1	0	$\overline{SCK}$	SO	SI	Input port
		1				Output port
1	0	0	Input port			$\overline{TMOUT}$
		1	Output port			
	1	0	$\overline{SCK}$	SO	SI	
		1				

Table 12-6. Contents Read from the Port Register (0.73H)

Port Mode		Contents Read from the Port Register (0.73H)
Input port		Pin status
Output port		Output latch contents
$\overline{SCK}$	An internal clock is selected as a serial clock.	Output latch contents
	An external clock is selected as a serial clock.	Pin status
SI		Pin status
SO		Not defined
$\overline{TMOUT}$		Output latch contents

Caution Using the serial interface causes the output latch for the $P0D_1$/SO pin to be affected by the contents of the SIOSFR (shift register). So, reset the output latch before using the pin as output port.

12.6 PORT 0E ($P0E_0$, $P0E_1/V_{ref}$) ... V_{ref}; μPD17132, 17133, 17P132, and 17P133 only

Port 0E is a 2-bit input/output port with an output latch. It is mapped into bits 0 and 1 of address 6FH in data memory. The output format is N-ch open-drain output. The mask option can be used to specify that a pin contain a pull-up resistor bit-by-bit.

$P0E_1/V_{ref}$ pin is also used as external reference voltage input of the comparator (incorporated only in the μPD17132, 17133, 17P132, and 17P133), and its function is changed from port to external reference voltage input depending on the value of the reference voltage selection resister. (CMPVREF0 to CMPVREF3) (Refer to **13.2 COMPARATOR**.)

Input or output can be specified bit-by-bit. Input/output is specified with P0EBIO0 and P0EBIO1 (bits 0 and 1 at address 32H) in the register file.

If P0EBIOn is 0 (n=0, 1), the each pin of P0E is used as input port. If a data read instruction is executed for the port register, the pin statuses are read.

If P0EBIOn is 1 (n=0, 1), the each pin of P0E is used as output port and the value written in the output latch are output pins.

If a data read instruction is executed regardless of the mode, the pin statuses, rather than output latch value, are fetched.

At reset, P0EBIOn is set to 0 (n=0, 1) and each of port 0E pin become input port. The contents of the port output latch become 0.

The write instruction to bits 2 and 3 at address 6FH becomes invalid. If the value is read, 0 is output.

Remark The μPD17P132 and 17P133 have no pull-up resistor by mask option.

Table 12-7. Writing into and Reading from the Port Registers (0.6FH.0, 0.6FH.1)

(n=0, 1)

P0EBIOn RF: 32H	Pin Input/ Output	BANK01 6FH	
		Write	Read
0	Input	Possible[Note]	P0E pin status
1	Output	Write to the output latch of P0E	

Note Port register 6FH is a write only register. The data which is written during input mode (P0EBIOn=0) will be ignored.

12.6.1 Cautions when Operating Port Registers

Among the input/output ports in the μPD17120 series, only port 0E is such that, even when in output mode, doing a read causes the status of the pins to be read.

Consequently, when executing port register read macro instructions (SETn/CLRn, etc.) or bit manipulation instructions such as AND/OR/XOR, etc., you may inadvertently change the status of the pins.

Be particularly careful when port 0E is being externally forced down to low level.

Figure 12-1 shows an example of the changes in the port register and microcontroller internal status when the CLR1 P0E1 instruction (equivalent to the AND 6F, #1101B instruction) is executed.

For example, consider the case where both the $P0E_1$ and $P0E_0$ pins of port 0E are used for output, high level is output from pins $P0E_1$ and $P0E_0$, and the $P0E_0$ pin is being externally forced down to a low level; then the status of each pin of port 0E is as shown in Figure 12-1 <1>. The ($P0E_3$ and $P0E_2$ pins do not exist in the μPD17120 subseries, but are handled as if they exist in the program.)

If the CLR1 P0E1 instruction is executed to bring the $P0E_1$ pin to low level, the status of each pin of port 0E changes as shown in Figure 12-1 <2>. In this case, the $P0E_1$ pin naturally changes to output low level, but the value of the port register changes so that pin $P0E_0$, which should output high level, also outputs low level. This result comes about because the CLR1 P0E1 instruction is executed not on the port register, but on the state of the pins.

To avoid this phenomenon, use a MOV or other instruction to set the state of all the pins, not just the pins that are to be changed. In this example, to set just the $P0E_1$ pin to a low level, you can use the MOV 6FH, #1101 instruction, and the problem will not occur.

For the same reason, when using port 0E for mixed input and output, be sure to put the pins that are being used for input into input mode (P0EBI0n = 0).

Figure 12-1. Changes in Port Register Due to Execution of the CLR1 P0E1 Instruction

<1> Before executing instructions

	$P0E_3$	$P0E_2$	$P0E_1$	$P0E_0$
Port register	Does not exist		1	1
Microcomputer state	–	–	H output	H output
Pin state	–	–	H	L (forced)

Execution of the CLR1 P0E1 instruction [AND 6FH, #1101B]

<2> After executing instructions

	$P0E_3$	$P0E_2$	$P0E_1$	$P0E_0$
Port register	Does not exist		0	0
Microcomputer state	–	–	L output	L output
Pin state	–	–	L	L

H: high level L: low level

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 0B is used as group I/O. The register shown in the figure below is used for input/output switching.

Figure 12-2. Input/Output Switching by Group I/O

RF: 24H

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

Read=R, write=W

P0BGIO	Function
0	Sets Port 0B to input mode.
1	Sets Port 0B 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 0A, port 0C, port 0D, and port 0E are used as bit I/O. The register shown in the figure below is used for input/output switching.

Figure 12-3. Bit I/O Port Control Register (1/4)

RF: 35H

	Bit 3	Bit 2	Bit 1	Bit 0
	P0ABIO3	P0ABIO2	P0ABIO1	P0ABIO0
Read/write	R/W			
Initial value when reset	0	0	0	0

Read=R, write=W

P0ABIO0	Function
0	Sets $P0A_0$ to input mode.
1	Sets $P0A_0$ to output mode.

P0ABIO1	Function
0	Sets $P0A_1$ to input mode.
1	Sets $P0A_1$ to output mode.

P0ABIO2	Function
0	Sets $P0A_2$ to input mode.
1	Sets $P0A_2$ to output mode.

P0ABIO3	Function
0	Sets $P0A_3$ to input mode.
1	Sets $P0A_3$ to output mode.

Figure 12-3. Bit I/O Port Control Register (2/4)

RF: 34H

	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 12-3. Bit I/O Port Control Register (3/4)

RF: 33H

	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.

Figure 12-3. Bit I/O Port Control Register (4/4)

RF: 32H

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	P0EBIO1	P0EBIO0
Read/write	R/W			
Initial value when reset	0	0	0	0

Read=R, write=W

P0EBIO0	Function
0	Sets $P0E_0$ to input mode.
1	Sets $P0E_0$ to output mode.

P0EBIO1	Function
0	Sets $P0E_1$ to input mode.
1	Sets $P0E_1$ to output mode.

CHAPTER 13 PERIPHERAL HARDWARE

13.1 8-BIT TIMER COUNTER (TM)

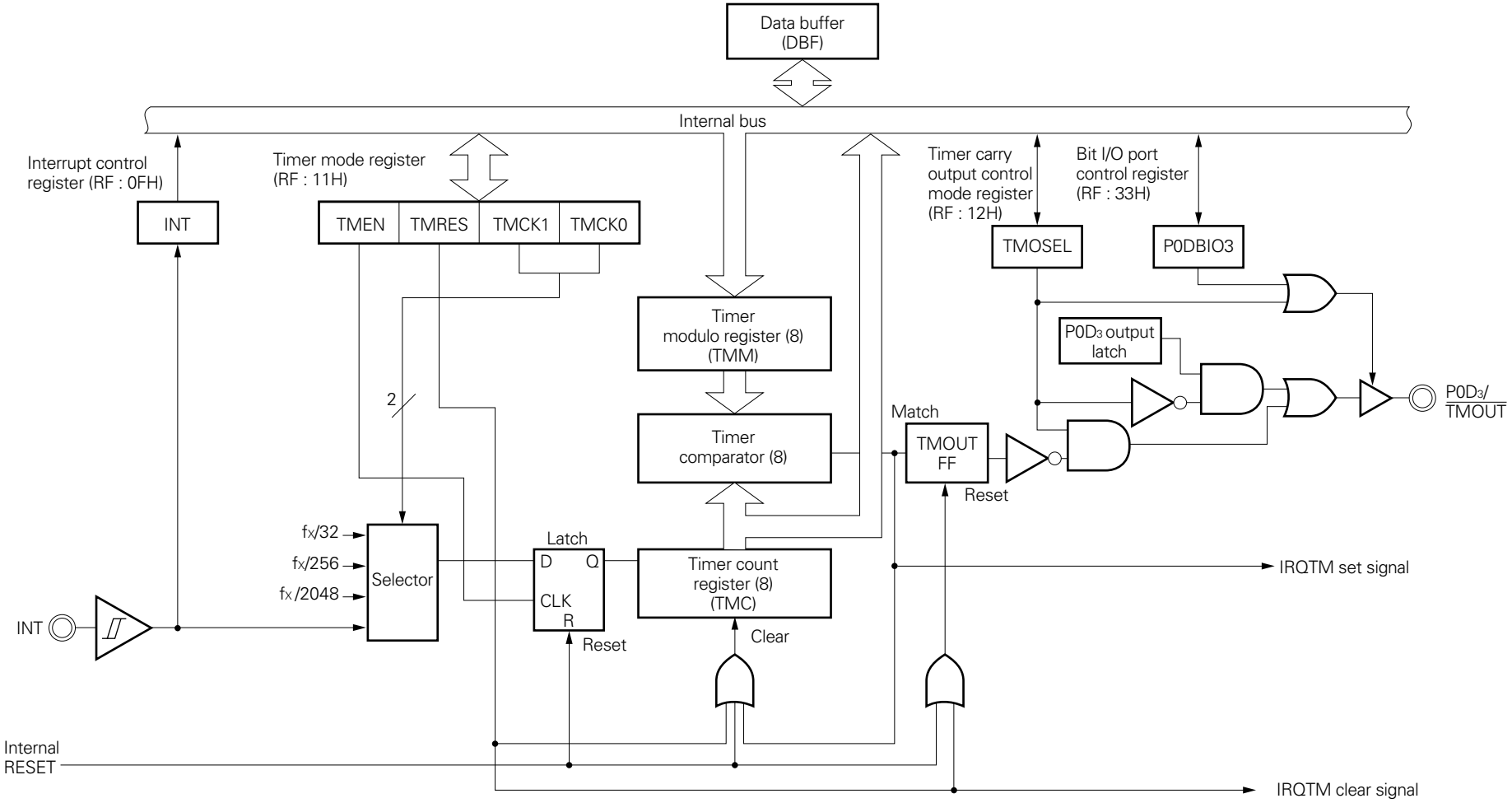
The μPD17120 subseries contains an 8-bit timer counter system. Control of the 8-bit timer counter is performed through hardware manipulation using the PUT/GET instruction or through register manipulation on the register file using the PEEK/POKE instruction.

13.1.1 8-Bit Timer Counter Configuration

Figure 13-1 shows the configuration of the 8-bit timer counter. The 8-bit timer counter consists of the comparator, which compares the 8-bit count register, 8-bit modulo register, count register, and modulo register values; and the separator, which selects the count pulse.

Caution The modulo register is for writing only. The count register is for reading only.

Figure 13-1. Configuration of the 8-bit Timer Counter



13.1.2 8-bit Timer Counter Control Register

There are two types of 8-bit timer counter control registers; timer mode register and timer carry output control mode register.

Figures 13-2 and 13-3 show the configuration of 8-bit timer counter control registers.

Figure 13-2. Timer Mode Register

RF: 11H

	Bit 3	Bit 2	Bit 1	Bit 0
	TMEN	TMRES	TMCK1	TMCK0
Read/write	R/W			
Initial value when reset	1	0	0	0

Read=R, write=W

TMCK1	TMCK0	Source Clock Selection
0	0	fx/256
0	1	fx/32
1	0	fx/2048
1	1	External clock from INT pin

TMRES	Timer Reset
0	No influence to timer
1	Reset count register (TMC) and IRQTM

Remark TMRES is automatically cleared (0) when set (1). When reading it, 0 is always read.

TMEN	Timer Start Instruction
0	Stop timer counting.
1	Start timer counting.

Remark TMEN can be used as the status flag which detects the count state of the timer (0 : count stop, 1 : counting).

13.1.3 Operation of 8-bit Timer Counters

(1) Count Register

Count register are 8-bit up counters whose initial values are 00H. They are incremented each time a count pulse is entered.

The counter register is initialized in the following situations:

- when the microcontroller is reset (refer to **CHAPTER 6 RESET**);
- when the content of the 8-bit modulo register coincides with the count register value, thus causing the comparator to generate the relevant signal; and
- when "1" is written in the TMRES of the register file.

(2) Modulo register

The modulo registers determine the count value of 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 output match signals when the value 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 contents of the count register to 0 and automatically sets the interrupt request flag (IRQTM) to 1. At this time interrupt handling occurs if the EI instruction (interrupt acceptance enable instruction) is executed and also the interrupt enable flag (IPTM) is set. When an interrupt is accepted, the interrupt request flag (IRQTM) is set to 0 and program control transfers to the interrupt handling routine.

13.1.4 Selecting Count Pulse

A count pulse is selected with TMCK0 or TMCK1.

One system clock fx can be selected from four types: a 2048-count pulse, 256-count pulse, 32-count pulse, and an external count pulse input from the INT pin.

At reset, TMCK0 and TMCK1 are 0 and fx/256 is selected.

At power start-up or reset, timer is used to generate stabilization wait time. For this purpose, the initial values are TMCK0=0 and TMCK1=0 and fx/256 is selected. Since the initial value is set to TMEN=1, the system starts at address 0000H after 8 ms after reset at fx=8 MHz (32 ms at fx=2 MHz). (Refer to **CHAPTER 16 RESET**.)

13.1.5 Setting a Count Value in Modulo Register and Calculation Method

Count value is set in the module register via the data buffer (DBF).

(1) Setting the count value in modulo register

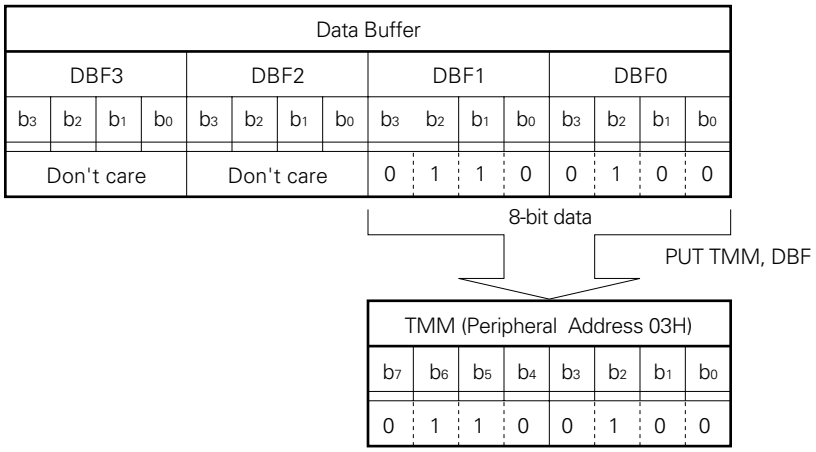
A count value is set in the modulo register via the data buffer using the PUT instruction. The peripheral address of the modulo register is 03H.

When a value is sent by the PUT instruction, data in the eight low-order bits (DBF1 and DBF0) of data buffer is sent to the modulo register. Figure 13-3 shows an example of count value setting.

Figure 13-3. Setting the Count Value in a Modulo Register

Example of setting count value 64H in timer 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   TMM, DBF        ; The value is transferred with reserved word TMM.
```



Caution The range of values that can be set in the module register is 01H to FFH. If 00H is set, normal counting operation is not performed.

The modulo register is for writing only. If is not possible to read a value from the modulo register. Neither is it possible, while the 8-bit timer counter is in operation, to stop the counting operation even by executing the PUT TMM and DBF instructions.

(2) Calculation method of count value

The time interval of the identity signal being emitted from the comparator is determined by the value that is set in the modulo register. The formula for finding the value N of the modulo register from the time interval T [sec] is shown below:

$$T = \frac{N+1}{f_{CP}} = (N+1) \times T_{CP}$$

$$N = T \times f_{CP} - 1 \text{ or } N = \frac{T}{T_{CP}} - 1 \quad (N=1 \text{ to } 255)$$

f_{CP} : Count pulse's frequency [Hz]
T_{CP} : Count pulse's frequency [sec] ($1/f_{CP}$ = resolution)

(3) Calculation example and program example when calculating count value by interval time

- **Example of assuming 7 ms as interval time for timer (System clock: fx=8 MHz)**
  Assuming 7 ms as interval time, it is impossible to set 7 ms interval time from the resolution of the timer. Therefore, count value should be calculated by selecting the source clock the resolution of which is maximum (f_X/256, resolution: 32 μs) to set the nearest interval time.

  Example t=7 ms, resolution: 32 μs

$$N = \frac{t}{(\text{Resolution})} - 1$$

$$= \frac{7 \times 10^{-3}}{32 \times 10^{-5}} - 1$$

$$= 217.75 = 218 \text{ (: DAH)}$$

The value of modulo register the interval time of which becomes nearest to 7 ms is DAH, and the interval time at that time becomes 7.008 ms.

(Program example)

```
MOV     DBF0,   #0AH        ; Stores DAH to DBF by using
MOV     DBF1,   #0DH        ; reserved words "DBF0" and "DBF1"
PUT     TMM,    DBF         ; Transfers the contents of DBF by using reserved word "TMM"

INITFLG TMEN, TMRES, NOT TMCK1, NOT TMCK0
                            ; Sets TMEN and TMRES by using built-in macro instruction "INITFLG".
                            ; Sets source clock of timer to "fx/256", and starts counting.
```

13.1.6 Margin of Error of Interval Time

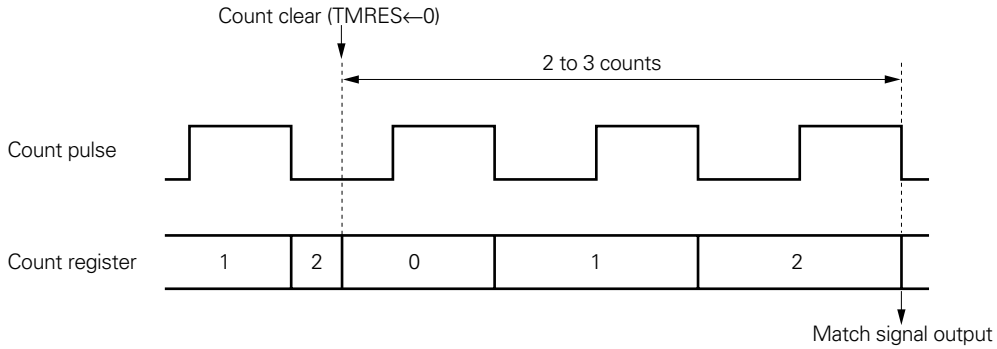
It is possible for the interval time to generate a margin of error of up to –1.5 counts. Be careful if set value of the modulo register is small.

(1) Error (up to –1 count) when the count register is cleared to 0 during counting

The count register of the 8-bit timer counter is cleared to 0 by setting (to 1) the TMRES flag. However, the scaler for generating the count pulse from the system clock is not reset.

Therefore, if, during counting, the TMRES flag is set (to 1) to clear the count to 0, an error margin of one cycle of the count pulse is generated in the timing of the first count. A count example when setting the modulo register to 2 is shown below:

Figure 13-4. Error in Zero-Clearing the Count Register during Counting



In the example above, the identity signal is generated every three counts. However, only for the first time after the count is cleared, the identity signal is generated for the minimal 2 counts.

The above error occurs not only when TMEN=1 ← 0 but when TMRES ← 1.

(2) Error in Starting Counting from the Count Halt State

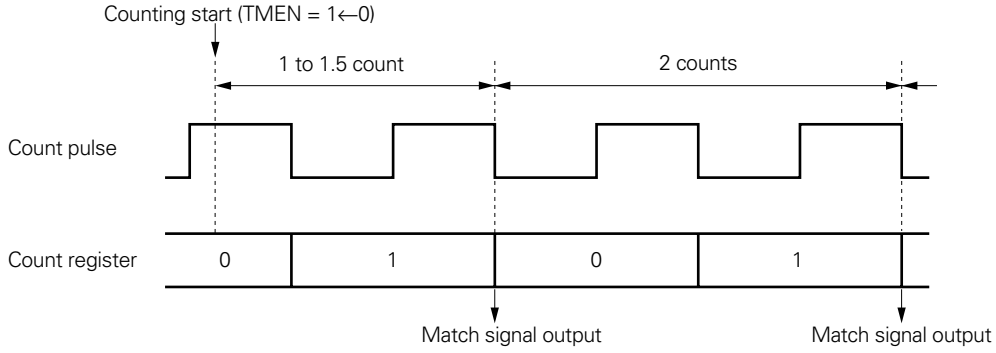
The count register of the 8-bit timer counter is cleared to zero by setting (to 1) the TMRES flag; however, the scaler for generating the count pulse from the system clock is not reset. When the TMEN flag is set (to 1) to start the counting from the count halt status, the timing of the first count varies as follows depending on whether the count pulse is started from the low level or from the high level.

When started from the high level: the next rising edge is the first count
When started from the low level: the count starting point is the first count

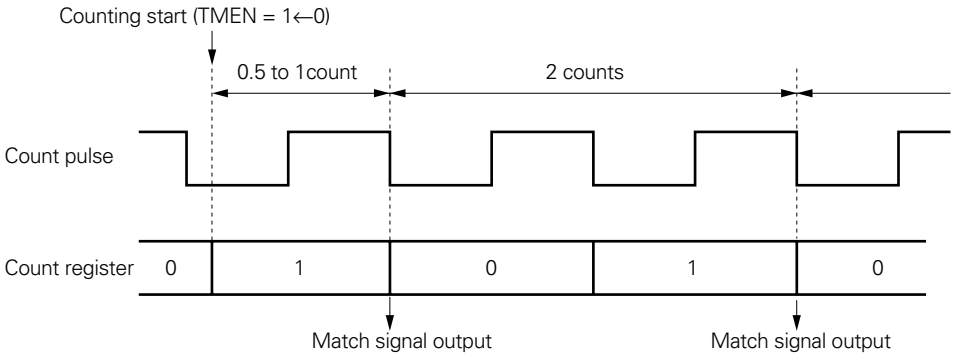
Therefore, only the first count after the counting is started generates an error of –0.5 to –1.5 counts during the time until the identity signal is issued. An example of counting when 1 is set for the modulo register is shown below.

Figure 13-5. Error in Starting Counting from the Count Halt State

(a) When the count pulse is stated from the high level (error: –0.5 to –1 count)



(b) When the count pulse is started from the low level (error: –1 to –1.5 counts)



In the example above, the identity signal is generated every 2 counts; however, only for the first count, the identity signal is issued for 1.5 counts maximum and for 0.5 count minimum (error: –0.5 to –1.5 counts). As the timer is in use even for generation of the oscillation stability wait time, the error margin above occurs even to this oscillation stability wait time.

13.1.7 Reading Count Register Values

(1) Reading Counter values

The counter values of count register are read at the same time via DBF (data buffer) using the GET instruction. The count register values of timer are assigned to peripheral address 02H.

Count register values of timer can be read into DBF by using the GET instruction. During execution of the GET instruction, timer count register stops count operation and a count value is retained. When a count pulse enters the timer in use during execution of the GET instruction, the count is retained.

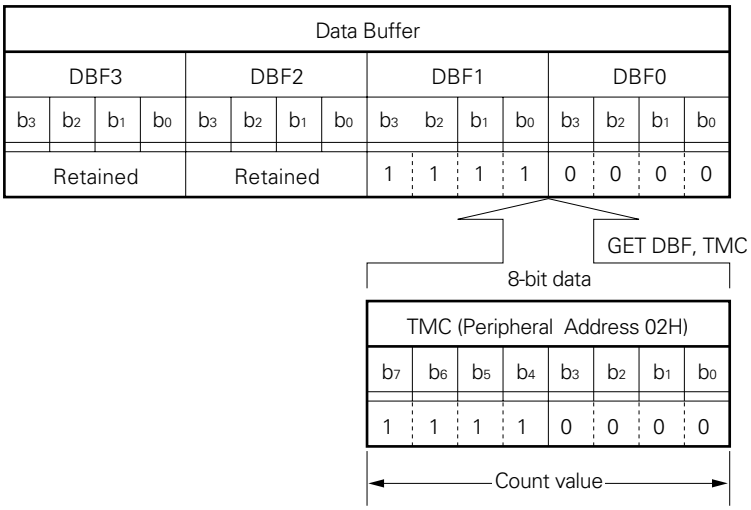
After execution of the GET instruction, the count register increments by one and continues counting.

The scheme prevents miscounting as long as two or more count pulses are not input in one instruction cycle, even when the GET instruction is executed during timer operation.

Figure 13-6. Reading 8-Bit Counter Count Values

The timer counter value is F0H.

GET DBF, TMC; Example of using reserved words DBF and TMC



(2) Program example

- **Measuring pulse width input from INT pin (system clock: fx=8 MHz)**
  The following is an example of measuring generation interval of external interrupt from INT pin by using timer. At this time the pulse width from INT pin should be within the count-up time of timer.

(Program example)

```
        CNTTMH    MEM     0.30H     ; Symbol definition of save area of count value
        CNTTML    MEM     0.31H     ;
                  ORG     00H       ; Specifies vector address of interrupt
                  BR      MAIN      ;
                  ORG     03H       ;
                  BR      INTJOB    ;
                  :
                  :
                  :
                  :
INITIAL:
        INITFLG   IP, NOT IPTM, NOT IPSIO
                                    ; Prohibits interrupts other than external interrupt
```

```
        INITFLG   NOT IEGMD1, IEGMD0
                                    ; Sets input from INT pin to falling edge
        CLR1      IRQ               ; Clears interrupt request signal from INT pin
        INITFLG   TMEM, TMRES, NOT TMCK1, TMCK0
                                    ; Sets source clock of timer to "fx/32"
                                    ; Clears timer count register and IRQTM,
                                    ; and starts timer
        EI

LOOP:
        BR        LOOP
        BR        TMRESTART

INTJOB:                             ; Vector interrupt becomes interrupt prohibit
        GET       DBF, TMC          ; state automatically immediately after accepting
                                    ; interrupt
        MOV       RPH, #.DM. (CNTTMH SHR 7) AND 0EH
                                    ; Sets general register pointer by using
                                    ; symbol-defined "CNTTMH" and "CNTTML"
        AND       RPL, #0001B       ;
        OR        RPL, # .DM. (CNTTMH SHR 3) AND 0EH
                                    ; At this time, BCD flag retains the previous state

        LD        CNTTMH, DBF1      ; Stores count value to count save area
        LD        CNTTML, DBF0      ;
        EI                          ; Makes interrupt permit state when executing
                                    ; main processing program
        RETI                        ; Returns to main processing program
```

13.1.8 Timer Output

The P0D3/$\overline{\text{TMOUT}}$ pin functions as a timer match signal output pin when the TMOSEL flag is set to 1. The P0DBIO3 value has nothing to do with this setting.

Timer contains a match signal output flip-flop. It reverses the output each time the comparator outputs a match signal. When the TMOSEL flag is set to 1, the contents of this flip-flop are output to the P0D3/$\overline{\text{TMOUT}}$ pin.

The P0D3/$\overline{\text{TMOUT}}$ 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 output flip-flop starts operating when TMEN is set to 1. To make the flip-flop start output beginning at an initial value, set 1 in TMRES and reset the flip-flop.

Remark The μPD17P132 and 17P133 have no mask option resistor.

Figure 13-7. Timer Output Control Mode Register

RF: 12H

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

Read=R, write=W

TMOSEL	Timer Output Control
0	P0D3/$\overline{\text{TMOUT}}$ pin functions as port.
1	P0D3/$\overline{\text{TMOUT}}$ pin functions as timer match signal output.

13.1.9 Timer Resolution and Maximum Setting Time

Table 13-1 shows the timer resolution in each source clock and maximum setting time.

Table 13-1. Timer Resolution and Maximum Setting Time

System Clock	Mode Register		Timer	
	TMCK1	TMCK0	Resolution	Maximum Setting Time
At 8 MHz **Note1**	0	0	32 μs	8.192 ms
	0	1	4 μs	1.024 ms
	1	0	256 μs	65.536 ms
	1	1	INT pin **Note 2**	
At 4.19 MHz **Note1**	0	0	approx. 61.1 μs	approx. 15.6 ms
	0	1	approx. 7.64 μs	approx. 1.9 ms
	1	0	approx. 489 μs	approx. 125 ms
	1	1	INT pin **Note 2**	
At 2 MHz	0	0	128 μs	32.768 ms
	0	1	16 μs	4.096 ms
	1	0	1.024 ms	262.144 ms
	1	1	INT pin **Note 2**	
At 500 kHz	0	0	512 μs	131.072 ms
	0	1	64 μs	16.384 ms
	1	0	4.096 ms	1.048576 s
	1	1	INT pin **Note 2**	

Notes 1. The guaranteed frequency range of oscillation for the μPD17120/17132/17P132 is f_{CC}=400 kHz to 2.4 MHz.

2. High/low level width of INT pin is 10 μs (MIN.) when V_{DD}=4.5 to 5.5 V, and 50 μs (MIN.) when V_{DD}=2.7 to 5.5 V. Refer to Data Sheet for detailed information.

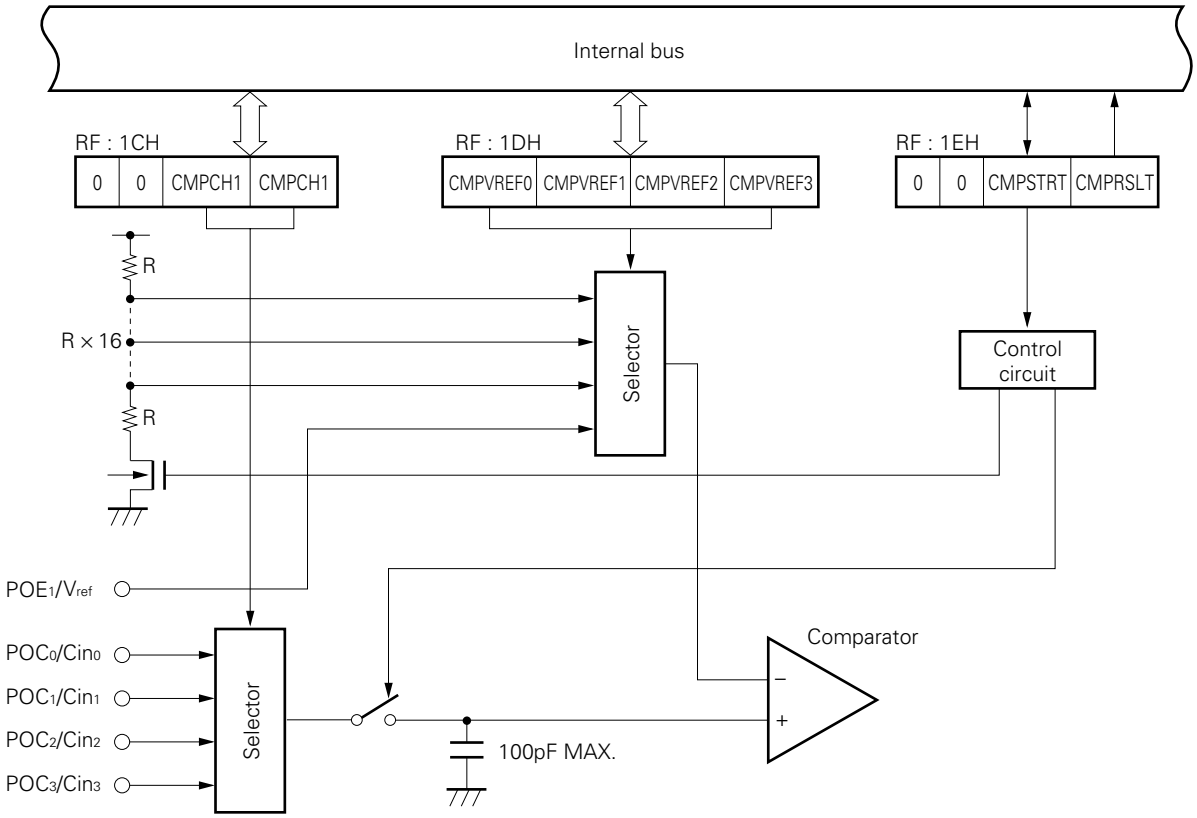
13.2 COMPARATOR (μPD17132, 17133, 17P132, AND 17P133 ONLY)

The comparator of the μPD17132, 17133, 17P132, and 17P133 compares the analog input (Cin_0 to Cin_3) and reference voltage (external: 1 type, internal: 15 types) and stores the comparison result to CMPRSLT (RF: 1EH, bit 0).

The comparator can also be used as a 4-bit A/D converter by software using 15 types of internal reference voltage.

13.2.1 Configuration of Comparator

Figure 13-8. Configuration of Comparator



Remark The sampling time of an analog input is as follows:

μPD17132, 17P132: 8/f_{CC} (4 μs, at 2 MHz)

μPD17133, 17P133: 28/f_X (3.5 μs, at 8 MHz)

13.2.2 Functions of Comparator

The comparator has a 4-channel analog input.

Concerning the pins used as analog input of the comparator, set 1 to P0CnIDI (n=0 to 3) at initial setting of the program (refer to **CHAPTER 12 PORTS**).

One of analog inputs (Cin_0 to Cin_3) can be selected by the comparator input channel selection flag (CMPCH1, CMPCH0 RF: 1CH, bits 1 and 0). One of the 16 types of reference voltages (external: 1, internal: 15) can be selected by the comparator reference voltage selection flag (CMPVREF0 to CMPVREF3 RF: 1DH).

After setting the comparator start flag to 1 (CMPSTRT RF: 1EH, bit 1), comparison takes 2 instruction execution cycles in the μPD17132 and 17P132, and 6 cycles in the μPD17133 and 17P133.

A comparison result is stored to the comparator comparison result flag (CMPRSLT RF: 1EH, bit 0).

Whether the comparator is operating or comparison is completed can be known by reading comparator start flag.

CMPSTRT=1... Comparator is operating (during analog voltage comparison)
CMPSTRT=0... Comparator is stopped (comparison is completed)

When comparing comparator analog voltage (CMPSTRT=1), manipulating comparator input channel selection flag (CMPCH1 CMPCH0) or comparator reference voltage selection flag (CMPVREF0 to CMPVREF3) is ignored and the data in these registers remain unchanged. Therefore, changing comparator operation modes are disabled.

CMPSTRT is cleared only when the voltage comparison operation of comparator is completed or when STOP instruction is executed.

Caution When using the standby function, be sure to wait for the comparator operation to stop before executing a standby instruction (HALT/STOP). If a STOP or HALT instruction is executed while the comparator is in operation, the comparator operation is halted. If the internal reference voltage has been selected at this time, current will keep flowing into the internal resistor ladder, thus resulting in increased current consumption during standby mode.

Comparator input channel selection register, reference voltage selection register, and comparator operation control register are shown in the Figures 13-9, 13-10, and 13-11, respectively.

Figure 13-9. Comparator Input Channel Selection Register

RF: 1CH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	CMPCH1	CMPCH0
Read/write	R/W			
Initial value when reset	0	0	0	0

Read=R, write=W

CMPCH1	CMPCH0	Comparator Input Channel Selection
0	0	Select Cin_0
0	1	Select Cin_1
1	0	Select Cin_2
1	1	Select Cin_3

Figure 13-10. Reference Voltage Selection Register

RF: 1DH

	Bit 3	Bit 2	Bit 1	Bit 0
	CMPVREF3	CMPVREF2	CMPVREF1	CMPVREF0
Read/write	R/W			
Initial value when reset	1	0	0	0

Read=R, write=W

CMPVREF3	CMPVREF2	CMPVREF1	CMPVREF0	Selected Reference Voltage
0	0	0	0	Voltage applied to V_{ref} pin
0	0	0	1	1/16 V_{DD}
0	0	1	0	2/16 V_{DD} (1/8 V_{DD})
0	0	1	1	3/16 V_{DD}
0	1	0	0	4/16 V_{DD} (1/4 V_{DD})
0	1	0	1	5/16 V_{DD}
0	1	1	0	6/16 V_{DD} (3/8 V_{DD})
0	1	1	1	7/16 V_{DD}
1	0	0	0	8/16 V_{DD} (1/2 V_{DD})
1	0	0	1	9/16 V_{DD}
1	0	1	0	10/16 V_{DD} (5/8 V_{DD})
1	0	1	1	11/16 V_{DD}
1	1	0	0	12/16 V_{DD} (3/4 V_{DD})
1	1	0	1	13/16 V_{DD}
1	1	1	0	14/16 V_{DD} (7/8 V_{DD})
1	1	1	1	15/16 V_{DD}

Caution When CMPSTRT =1, a write instruction to this register is ignored (the data in the register remains unchanged).

Figure 13-11. Comparator Operation Control Register

RF: 1EH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	CMPSTRT	CMPRSLT
Read/write	R/W			R
Initial value when reset	0	0	0	1

Read = R, write = W

CMPRSLT	Comparator Operation Comparison Result
0	When the voltage from analog input (Cin_0 to Cin_3) is lower than the external/internal reference voltage
1	When the voltage from analog input (Cin_0 to Cin_3) is higher than the external/internal reference voltage

CMPSTRT	Comparator Operation Check (at Reading)
0	During comparator operation is stopped or comparator voltage comparison operation is completed
1	During comparator is operating

Remark CMPSTRT is cleared to 0 only when the comparator voltage comparison operation is completed or STOP instruction is executed.

CMPSTRT	Comparator Operation Start (at Writing)
0	Invalid
1	Start comparator operation

13.3 SERIAL INTERFACE (SIO)

The serial interfaces of the μPD17120 subseries consists of a shift register (SIOSFR, 8 bits), serial mode register, and serial clock counter. It is used for serial data input/output.

13.3.1 Functions of the Serial Interface

This serial interface provides three signal lines: serial clock input pin ($\overline{SCK}$), 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 can be selected. If an internal clock is selected as a serial clock, it is automatically output to the $P0D_0/\overline{SCK}$ pin.

Table 13-2. List of Serial Clock

SIOCK1	SIOCK0	Serial clock to be selected
0	0	External clock from the $\overline{SCK}$ pin
0	1	$f_X/16$
1	0	$f_X/128$
1	1	$f_X/1024$

(2) Transmission operation

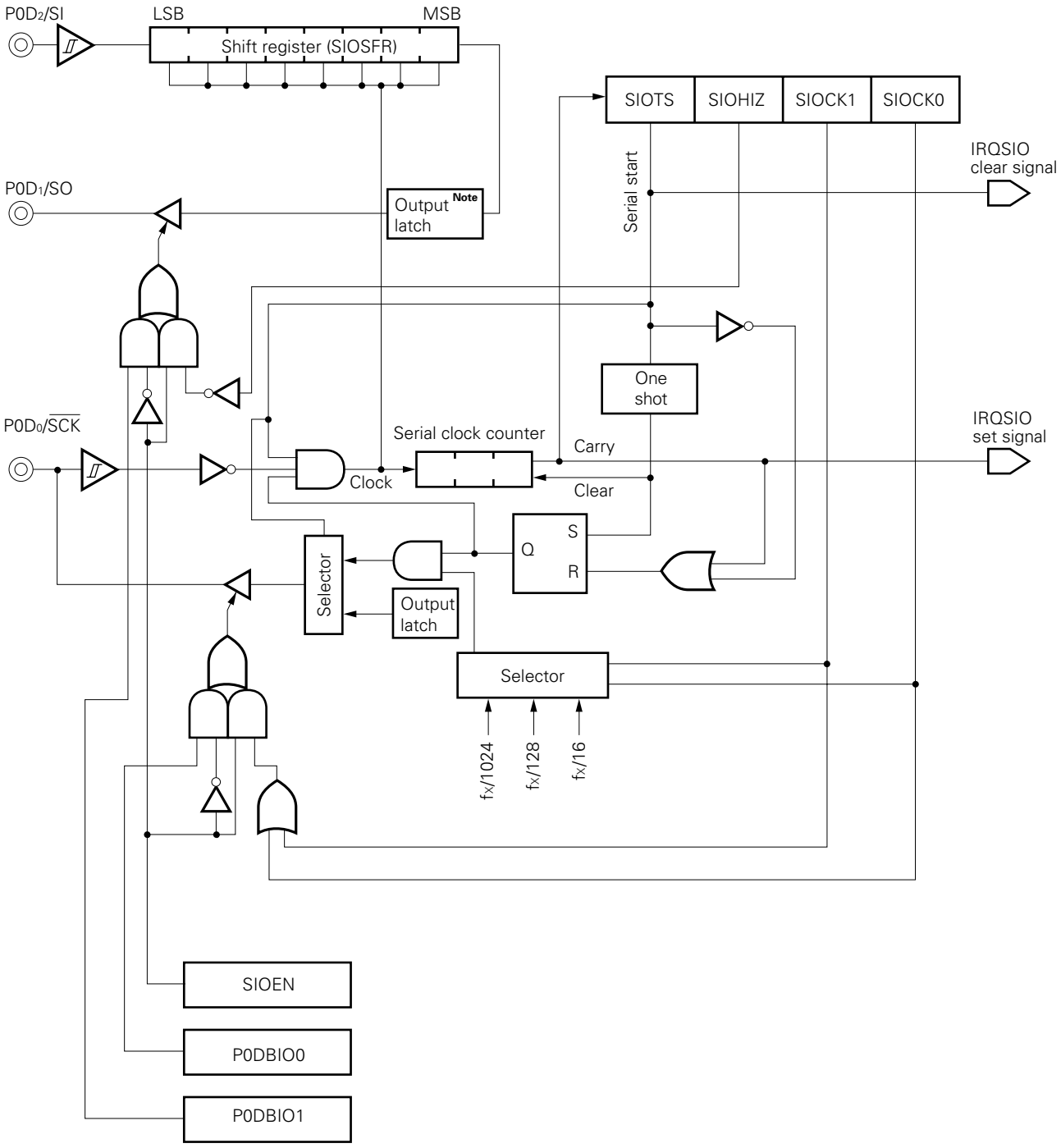
By setting (to 1) SIOEN, each pin of port 0D ($P0D_0/\overline{SCK}$, $P0D_1$/SO, $P0D_2$/SI) functions as a pin for serial interfacing. At this time, if SIOTS is set (to 1), the operation is started synchronously with the falling edge of the serial clock. Also, setting SIOTS will result in automatically clearing IRQSIO.

The transfer is started from the most significant bit of the shift register synchronously with the falling edge of the serial clock. And, the information on the SI pin is stored in the shift register from the most significant bit, synchronously with the rising edge of the clock.

If the 8-bit data transfer 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 13-12. Block Diagram of the Serial Interface



Note The output latch of the shift register is common with the output latch of P0D1. Therefore, if an output instruction is executed for P0D1, the output latch state of the shift register is also changed.

13.3.2 3-wire Serial Interface Operation Modes

Two modes can be used for the serial interface. If the serial interface function is selected, the P0D2/SI pin always takes in data in synchronization with the serial clock.

- 8-bit send/receive mode (concurrent send/receive)
- 8-bit receive mode (SO pin: high-impedance state)

Table 13-3. Serial Interface's Operation Mode

SIOEN	SIOHIZ	P0D2/SI pin	P0D1/SO pin	Serial Interface Operation Mode
1	0	SI	SO	8-bit send/receive mode
1	1	SI	P0D1 (input)	8-bit receive mode
0	×	P0D2 (input/output)	P0D1 (input/output)	General-purpose port mode

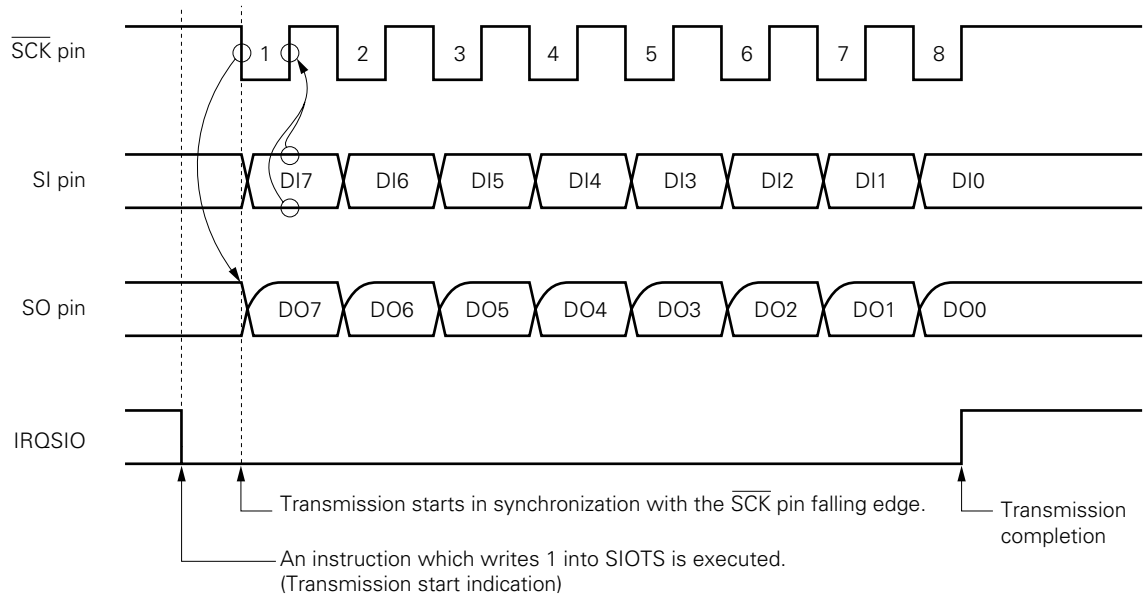
×: Don't care

(1) 8-bit transmission and reception mode (simultaneous transmission and reception)

Serial data input/output is controlled by a serial clock. The MSB of the shift register is output from the SO line with a falling edge of the serial clock ($\overline{SCK}$). The contents of the shift register is shifted one bit and at the same time, data on the SI line is loaded into the LSB of the shift register.

The serial clock counter counts serial clock pulses. Every time it counts eight clocks, the interrupt request flag is set (IRQSIO ← 1).

Figure 13-13. Timing of 8-Bit Transmission and Reception Mode (Simultaneous Transmission Reception)



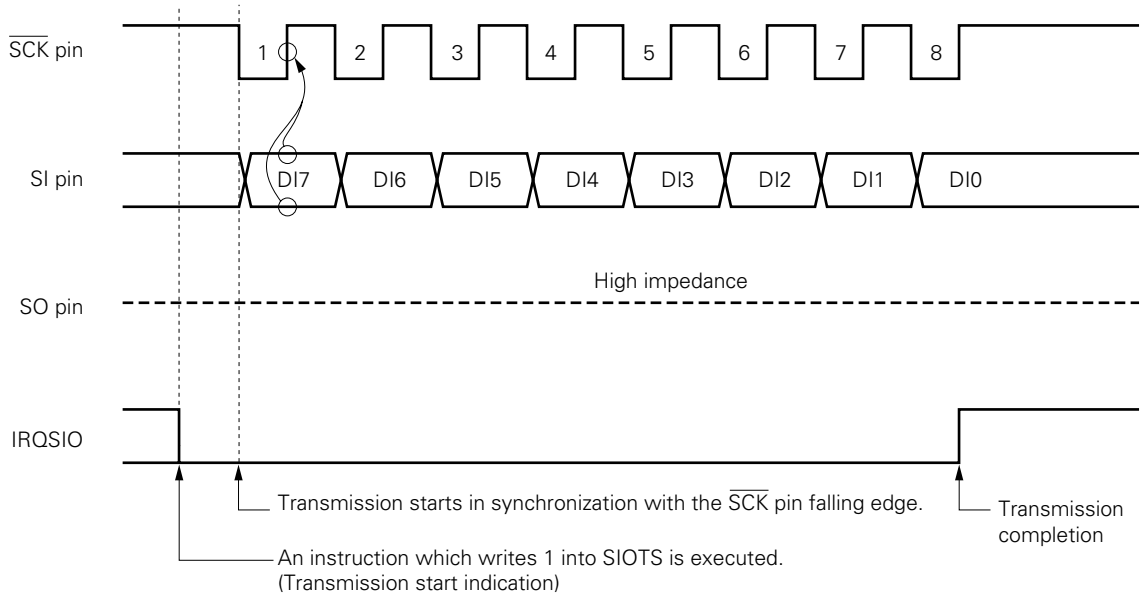
Remark DIn : Serial input data
DOn : Serial output data

(2) 8-bit receive mode (SO pin: high impedance state)

When SIOHIZ=1, the P0D1/SO pin is placed in the high-impedance state. At this time, if "1" is written into SIOTS to start supply of the serial clock, the serial interface operates only the receiving function.

Because the P0D1/SO pin is placed in the high-impedance state, it can be used as an input port (P0D1).

Figure 13-14. Timing of the 8-Bit Reception Mode



Remark DIn: Serial input data

(3) Operation stop mode

If the value in SIOTS (RF: address 1AH, 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 13-15. Serial Interface Control Register (1/2)

RF: 1AH

	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 ($\overline{SCK}$ 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	Confirmation of Shift Register Operation Status (at Reading)
0	The shift register is in the stop status.
1	The shift register is operating.

SIOTS	Start and Stop of Serial Transmission (at Writing)
0	Forced termination of the shift register. Disables intermediate restart.
1	Start of shift register operation • At internal clock selection Starts operating internal devided signal of a system clock (f_X) as a serial clock • At external clock selection Starts operation in synchronization with an $\overline{SCK}$ pin falling edge.

Remark SIOTS is automatically cleared to 0 when serial transmission is completed.

Figure 13-15. Serial Interface Control Register (2/2)

RF: 0AH

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

Read = R, write = W

SIOEN	Serial Interface Enable
0	The pins of port 0D ($P0D_0/\overline{SCK}$, $P0D_1$/SO, $P0D_2$/SI) function as ports.
1	The pins of port 0D ($P0D_0/\overline{SCK}$, $P0D_1$/SO, $P0D_2$/SI) function as the serial interface.

Remark Refer to **CHAPTER 12 PORTS**

13.3.3 Setting Values in the Shift Register

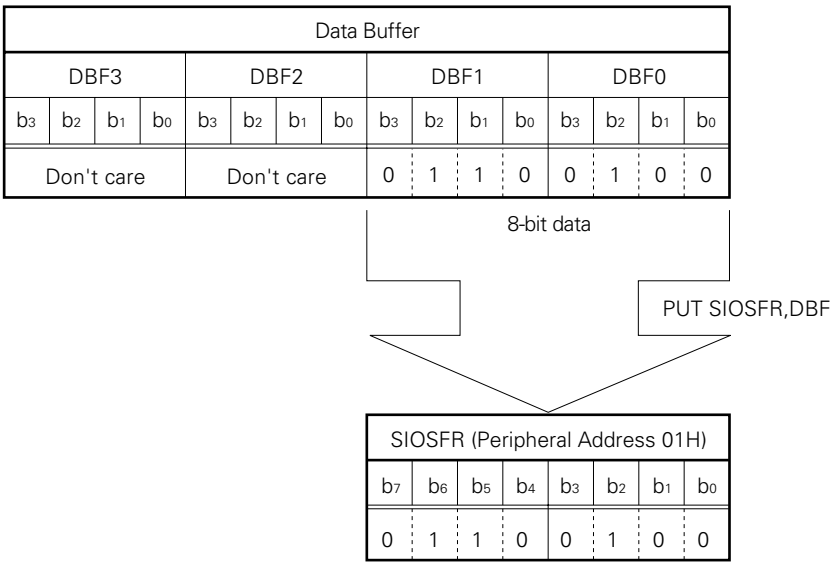
Values are set in the shift register via the data buffer (DBF) using the PUT instruction.

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 13-16. 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.
```

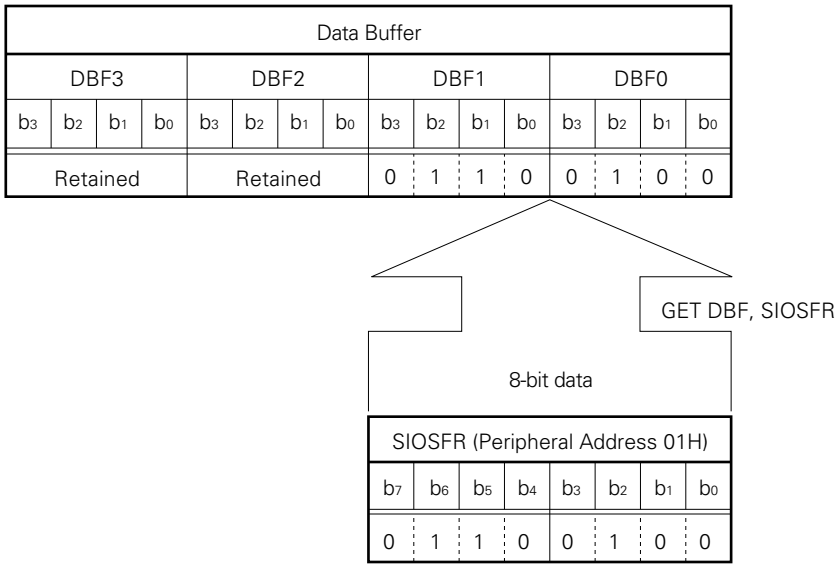


13.3.4 Reading Values from the Shift Register

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 13-17. Reading a Value from the Shift Register

GET DBF, SIOSFR; Example of using reserved words DBF and SIOSFR



13.3.5 Program Example of Serial Interface

(1) Program example of data transmission/reception by 8-bit transmission/reception mode (synchronous transmission/reception)

This program executes data transmission/reception synchronizing with fx/128. Judgment of finishing serial data transmission/reception is executed by checking interrupt request flag.

Example

```
MAIN:
        ⋮
    CALL    SIOJOB
        ⋮
    BR      MAIN
SIOJOB:
    DI                                  ; Prohibits interrupt in SIOJOB
    CLR1    IRQSIO                      ; Clears interrupt request flag of SIO
    SET1    SIOEN                       ; Enables SIO
    MOV     DBF0, #SIODATL              ; Sets transmitted data
    MOV     DBF1, #SIODATH
    PUT     SIOSFR, DBF
    INITFLG SIOTS, NOT SIOHIZ, SIOCK1, NOT SIOCK0
                                        ; Sets serial clock to "fx/128", starts shift
                                        ; register operation, and outputs serial data
LOOP:
    SKT1    IRQSIO                      ; Transmission/reception finish judgment
    BR      LOOP                        ; Waits finishing transmission/reception
    GET     DBF, SIOSFR                 ; Reads reception data
    EI                                  ; Permits interrupt and returns to main processing
    RET                                 ;
```

(2) Program example of data reception by 8-bit reception mode

This program executes data reception synchronizing with external clock, and reads reception data by using interrupt processing.

Example

```
SIODATH   MEM     0.50H
SIODATL   MEM     0.51H
          ORG     0H
          BR      SIO_INIT
          ORG     01H
          BR      SIOJOB

SIO_INIT:
          MOV     SIODATH, #0H
          MOV     SIODATL, #0H
          CLR1    IRQSIO              ; Clears interrupt request flag of SIO
          SET1    SIOEN               ; Enables SIO
          INITFLG SIOTS, SIOHIZ, NOT SIOCK1, NOT SIOCK0
                                      ; Sets serial clock to external clock, starts receiving
                                      ; serial data, and sets P0D1/SO pins to input port
                                      ; (output high impedance)
          EI                          ; Permits all interrupts
                                      ; Main processing
MAIN:
          CALL    ××JOB
          CALL    ××JOB
            ⋮
          BR      MAIN
SIOJOB:
          GET     DBF, SIOSFR         ; Reads reception data
          MOV     RPH, #0000B         ; Sets general register to low address 5H of BANK0
          MOV     RPL, #1010B         ; BCD ← 0
          LD      SIODATH, DBF1       ; Stores reception data on RAM
          LD      SIODATL, DBF0       ;
          EI
          RETI
```

CHAPTER 14 INTERRUPT FUNCTIONS

The μPD17120 subseries has two internal interrupt functions and one external interrupt function. It can be used in various applications.

The interrupt control circuit of this product 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) Standby mode (STOP, HALT) can be released by an interrupt request. (Release source can be selected by the interrupt enable flag.)

Cautions 1. In interrupt handling, the BCD, CMP, CY, Z, and IXE flags are saved in the stack automatically by the hardware for one level of multiple interrupts. The DBF and WR are not saved by the hardware when peripheral hardware such as the timers or serial interface 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.

2. Because the interrupt stack is only one level, multi-interrupt by hardware cannot be executed. If the interrupt over one level is accepted, the first data is lost.

14.1 INTERRUPT SOURCES AND VECTOR ADDRESS

For every interrupt in the μPD17120 subseries, 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 the retained 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	0003H	IRQ RF: 3FH, bit 0	IP RF: 2FH, bit 0	IEGMD0, 1 RF: 1FH	External	Rising edge or falling Edge can be selected.
Timer	2	0002H	IRQTM RF: 3EH, bit 0	IPTM RF: 2FH, bit 1	–	Internal	
SIO	3	0001H	IRQSIO RF: 3DH, bit 0	IPSIO RF: 2FH, bit 2	–	Internal	

14.2 HARDWARE COMPONENTS OF THE INTERRUPT CONTROL CIRCUIT

The flags of the interrupt control circuit are explained below.

14.2.1 Interrupt Request Flag (IRQ×××) and the Interrupt Enable Flag (IP×××)

The interrupt request flag (IRQ×××) 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 (IP×××) is provided for each interrupt request flag. If the flag is 1, an interrupt is enabled. If it is 0, the interrupt is disabled.

14.2.2 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, INTE for enabling interrupt reception is set. 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 Enagle Flag
IRQ	Set by edge detection of an INT pin input signal. A detection edge is selected by IEGMD0 or IEGMD1.	IP
IRQTM	Set by a match signal from timer.	IPTM
IRQSIO	Set by a serial data transmission end signal from the serial interface.	IPSIO

Figure 14-1. Interrupt Control Register (1/4)

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	State of INT Pin
0	INT pin noise elimination circuit sets logical status to 0 during PEEK instruction execution.
1	INT pin noise elimination circuit sets logical status to 1 during PEEK instruction execution.

Note Since the INT flags are not latched, they change all the time in response to the logical state of the pin, However, once the IRQ flag is set, it stays set until an interrupt is accepted. The POKE instruction to address 0FH is invalid.

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 (2/4)

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

IRQ	INT Pin Interrupt Request (at Reading)
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.

IRQ	INT Pin Interrupt Request (at Writing)
0	An interrupt request from the INT pin is forcibly released.
1	An interrupt request from the INT pin is forced to occur.

RF: 3FH

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

Read=R, write=W

IRQTM	TM Interrupt Request (at Reading)
0	No interrupt request has been issued from timer or a timer interrupt is being handled.
1	The contents of the timer count register matches that of the timer modulo register and an interrupt request occurs. Or a timer interrupt request is being held.

IRQTM	TM Interrupt Request (at Writing)
0	An interrupt request from timer is forcibly released.
1	An interrupt request from timer is forced to occur.

Remark If TMRES is set to 1, IRQTM is cleared to 0. IRQTM is cleared to 0 immediately after STOP instruction is executed.

Figure 14-1. Interrupt Control Register (3/4)

RF: 3DH

	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

IRQSIO	SIO Interrupt Request (at Reding)
0	No interrupt request has been issued from the serial interface or a serial interface interrupt is being handled.
1	Serial interrupt transmission is completed and an interrupt request occurs. Or, a serial interface intrrupt request is being held.

IRQSIO	SIO Interrupt Request (at Writing)
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 (4/4)

RF: 2FH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	IPSIO	IPTM	IP
Read/write	R/W			
Initial value when reset	0	0	0	0

Read=R, write=W

IP	INT Pin Interrupt Enable
0	Disables an interrupt from the INT pin. Holds interrupt handling when the IRQ flag is set to 1.
1	Enables an interrupt from the INT pin. Executes the EI instruction. If the IRQ flag is set to 1, executes interrupt handling.

IPTM	TM Interrupt Enable
0	Disables an interrupt from timer. Holds an interrupt if the IRQTM flag is set to 1.
1	Enables an interrupt from timer. Executed the EI instruction. If the IRQTM flag is set to 1, executes interrupt handling.

IPSIO	SIO Interrupt Enable
0	Disables an interrupt from serial interface. Holds an interrupt if the IRQSIO flag is set to 1.
1	Enables an interrupt from serial interface. Executed the EI instruction. If the IRQSIO flag is set to 1, executes interrupt handling.

14.3 INTERRUPT SEQUENCE

14.3.1 Acceptance of Interrupts

The moment an interrupt is accepted, the instruction cycle of the instruction which has been executed is terminated, and the interrupt operation is started thus altering the flow of the program to the vector address. However, if the interrupt occurs during execution of the MOVT instruction, the EI instruction or an instruction which has satisfied the skip condition, the processing of this interrupt is started after two instruction cycles are completed.

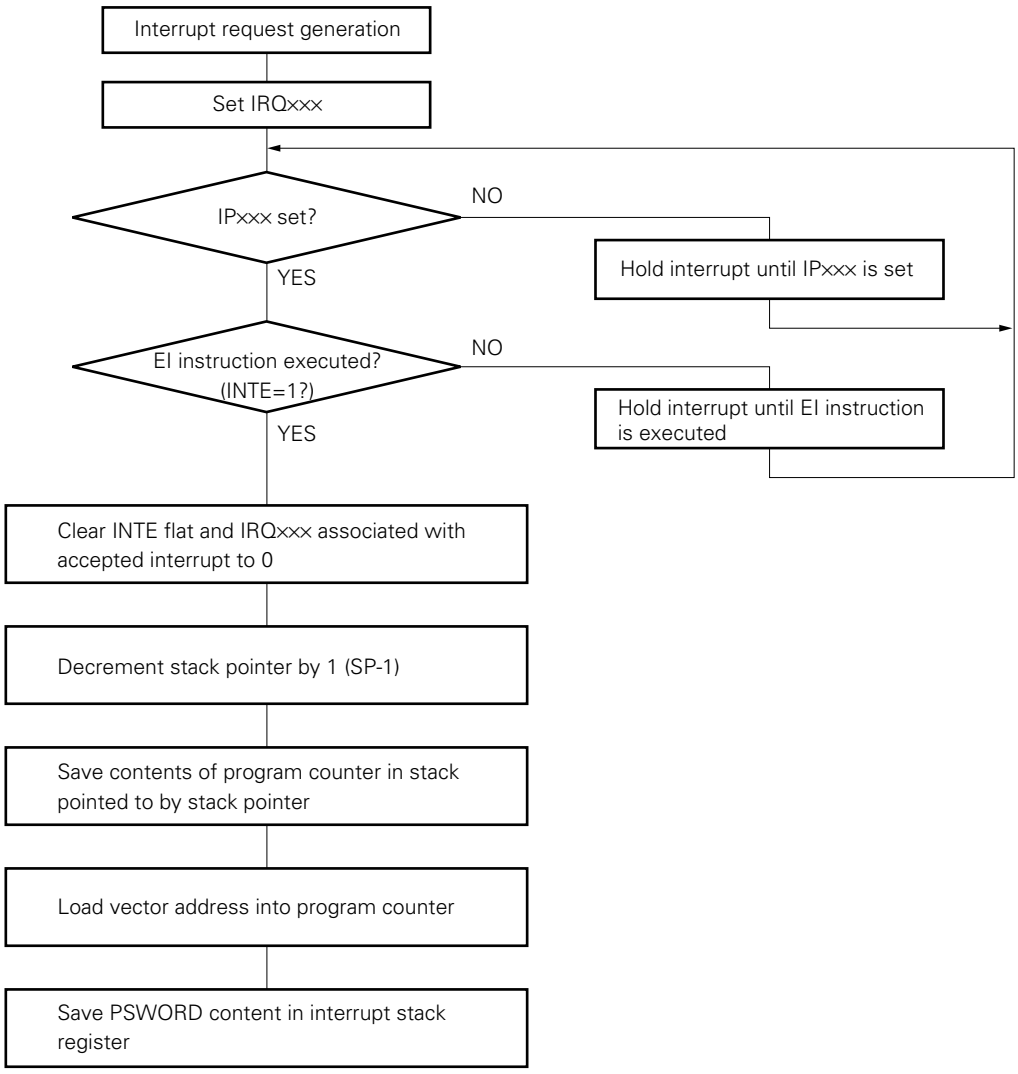
If interrupt operation is started, one level of the address stack register is consumed to store the return address of the program, and also a level of the interrupt stack register is used to save the PSWORD in the system register.

If multiple interrupts are enabled and occure simultaneously, the interrupts are processed in order of higher priority. In this case, an interrupt with a lower priority is put on hold until the interrupts with higher priority are processed.

For details of the priority levels, refer to **Table 14-1. Types of Interrupt Factors**.

Caution The PSWORD is automatically reset to 00000B after being saved in the interrupt stack register.

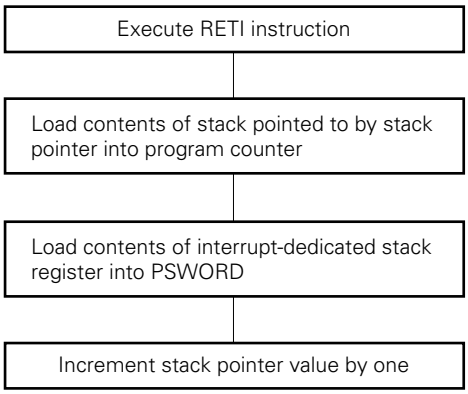
Figure 14-2. Interrupt Handling Procedure



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.

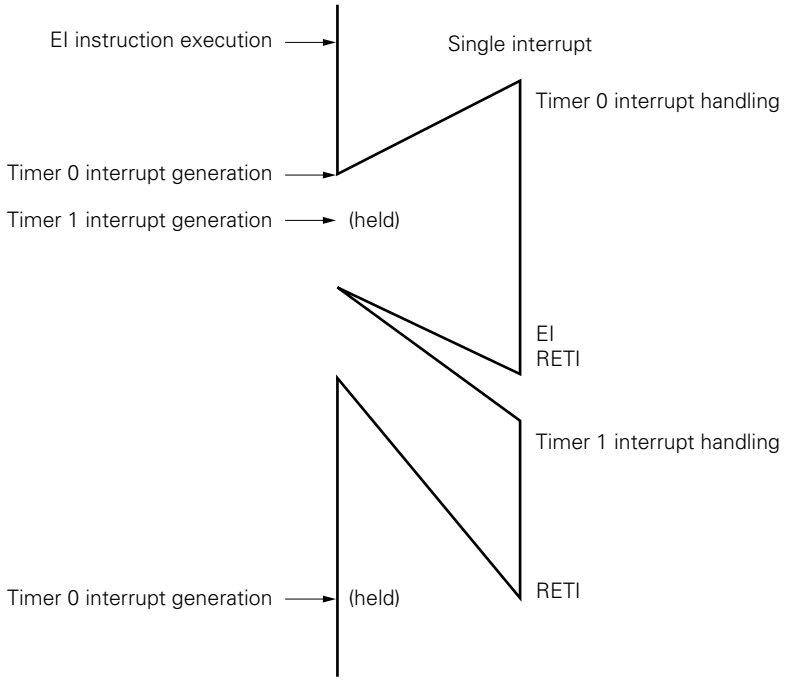
Figure 14-3. Return from Interrupt Handling



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



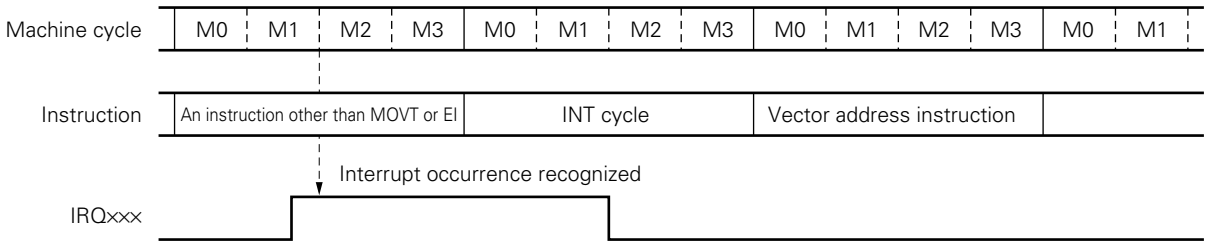
14.3.3 Interrupt Acceptance Timing

Figure 14-4 shows the interrupt acceptance timing chart. The μPD17120 subseries executes an instruction with 16 clocks, which is one instruction cycle. One instruction cycle is subdivided into M0-M3 in terms of 4 clocks as a unit.

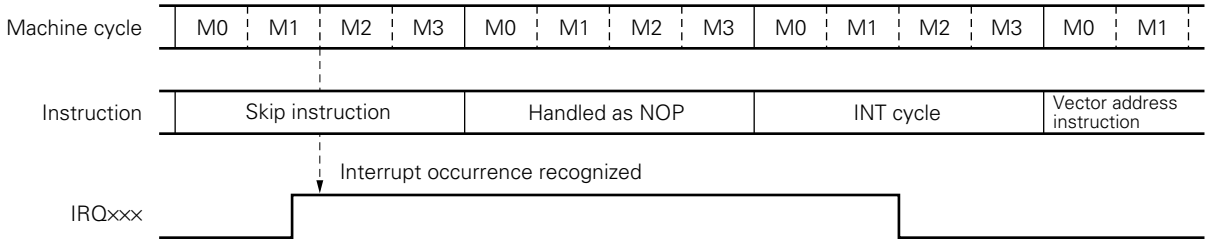
The timing of the program recognizing the interrupt occurrence coincides with the edge preceding the M2.

Figure 14-4. Interrupt Acceptance Timing Chart (when INTE=1 and IP×××=1) (1/3)

<1> When an interrupt has occurred before M2 of an instruction other than MOVT or EI



<2> When the skip condition for the skip instruction is materialized in <1>



<3> When an interrupt has occurred after M2 of an instruction other than MOVT or EI

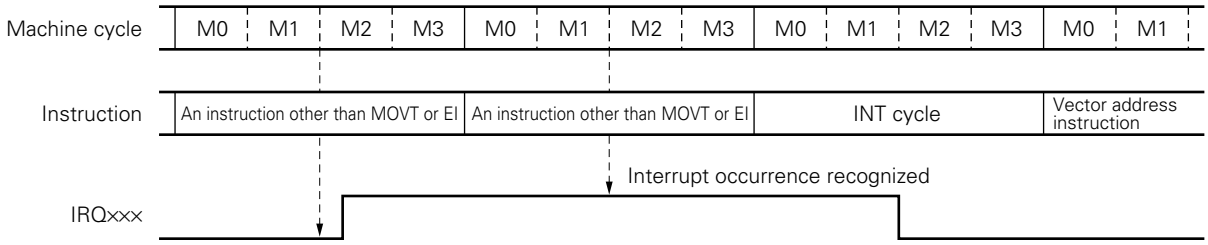
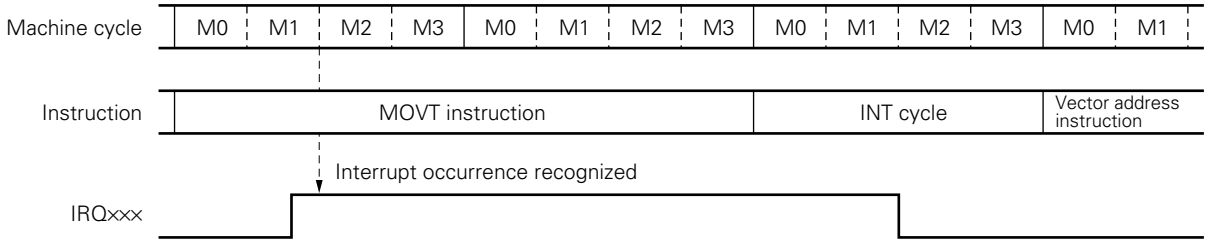
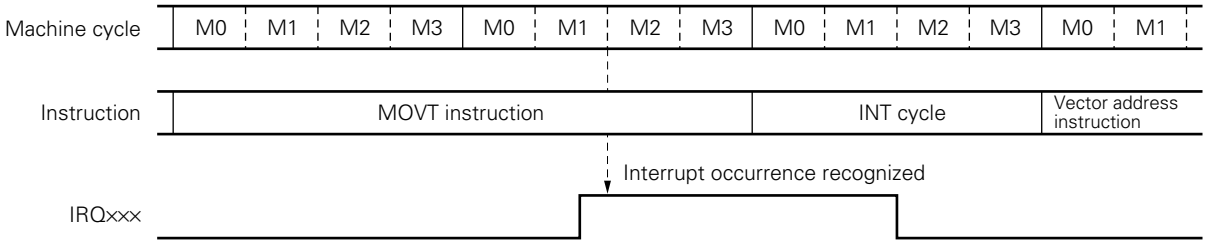


Figure 14-4. Interrupt Acceptance Timing Chart (when INTE=1, IP×××=1) (2/3)

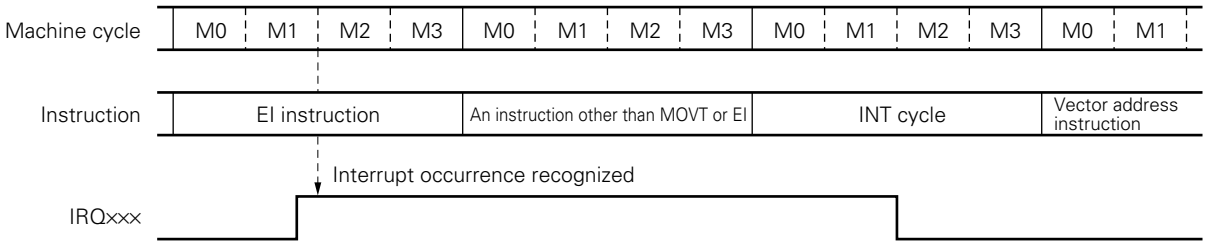
<4> When an interrupt has occurred before M2 of a MOVT instruction



<5> When an interrupt has occurred before M2' of a MOVT instruction



<6> When an interrupt has occurred before M2 of an EI instruction



<7> When an interrupt has occurred after M2 of an EI instruction

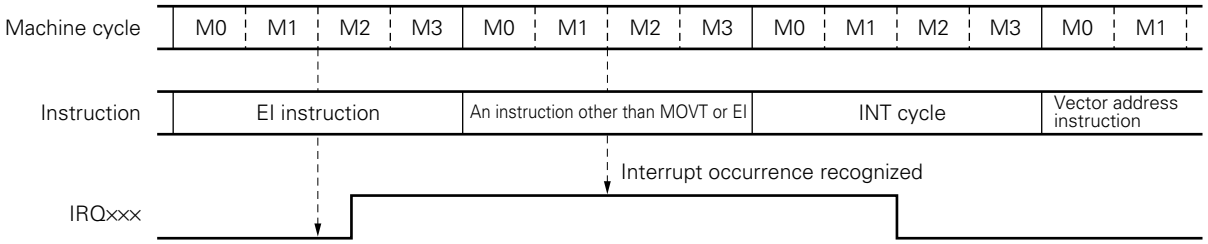
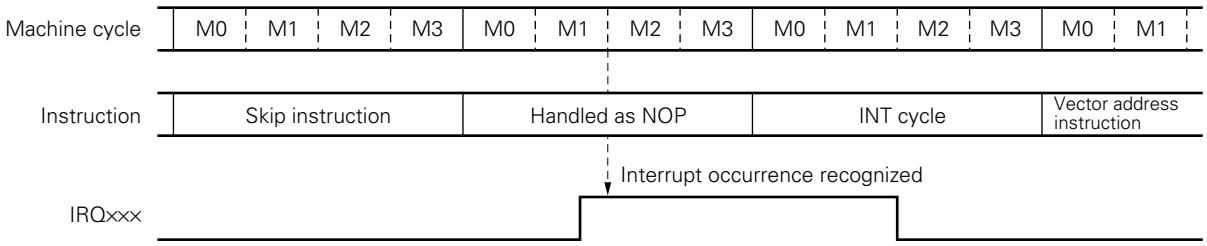


Figure 14-4. Interrupt Acceptance Timing Chart (INTE=1 and IP×××=1) (3/3)

<8> When an interrupt has occurred during skipping (NOP handling) by a skip instruction



Remarks 1. The INT cycle is for preparing interrupts. During this cycle, PC and PSWORD saving and IRQ clearing are performed.

2. For execution of the MOVT instruction, two instruction cycles are exceptionally required.

3. The EI instruction is considered to prevent multiple interrupts from occurring when returning from the interrupt operation.

14.4 PROGRAM EXAMPLE OF INTERRUPT

- **Program example of contermeasure for noise reduction of external interrupt (INT pin)**

  This example assumes the case of assigning INT pin for key input, etc.

  When taking into the microcontroller data in kind of switch such as key input processing, it takes some time for the level of input voltage to be stabilized after pushing the key or switch. Accordingly, the countermeasures for removing the noise generated by key, etc. should be executed by software.

  In the following program, after generating external interrupt, the signal from INT pin becomes effective after confirming that there is not change in the level of INT pin two times in every 100 μs.

Example

```
WAITCNT   MEM     0.00H            ; Counter of wait processing
KEYON     FLG     0.01H.3          ; If key ON is determined (even just once), KEYON=1
SECOND    FLG     0.01H.0          ; A flag describing key-checking for the second time.
          ORG     0H
          BR      JOB_INIT
          ORG     3H
          BR      INT_JOB
           ⋮
JOB_INIT:
          MOV     WAITCNT, #0      ; Clears RAM and the flag on RAM
          CLR2    KEYON, SECOND    ;
          INITFLG NOT IEGMD1, IEGMD0
                                   ; Rising edge is effective for the interrupt from INT pin
          CLR1    IRQ
          SET1    IP
          EI
           ⋮
MAIN:
          CALL    ××JOB            ; arbitrary processing
          CALL    ××JOB            ; arbitrary processing
           ⋮
          BR      MAIN
```

```
INT_JOB:
            NOP                 ; Loop which executes waiting for 100 μs at 8 MHz
            NOP                 ; 2 μs (1 instruction) × 5 instructions × 10 times
                                ; (count value at WAIT)
            ADD     WAITCNT, #01 ;
            SKE     WAITCNT, #0AH ;
            BR      INT_JOB     ;
            SKF1    INT         ; Check the level of INT pin
            BR      KEY_OFF     ; If INT pin is high level, interrupt is invalid, and returns
                                ; to main processing
            SKF1    SECOND      ; First wait?
            BR      WAIT_END    ; If it is the first time, wait again after setting SECOND.
                                ; In the case of the second time, finish wait processing
            SET1    SECOND      ;
            MOV     WAITCNT, #0
            BR      INT_JOB
WAIT_END:
            SET1    KEYON       ; Judges that there is key input
            BR      INT_JOB_END
KEY_OFF:
            CLR1    SECOND      ; SECOND←0
INT_JOB_END:
            MOV     WAITCNT, #0
            EI
            RETI
```

[MEMO]

CHAPTER 15 STANDBY FUNCTIONS

15.1 OUTLINE OF STANDBY FUNCTION

The μPD17120 subseries reduces current consumption by using a standby function. In standby mode, the series uses STOP mode or HALT mode depending on the application.

STOP mode is a mode that stops the system clock. In this mode, the CPU's current consumption is mostly limited to the leakage current. Therefore, this is useful for retaining the contents of the data memory without operating the CPU.

HALT mode is a mode that halts CPU operation because the clock supplying the CPU is stopped even when the system clock's oscillation continues. Although, compared with STOP mode, this mode does not reduce the current consumption, operation can start immediately after HALT is canceled because the system clock is oscillating. Also, in either the STOP mode or HALT mode, the status of items such as the data memory, registers, the output port's output latch, etc. immediately before being set to standby mode are retained (STOP 0000B excluded).

Therefore, before placing the system in standby mode, please set the port's status in a way that the current consumption of the whole system is reduced.

Table 15-1. States during Standby Mode

		STOP mode	HALT Mode
Instruction to set		STOP instruction	HALT instruction
Clock Oscillation Circuit		Oscillation stopped	Oscillation continued
Operation Statuses	CPU	• Operation stopped	
	RAM	• Immediately-preceding status retained	
	Port	• Immediately-preceding status retained[Note 1]	
	TM	• Operation stopped (The count value is reset to "0".) (The count up is also disabled.)	• Operable
	SIO	• Operable only when an external clock has been selected for the shift clock[Note 1]	• Operable
	Comparator[Note 2]	• Operation stopped[Note 1]	• Operation stopped (The result after resumption of the operation is "undefined".)
	INT	• Operable	

Notes 1. At the point where STOP 0000B has been executed, the pin's status is placed in input port mode even when the pin is used with its dual function.

2. Limited to μPD17132, 17133, 17P132, and 17P133.

Cautions 1. Be sure to place a NOP instruction immediately before the STOP instruction or the HALT instruction.

2. Both the interrupt request flag and the interrupt enable flag are set and are not placed in standby mode if their interruption is specified in the condition for canceling the standby mode.

15.2 HALT MODE

15.2.1 HALT Mode Setting

The system is placed in HALT mode by executing the HALT instruction. The HALT instruction's operands $b_3b_2b_1b_0$ are the conditions for canceling HALT mode.

Table 15-2. HALT Mode Cancellation Condition

Format: HALT $b_3b_2b_1b_0$B

Bit	Condition for Canceling HALT Mode[Note 1]
b_3	At 1, cancellation by IRQxxx is enabled.[Notes 2, 4]
b_2	"Fixed to 0"
b_1	At 1, forced cancellation by IRQTM is enabled.[Note 3, 4]
b_0	"Fixed to 0"

Notes
1. At HALT 0000B, only the resets ($\overline{\text{RESET}}$ input; power-ON/power-DOWN reset) are valid.
2. It is required that IPxxx=1.
3. Regardless of the PITM's state, HALT mode is canceled.
4. Even if the HALT instruction is executed when IRQxxx=1, the HALT instruction is ignored (handled as the NOP instruction) thus failing to place the system in HALT mode.

15.2.2 Start Address after HALT Mode is Canceled

The start address varies depending on the cancellation condition and the interrupt enable condition.

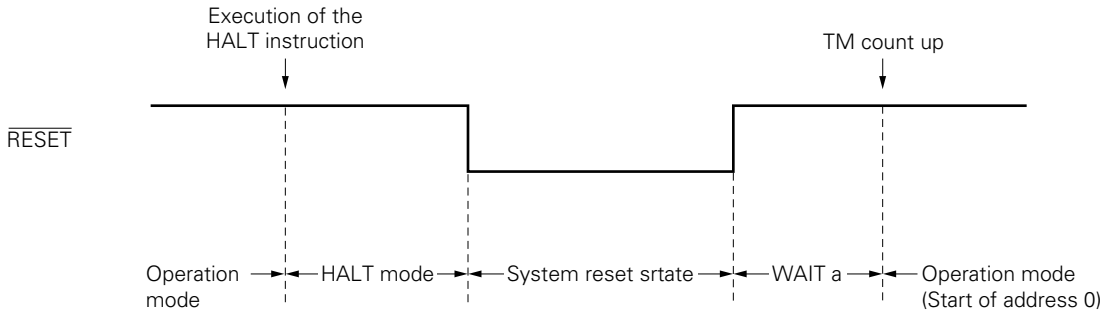
Table 15-3. Start Address After HALT Mode Cancellation

Cancellation Condition	Start Address After Cancellation
Reset[Note 1]	Address 0
IRQxx[Note 2]	If DI, the start address is the one following the HALT instruction
	If EI, the start address is the interrupt vector. (If more than one IRQxxx have been set, the start address is the interrupt vector with a higher priority.)

Notes
1. Valid resets include the $\overline{\text{RESET}}$ input and power-ON/power-DOWN resets.
2. Except for forced cancellation by IRQTM, it is required that IPxxx=1.

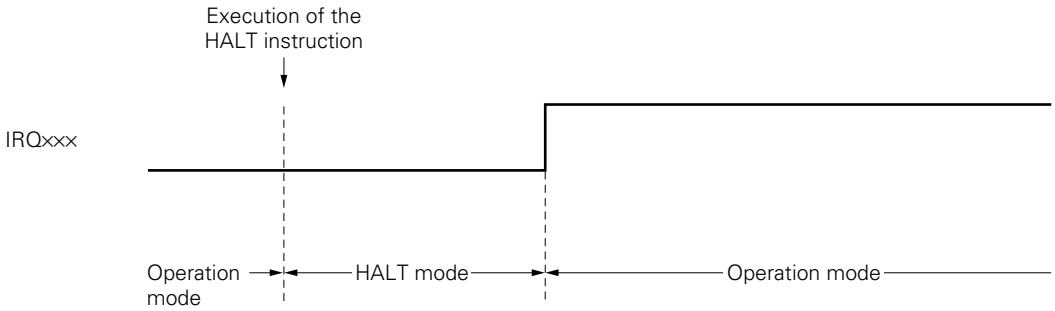
Figure 15-1. Cancellation of HALT Mode

(a) HALT cancellation by $\overline{\text{RESET}}$ input

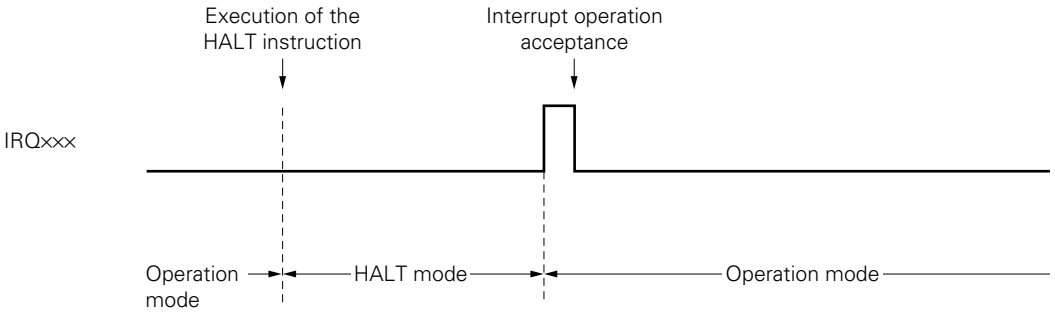


WAIT a: This refers to the wait time until TM counts the divide-by-256 clock up to 256.
$256 \times 256/f_X$ (when approximately 32 ms and f_X=2MHz)

(b) HALT cancellation by IRQ××× (if DI)



(c) HALT cancellation by IRQ××× (if EI)



15.2.3 HALT Setting Condition

(1) Forced cancellation by IRQTM

- The timer is in the operable state (TMEN=1)
- The timer's interrupt request flag is cleared (IRQTM=0).

(2) Cancellation by the interrupt request flag (IRQ×××)

- Setting in a way that places beforehand the peripheral hardware used for HALT cancellation in an operable state.

Timer	Operable state (TMEN=1)
Serial Interface	Serial interface circuit placed in operable state (SIOTS=1, SIOEN=1)
INT Pin	Setting the edge selection

- The interrupt request flag (IRQ×××) of the peripheral hardware used for HALT cancellation is cleared (to 0).
- The interrupt enable flag (IP×××) of the peripheral hardware used for HALT cancellation is set (to 1).

Caution Be sure to code a NOP instruction immediately before the HALT instruction. The time of one instruction is generated between the IRQ××× operation instruction and the HALT instruction by coding the NOP instruction immediately before the HALT instruction. Therefore, in the case of the CLR1 IRQ××× instruction, for example, the clearance of the IRQ××× is correctly reflected in the HALT instruction (Example 1). If a NOP instruction is not coded immediately before the HALT instruction, the CLR1 IRQ××× instruction is not reflected in the HALT instruction thus failing to place the system in HALT mode (Example 2).

Example 1. A correct program example

```
        :
(Setting of IRQxxx)
        :

CLR1   IRQxxx
NOP            ; Codes a NOP instruction immediately before the HALT instruction
               ; (Clearance of IRQxxx is reflected correctly to the HALT instruction.
HALT   1000B   ; Executes the HALT instruction correctly (placing the system in HALT mode).
        :
        :
```

2. An incorrect program example

```
        :
(Setting of IRQxxx)
        :

CLR1   IRQxxx  ; Clearance of IRQxxx is not reflected as to the HALT instruction.
               ; (It is the instruction following the HALT instruction that is reflected.)
HALT   1000B   ; The HALT instruction is ignored (not placing the system in HALT mode.)
        :
        :
```

15.3 STOP MODE

15.3.1 STOP Mode Setting

Executing the STOP instruction places the system in STOP mode.

Operand $b_3b_2b_1b_0$ of the STOP instruction is the condition for canceling STOP mode.

Table 15-4. STOP Mode Cancellation Condition

Format: STOP $b_3b_2b_1b_0$B

Bit	STOP Mode Cancellation Condition[Note 1]
b_3	At 1, this bit enables cancellation by IRQ×××.[Note 2, 3]
b_2	"Fixed to 0"
b_1	"Fixed to 0"
b_0	"Fixed to 0"

Notes 1. At STOP 0000B, only the resets ($\overline{\text{RESET}}$ input; power-ON/power-DOWN reset) are valid. The microcontroller is internally initialized to the state immediately following the resetting when STOP 0000B is executed.

2. It is required that IP×××=1. Cancellation by IRQTM is not possible.

3. Even if the STOP instruction is executed when IRQ×××=1, the STOP instruction is ignored (handled as a NOP instruction) thus failing to place the system in STOP mode.

15.3.2 Start Address after STOP Mode Cancellation

The start address varies depending on the cancellation condition and the interrupt enable condition.

Table 15-5. Start Address After STOP Mode Cancellation

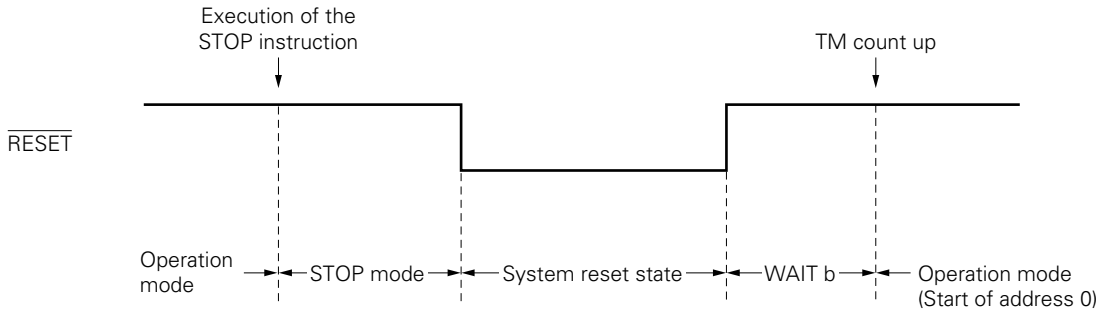
Cancellation Condition	Start Address after Cancellation
Reset[Note 1]	Address 0
IRQ×××[Note 2]	If DI, the start address is the one following the STOP instruction
	If EI, the start address is the interrupt vector. (If more than one IRQ××× have been set, the start address is the interrupt vector with highest priority.)

Notes 1. Valid resets include the $\overline{\text{RESET}}$ input and power-ON/power-DOWN resets.

2. It is required that IP×××=1. Cancellation by IRQTM is not possible.

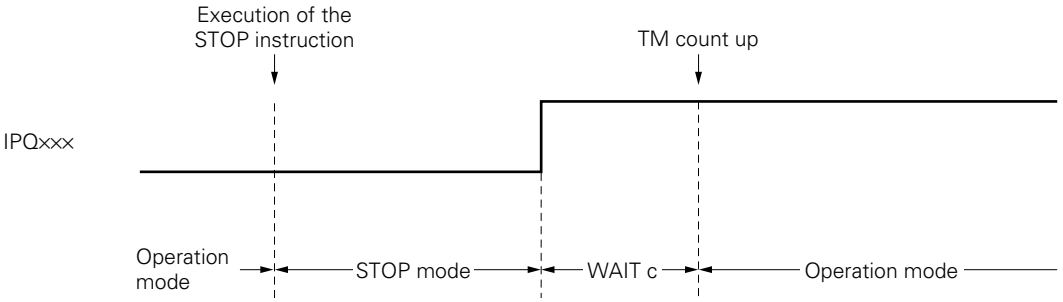
Figure 15-2. Cancellation of STOP Mode

(a) STOP cancellation by $\overline{\text{RESET}}$ input



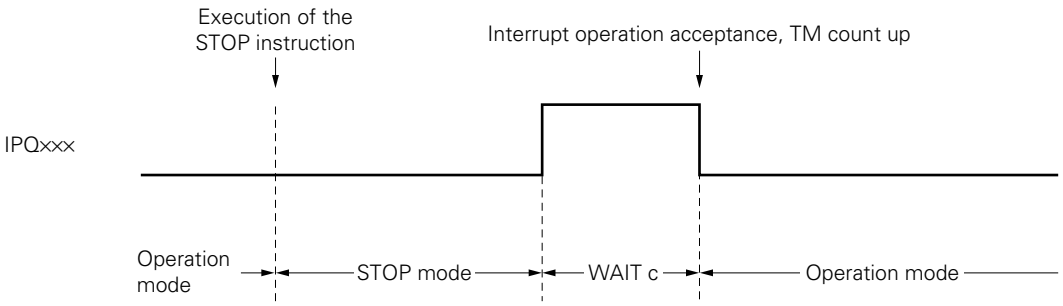
WAIT b: This refers to the wait time until TM counts the divide-by-256 clock up to 256.
$256 \times 256/f_X+\alpha$ (when approximately 32 ms+α and f_X=2MHz)
α: Oscillation growth time (Varies depending on the resonator)

(b) STOP cancellation by IRQ××× (if DI)



WAIT c: This refers to the wait time until TM counts the divide-by-m clock up to (n+1).
$(n+1) \times m/f_X+\alpha$ (n and m: values immediately before the system is placed in STOP mode)
α: Oscillation growth time (Varies depending on the resonator)

(c) STOP cancellation by IRQ××× (if EI)



WAIT c: This refers to the wait time until TM counts the divide-by-m clock up to (n+1).
$(n+1) \times m/f_X+\alpha$ (n and m: values immediately before the system is placed in STOP mode)
α: Oscillation growth time (Varies depending on the resonator)

15.3.3 STOP Setting Condition

Cancellation by IRQ×××

Cancellation by IRQ	• Sets the edge selection (IEGMD1, IEGMD0) for the signal that is input from the INT pin. • Sets the modulo register value of the timer (wait time for generation of oscillation stability). • Clears the interrupt request flag (IRQ) of the INT pin (to 0). • Sets the interrupt enable flag (IP) of the INT pin (to 1.)
Cancellation by IRQSIO	• Sets the source clock to the external clock (SIOCK1=0, SIOCK0=0) that is input from the $\overline{SCK}$ pin. • Sets the serial interface to the operable state (SIOTS=1). • Sets the modulo register value of the timer (wait time for generation of oscillation stability). • Clears the interrupt request flag (IRQSIO) of the serial interface (to 0). • Sets the interrupt enable flag (IPSIO) of the serial interface (to 1).

Caution Be sure to code a NOP instruction immediately before the STOP instruction. The time of one instruction is generated between the IRQ××× operation instruction and the STOP instruction by coding the NOP instruction immediately before the STOP instruction. Therefore, in the case of the CLR1 IRQ××× instruction, for example, the clearance of IRQ××× is correctly reflected in the STOP instruction (Example 1). If a NOP instruction is not coded immediately before the STOP instruction, the CLR1 IRQ××× instruction is not reflected in the STOP instruction thus failing to place the system in STOP mode (Example 2).

Example 1. A correct program example

```
        :
(Setting of IRQxxx)
        :

CLR1   IRQxxx
NOP            ; Codes a NOP instruction immediately before the STOP instruction.
               ; (Clearance of IRQxxx is reflected correctly to the STOP instruction.
STOP   1000B   ; Executes the STOP instruction correctly (placing the system in STOP mode).
        :
        :
```

2. An incorrect program example

```
        :
(Setting of IRQxxx)
        :

CLR1   IRQxxx  ; Clearance of IRQxxx is not reflected to the STOP instruction.
               ; (It is the instruction following the STOP instruction that is reflected.)
STOP   1000B   ; The STOP instruction is ignored (not placing the system in STOP mode.)
        :
        :
```

CHAPTER 16 RESET

The following 3 types of resets are provided in the μPD17120 series.

<1> Reset by input to $\overline{\text{RESET}}$.

<2> The power-on/power-down reset function when power is turned on or supply voltage drops.

<3> The address stack overflow/underflow reset function.

16.1 RESET FUNCTIONS

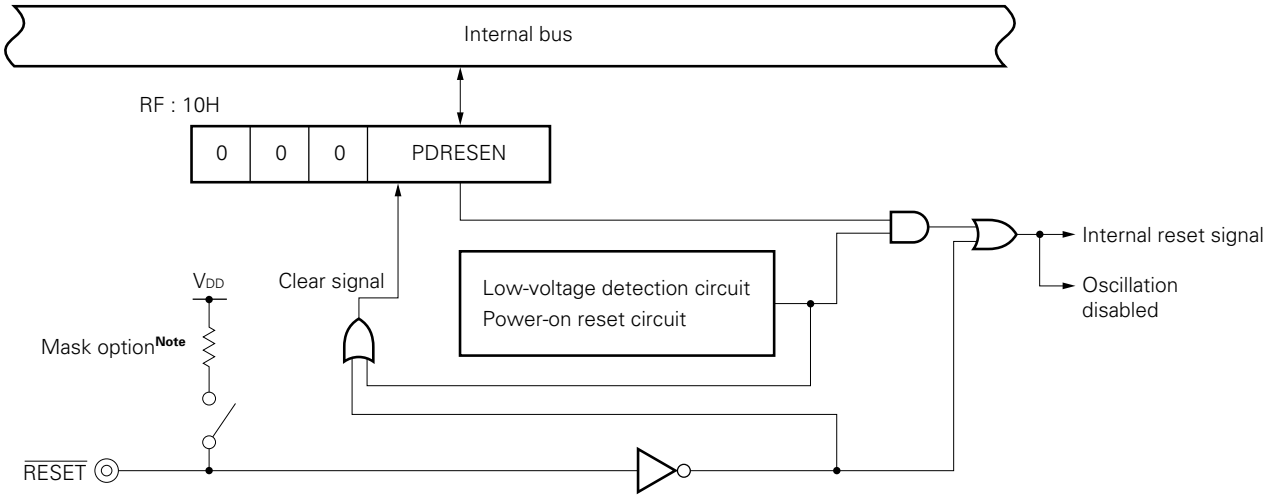
The reset functions are used to initialize device operations. The state to be initialized depends on the type of reset.

Table 16-1. State of Each Hardware Unit When Reset

Hardware \ Reset Type		• $\overline{\text{RESET}}$ Input during Operation • Built-in Power-ON/Power-DOWN Reset during Operation	• $\overline{\text{RESET}}$ Input during Standby Mode • Built-in Power-ON/Power-DOWN Reset during Standby Mode	• Stack Overflow or Underflow
Program Counter		0000H	0000H	0000H
Port	Input/Output mode	Input	Input	Input
	Output latch	0	0	Undefined
General-Purpose Data Memory	Other than DBF	Undefined	Retains the status immediately preceding the resetting.	Undefined
	DBF	Undefined	Undefined	Undefined
System Register	Other than WR	0	0	0
	WR	Undefined	Retains the status immediately preceding the resetting.	Undefined
Control Register		SP=5H; IRQTM1=1; TMEN=1; CMPVREF23=1**Note**; CMPRSLT=1**Note**; INT retains the status of the INT pin at the time; all the others go to 0. Refer to **19.2 RESERVED SYMBOLS**.		SP=5H; INT retains the status of the INT pin at the time; all the others go to 0.
Timer	Count register	00H	00H	Undefined
	Modulo register	FFH	FFH	FFH
Serial Interface's Shift Register (SIOSFR)		Undefined	Retains the status immediately preceding the resetting.	Undefined

Note μPD17132, 17133, 17P132, and 17P133 only.

Figure 16-1. Reset Block Configuration



Note The μPD17P132 and 17P133 have no pull-up resistor by mask option, and are always open.

16.2 RESETTING

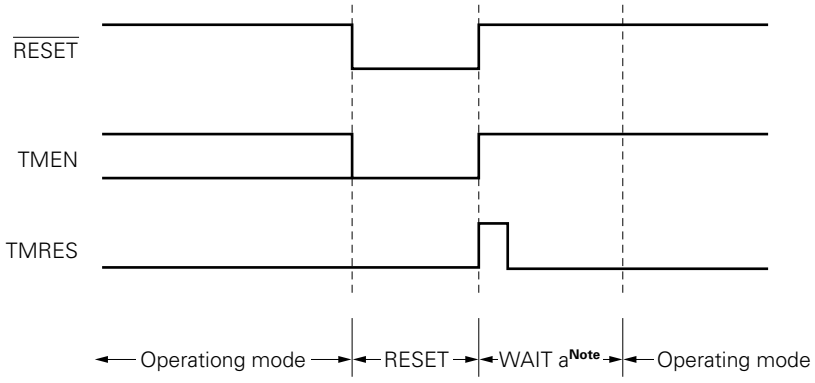
Operation when reset is caused by the $\overline{\text{RESET}}$ input is shown in the figure below.

If the $\overline{\text{RESET}}$ pin is set from low to high, system clock generation starts and an oscillation stabilization wait occurs with the timer. Program execution starts from address 0000H.

If power-on reset function is used, the reset signals shown in Figure 16-2 are internally generated. Operation is the same as that when reset is caused externally by the $\overline{\text{RESET}}$ input.

At address stack overflow and underflow reset, oscillation stabilization wait time (WAIT a) does not occur. Operation starts from address 0000H after initial statuses are internally set.

Figure 16-2. Resetting



Note This is oscillation stabilization wait time. Operating mode is set when timer counts system clocks 256 × 256 time (approx. 8ms, at f_X=8 MHz/approx. 32 ms, at f_{CC}=2 MHz)

16.3 POWER-ON/POWER-DOWN RESET FUNCTION

The μPD17120 subseries is provided with two reset functions to prevent malfunctions from occurring in the microcontroller. They are the power-on reset function and power-down reset function. The power-on reset function resets the microcontroller when it detects that power was turned on. The power-down reset function resets the microcontroller when it detects drops in the power voltage.

These functions are implemented by the power-voltage monitoring circuit whose operating voltage has a different range from the logic circuits in the microcontroller and the oscillation circuit (which stops oscillation at reset to put the microcontroller in a temporary stop state). Conditions required to enable these functions and their operations will be described next.

Caution When designing an applied circuit requiring a high level of reliability, make sure that its resetting relies on the built-in power-on/power-down reset function only. Also, design the circuit in such a way that the $\overline{\text{RESET}}$ signal is input externally.

16.3.1 Conditions Required to Enable the Power-On Reset Function

This function is effective when used together with the power-down reset function.

The following conditions are required to validate the power-on reset function:

<1> The power voltage must be 4.5 to 5.5 V during normal operation, including the standby state.
<2> The frequency of the system clock oscillator must be 400 kHz to 4 MHz.**Note**
<3> The power-down reset function must be enabled during normal operation, including the standby state.
<4> The power voltage must rise from 0 V to the specified voltage.
<5> The time it takes for the power voltage to rise from 0 to 2.7 V must be shorter than the oscillation stabilization wait time counted in timer of the μPD17120 subseries. (System clock 256 × 256 counts: approx. 16 ms, at f_X=4 MHz/approx. 32 ms, at f_{CC}=2 MHz)

Note μPD17121/17133/17P133 only

Cautions 1. If the above conditions are not satisfied, the power-on reset function will not operate effectively. In this case, an external reset circuit needs to be added.

2. In the standby state, even if the power-down reset function operates normally, general-purpose data memory (except for DBF) retains data up to V_{DD}=2.7 V. If, however, data is changed due to an external error, the data in memory is not guaranteed.

16.3.2 Description and Operation of the Power-On Reset Function

The power-on reset function resets the microcontroller when it detects that power was turned on in the hardware, regardless of the software state.

The power-on reset circuit operates under a lower voltage than the other internal circuits in the μPD17120 subseries. It initializes the microcontroller regardless whether the oscillation circuit is operating. When the reset operation is terminated, timer counts the number of oscillation pulses sent from the oscillator until it reaches the specified value. Within this period, oscillation becomes stable and the power voltage applied to the microcontroller enters the range (V_{DD}=2.7 to 5.5 V at 400 kHz to 4 MHz**Note**) in which the microcontroller is guaranteed to operate.

When this period elapses, the microcontroller enters normal operation mode. Figure 16-3 shows an example of the power-on reset operation.

Note μPD17121/17133/17P133 only

Operation of the power-on reset circuit

<1> This circuit always monitors the voltage applied to the V_{DD} pin.

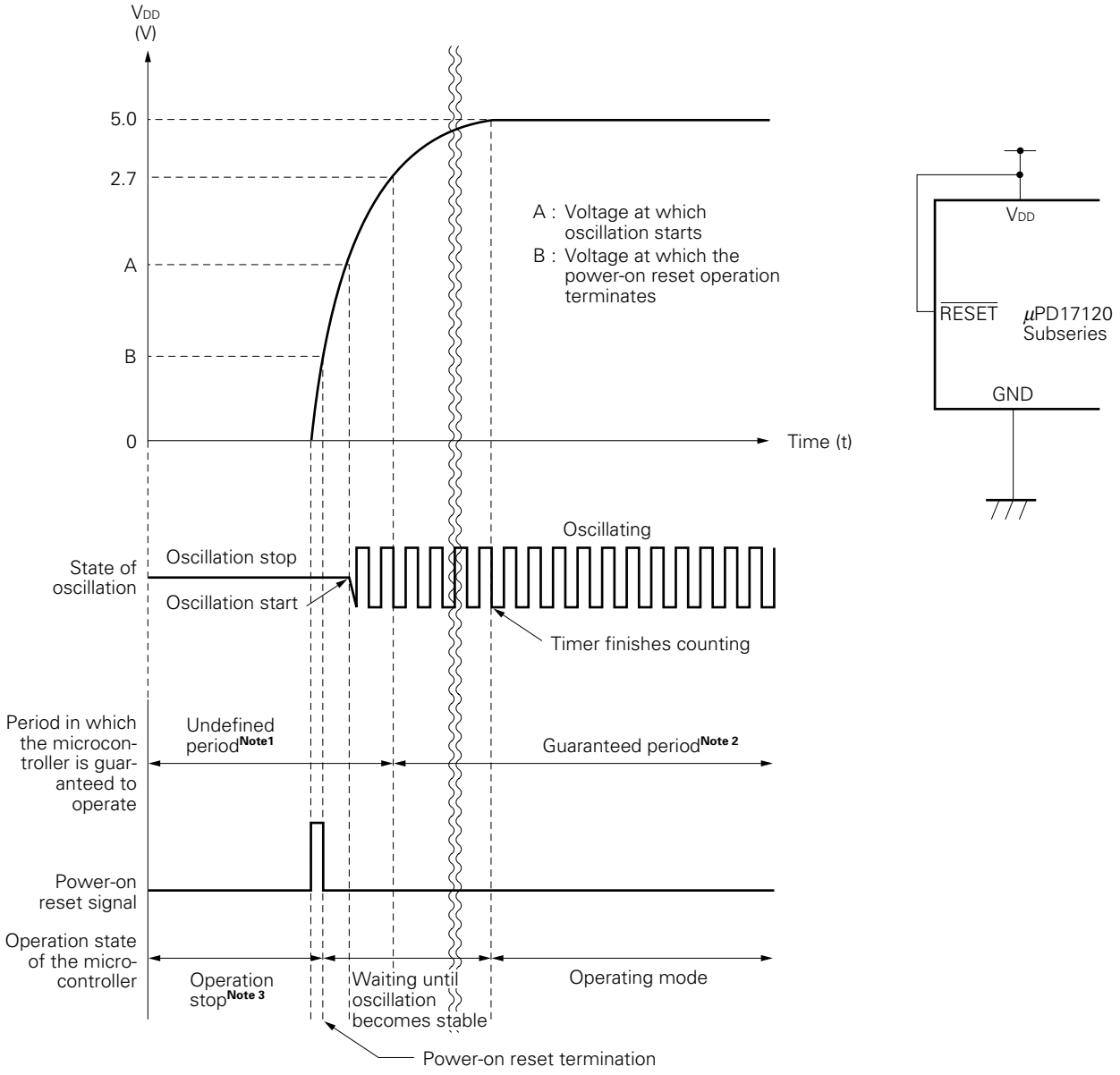
<2> This circuit resets the microcontroller **Note** until power reaches a particular voltage (typically 1.5 V), regardless whether the oscillation circuit is operating.

<3> This circuit stops oscillation during the reset operation.

<4> When reset is terminated, timer counts oscillation pulses. The microcontroller waits until oscillation becomes stable and the power voltage becomes V_{DD}=2.7 V or higher.

Note It is from the point when the supply voltage has reached a level allowing the internal circuit to be operable (accepting the internal reset signal) that the resetting takes effect within the microcontroller.

Figure 16-3. Example of the Power-On Reset Operation



Notes **1.** During the operation-undefined period, certain operations on the μPD17120 subseries are not guaranteed. However, the power-on reset function is guaranteed in this period.

2. The operation-guaranteed period refers to the time in which all the operations specified for the μPD17120 subseries are guaranteed.

3. An operation stop state refers to the state in which all of the functions of the microcontroller are stopped.

16.3.3 Condition Required for Use of the Power-Down Reset Function

The power-down reset function can be enabled or disabled using software. The following conditions are required to use this function:

- The power voltage must be 4.5 to 5.5 V during normal operation, including the standby state.
- The frequency of the system clock oscillator must be 400 kHz to 4 MHz.**Note**

Note μPD17121/17133/17P133 only

Caution When the microcontroller is used with a power voltage of 2.7 to 4.5 V, add an external reset circuit instead of using the internal power-down reset circuit. If the internal power-down reset circuit is used with a power voltage of 2.7 to 4.5 V, reset operation may not terminate.

16.3.4 Description and Operation of the Power-Down Reset Function

This function is enabled by setting the power-down reset enable flag (PDRESEN) using software.

When this function detects a power voltage drop, it issues the reset signal to the microcontroller. It then initializes the microcontroller. Stopping oscillation during reset prevents the power voltage in the microcontroller from fluctuating out of control. When the specified power voltage recovers and the power-down reset operation is terminated, the microcontroller waits the time required for stable oscillation using the timer. The microcontroller then enters normal operation (starts from the top of memory).

Figure 16-4 shows an example of the power-down operation. Figure 16-5 shows an example of reset operation during the period from power-down reset to power recovery.

Operation of the power-down reset circuit

<1> This circuit always monitors the voltage applied to the V_{DD} pin.

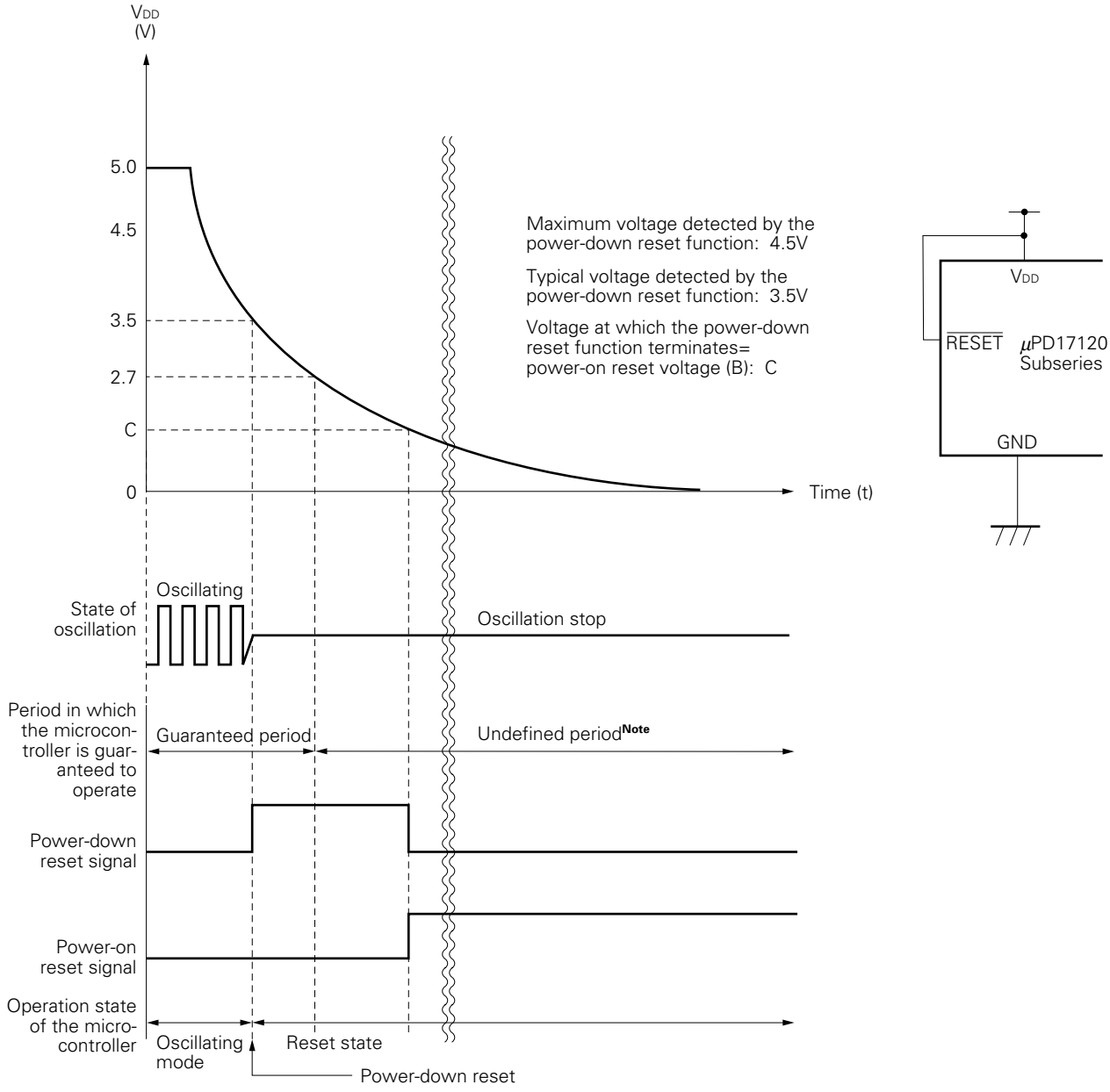
<2> When this circuit detects a power voltage drop, it issues a reset signal to the other parts of the microcontroller. It continues to send this reset signal until the power voltage recovers or all the functions in the microcontroller stop.

<3> This circuit stops oscillation during the reset operation to prevent software crashes.
When the power voltage recovers to the low-voltage detection level (typically 3.5 V, 4.5 V maximum) before the power-down reset function stops, the microcontroller waits the time required for stable oscillation using timer, then enters normal operation mode.

<4> When the power voltage recovers from 0 V, the power-on reset function has priority.

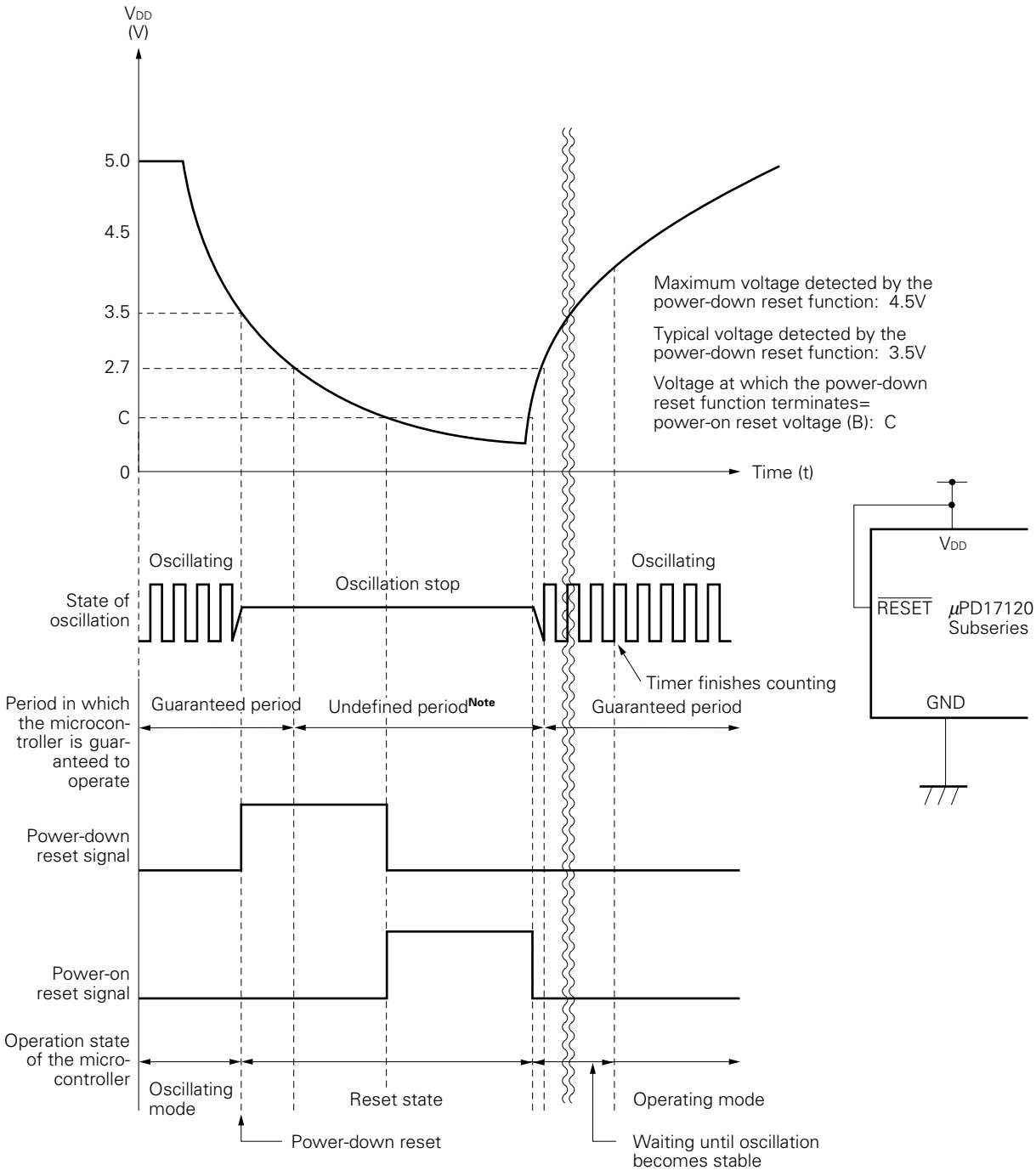
<5> After the power-down reset function stops and the power voltage recovers before it reaches 0 V, the microcontroller waits using timer until oscillation becomes stable and the power voltage (V_{DD}) reaches 2.7 V. The microcontroller then enters normal operation mode.

Figure 16-4. Example of the Power-Down Reset Operation



Note The undefined operation area refers to the area in which operation specified for the μPD17120 subseries is not assured. However, even in this area, the power-down reset function operates, thus continuing to generate resets until all the other functions within the microcontroller are stopped.

Figure 16-5. Example of Reset Operation during the Period from Power-Down Reset to Power Recovery



Note The undefined operation area refers to the area in which operation specified for the μPD17120 subseries is not assured. However, even in this area, the power-down reset function operates, thus continuing to generate resets until all the other functions within the microcontroller are stopped.

CHAPTER 17 ONE-TIME PROM WRITING/VERIFYING

The on-chip program memory of the μPD17P132 and 17P133 is a 1024 × 16-bit one-time PROM.

Pins listed in Table 17-1 are used for one-time PROM writing/verifying. The address is updated by the clock signal input from the CLK pin.

Caution INT/V_{PP} pin is used as V_{PP} pin in program writing/verifying mode. Therefore, there is a possibility of overrunning of the microcontroller when voltage higher than V_{DD} + 0.3 V is applied to INT/V_{PP} pin in normal operation mode. Pay careful attention to ESD protection.

Table 17-1. Pins Used for Writing/Verifying Program Memory

Pin	Function
V_{PP}	Applies program voltage. Apply 12.5 V to this pin.
V_{DD}	Power supply pin. Apply 6 V to this pin.
CLK	Clock input for updating address. Updates program memory address by inputting four pulses.
MD_0-MD_3	Select operation mode.
D_0-D_7	8-bit data I/O pins.

17.1 DIFFERENCES BETWEEN MASK ROM VERSION AND ONE-TIME PROM VERSION

The μPD17P132 and 17P133 are microcontrollers replacing the program memory of the on-chip mask ROM version μPD17132 and 17133 to one-time PROM. Table 17-2 shows the differences between mask ROM version and one-time PROM version.

Differences between each products are only its capacity of ROM/RAM, and whether it can specify mask option or not. The CPU function and internal peripheral hardware of each product (excluding comparator) are the same. Therefore, at system designing, the μPD17P132 can be used for evaluating program of the μPD17120/17132 . Also, the μPD17P133 can be used for evaluating the μPD17121/17133 in the same way.

Table 17-2. Differences Between Mask ROM Version and One-Time PROM Version

Item	μPD17120	μPD17132	μPD17P132	μPD17121	μPD17133	μPD17P133
ROM	Mask ROM		One-time PROM	Mask ROM		One-time PROM
	768 × 16 bit (0000H-02FFH)	1024 × 16 bit (0000H-03FFH)		768 × 16 bit (0000H-02FFH)	1024 × 16 bit (0000H-03FFH)	
RAM	64 × 4 bit	111 × 4 bit		64 × 4 bit	111 × 4 bit	
P0D and P0E pins and pull-up resistor of $\overline{\text{RESET}}$ pin	Mask option		Not available	Mask option		Not available
V_{PP} pin, operating mode selection pin	Not available		Available	Not available		Available
Operating frequency	f_{CC}=400 kHz to 2.4 MHz			f_X=400 KHz to 4 MHz (V_{DD}=2.7 to 5.5 V) f_X=400 kHz to 8 MHz (V_{DD}=4.5 to 5.5 V)		
Comparator	Not available	Available		Not available	Available	

Caution Although, functionally, the PROM product is highly compatible with the masked ROM product, they still differ from each other in terms of their internal ROM circuits and some electrical features. When switching from a PROM product to a ROM product, ensure to make sufficient application evaluations based on masked-ROM product samples.

17.2 OPERATING MODE IN PROGRAM MEMORY WRITING/VERIFYING

The μPD17P132 and 17P133 become program memory writing/verifying mode by applying +6V to V_{DD} pin and +12.5 V to V_{PP} pin after reset state for a fixed time (V_{DD}=5 V, $\overline{\text{RESET}}$=0 V). This mode becomes the following operating mode by the setting of pins MD_0 to MD_3. Regarding pins other than those shown in Table 17-1, connect them all individually to the GND through pull-down resistors.

For details, refer to **1.4 (2)**.

Table 17-3. Operating Mode Setting

Operating Mode Setting						Operating Mode
V_{PP}	V_{DD}	MD_0	MD_1	MD_2	MD_3	
+12.5 V	+6 V	H	L	H	L	Clear program memory address to 0.
		L	H	H	H	Write mode
		L	L	H	H	Verify mode
		H	×	H	H	Program inhibit mode

Remark ×: don't care (L or H)

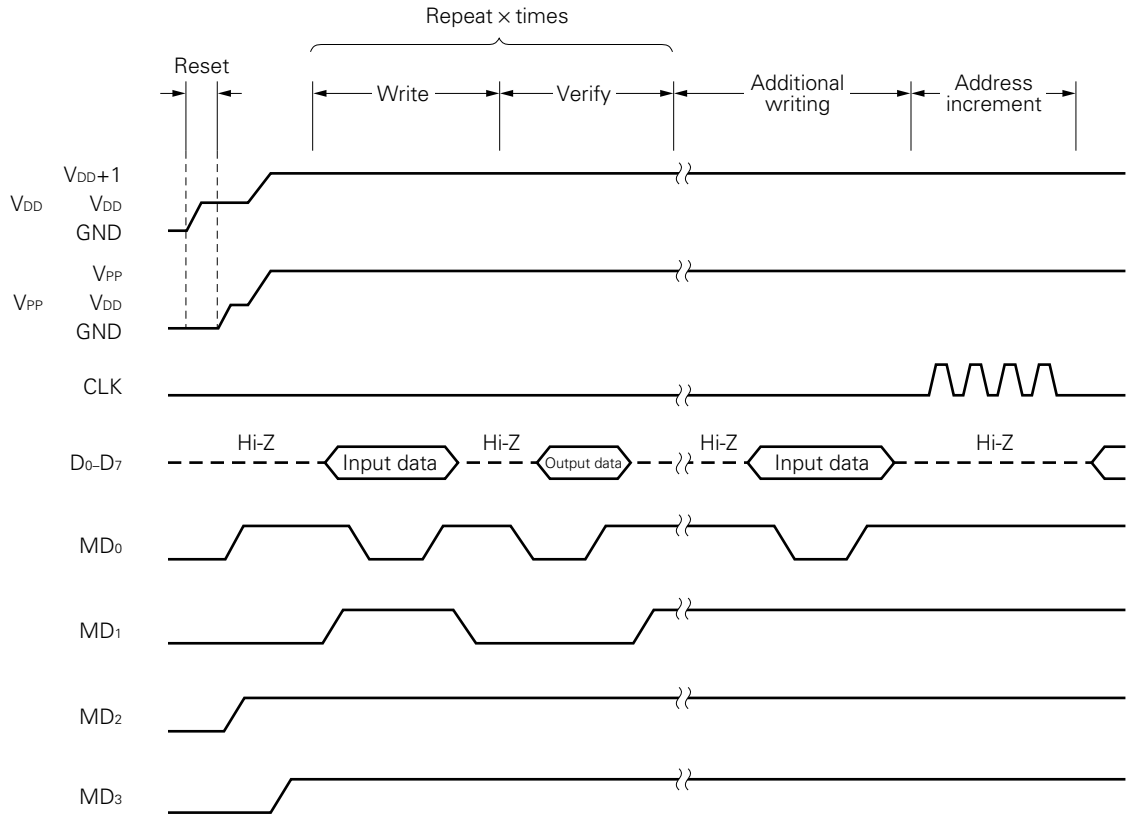
17.3 WRITING PROCEDURE OF PROGRAM MEMORY

The program memory can be written at high speeds in the following procedure.

(1) Pull down the unused pins to GND. (X_{OUT} pin is open.) Mask the CLK pin low.
(2) Apply 5 V to the V_{DD} pin. Make V_{PP} pin low.
(3) Wait for 10 μs. Then, apply 5 V to V_{PP} pin.
(4) Set the program memory address 0 clear mode using mode selector pins.
(5) Apply 6 V to V_{DD} and 12.5 V to V_{PP}.
(6) Set the program inhibit mode.
(7) Write data in mode for 1 ms writing.
(8) Set the program inhibit mode.
(9) Set the verify mode (MD_0-MD_3=LLHH). If the program has been correctly written, proceed to (10). If not, repeat (7) through (9).
(10) Additional writing of (number of times (×) the program has been written in (7) through (9)) × 1ms.
(11) Set the program inhibit mode.
(12) Input four pulses to the CLK pin to update the program memory address by one.
(13) Repeat (7) through (12) until the last address in programmed.
(14) Set the program memory address 0 clear mode.
(15) Change the voltage of V_{DD} and V_{PP} pins to 5 V.
(16) Turn off the power.

Figure 17-1 shows the procedures of (2) through (12).

Figure 17-1. Procedure of program Memory Writing

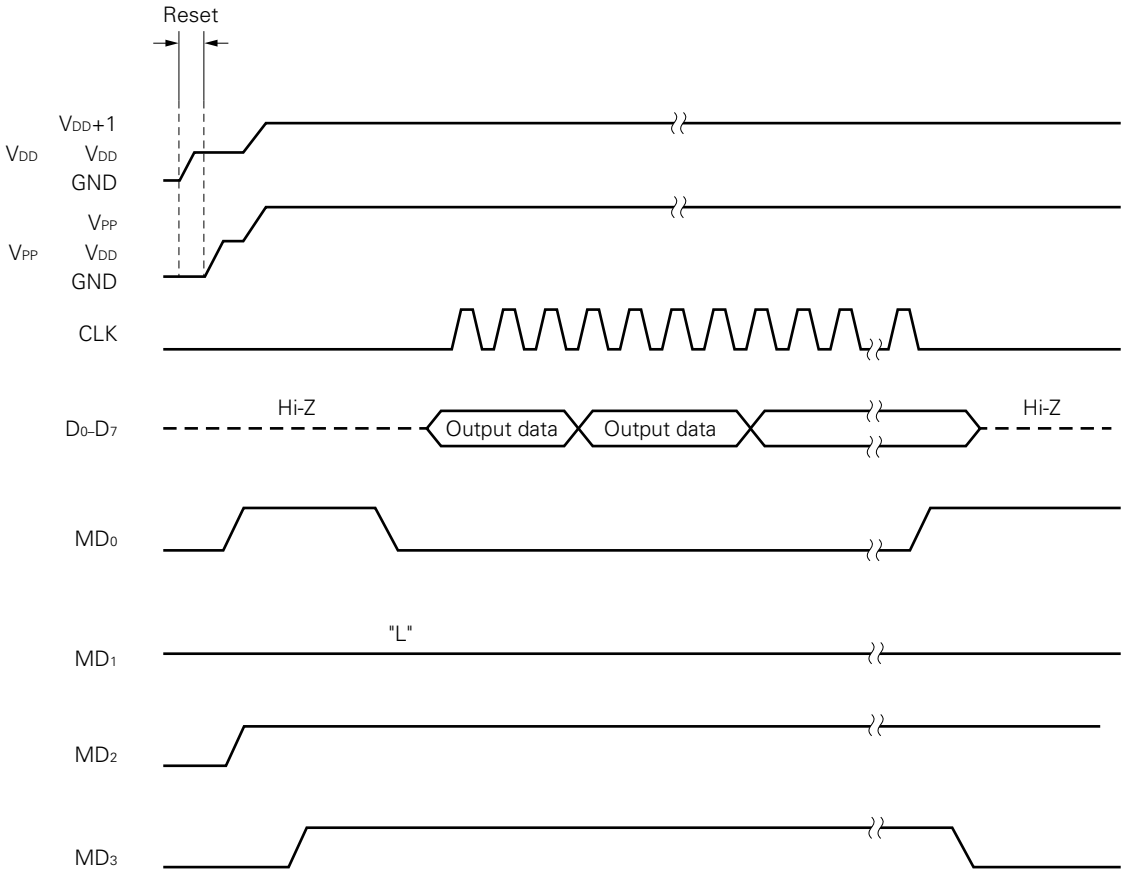


17.4 READING PROCEDURE OF PROGRAM MEMORY

(1) Pull down the unused pins to GND. (X_{OUT} pin is open.) Make the CLK pin low.
(2) Apply 5 V to the V_{DD} pin. Make V_{PP} pin low.
(3) Wait for 10 μs. Then, apply 5 V to V_{PP} pin.
(4) Set the program memory address 0 clear mode using mode selector pins.
(5) Apply 6 V to V_{DD} and 12.5 V to V_{PP}.
(6) Set mode selector pins to the program inhibit mode.
(7) Set the verify mode. When clock pulses are input to the CLK pin, data for each address can be sequentially output with four clocks as one cycle.
(8) Set the program inhibit mode.
(9) Set the program memory address 0 clear mode.
(10) Change the voltage of V_{DD} and V_{PP} pins to 5 V.
(11) Turn off the power.

Figure 17-2 shows the program reading procedure (2) through (9).

Figure 17-2. Procedure of Program Memory Reading



[MEMO]

CHAPTER 18 INSTRUCTION SET

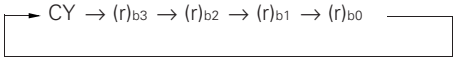
18.1 OVERVIEW OF THE INSTRUCTION SET

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			
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	CALL	addr
1101	D			MOV	m, #n4
1110	E			SKT	m, #n
1111	F			SKF	m, #n

18.2 LEGEND

AR	:	Address register
ASR	:	Address stack register indicated by stack pointer
addr	:	Program memory address (11 bits, the most-significant bit is fixed to 0)
BANK	:	Bank register
CMP	:	Compare flag
CY	:	Carry flag
DBF	:	Data buffer
h	:	Halt release condition
INTEF	:	Interrupt enable flag
INTR	:	Register saved automatically to stack when interrupt occurs
INTSK	:	Interrupt stack register
IX	:	Index register
MP	:	Data memory row address pointer
MPE	:	Memory pointer enable flag
m	:	Data memory address indicated by m_R and m_C
m_R	:	Data memory row address (upper)
m_C	:	Data memory column address (lower)
n	:	Bit position (4 bits)
n4	:	Immediate data (4 bits)
PC	:	Program counter
p	:	Peripheral address
p_H	:	Peripheral address (upper 3 bits)
p_L	:	Peripheral address (lower 4 bits)
r	:	General register column address
rf	:	Register file address
rf_R	:	Register file row address (upper 3 bits)
rf_C	:	Register file column address (lower 4 bits)
SP	:	Stack pointer
s	:	Stop release condition
WR	:	Window register
(×)	:	Contents addressed by ×

18.3 LIST OF THE INSTRUCTION SET

Group	Mnemonic	Operand	Operation	Machine Code			
				OP Code	Operand		
Add	ADD	r, m	(r) ← (r) + (m)	00000	m_R	mc	r
		m, #n4	(m) ← (m) + n4	10000	m_R	mc	n4
	ADDC	r, m	(r) ← (r) + (m) + CY	00010	m_R	mc	r
		m, #n4	(m) ← (m) + n4 + CY	10010	m_R	mc	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	mc	r
		m, #n4	(m) ← (m) – n4	10001	m_R	mc	n4
	SUBC	r, m	(r) ← (r) – (m) –CY	00011	m_R	mc	r
		m, #n4	(m) ← (m) – n4 –CY	10011	m_R	mc	n4
Logical Operation	OR	r, m	(r) ← (r) ∨ (m)	00110	m_R	mc	r
		m, #n4	(m) ← (m) ∨ n4	10110	m_R	mc	n4
	AND	r, m	(r) ← (r) ∧ (m)	00100	m_R	mc	r
		m, #n4	(m) ← (m) ∧ n4	10100	m_R	mc	n4
	XOR	r, m	(r) ← (r) ∀ (m)	00101	m_R	mc	r
		m, #n4	(m) ← (m) ∀ n4	10101	m_R	mc	n4
Test	SKT	m, #n	CMP ← 0, if (m) ∧ n=n, then skip	11110	m_R	mc	n
	SKF	m, #n	CMP ← 0, if (m) ∧ n=0, then skip	11111	m_R	mc	n
Compare	SKE	m, #n4	(m) –n4, skip if zero	01001	m_R	mc	n4
	SKNE	m, #n4	(m) –n4, skip if not zero	01011	m_R	mc	n4
	SKGE	m, #n4	(m) –n4, skip if not borrow	11001	m_R	mc	n4
	SKLT	m, #n4	(m) –n4, skip if borrow	11011	m_R	mc	n4
Rotate	RORC	r		00111	000	0111	r
Transfer	LD	r, m	(r) ← (m)	01000	m_R	mc	r
	ST	m, r	(m) ← (r)	11000	m_R	mc	r
	MOV	@r, m	if MPE = 1: (MP, (r)) ← (m) if MPE = 0: (BANK, m_R, (r)) ← (m)	01010	m_R	mc	r
		m, @r	if MPE = 1: (m) ← (MP, (r)) if MPE = 0: (m) ← (BANK, m_R, (r))	11010	m_R	mc	r
		m, #n4	(m) ← n4	11101	m_R	mc	n4
	MOVT[Note]	DBF, @AR	SP ← SP–1, ASR ← PC, PC ← AR, DBF ← (PC), PC ← ASR, SP ← SP+1	00111	000	0001	0000

Note As an exception, execution of MOVT instruction requires two instruction cycles.

Group	Mnemonic	Operand	Operation	Machine Code			
				OP Code	Operand		
Transfer	PUSH	AR	SP ← SP – 1, ASR ← AR	00111	000	1101	0000
	POP	AR	AR ← ASR, SP ← SP + 1	00111	000	1100	0000
	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	PC ← addr	01100	addr		
		@AR	PC ← AR	00111	000	0100	0000
Sub-routine	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

18.4 ASSEMBLER (AS17K) MACRO INSTRUCTIONS

Legend

flag n : FLG symbol

< > : Can be omitted

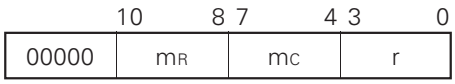
	Mnemonic	Operand	Operation	n
Macro Instructions	SKTn	flag 1, …flag n	if (flag 1) ~ (flag n)=all "1" then skip	1≤n≤4
	SKFn	flag 1, …flag n	if (flag 1) ~ (flag n)=all "0", then skip	1≤n≤4
	SETn	flag 1, …flag n	(flag 1) ~ (flag n) ← 1	1≤n≤4
	CLRn	flag 1, …flag n	(flag 1) ~ (flag n) ← 0	1≤n≤4
	NOTn	flag 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

18.5 INSTRUCTIONS

18.5.1 Addition Instructions

(1) Add r, m **Add data memory to general register**

<1> OP code



<2> Function

When CMP=0, (r) ← (r) + (m)

Adds the data memory contents to the general register contents, and stores the result in general register.

When CMP=1, (r) + (m)

The result is not stored in the register. Carry flag CY and zero flag Z are changed, according to the result.

Sets carry flag CY, if a carry occurs as a result of the addition. Resets the carry flag CY, if no carry occurs.

If the addition result is other than zero, zero flag Z is reset, regardless of compare flag CMP.

If the addition result is zero, with the compare flag reset (CMP=0), the zero flag Z is set.
If the addition result is zero, with the compare flag set (CMP=1), the zero flag Z is not changed.
Addition can be executed in binary 4-bit or BCD. The BCD flag for the PSWORD specifies which kind of addition is to be executed.

<3> Example 1

Adds the address 0.2FH contents to the address 0.03H contents, when row address 0 (0.00H-0.0FH) in bank 0 is specified as the general register (RPH=0, RPL=0), and stores the result in address 0.03H:

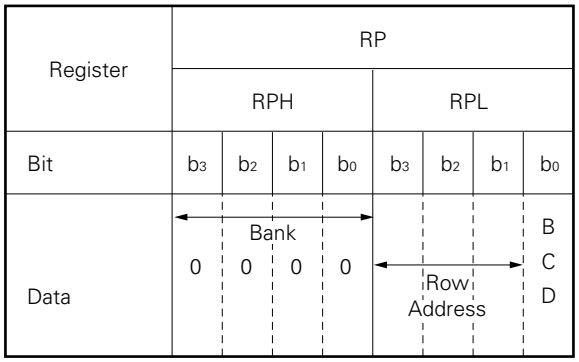
```
            (0.03H) ← (0.03H) + (0.2FH)
MEM003      MEM   0.03H
MEM02F      MEM   0.2FH
            MOV   BANK, #00H      ; Data memory bank 0
            MOV   RPH, #00H       ; General register bank 0
            MOV   RPL, #00H       ; General register row address 0
            ADD   MEM003, MEM02F
```

Example 2

Adds the address 0.2FH contents to the address 0.23H contents, when row address 2 (0.20H-0.2FH) in bank 0 is specified as the general register (RPH=0, RPL=4), and stores the result in address 0.23H:

```
            (0.23H) ← (0.23H) + (0.2FH)
MEM023      MEM   0.23H
MEM02F      MEM   0.2FH
            MOV   BANK, #00H      ; Data memory bank 0
            MOV   RPH, #00H       ; General register bank 0Note
            MOV   RPL, #04H       ; General register row address 2
            ADD   MEM023, MEM02F
```

Note



RP (general register pointer) is assigned in the system register, as shown above.

Therefore, to set bank 0 and row address 2 in a general register, 00H must be stored in RPH and 04H, in RPL.

In this case, the subsequent arithmetic operation is executed in binary 4-bit operation, because the BCD flag is reset.

Example 3

Adds the address 0.6FH contents to the address 0.03H contents and stores the result in address 0.03H. At this time, data memory address 0.6FH can be specified, by selecting data memory address 2FH, if IXE=1, IXH=0, IXM=4, and IXL=0, i.e., IX=0.40H.

(0.03H) ← (0.03H) + (0.6FH)

Address obtained as result of ORing index register contents, 0.40H, and data memory address 0.2FH

```
MEM003  MEM   0.03H
MEM02F  MEM   0.2FH
        MOV   RPH, #00H          ; General register bank 0
        MOV   RPL, #00H          ; General register row address 0
        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

Adds the address 0.3FH contents to the address 0.03H contents and stores the result in address 0.03H. At this time, data memory address 0.3FH can be specified by specifying data memory address 2FH, if IXE=1, IXH=0, IXM=1, and IXL=0, i.e., IX=0.10H.

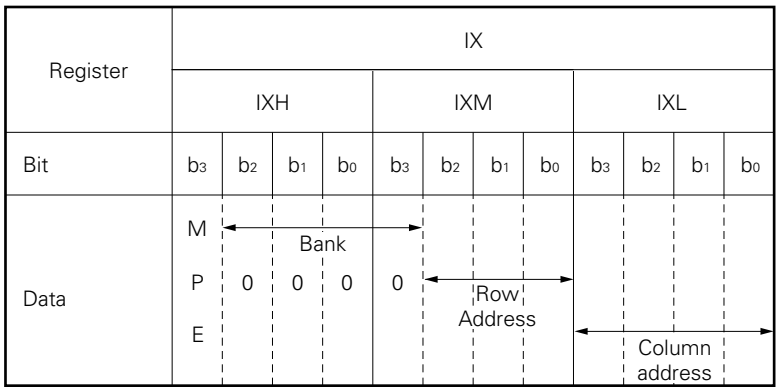
(0.03H) ← (0.03H) + (0.3FH)

Address obtained as result of ORing index register contents, 0.10H, and data memory address 0.2FH

```
MEM003  MEM   0.03H
MEM02F  MEM   0.2FH
```

```
MOV   BANK, #00H
MOV   RPH, #00H          ; General register bank 0
MOV   RPL, #00H          ; General register row address 0
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



IX (index register) is assigned in the system register, as shown above.

Therefore, to specify IX=0.10H, 00H must be stored in IXH. 01H in IXM, and 00H in IXL.

In this case, MP (memory pointer) for general register indirect transfer is invalid, because the MPE flag (memory pointer enable) is reset.

<4> Note

The first operand for the ADD r, m instruction is a column address in general register. Therefore, if the instruction is described as follows, the column address for the general register is 03H.

```
MEM013   MEM   0.13H
MEM02F   MEM   0.2FH
         ADD   MEM013, MEM02F
```

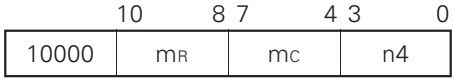
Indicates the general register column address.
The lower 4 bits (in this case, 03H) are valid

When CMP flag=1, the addition result is not stored.

When BCD flag=1, the BCD result is stored.

(2) ADD m, #n4 **Add immediate data to data memory**

<1> OP code



<2> Function

When CMP=0, (m) ← (m) + n4

Adds immediate data to the data memory contents, and stores the result in data memory.

When CMP=1, (m) + n4

The result is not stored in the data memory. Carry flag CY and zero flag Z are changed, according to the result.

Sets carry flag CY, if a carry occurs as a result of the addition; resets the carry flag CY if no carry occurs.

If the addition result is other than zero, zero flag Z is reset, regardless of compare flag CMP.

If the addition result is zero with the compare flag reset (CMP=0), the zero flag Z is set.

If the addition result is zero with the compare flag set (CMP=1), the zero flag Z is not changed.

Addition can be executed in binary 4-bit or BCD. The BCD flag for the PSWORD specifies which kind of addition is to be executed.

<3> Example 1

Adds 5 to the address 0.2FH contents, and stores the result in address 0.2FH:

```
                (0.2FH) ← (0.2FH) + 5
MEM02F    MEM   0.2FH
          ADD MEM02F, #05H
```

Example 2

Adds 5 to the address 0.6FH contents and stores the result in address 0.6FH. At this time, data memory address 0.6FH can be specified by selecting data memory address 2FH, if IXE=1, IXH=0, IXM=4, and IXL=0, i.e., IX=0.40H.

(0.6FH) ← (0.6FH) + 05H

Address obtained as result of ORing index register contents, 0.40H, and data memory address 0.2FH

```
MEM02F  MEM   0.2FH
        MOV   BANK, #00H        ; Data memory bank 0
        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

Adds 5 to the address 0.2FH contents and stores the result in address 0.2FH. At this time, data memory address 0.2FH can be specified by selecting data memory address 2FH, if IXE=1, IXH=0, IXM=0, and IXL=0, i.e., IX=0.00H.

(2.2FH) ← (0.2FH) + 05H

Address obtained as result of ORing index register contents, 0.00H, and data memory address 0.2FH

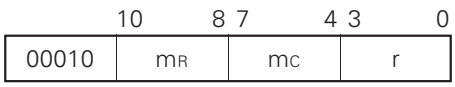
```
MEM02F  MEM   0.2FH
        MOV   BANK, #00H        ; Data memory bank 0
        MOV   IXH, #00H         ; IX ← 00000000000B
        MOV   IXM, #00H
        MOV   IXL, #00H         ;
        SET1  IXE               ; IXE flag ← 1
        ADD   MEM02F, #05H      ; IX                  00000000000B (0.00H)
                                ; Bank operand OR)    00000101111B (0.2FH)
                                ; Specified address   00000101111B (0.2FH)
```

<4> Note

When the CMP flag=1, the addition result is not stored.
When the BCD flag=1, the BCD result is stored.

(3) ADDC r, m — Add data memory to general register with carry flag

<1> OP code



<2> Function

When CMP=0, (r) ← (r) + (m) +CY

Adds the data memory contents to the general register contents with carry flag CY, and stores the result in general register indentified as r.

When CMP=1, (r) + (m) + CY

The result is not stored in the register. Carry flag CY and zero flag Z are changed according to the result.

By using this ADDC instruction, two or more words can be easily added.

Sets carry flag CY, if a carry occurs as a result of the addition; resets the carry flag CY if no carry occurs.

If the addition result is other than zero, zero flag Z is reset, regardless of compare flag CMP.

If the addition result is zero, with the compare flag reset (CMP=0), the zero flag Z is set.

If the addition result is zero, with the compare flag set (CMP=1), the zero flag Z is not changed.

Addition can be executed in binary 4-bit or BCD. The BCD flag for program status word PSWORD specifies which kind of addition is to be executed.

<3> Example 1

Adds the 12-bit contents for addresses 0.0DH through 0.0FH to the 12-bit contents for addresses 0.2DH through 0.2FH, and stores the result in the 12-bit contents for address 0.0DH to 0.0FH, when row address 0 (0.00H-0.0FH) of bank 0 is specified as a general register:

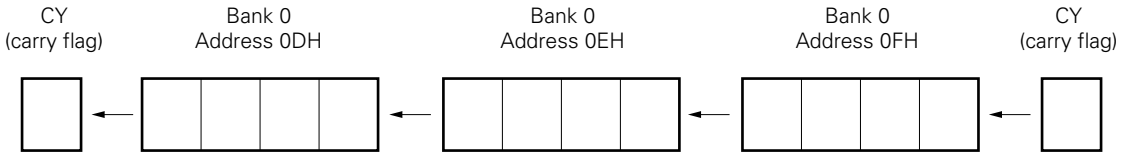
(0.0FH) ← (0.0FH) + (0.2FH)
(0.0EH) ← (0.0EH) + (0.2EH) + CY
(0.0DH) ← (0.0DH) + (0.2DH) + CY

```
MEM00D   MEM   0.0DH
MEM00E   MEM   0.0EH
MEM00F   MEM   0.0FH
```

```
MEM02D  MEM   0.2DH
MEM02E  MEM   0.2EH
MEM02F  MEM   0.2FH
        MOV   BANK, #00H        ; Data memory bank 0
        MOV   RPH, #00H         ; General register bank 0
        MOV   RPL, #00H         ; General register row address 0
        ADD   MEM00F, MEM02F
        ADDC  MEM00E, MEM02E
        ADDC  MEM00D, MEM02D
```

Example 2

Shifts the 12-bit contents for addresses 0.2DH through 0.2FH and the carry flag by 1 bit to the left, when row address 2 in bank 0 (0.20H-0.2FH) is specified as a general register.




```
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         ; General register bank 0
        MOV   RPL, #04H         ; General register row address 2
        MOV   BANK, #00H        ; Data memory bank 0
        ADDC  MEM00F, MEM02F
        ADDC  MEM00E, MEM02E
        ADDC  MEM00D, MEM02D
```

Example 3

Adds the address 0.0FH contents to the addresses 0.40H through 0.4FH contents, and stores the result in address 0.0FH:

```
                (0.0FH) ← (0.0FH) + (0.40H) + (0.41H) + ··· + (0.4FH)
MEM00F  MEM     0.0FH
MEM000  MEM     0.00H
        MOV     BANK, #00H          ; Data memory bank 0
        MOV     RPH, #00H           ; General register bank 0
        MOV     RPL, #00H           ; General register row address 0
        MOV     IXH, #00H           ; IX ← 00001000000B (0.40H)
        MOV     IXM, #04H
        MOV     IXL, #00H
LOOP1:
        SET1    IXE                 ; IXE flag ← 1
        ADD     MEM00F, MEM000
        CLR1    IXE                 ; IXE flag ← 0
        INC     IX                  ; IX ← IX + 1
        SKE     IXL, #0
        JMP     LOOP1
```

Example 4

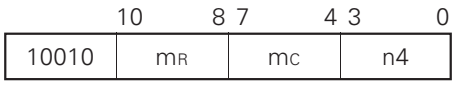
Adds the 12-bit contents for addresses 0.40H through 0.42H to the 12-bit contents for addresses 0.0DH through 0.0FH, and stores the result in 12-bit contents for addresses 0.0DH through 0.0FH:

```
                (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          ; Data memory bank 0
        MOV     RPH, #00H           ; General register bank 0
        MOV     RPL, #00H           ; General register row address 0
        MOV     IXH, #00H           ; IX ← 00001000000 (0.40H)
        MOV     IXM, #04H
        MOV     IXL, #00H
        SET1    IXE                 ; IXE flag ← 1
        ADD     MEM00D, MEM000      ; (0.0DH) ← (0.0DH) + (0.40H)
```

```
ADDC   MEM00E, MEM001   ; (0.0EH) ← (0.0EH) + (0.41H)
ADDC   MEM00F, MEM002   ; (0.0FH) ← (0.0FH) + (0.42H)
```

(4) ADDC m, #n4 **Add immediate data to data memory with carry flag**

<1> OP code



<2> Function

When CMP=0, (m) ← (m) + n4 + CY

Add immediate data to the data memory contents with carry flag (CY), and stores the result in data memory.

When CMP=1, (m) + n4 + CY

The result is not stored in the data memory, and carry flag CY and zero flag Z are changed, according to the result.

Sets carry flag CY, if a carry occurs as a result of the addition. Resets the carry flag CY, if no carry occurs.

If the addition result is other than zero, zero flag Z is reset, regardless of compare flag CMP.

If the addition result is zero, with the compare flag reset (CMP=0), the zero flag Z is set.

If the addition result is zero, with the compare flag set (CMP=1), the zero flag Z is not changed.

Addition can be executed in binary 4-bit or BCD. The BCD flag for PSWORD specifies which kind of addition is to be executed.

<3> Example 1

Adds 5 to the 12-bit contents for addresses 0.0DH through 0.0FH, and stores the result in addresses 0.0DH through 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        ; Data memory bank 0
        ADD   MEM00F, #05H
        ADDC  MEM00E, #00H
        ADDC  MEM00D, #00H
```

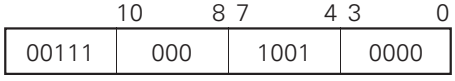
Example 2

Adds 5 to the 12-bit contents for addresses 0.4DH through 0.4FH, and stores the result in addresses 0.4DH through 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        ; Data memory bank 0
        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> OP code



<2> Function

AR ← AR + 1

Increments the address register AR contents.

<3> Example 1

Adds 1 to the 16-bit contents for AR3 through AR0 (address registers) in the system register and stores the result in AR3 through AR0:

```
    AR0 ← AR0 + 1
    AR1 ← AR1 + CY
    AR2 ← AR2 + CY
    AR3 ← AR3 + CY
INC AR
```

This instruction can be rewritten as follows, with addition instructions:

```
ADD     AR0,  #01H
ADDC    AR1,  #00H
ADDC    AR2,  #00H
ADDC    AR3,  #00H
```

Example 2

Transfers table data, 16 bits (1 address) at a time, to DBF (data buffer), using the table reference instruction (for details, refer to **10.2.3 Table Reference**):

```
; Address           Table data
010H    DW          0F3FFH
011G    DW          0A123H
012H    DW          0FFF1H
013H    DW          0FFF5H
014H    DW          0FF11H

        :
        :
        :
        :
        :

        MOV         AR3, #0H        ; Sets table data address
        MOV         AR2, #0H        ; 0010H in address register
        MOV         AR1, #1H        ;
        MOV         AR0, #0H
LOOP:
        MOVT        @AR             ; Reads table data to DBF
        :
        :
        :                           ; Table data reference processing
        :
```

```
        INC     AR              ; Increments address register by 1
        BR      LOOP
```

<4> Note

The higher 6 bits of address register are fixed to 0. Only lower 10 bits can be used.

(6) INC IX — Increment index register

<1> OP code

10	8 7	4 3	0
00111	000	1000	0000

<2> Function

IX ← IX + 1

Increments the index register IX contents.

<3> Example 1

Adds 1 to the total of 12-bit contents for IXH, IXM, and IXL (index registers) in the system register and stores the result in IXH, IXM, and IXL;

```
    ;   IXL ← IXL + 1
    ;   IXM ← IXM + CY
    ;   IXH ← IXH + CY
        INC IX
```

This program can be rewritten as follows, with addition instructions:

```
    ADD     IXL, #01H
    ADDC    IXM, #00H
    ADDC    IXH, #00H
```

Example 2

Clears all the contents for data memory addresses 0.00H through 0.73H to 0, using the index register:

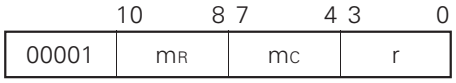
```
                MOV     IXH, #00H       ; Sets index register contents in 00H in bank 0
                MOV     IXM, #00H
                MOV     IXL, #00H
RAM clear:
MEM000          MEM     0.00H
                SET1    IXE             ; IXE flag ← 1
```

```
MOV   MEM000, #00H    ; Writes 0 to data memory indicated by index register
CLR1  IXE             ; IXE flag ← 0
INC   IX
SET2  CMP, Z          ; CMP flag ← 1, Z flag ← 1
SUB   IXL, #03H       ; Checks whether index register contents
SUBC  IXM, #07H       ; are 73H in bank 0
SUBC  IXH, #00H       ;
SKT1  Z               ; Loops until contents of index register becomes
BR    RAM clear       ; 73H of bank 0
```

18.5.2 Subtraction Instructions

(1) SUB r, m **Subtract data memory from general register**

<1> OP code



<2> Function

When CMP=0, (r) ← (r) – (m)

Subtracts the data memory contents from the general register contents, and stores the result in general register.

When CMP=1, (r) – (m)

The result is not stored in the register. Carry flag CY and zero flag Z are changed, according to the result.

Sets carry flag CY, if a borrow occurs as a result of the subtraction. Resets the carry flag, if no borrow occurs.
If the subtraction result is other than zero, zero flag Z is reset, regardless of compare flag CMP.
If the subtraction result is zero, with the compare flag reset (CMP=0), the zero flag Z is set.
If the subtraction result is zero, with the compare flag set (CMP=1), the zero flag Z is not changed.
Subtraction can be executed in binary 4-bit or BCD. The BCD flag for program status word PSWORD specifies which kind of subtraction is to be executed.

<3> Example 1

Subtracts the address 0.2FH contents from the address 0.03H contents, and stores the result in address 0.03H, when row address 0 (0.00H-0.0FH) in bank 0 is specified as a general register (RPH=0, RPL=0):

```
            (0.03H) ← (0.03H) + (0.2FH)
MEM003      MEM   0.03H
MEM02F      MEM   0.2FH
            SUB   MEM003, MEM02F
```

Example 2

Subtracts the address 0.2FH contents from the address 0.23H contents, when row address 2 (0.20H-0.2FH) in bank 0 is specified as the general register (RPH=0, RPL=4), and stores the result in address 0.23H:

```
            (0.23H) ← (0.23H) – (0.2FH)
MEM023      MEM   0.23H
MEM02F      MEM   0.2FH
            MOV   BANK, #00H          ; Data memory bank 0
            MOV   RPH, #00H           ; General register bank 0
            MOV   RPL, #04H           ; General register row address 2
            SUB   MEM023, MEM02F
```

Example 3

Subtracts the address 0.6FH contents from the address 0.03H contents and stores the result in address 0.03H. At this time, data memory address 0.6FH can be specified by selecting data memory address 2FH, if IXE=1, IXH=0, IXM=4, and IXL=0, i.e., IX=0.40H.

```
            (0.03H) ← (0.03H) + (0.6FH)
MEM003      MEM   0.03H
MEM02F      MEM   0.2FH
            MOV   BANK, #00H          ; Data memory bank 0
            MOV   RPH, #00H           ; General register bank 0
            MOV   RPL, #00H           ; General register row address 0
            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

Subtracts the address 0.3FH contents from the address 0.03H contents and stores the result in address 0.03H. At this time, data memory address 0.3FH can be specified by selecting data memory address 2FH, if IXE=1, IXH=0, IXM=1, and IXL=0, i.e., IX=0.10H.

```
                (0.03H) ← (0.03H) + (0.3FH)
MEM003  MEM     0.03H
MEM02F  MEM     0.2FH
        MOV     BANK, #00H          ; Data memory bank 0
        MOV     RPH, #00H           ; General register bank 0
        MOV     RPL, #00H           ; General register row address 0
        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> Note

The first operand for the SUB r, m instruction must be a general register address. Therefore, if the instruction is described as follows, address 03H is specified as a register:

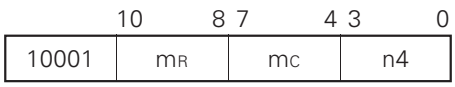
```
MEM013  MEM     0.13H
MEM02F  MEM     0.2FH
        SUB     MEM013, MEM02F
                └── General register address must be in 00H-0FH range
                    (set register pointer row address other than 1).
```

When the CMP flag=1, the subtraction result is not stored.

When the BCD flag=1, the BCD result is stored.

(2) SUB m, #n4 **Subtract immediate data from data memory**

<1> OP code



<2> Function

When CMP=0, (m) ← (m) – n4

Subtracts immediate data from the data memory contents, and stores the result in data memory.

When CMP=0, (m) – n4

The result is not stored in data memory. Carry flag CY and zero flag Z are changed, according to the result.

Sets carry flag CY, if a borrow occurs as a result of the subtraction. Resets the carry flag CY, if no borrow occurs.

If the subtraction result is other than zero, zero flag Z is reset, regardless of compare flag CMP.

If the subtraction result is zero, with the compare flag reset (CMP=0), the zero flag Z is set.

If the subtraction result is zero, with the compare flag set (CMP=1), the zero flag Z is not changed.

Subtraction can be executed in binary 4-bit or BCD. The BCD flag for program status word PSWORD specifies which kind of subtraction is to be executed.

<3> Example 1

Subtracts 5 from the address 0.2FH contents, and stores the result in address 0.2FH:

```
                (0.2FH) ← (0.2FH) – 5
MEM02F    MEM   0.2FH
          SUB   MEM02F, #05H
```

Example 2

To subtract 5 from the address 0.6FH contents and store the result in address 0.6FH. At this time, data memory address 0.6FH can be specified by selecting data memory address 2FH, if IXE=1, IXH=0, IXM=4, and IXL=0, i.e., IX=0.40H.

(0.6FH) ← (0.6FH) – 5

Address obtained as a result of ORing index register contents, 0.40H, and data memory address 0.2FH

```
MEM02F  MEM   0.2FH
        MOV   BANK, #00H        ; Data memory bank 0
        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

Subtracts 5 from the address 0.2FH contents and stores the result in address 0.2FH. At this time, data memory address 0.2FH can be specified by selecting data memory address 2FH, if IXE=1, IXH=0, IXM=0, and IXL=0, i.e., IX=0.00H.

(0.2FH) ← (0.2FH) – 5

Address obtained as a result of ORing index register contents, 0.00H, and data memory address 0.2FH

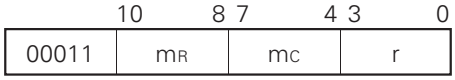
```
MEM02F  MEM   0.2FH
        MOV   BANK0, #00H       ; Data memory bank 0
        MOV   IXH, #00H         ; IX ← 00000000000B (0.00H)
        MOV   IXM, #00H         ;
        MOV   IXL, #00H         ;
        SET1  IXE               ; IXE flag ← 1
        SUB   MEM02F, #05H      ; IX                    00000000000B (0.00H)
                                ; Bank operand OR)      00000101111B (0.2FH)
                                ; Specified address     00000101111B (0.2FH)
```

<4> Note

When the CMP flag=1, the subtraction result is not stored.
When the BCD flag=1, the BCD result is stored.

(3) SUBC r, m — Subtract data memory from general register with carry flag

<1> OP code



<2> Function

When CMP=0, (r) ← (r) – (m) – CY

Subtracts the data memory contents and the value of carry flag CY from the general register contents. Stores the result in general register. By using this SUBC instruction, 2 or more words can be easily subtracted.

When CMP=1, (r) – (m) – CY

The result is not stored in the register. Carry flag CY and zero flag Z are changed, according to the result.

Sets carry flag CY, if a borrow occurs as a result of the subtraction. Resets the carry flag CY, if no borrow occurs.

If the subtraction result is other than zero, zero flag Z is reset, regardless of compare flag CMP.

If the subtraction result is zero, with the compare flag reset (CMP=0), the zero flag Z is set.

If the subtraction result is zero, with the compare flag set (CMP=1), the zero flag Z is not changed.

Subtraction can be executed in binary 4-bit or BCD. The BCD flag for program status word PSWORD specifies which kind of subtraction is to be executed.

<3> Example 1

Subtracts the 12-bit contents for addresses 0.2DH through 0.2FH from the 12-bit contents for addresses 0.0DH through 0.0FH and stores the result in 12 bits for addresses 0.0DH through 0.0FH, when row address 0 (0.00H-0.0FH) in bank 0 is specified as a general register:

(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
        SUBC  MEM00E, MEM02E
        SUBC  MEM00D, MEM02D
```

Example 2

Subtracts the 12-bit contents for addresses 0.40H through 0.42H from the 12-bit contents for addresses 0.0DH through 0.0FH, and stores the result in 12 bits for addresses 0.0DH through 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          ; Data memory bank 0
        MOV   RPH, #00H           ; General register bank 0
        MOV   RPL, #00H           ; General register row address 0
        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)
```

Example 3

Compares the 12-bit contents for addresses 0.0CH through 0.0FH with the 12-bit contents for addresses 0.00H through 0.03H. Jumps to LAB1, if the contents are the same, if not, jumps to LAB2:

```
MEM000  MEM   0.00H
MEM001  MEM   0.01H
MEM002  MEM   0.02H
MEM003  MEM   0.03H
```

```
MEM00C   MEM   0.0CH
MEM00D   MEM   0.0DH
MEM00E   MEM   0.0EH
MEM00F   MEM   0.0FH
         SET2  CMP, Z              ; CMP flag ← 1, Z flag ← 1
         SUB   MEM000, MEM00C      ; Contents for addresses 0.00H-0.03H do not change,
         SUBC  MEM001, MEM00D      ; because CMP flag is set
         SUBC  MEM002, MEM00E      ;
         SUBC  MEM003, MEM00F      ;
         SKF1  Z                   ; Z flag=1, if contents are the same; if not, Z flag=0
         BR    LAB1                ;
         BR    LAB2

   LAB1:      .
              .
              .
   LAB2:      .
              .
              .
```

(4) SUBC m, #n4 — Subtract immediate data from data memory with carry flag

<1> OP code

10	8 7	4 3	0
10011	m_R	m_C	n4

<2> Function

When CMP=0, (m) ← (m) – n4 – CY

Subtracts immediate data and the value of carry flag CY from the data memory contents, and stores the result in data memory.

When CMP=1, (m) – n4 – CY

The result is not stored in the data memory. Carry flag CY and zero flag Z are changed, according to the result.

Sets carry flag CY, if a borrow occurs as a result of the subtraction. Resets the carry flag CY, if no borrow occurs.

If the subtraction result is other than zero, zero flag Z is reset, regardless of compare flag CMP.

If the subtraction result is zero, with the compare flag reset (CMP=0), the zero flag Z is set.
If the subtraction result is zero, with the compare flag set (CMP=1), the zero flag Z is not changed.
Subtraction can be executed in binary or BCD. The BCD flag for program status word PSWORD specifies which kind of subtraction is to be executed.

<3> Example 1

Subtracts 5 from the 12-bit contents for addresses 0.0DH through 0.0FH and stores the result in addresses 0.0DH through 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

To subtract 5 from the 12-bit contents for addresses 0.4DH through 0.4FH and store the result in addresses 0.4DH through 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          ; Data memory bank 0
        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
```

Example 3

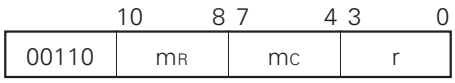
Compares the 12-bit contents for addresses 0.00H through 0.03H with immediate data 0A3FH.
Jumps to LAB1, if the contents are the same; if not, jumps to LAB2:

```
MEM000   MEM   0.00H
MEM001   MEM   0.01H
MEM002   MEM   0.02H
MEM003   MEM   0.03H
         SET2  CMP, Z          ; CMP flag ← 1, Z flag ← 1
         SUB   MEM000, #0H     ; Contents for addresses 0.00H–0.03H do not
         SUBC  MEM001, #0AH    ; change, because CMP flag is set
         SUBC  MEM002, #3H     ;
         SUBC  MEM003, #0FH    ;
         SKF1  Z               ; Z flag=1, if contents are the same; if not, Z flag=0
         BR    LAB1            ;
         BR    LAB2
                 :
LAB1:            :
                 :
                 :
LAB2:            :
                 :
```

18.5.3 Logical Operation Instructions

(1) OR r, m **OR between general register and data memory**

<1> OP code



<2> Function

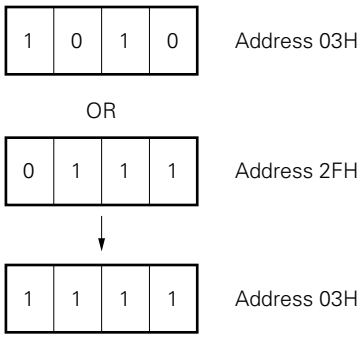
(r) ← (r) ∨ (m)

ORs the general register contents with data memory contents. Stores the result in general register.

<3> Example 1

To OR the address 0.03H contents (1010B) and the address 0.2FH contents (0111B) and store the result (1111B) in address 0.03H:

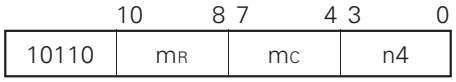
(0.03H) ← (0.03H) ∨ (0.2FH)




```
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> OP code



<2> Function

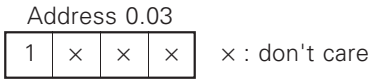
(m) ← (m) ∨ n4

ORs the data memory contents and immediate data. Stores the result in data memory.

<3> Example 1

To set bit 3 (MSB) for address 0.03H:

(0.03H) ← (0.03H) ∨ 1000B




```
MEM003   MEM   0.03H
         OR    MEM003, #1000B
```

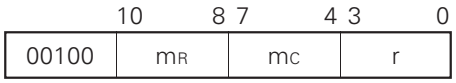
Example 2

Sets all the bits for address 0.03H:

```
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> OP code



<2> Function

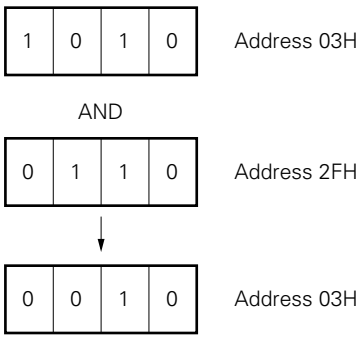
(r) ← (r) ∧ (m)

ANDs the general register contents with data memory contents and stores the result in general register.

<3> Example 1

ANDs the address 0.03H (1010B) contents and the address 0.2FH (0110B) contents. Stores the result (0010B) in address 0.03H:

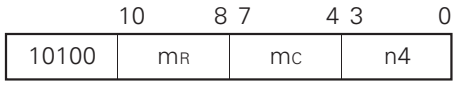
(0.03H) ← (0.03H) ∧ (0.2FH)




```
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> OP code



<2> Function

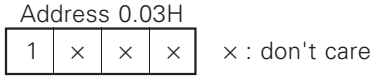
(m) ← (m) ∧ n4

ANDs the data memory contents and immediate data. Stores the result in data memory.

<3> Example 1

To reset bit 3 (MSB) for address 0.03H.

(0.03H) ← (0.03H) ∧ 0111B




```
MEM003   MEM   0.03H
         AND   MEM003, #0111B
```

Example 2

To reset all the bits for address 0.03H:

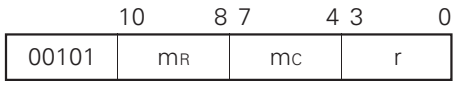
```
MEM003   MEM   0.03H
         AND   MEM003, #0000B
```

or,

```
MEM003   MEM   0.03H
         MOV   MEM003, #00H
```

(5) XOR r, m **Exclusive OR between general register and data memory**

<1> OP code



<2> Function

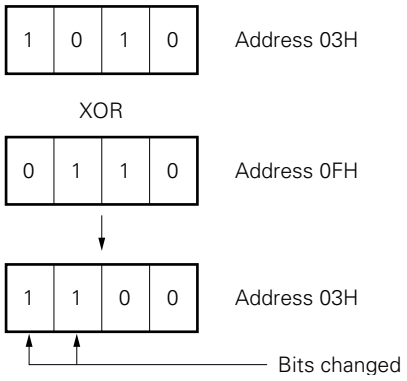
(r) ← (r) $\forall$ (m)

Exclusive-ORs (XOR) the general register contents with data memory contents. Stores the result in general register.

<3> Example 1

Compares the address 0.03H contents and the address 0.0FH contents. If different bits are found, set and store them in address 0.03H. If all the bits in address 0.03H are reset (i.e., the address 0.03H contents are the same as those for address 0.0FH), jumps to LBL1; otherwise, jumps to LBL2.

This example is to compare the status of an alternate switch (address 0.03H contents) with the internal status (address 0.0FH contents) and to branch to changed switch processing.




```
MEM003   MEM   0.03H
MEM00F   MEM   0.0FH
         XOR   MEM003, MEM00F
         SKNE  MEM003, #00H
         BR    LBL1
         BR    LBL2
```

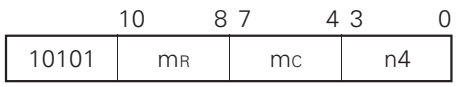
Example 2

Clears the address 0.03H contents:


```
MEM003   MEM   0.03H
         XOR   MEM003, MEM003
```

(6) XOR m, #n4 **Exclusive OR between data memory and immediate data**

<1> OP code



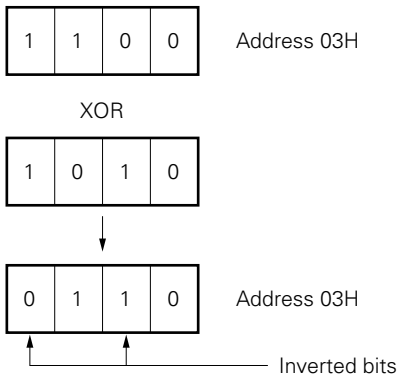
<2> Function

(m) ← (m) ∀ n4

Exclusive-ORs the data memory contents and immediate data. Stores the result in data memory.

<3> Example

Inverts bits 1 and 3 in address 0.03H and store the result in address 03H:

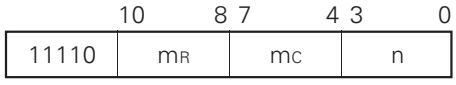



```
MEM003    MEM    0.03H
          XOR    MEM003, #1010B
```

18.5.4 Judgment Instruction

(1) SKT m, #n **Skip next instruction if data memory bits are true**

<1> OP code



<2> Function

CMP ← 0, if (m) ∧ n=n, then skip

Skip the next one instruction, if the result of ANDing the data memory contents and immediate data n is equal to n (Executes as NOP instruction)

<3> Example 1

Jumps to AAA, if bit 0 in address 03H is 1; if it is 0, jumps to BBB:

```
SKT     03H, #0001B
BR      BBB
BR      AAA
```

Example 2

Skips the next instruction, if both bits 0 and 1 in address 03H are 1.

```
SKT     03H, #0011B
```

	b_3	b_2	b_1	b_0	
Skip condition 03H	×	×	1	1	× : don't care

Example 3

The results of executing the following two instructions are the same:

```
SKT     13H, #1111B
SKE     13H, #0FH
```

(2) SKF m, #n **Skip next instruction if data memory bits are false**

<1> OP code

10	8 7	4 3	0
11111	m_R	m_C	n

<2> Function

CMP ← 0, if (m) ∧ n=0, then skip

Skips the next one instruction, if the result of ANDing the data memory contents and immediate data n is 0 (Executes as NOP instruction).

<3> Example 1

Stores immediate data 00H to address 0FH in the data memory content, if bit 2 in address 13H is 0; if it is 1, jumps to ABC:

```
MEM013  MEM   0.13H
MEM00F  MEM   0.0FH
        SKF   MEM013, #0100B
        BR    ABC
        MOV   MEM00F, #00H
```

Example 2

Skips the next instruction, if both bits 3 and 0 in address 29H are 0.

```
SKF     29H, #1001B
```

		b_3	b_2	b_1	b_0	
Skip condition	29H	0	×	×	0	× : don't care

Example 3

The results of executing the following two instructions are the same:

```
SKF     34H, #1111B
SKE     34H, #00H
```

18.5.5 Comparison Instructions

(1) SKE m, #n4 **Skip if data memory equal to immediate data**

<1> OP code

10	8 7	4 3	0
01001	m_R	m_C	n4

<2> Function

(m) –n4, skip if zero

Skip the next one instruction, if the data memory contents are equal to the immediate data value (Executes as NOP instruction).

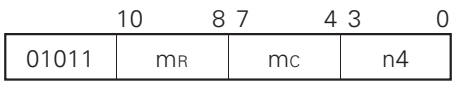
<3> Example

To transfer 0FH to address 24H, if the address 24H contents are 0; if not, jumps to OPE1:

```
MEM024  MEM   0.24H
        SKE   MEM024, #00H
        BR    OPE1
        MOV   MEM024, #0FH
OPE1    :
```

(2) SKNE m, #n4 **Skip if data memory not equal to immediate data**

<1> OP code



<2> Function

(m) –n4, skip if not zero

Skips the next one instruction, if the data memory contents are not equal to the immediate data value (Executes as NOP instruction).

<3> Example

Jumps to XYZ, if the address 1FH contents are 1 and the address 1EH contents are 3; otherwise, jumps to ABC.

To compare 8-bit data, this instruction is used in the following combination:


```
MEM01E   MEM   0.1EH
MEM01F   MEM   0.1FH
         SKNE  MEM01F, #01H
         SKE   MEM01E, #03H
         BR    ABC
         BR    XYZ
```

The above program can be rewritten as follows, using compare and zero flags:

```
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 data memory greater than or equal to immediate data**

<1> OP code

10	8	7	4	3	0
11001	mR		mC	n4	

<2> Function

(m) –n4, skip if not borrow

Skips the next one instruction, if the data memory contents are equal to or greater than the immediate data value (Executes as NOP instruction).

<3> Example

Executes RET, if 8-bit data stored in addresses 1FH (higher) and 2FH (lower) is greater than immediate data '17H'; if not, executes RETSK:

```
MEM01E  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 data memory less than immediate data**

<1> OP code

10	8	7	4	3	0
11011	mR		mC	n4	

<2> Function

(m) –n4, skip if borrow

Skips the next one instruction, if the data memory contents are less than the immediate data value (Executes as NOP instruction).

<3> Example

Stores 01H in address 0FH, if the address 10H contents are greater than immediate data '6'; if not, stores 02H in address 0FH:

```
MEM00F   MEM   0.0FH
MEM010   MEM   0.10H
         MOV   MEM00F, #02H
         SKLT  MEM010, #06H
         MOV   MEM00F, #01H
```

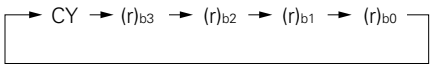
18.5.6 Rotation Instructions

(1) RORC r **Rotate right general register with carry flag**

<1> OP code

			3 0
00111	000	0111	r

<2> Function



Rotates the contents of general register indicated by r including carry flag to the right by 1 bit.

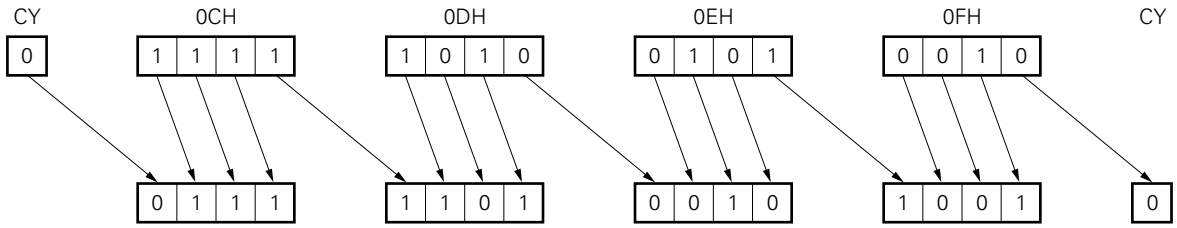
<3> Example 1

When row address 0 of bank 0 (0.00H-0.0FH) is specified as general register (RPH=0, RPL=0), rotate the value of address 0.00H (1000B) to the right by 1 bit to make it 0100B.

```
         (0.00H) ← (0.00H)÷2
MEM000   MEM   0.00H
         MOV   RPH, #00H          ; General register bank 0
         MOV   RPL, #00H          ; General register row address 0
         CLR1  CY                 ; CY flag ← 0
         RORC  MEM000
```

Example 2

When row address 0 of bank 0 (0.00H-0.0FH) is specified as general register (RPH=0, RPL=0), rotate the data buffer DBF contents 0FA52H to the right by 1 bit to make DBF contents 7D29H.




```
MEM00C   MEM   0.0CH
MEM00D   MEM   0.0DH
MEM00E   MEM   0.0EH
MEM00F   MEM   0.0FH
         MOV   RPH, #00H          ; General register bank 0
         MOV   RPL, #00H          ; General register row address 0
         CLR1  CY                 ; CY flag ← 0
         RORC  MEM00C
         RORC  MEM00D
         RORC  MEM00E
         RORC  MEM00F
```

18.5.7 Transfer Instructions

(1) LD r, m **Load data memory to general register**

<1> OP code

10	8 7	4 3	0
01000	m_R	m_C	r

<2> Function

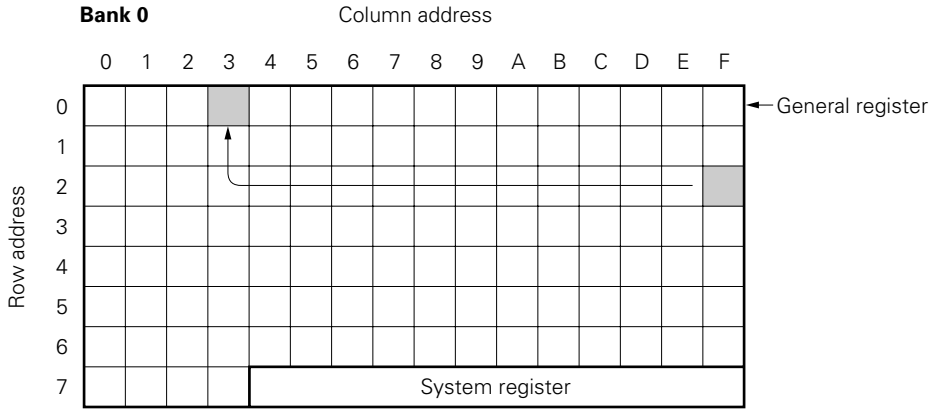
(r) ← (m)

Stores the data memory contents to general register.

<3> Example 1

To store the address 0.2FH contents to address 0.03H:

```
                 (0.03H) ← (0.2FH)
MEM003   MEM   0.03H
MEM02F   MEM   0.2FH
         LD    MEM003, MEM02F
```



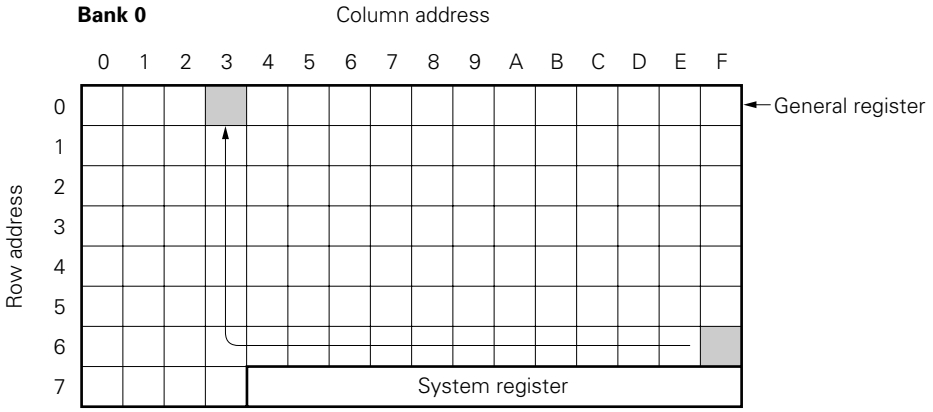
Example 2

Stores the address 0.6FH contents to address 0.03H. At this time, data memory address 0.6FH can be specified by selecting data memory address 2FH, if IXE=1, IXH=0, IXM=4, and IXL=0, i.e., IX=0.40H.

IXH ← 00H
IXM ← 04H
IXL ← 00H
IXE flag ← 1
(0.03H) ← (0.6FH)

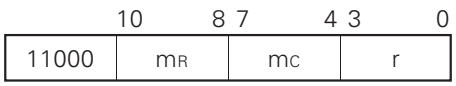
Address obtained as result of ORing index register contents, 040H, and data memory contents, 0.2FH

```
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> OP code



<2> Function

(m) ← (r)

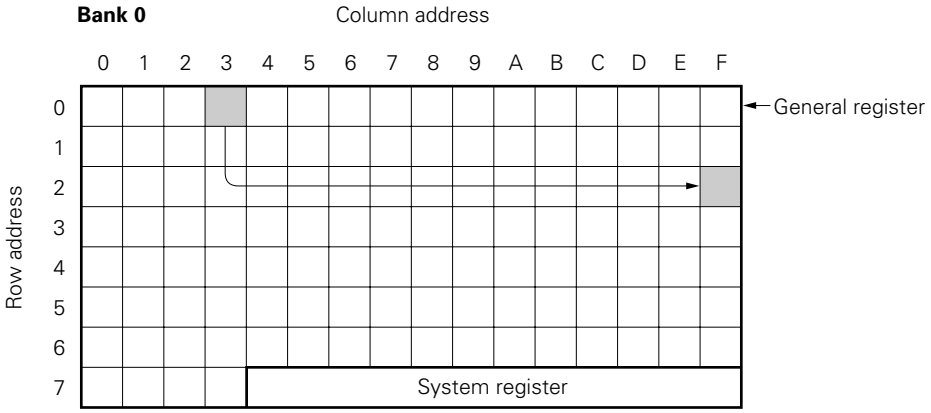
Stores the general register contents to data memory.

<3> Example 1

Stores the address 0.03H contents to address 0.2FH:

(0.2FH) ← (0.03H)

```
ST      2FH, 03H        ; Transfer general register contents to data memory
```



Example 2

Stores the address 0.00H contents to addresses 0.18H through 0.1FH. The data memory addresses (18H-1FH) are specified by the index register.

(0.18H) ← (0.00H)

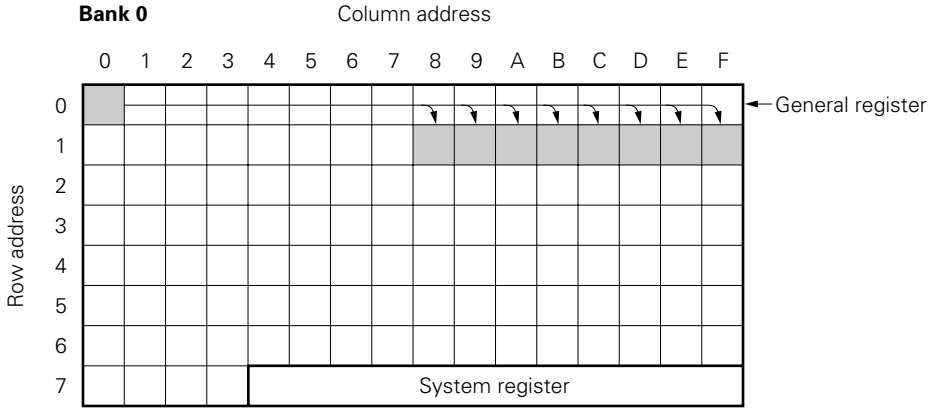
(0.19H) ← (0.00H)

⋮

(0.1FH) ← (0.00H)

```
        MOV     IXH, #00H       ; IX ← 00000000000B (0.00H)
        MOV     IXM, #00H
        MOV     IXL, #00H       ; Specifies data memory address 0.00H
```

```
MEM018   MEM   0.18H
MEM000   MEM   0.00H
   LOOP1:
         SET1  IXE                ; IXE flag ← 1
         ST    MEM018, MEM000     ; (0.1×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> OP 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)

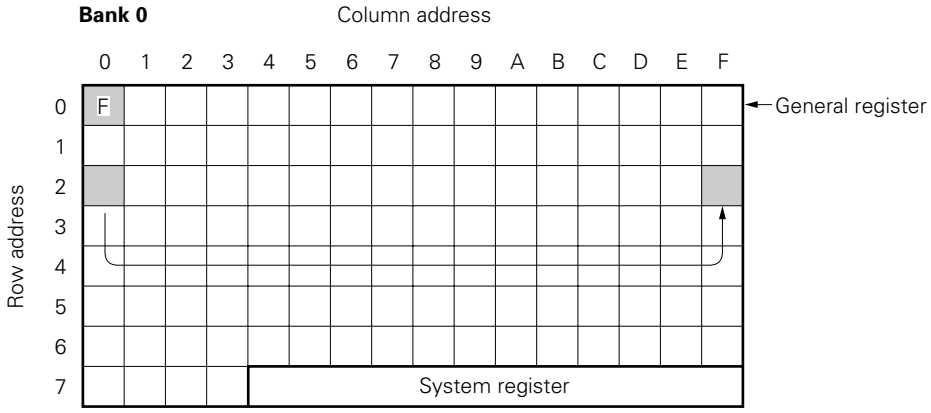
Stores the data memory contents to the data memory addressed by the general register contents.

When MPE=0, transfer is performed in the same row address in the same bank.

<3> Example 1

Stores the address 0.20H contents to address 0.2FH with the MPE flag cleared to 0. The transfer destination data memory address is at the same row address as the transfer source, and the column address is specified by the general register contents at address 0.00H.

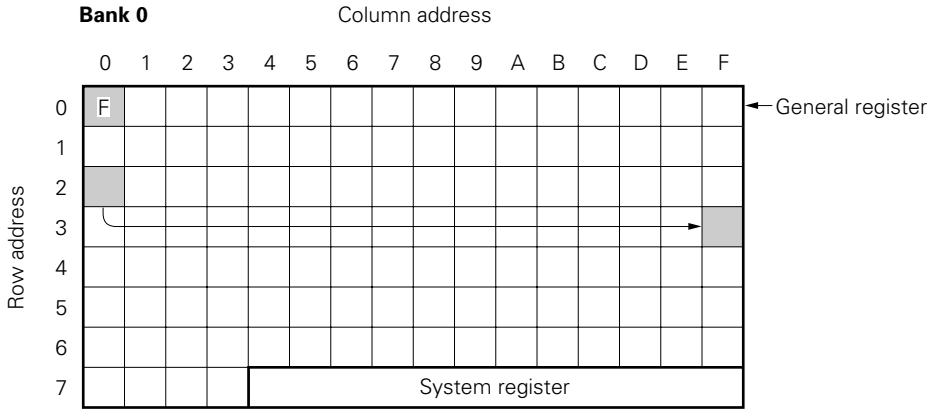
```
                (0.2FH) ← (0.20H)
MEM000  MEM     0.00H
MEM020  MEM     0.20H
        CLR1    MPE                 ; MPE flag ← 0
        MOV     MEM000, #0FH        ; Sets column address in general register
        MOV     @MEM000, MEM020     ; Store
```



Example 2

Stores the address 0.20H contents to address 0.3FH, with the MPE flag set to 1. The row address for the transfer destination data memory address is specified by the memory pointer MP contents. The column address is specified by the general register contents at address 0.00H.

```
                (0.3FH) ← (0.20H)
MEM000  MEM     0.00H
MEM020  MEM     0.20H
        MOV     RPH, #00H           ; General register bank 0
        MOV     RPL, #00H           ; General register row address 0
        MOV     00H, #0FH           ; Sets column address in general register
        MOV     MPH, #00H           ; Sets row address in memory pointer
        MOV     MPL, #03H           ;
        SET1    MPE                 ; MPE flag ← 1
        MOV     @MEM000, MEM020     ; Store
```



(4) MOV m, @r **Move data memory to destination indirect**

<1> OP 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))

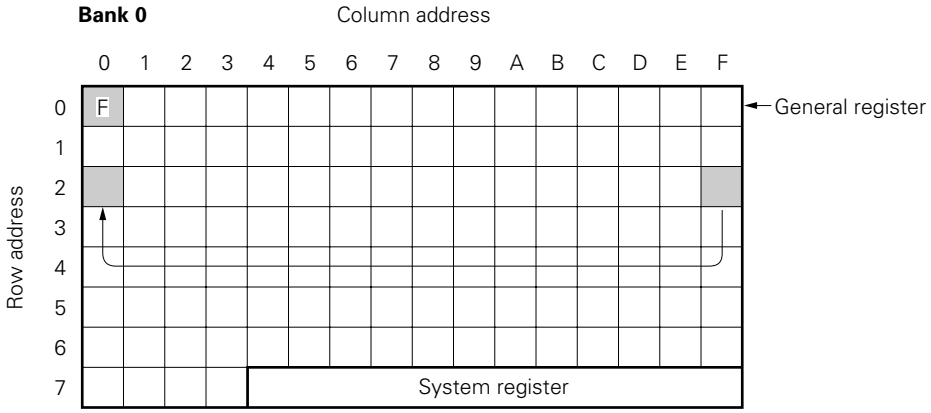
Stores the data memory contents addressed by the general register contents to data memory.

When MPE=0, transfer is performed in the same row address in the same bank.

<3> Example 1

Stores the address 0.2FH contents to address 0.20H, with the MPE falg cleared to 0. The transfer destination data memory address is at the same row address as the transfer source. The column address is specified by the general register contents at address 0.00H.

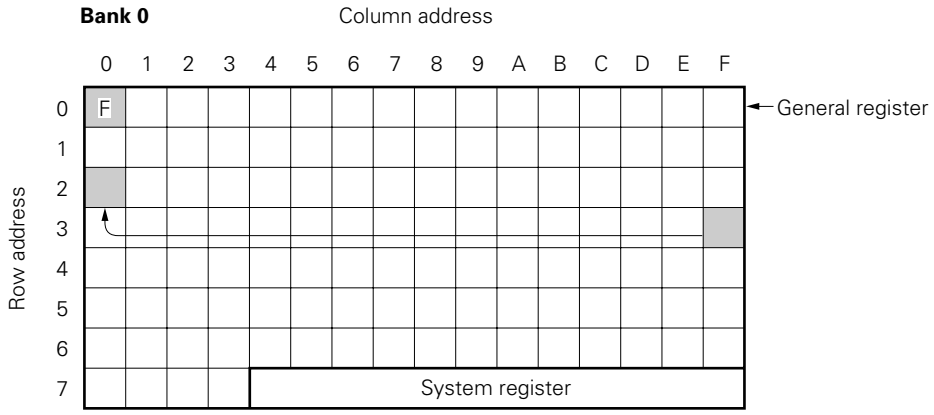
```
                (0.20H) ← (0.2FH)
MEM000  MEM     0.00H
MEM020  MEM     0.20H
        CLR1    MPE                 ; MPE flag ← 0
        MOV     MEM000, #0FH        ; Sets column address in general register
        MOV     MEM020, @MEM000     ; Store
```



Example 2

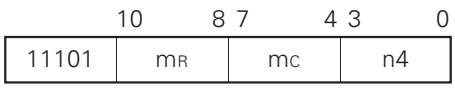
Stores the address 0.3FH contents to address 0.20H, with the MPE flag set to 1. The row address for the transfer source data memory is specified by the memory pointer MP contents. The column address is specified by the general register contents at address 0.00H.

```
                (0.20H) ← (0.3FH)
MEM000  MEM     0.00H
MEM020  MEM     0.20H
        MOV     MEM000, #0FH        ; Sets column address in general register
        MOV     MPH, #00H           ; Sets row address in memory pointer
        MOV     MPL, #03H           ;
        SET1    MPE                 ; MPE flag ← 1
        MOV     MEM020, @MEM000     ; Store
```



(5) MOV m, #n4 **Move immediate data to data memory**

<1> OP code



<2> Function

(m) ← n4

Stores immediate data to data memory.

<3> Example 1

Stores immediate data 0AH to data memory address 0.50H:

```
                (0.50H) ← 0AH
MEM050  MEM     0.50H
        MOV     MEM050, #0AH
```

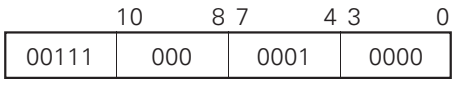
Example 2

Stores immediate data 07H to address 0.32H, when data memory address 0.00H is specified with IXH=0, IXM=3, IXL=2, and IXE flag=1:

```
                (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> OP code



<2> Function

SP ← SP–1, ASR ← PC, PC ← AR,

DBF ← (PC), PC ← ASR, SP ← SP+1

Stores the program memory contents, addressed by address register AR, to data buffer DBF.
Since this instruction temporarily uses one stack level, pay attention to nesting such as subroutines and interrupts.

<3> Example

To transfer 16 bits of table data, specified by the values for address registers AR3, AR2, AR1, and AR0 in the system register, to data buffers DBF3, DBF2, DBF1, and DBF0:

```
                ; *
                ; ** Table data
                ; *
Address         ORG     0010H
0010H           DW      0000000000000000B   ; (0000H)
0011H           DW      1010101111001101B   ; (0ABCDH)
                              :
                              :
                              :


                ; *
                ; ** Table reference program
                ; *
                MOV     AR3, #00H           ; AR3 ← 00H     Sets 0011H in address register
                MOV     AR2, #00H           ; AR2 ← 00H
                MOV     AR1, #01H           ; AR1 ← 01H
                MOV     AR0, #01H           ; AR0 ← 01H
                MOVT    DBF, @AR            ; Transfers address 0011H data to DBF
```

In this case, the data are stored in DBF, as follows:

DBF3=0AH
DBF2=0BH
DBF1=0CH
DBF0=0DH

(7) PUSH AR **Push address register**

<1> OP code

00111	000	1101	0000

<2> Function

SP ← SP–1,

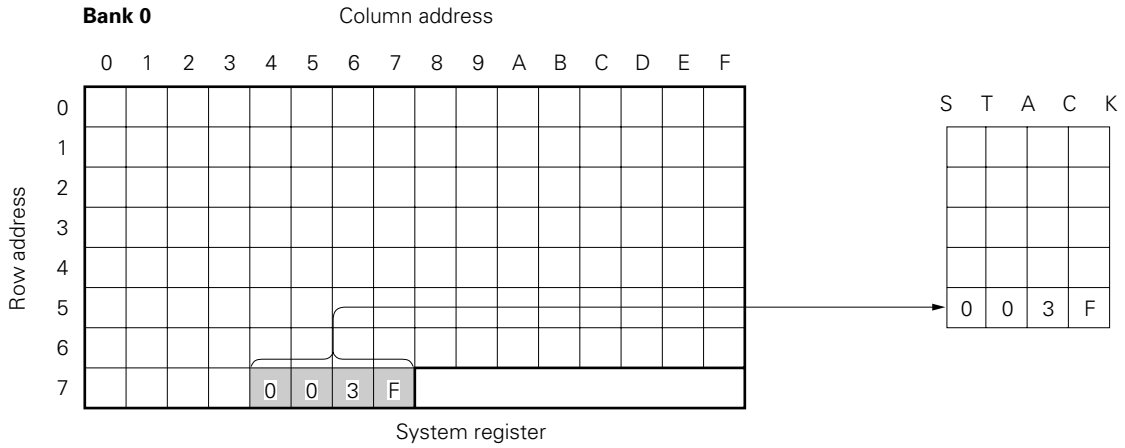
ASR ← AR

Decrements stack pointer SP and stores the address register AR value to address stack register specified by stack pointer.

<3> Example 1

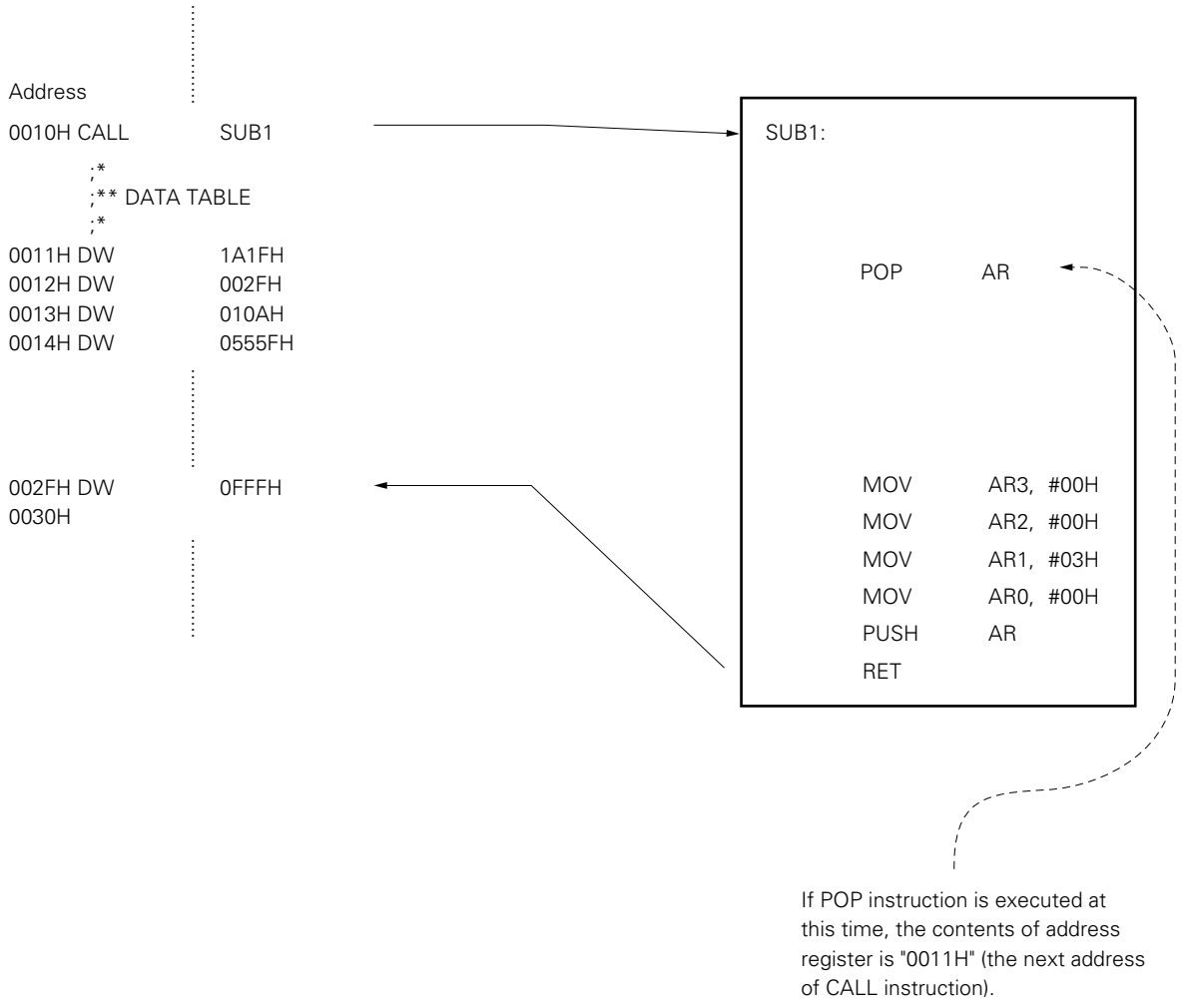
Sets 003FH in address register and stores it in stack:

```
MOV     AR3, #00H
MOV     AR2, #00H
MOV     AR1, #03H
MOV     AR0, #0FH
PUSH    AR
```



Example 2

Sets the return address for a subroutine in the address register. Returns execution, if a data table exists after a subroutine:



(8) POP AR — Pop address register

<1> OP code

00111	000	1100	0000

<2> Function

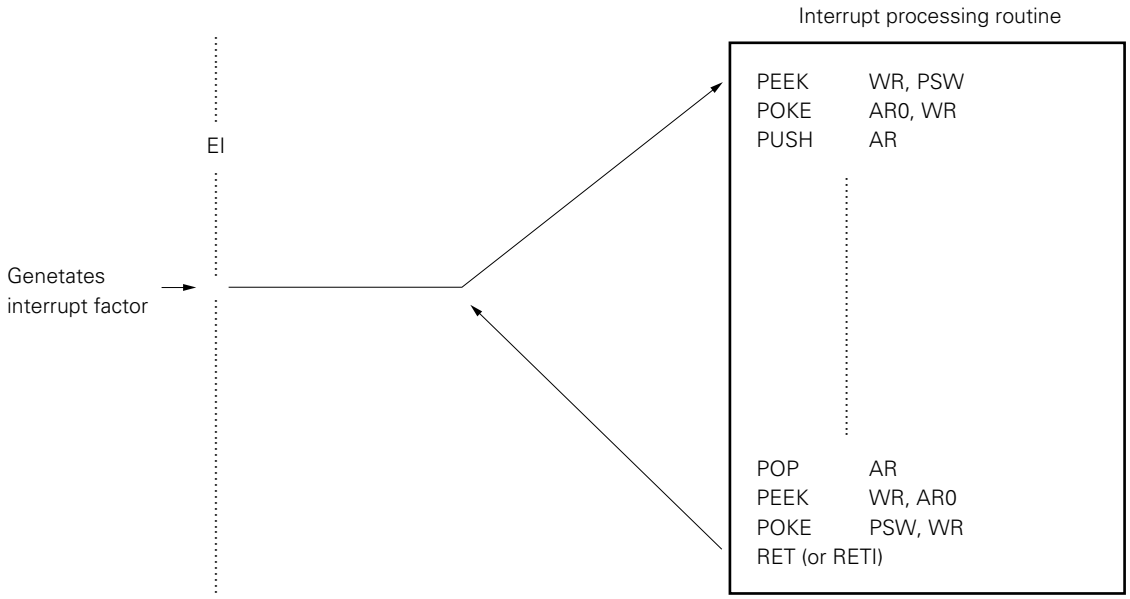
AR ← ASR,

SP ← SP+1

Pops the contents of address stack register indicated by stack pointer to address register AR and then increments stack pointer SP.

<3> Example

If the PSW contents are changed, while an interrupt processing routine is being executed, the PSW contents are transferred to the address register through WR at the beginning of the interrupt processing and saved to address stack register by the PUSH instruction. Before the execution returns from the interrupt routine, the address register contents are restored through WR to PSW by the POP instruction.



(9) PEEK WR, rf **Peek register file to window register**

<1> OP code

00111	rfR	0011	rfC

<2> Function

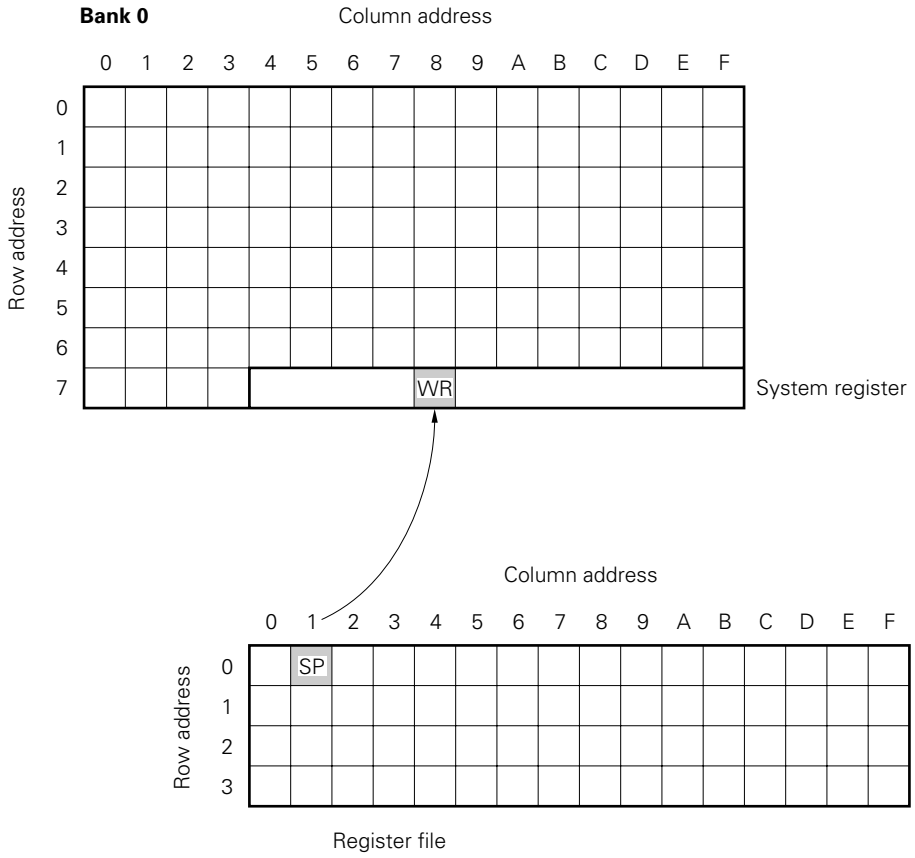
WR ← (rf)

Stores the register file contents to window register WR.

<3> Example

Stores the stack pointer SP contents at address 01H in the register file to the window register:

PEEK WR, SP



(10) POKE rf, WR — Poke window register to register file

<1> OP code

10	8 7	4 3	0
00111	rf_R	0010	rf_C

<2> Function

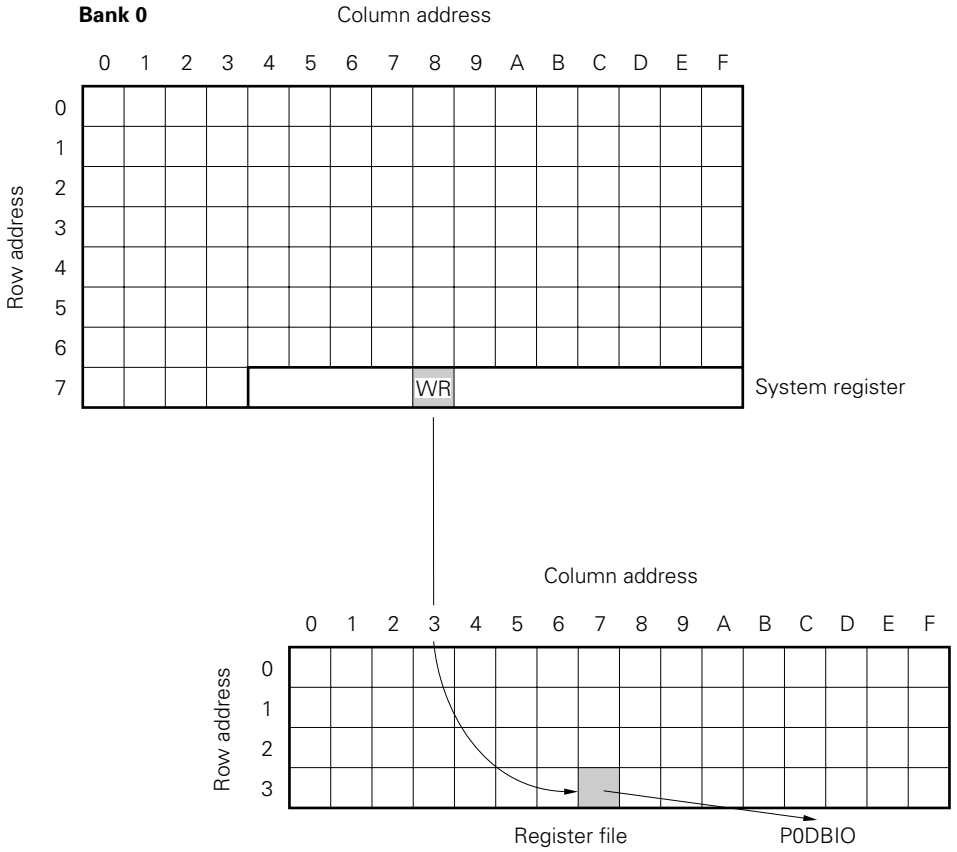
(rf) ← WR

Stores the window register WR contents to register file.

<3> Example 1

Stores immediate data 0FH to P0DBIO for the register file through the window register:

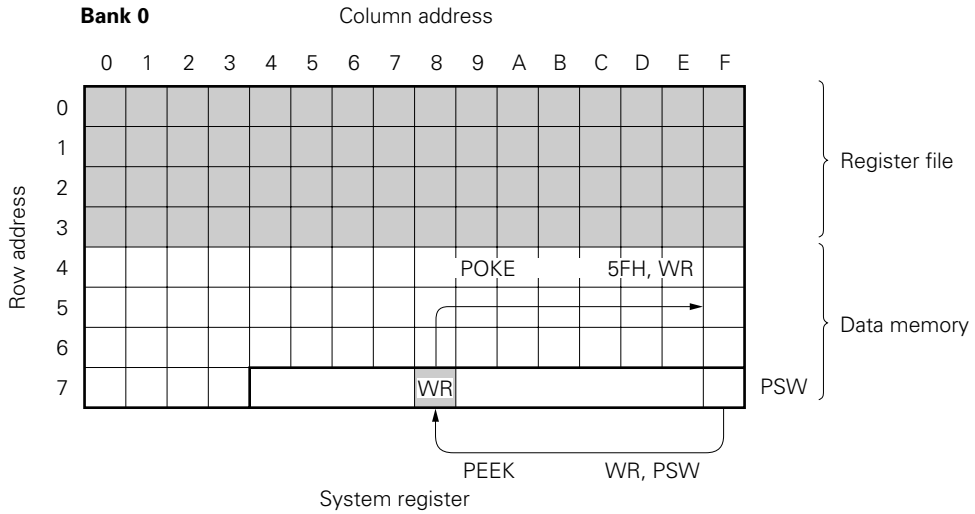
```
MOV     WR, #0FH
POKE    P0DBIO, WR          ; Sets all of P0D0, P0D1, P0D2, and P0D3 in output mode
```



<4> Note

Among register files, data memories can be seen at 40H-7FH (74H-7FH is system register). Therefore, the PEEK and POKE instructions can access addresses 40H through 7FH in each data memory bank, in addition to the register file. For example, these instructions can be used as follows:

```
MEM05F   MEM   0.5FH
         PEEK  WR, PSW          ; Stores PSW (7FH) contents in system register to WR
         POKE  MEM05F, WR       ; Stores WR contents to address 5FH in data memory
```



(11) GET DBF, p **Get peripheral data to data buffer**

<1> OP code

10	8 7	4 3	0
00111	P_H	1011	P_L

<2> Function

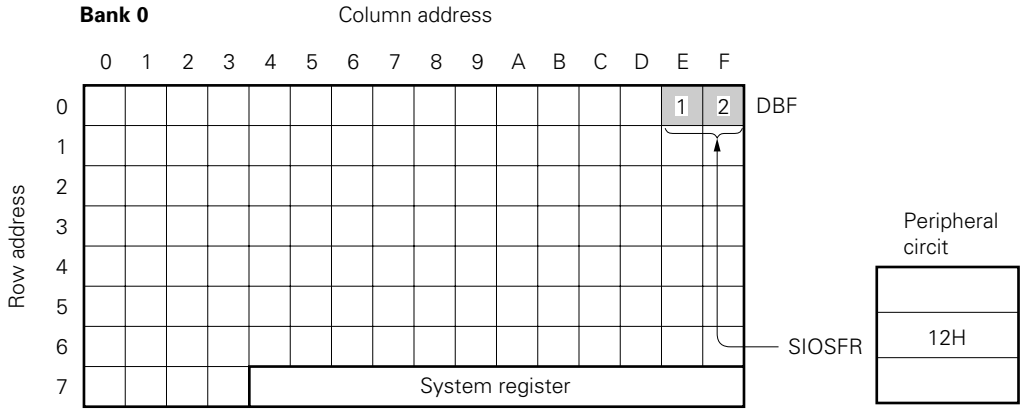
DBF ← (p)

Stores the peripheral register contents to data buffer DBF.

<3> Example 1

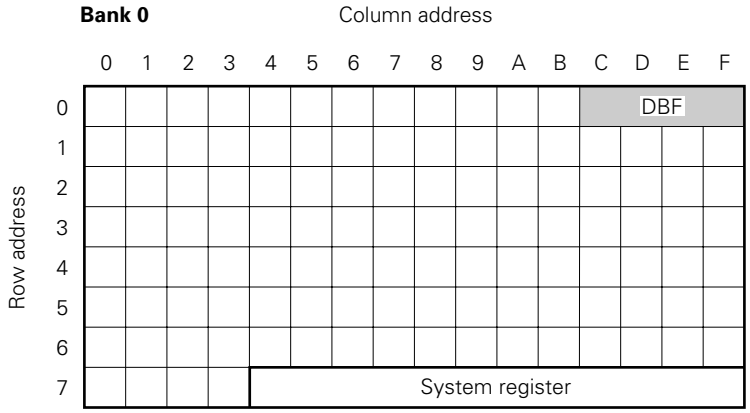
Stores the 8-bit contents for shift register SIOSFR in the serial interface to data buffers DBF0 and DBF1:

```
GET    DBF, SIOSFR
```



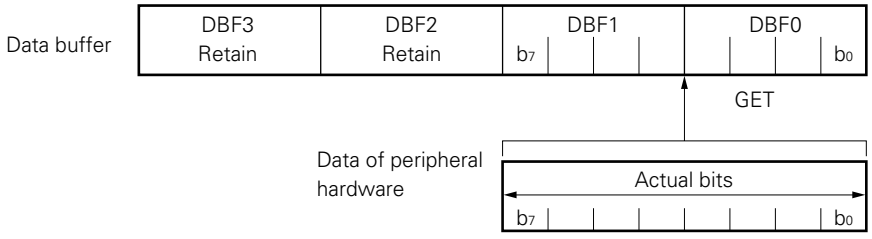
<4> Note 1

The data buffer is assigned to addresses 0CH, 0DH, 0EH, and 0FH in bank 0 for the data memory, regardless of the bank register value.



Note 2

Up to 16 bits in the data buffer are available. When a peripheral circuit is accessed by the GET instruction, the number of bits, by which the circuit is to be accessed, differs depending on the circuit. For example, if the GET instruction is executed to access a peripheral circuit, which should be accessed in 8-bit units, data is stored in the lower 8 bits for the data buffer DBF (DBF1, DBF0).



(12) PUT p, DBF **Put data buffer to peripheral**

<1> OP code

10	8 7	4 3	0
00111	P_H	1010	P_L

<2> Function

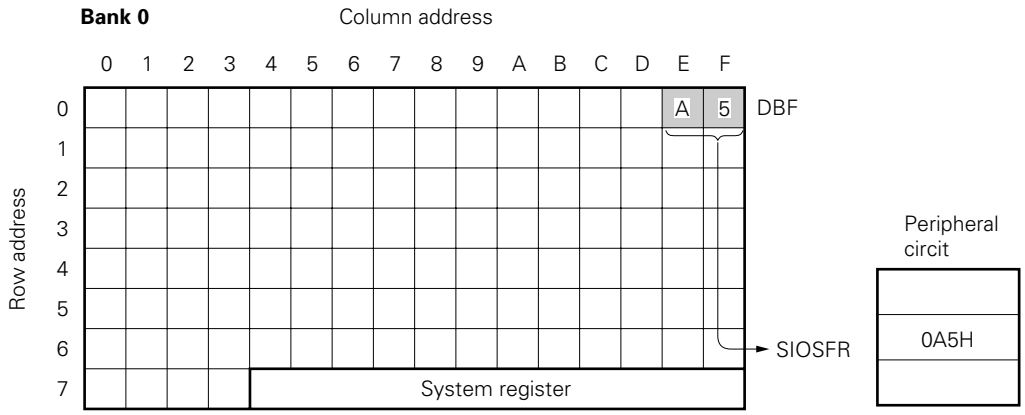
(p) ← DBF

Stores the data buffer DBF contents to peripheral register.

<3> Example

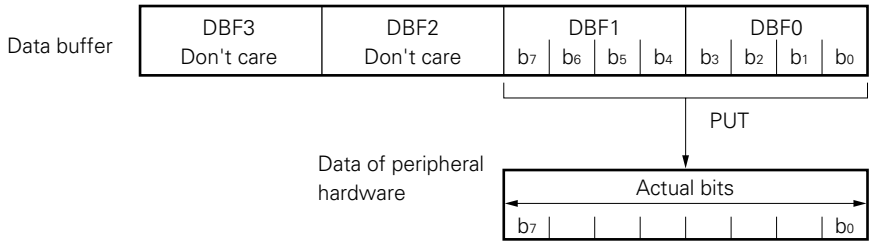
Sets 0AH and 05H to data buffers DBF1 and DBF0, respectively, and transfers them to a peripheral register, shift register (SIOSFR) for serial interface:

```
MOV     BANK,   #00H    ; Data memory bank 0
MOV     DBF0,   #05H
MOV     DBF1,   #0AH
PUT     SIOSFR
        DBF
```



<4> Note

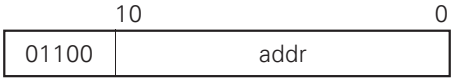
Up to 16 bits in the data buffer are available. When a peripheral circuit is accessed by the PUT instruction, the number of bits, by which the circuit is to be accessed, differs depending on the circuit. For example, if the PUT instruction is executed to access the shift register SIO, which should be accessed in 8-bits units, only the lower 8 bits for the data buffer DBF (DBF1, DBF0) are transferred (DBF3 and DBF2 are not transferred).



18.5.8 Branch Instructions

(1) BR addr **Branch to the address**

<1> OP code



<2> Function

PC ← addr

Branches to an address specified by addr.

<3> Example

```
FLY     LAB     0FH         ; Defines FLY=0FH
          :
          :
        BR      FLY         ; Jumps to address 0FH
          :
          :
        BR      LOOP1       ; Jumps to LOOP1
          :
          :
        BR      $+2         ; Jumps to an address 2 addresses lower than current address
          :
          :
        BR      $–3         ; Jumps to an address 3 addresses higher than current address
          :
          :
LOOP1:
```

(2) BR @AR — Branch to the address specified by address register

<1> OP code

00111	000	0100	0000

<2> Function

PC ← AR

Branches to the program address, specified by address register AR.

<3> Example 1

Sets 003FH in address register AR (AR0-AR3) and jumps to address 003FH by using the BR @AR instruction:

```
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

Changes the branch destination according to the data memory address 0.10H contents, as follows:

```
    0.10H contents      Branch destination label
        00H       →     AAA
        01H       →     BBB
        02H       →     CCC
        03H       →     DDD
        04H       →     EEE
        05H       →     FFF
        06H       →     GGG
        07H       →     HHH
      08H-0FH     →     ZZZ
      ; *
      ; ** Jump table
Address ; *
0010H   BR      AAA
0011H   BR      BBB
0012H   BR      CCC
```

```
0013H   BR      DDD
0014H   BR      EEE
0015H   BR      FFF
0016H   BR      GGG
0017H   BR      HHH
0018H   BR      ZZZ
                  :
                  :
                  :
MEM010  MEM     0.10H
        MOV     RPH,    #00H      ; General register bank 0
        MOV     RPL,    #02H      ; General register row address 1
        MOV     AR3,    #00H      ; AR3 ← 00H Sets AR to 001×H
        MOV     AR2,    #00H      ; AR2 ← 00H
        MOV     AR1,    #01H      ; AR1 ← 01H
        ST      AR0,    #MEM010   ; AR0 ← 0.10H
        SKF     AR0,    #1000B    ; Sets 08H in AR0, if AR0 contents are greater than 08H
        AND     AR0,    #1000B    ;
        BR      @AR
```

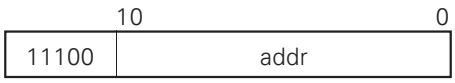
<4> Note

The higher 6 bits of address register are fixed to 0. Only lower 10 bits can be used.

18.5.9 Subroutine Instructions

(1) CALL addr **Call subroutine**

<1> OP code

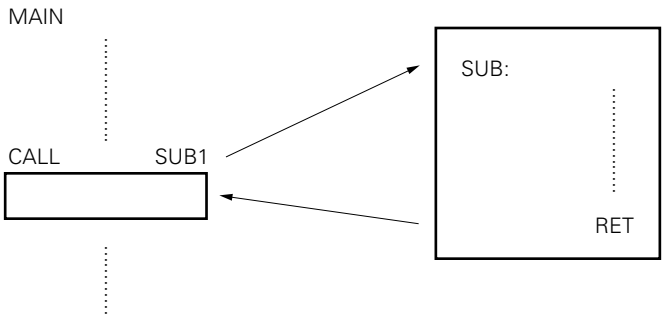


<2> Function

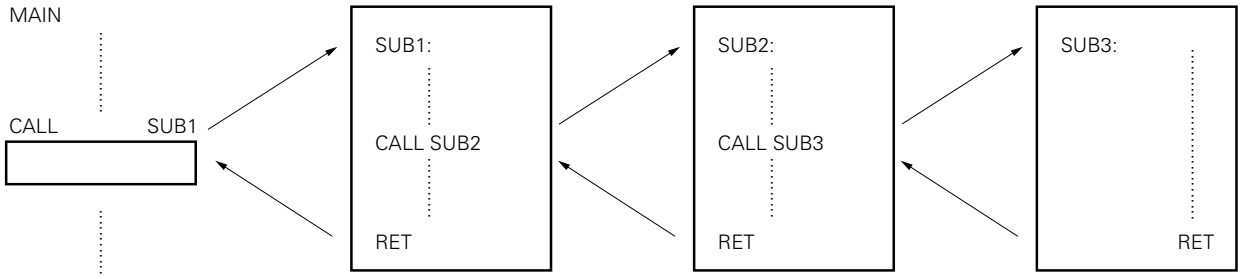
SP ← SP–1, ASR ← PC,
PC ← addr

Increments the program counter PC value, stores it to stack, and branches to a subroutine specified by addr.

<3> Example 1



Example 2



(2) CALL @AR **Call subroutine specified by address register**

<1> OP code

00111	000	0101	0000

<2> Function

SP ← SP–1,
ASR ← PC,
PC ← AR

Increments and saves to the stack the program counter PC value, and branches to a subroutine that starts from the address specified by address register AR.

<3> Example 1

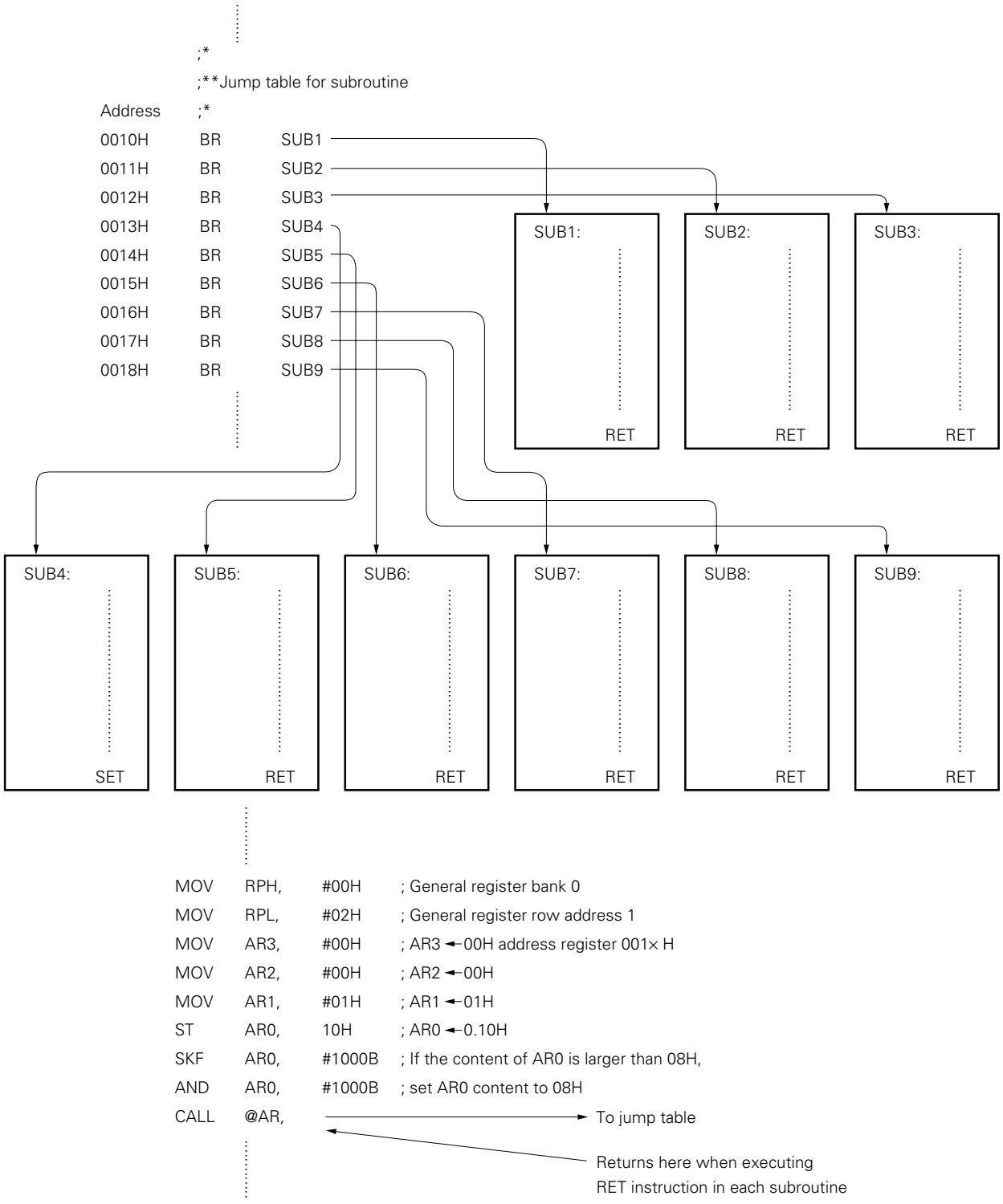
Sets 0020H in address register AR (AR0-AR3) and calls the subroutine at address 0020H with the CALL @AR instruction:

```
MOV     AR3,  #00H   ; AR3 ← 00H
MOV     AR2,  #00H   ; AR2 ← 00H
MOV     AR1,  #02H   ; AR1 ← 02H
MOV     AR0,  #00H   ; AR0 ← 00H
CALL    @AR          ; Calls subroutine at address 0020H
```

Example 2

Calls the following subroutine by the data memory address 0.10H contents:

0.10H Contents		Subroutine
00H	→	SUB1
01H	→	SUB2
02H	→	SUB3
03H	→	SUB4
04H	→	SUB5
05H	→	SUB6
06H	→	SUB7
07H	→	SUB8
08H-0FH	→	SUB9



<4> Note

The higher 6 bits of address register are fixed to 0. Only lower 10 bits can be used.

(3) RET **Return to the main program from subroutine**

<1> OP code

10	8 7	4 3	0
00111	000	1110	0000

<2> Function

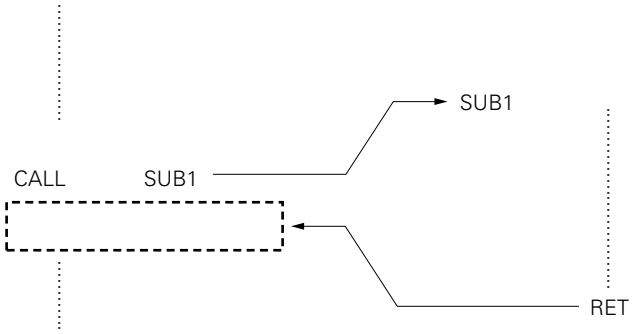
PC ← ASR,

SP ← SP+1

Instruction to return to the main program from a subroutine.

Restores the return address, saved to the stack by the CALL instruction, to the program counter.

<3> Example



(4) RETSK **Return to the main program then skip next instruction**

<1> OP code

00111	001	1110	0000

<2> Function

PC ← ASR, SP ← SP+1 and skip

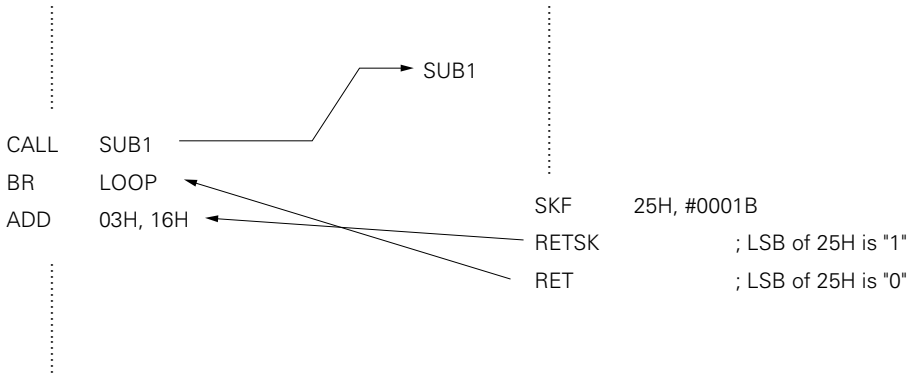
Instruction to return to the main program from a subroutine.

Skips the instruction next to the CALL instruction (Executes as NOP instruction).

Therefore, restores the return address, saved to the stack by the CALL instruction, to program counter PC and then increments the program counter.

<3> Example

Executes the RET instruction, if the LSB (least significant bit) content for address 25H in the data memory (RAM) is 0. The execution is returned to the instruction next to the CALL instruction. If the LSB is 1, executes the RETSK instruction. The execution is returned to the instruction following the one next to the CALL instruction (in this example, ADD 03H, 16H).



(5) RETI **Return to the main program from interrupt service routine**

<1> OP code

00111	100	1110	0000

<2> Function

PC ← ASR, INTR ← INTSK, SP ← SP+1

Instruction to return to the main program, from an interrupt service program.

Restores the return address, saved to the stack by a vector interrupt, to the program counter.

Part of the system register is also returned to the status before the occurrence of the vector interrupt.

<3> Note 1

The system register contents that are automatically saved (i.e., that can be restored by the RETI instruction) when an interrupt occurs is PSWORD.

Note 2

If the RETI instruction is used, instead of the RET instruction, in an ordinary subroutine, the contents of the bank (which are to be saved when an interrupt occurs) are changed to the contents of the interrupt stack, when the execution has returned to the return address. Consequently, an unpredictable status may be assumed. Therefore, use the RET (or RETSK) instruction to return from a subroutine.

18.5.10 Interrupt Instructions

(1) EI **Enable Interrupt**

<1> OP code

00111	000	1111	0000

<2> Function

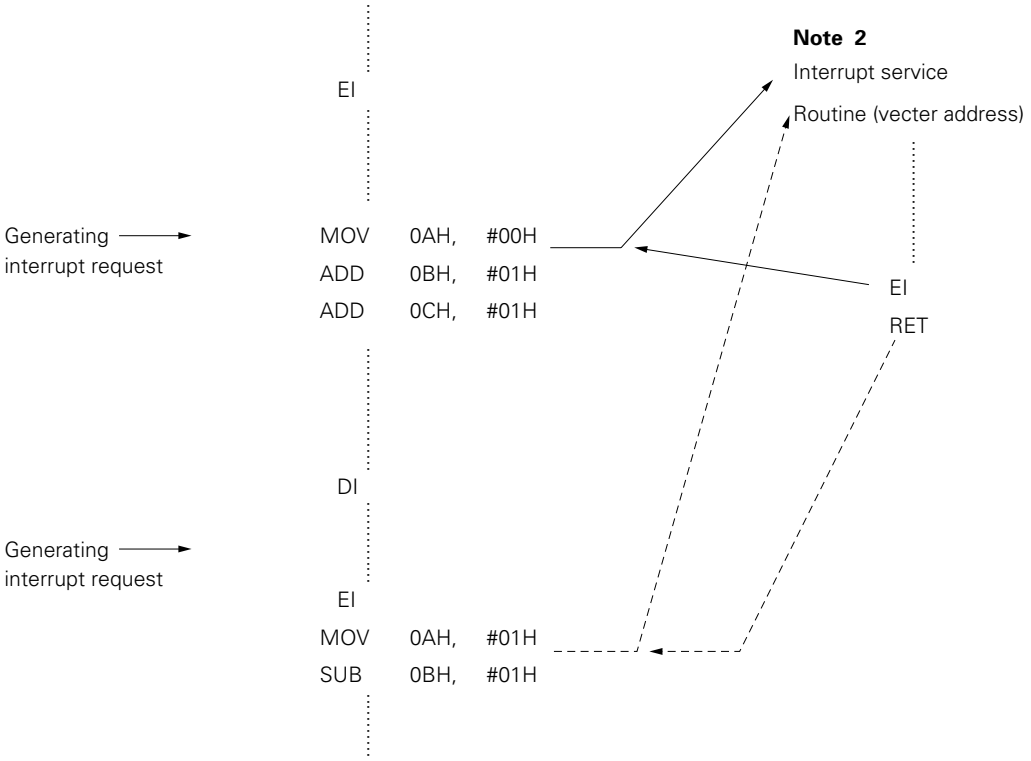
INTEF ← 1

Enables a vectored interrupt.

The interrupt is enabled, after the instruction next to the EI instruction has been executed.

<3> Example 1

As shown in the following example, the interrupt request is accepted after the instruction next to that, that has accepted the interrupt, has been completely executed (excluding an instruction that manipulates program counter). The flow then shifts to the vector address**Note1**.

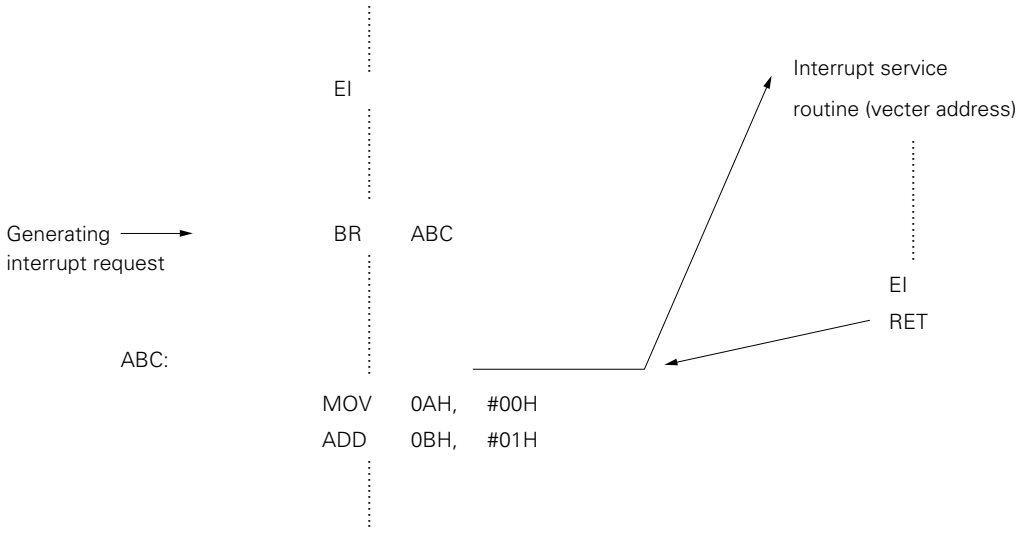


Notes 1. The vector address differs, depending on the interrupt to be accepted. Refer to Table 14-1 Interrupt Source Types.

2. **The interrupt accepted in this example (an interrupt request is generated after the EI instruction has been executed and the execution flow shifts to an interrupt service routine) is the interrupt, whose interrupt enable flag (IP×××) is set. The interrupt request generation without the interrupt enable flag set does not change the program flow, after the EI instruction has been executed (therefore, the interrupt is not accepted). However, interrupt request flag (IRQ×××) is set, and the interrupt is accepted, as soon as the interrupt enable flag is set.**

Example 2

An example of an interrupt, which occurs in response to an interrupt request being accepted when program counter PC is being executed:



(2) DI **Disable interrupt**

<1> OP code

00111	001	1111	0000

<2> Function

INTEF ← 0

Instruction to disable a vectored interrupt.

<3> Example

Refer to Example 1 in (1) EI.

18.5.11 Other Instructions

(1) STOP s **Stop CPU and release by condition s**

<1> OP code

		3	0
00111	010	1111	s

<2> Function

Stops the system clock and places the device in the STOP mode.
In the STOP mode, the power dissipation for the device is minimized.
The condition, under which the STOP mode is to be released, is specified by operand (s).
For the stop releasing condition (s), refer to 15.3.

(2) HALT h **Halt CPU and release by condition h**

<1> OP code

		3	0
00111	011	1111	h

<2> Function

Places the device in the halt mode.
In the halt mode, the power dissipation for the device is reduced.
The condition, under which the halt mode is to be released, is specified by operand (h).
For halt releasing condition (h), refer to 15.2 HALT MODE.

(3) NOP **No operation**

<1> OP code

00111	100	1111	0000

<2> Function

Performs nothing and consumes one machine cycle.

[MEMO]

CHAPTER 19 ASSEMBLER RESERVED WORDS

19.1 MASK OPTION PSEUDO INSTRUCTIONS

To create μPD17120, 17121, 17132, and 17133 programs, it is necessary to specify whether pins that can have pull-up resistors have pull-up resistors. This is done in the assembler source program using mask option pseudo instructions. To set the mask option, note that D171××.OPT file in the AS171×× (μPD171×× device file) must be in the current directory at assembly time.

Specify mask options for the following pins:

- $\overline{\text{RESET}}$ pin
- Port 0D ($P0D_3$, $P0D_2$, $P0D_1$, $P0D_0$)
- Port 0E ($P0E_1$, $P0E_0$)

19.1.1 OPTION and ENDOP Pseudo Instructions

The block from the OPTION pseudo instruction to the ENDOP pseudo instruction is defined as the option definition block.

The format for the mask option definition block is shown below. Only the three pseudo instructions listed in Table 19-1 can be described in this block.

Format:

Symbol	Mnemonic	Operand	Comment
[label:]	OPTION		[;comment]
	•		
	•		
	•		
	•		
	ENDOP		

19.1.2 Mask Option Definition Pseudo Instructions

Table 19-1 lists the pseudo instructions which define the mask options for each pin.

Table 19-1. Mask Option Definition Pseudo Instructions

Pin	Mask Option Pseudo Instruction	Number of Operands	Parameter Name
$\overline{\text{RESET}}$	OPTRES	1	OPEN (without pull-up resistor) PULLUP (with pull-up resistor)
$P0D_3$-$P0D_0$	OPTP0D	4	OPEN (without pull-up resistor) PULLUP (with pull-up resistor)
$P0E_1$, $P0E_0$	OPTP0E	2	OPEN (without pull-up resistor) PULLUP (with pull-up resistor)

The OPTRES format is shown below. Specify the $\overline{\text{RESET}}$ mask option in the operand field.

Symbol	Mnemonic	Operand	Comment
[label:]	OPTRES	($\overline{\text{RESET}}$)	[;comment]

The OPTP0D format is shown below. Specify mask options for all pins of port 0D. Specify the pins in the operand field starting at the first operand in the order $P0D_3$, $P0D_2$, $P0D_1$, then $P0D_0$.

Symbol	Mnemonic	Operand	Comment
[label:]	OPTP0D	($P0D_3$), ($P0D_2$), ($P0D_1$), ($P0D_0$)	[;comment]

The OPTP0E is shown below. Specify mask options for all pins of port 0E. Specify the pins in the operand field starting at the first operand in the order $P0E_1$, $P0E_0$.

Symbol	Mnemonic	Operand	Comment
[label:]	OPTP0E	($P0E_1$), ($P0E_0$)	[;comment]

Example of describing mask options

$\overline{\text{RESET}}$ pin: Pull-up

$P0D_3$: Open, $P0D_2$: Open, $P0D_1$: Pull-up, $P0D_0$: Pull-up,

$P0E_1$: Pull-up, $P0E_0$: Open

```
Symbol              Mnemonic    Operand                      Comment
;μPD17133
Setting mask options:  OPTION
                    ;
                       OPTRES   PULLUP
                       OPTP0D   OPEN, OPEN, PULLUP, PULLUP
                       OPTP0E   PULLUP, OPEN
                    ;
                       ENDOP
```

19.2 RESERVED SYMBOLS

The reserved symbols defined in the μPD17120 subseries device file (AS1712×, AS1713×) are listed below.

19.2.1 List of Reserved Symbols (μPD17120, 17121)

System register (SYSREG)

Symbol 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

Data buffer (DBF)

Symbol 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

Symbol Name	Attribute	Value	Read/Write	Description
P0E1	FLG	0.6FH.1	R/W	Port 0E bit 1
P0E0	FLG	0.6FH.0	R/W	Port 0E bit 0
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

Register file (control register)

(1/2)

Symbol Name	Attribute	Value	Read/Write	Description
SP	MEM	0.81H	R/W	Stack pointer
SIOEN	FLG	0.8AH.0	R/W	SIO enable flag
INT	FLG	0.8FH.0	R	INT pin status flag
PDRESEN	FLG	0.90H.0	R/W	Power-down reset enable flag
TMEN	FLG	0.91H.3	R/W	Timer enable flag
TMRES	FLG	0.91H.2	R/W	Timer reset flag
TMCK1	FLG	0.91H.1	R/W	Timer count pulse selection flag bit 1
TMCK0	FLG	0.91H.0	R/W	Timer count pulse selection flag bit 0
TMOSEL	FLG	0.92H.0	R/W	Timer output port/port selection flag
SIOTS	FLG	0.9AH.3	R/W	SIO start flag
SIOHIZ	FLG	0.9AH.2	R/W	SO pin status
SIOCK1	FLG	0.9AH.1	R/W	Serial clock selection flag bit 1
SIOCK0	FLG	0.9AH.0	R/W	Serial clock selection flag bit 0

Register file (control register)

(2/2)

Symbol Name	Attribute	Value	Read/Write	Description
IEGMD1	FLG	0.9FH.1	R/W	INT pin edge detection selection flag bit 1
IEGMD0	FLG	0.9FH.0	R/W	INT pin edge detection selection flag bit 0
P0BGIO	FLG	0.A4H.0	R/W	P0B group input/output selection flag (1= all P0Bs are output ports.)
IPSIO	FLG	0.AFH.2	R/W	SIO interrupt flag
IPTM	FLG	0.AFH.1	R/W	Timer interrupt enable flag
IP	FLG	0.AFH.0	R/W	INT pin interrupt enable flag
P0EBIO1	FLG	0.B2H.1	R/W	$P0E_1$ input/output selection flag (1=output port)
P0EBIO0	FLG	0.B2H.0	R/W	$P0E_0$ input/output selection flag (1=output port)
P0DBIO3	FLG	0.B3H.3	R/W	$P0D_3$ input/output selection flag (1=output port)
P0DBIO2	FLG	0.B3H.2	R/W	$P0D_2$ input/output selection flag (1=output port)
P0DBIO1	FLG	0.B3H.1	R/W	$P0D_1$ input/output selection flag (1=output port)
P0DBIO0	FLG	0.B3H.0	R/W	$P0D_0$ input/output selection flag (1=output port)
P0CBIO3	FLG	0.B4H.3	R/W	$P0C_3$ input/output selection flag (1=output port)
P0CBIO2	FLG	0.B4H.2	R/W	$P0C_2$ input/output selection flag (1=output port)
P0CBIO1	FLG	0.B4H.1	R/W	$P0C_1$ input/output selection flag (1=output port)
P0CBIO0	FLG	0.B4H.0	R/W	$P0C_0$ input/output selection flag (1=output port)
P0ABIO3	FLG	0.B5H.3	R/W	$P0A_3$ input/output selection flag (1=output port)
P0ABIO2	FLG	0.B5H.2	R/W	$P0A_2$ input/output selection flag (1=output port)
P0ABIO1	FLG	0.B5H.1	R/W	$P0A_1$ input/output selection flag (1=output port)
P0ABIO0	FLG	0.B5H.0	R/W	$P0A_0$ input/output selection flag (1=output port)
IRQSIO	FLG	0.BDH.0	R/W	SIO interrupt request flag
IRQTM	FLG	0.BEH.0	R/W	Timer interrupt request flag
IRQ	FLG	0.BFH.0	R/W	INT pin interrupt request flag

Peripheral hardware register

Symbol Name	Attribute	Value	Read/Write	Description
SIOSFR	DAT	01H	R/W	Peripheral address of the shift register
TMC	DAT	02H	R	Peripheral address of the timer count register
TMM	DAT	03H	W	Peripheral address of the timer modulo register
AR	DAT	40H	R/W	Peripheral address of the address register for GET, PUT, PUSH, CALL, BR, MOVT, and INC instructions

Others

Symbol Name	Attribute	Value	Description
DBF	DAT	0FH	Fix operand value of PUT, GET, MOVT instructions
IX	DAT	01H	Fix operand value of INC instruction

Figure 19-1. Configuration of Control Register (μPD17120, 17121) (1/2)

Column address / Row address	Item	0	1	2	3	4	5	6	7
0 (8)	Symbol		0 SP						
	At reset		0 0 0 0						
	Read/ Write		R/W						
1 (9)	Symbol	0 0 0 PDRESEN	TMEN TMRES TMCK1 TMCK0	0 0 0 TMOSEL					
	At 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 P0BGIO			
	At reset					0 0 0 0			
	Read/ Write					R/W			
3 (B)	Symbol			0 0 P0EBIO1 P0EBIO0	P0DBIO3 P0DBIO2 P0DBIO1 P0DBIO0	P0CBIO3 P0CBIO2 P0CBIO1 P0CBIO0	P0ABIO3 P0ABIO2 P0ABIO1 P0ABIO0		
	At reset			0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0		
	Read/ Write			R/W	R/W	R/W	R/W		

Remark ( ) means the address when using assembler (AS17K).
All flags of the control register are registered in device file as assembler reserved words. It is convenient for program design to use the reserved words.

Figure 19-1. Configuration of Control Register (μPD17120, 17121) (2/2)

8	9	A	B	C	D	E	F
		0 0 0 SIOEN					0 0 0 INT
		0 0 0 0					0 0 0 Note
		R/W					R
		SIOTS SIOHIZ SIOCK1 SIOCK0					0 0 IEGMD1 IEGMD0
		0 0 0 0					0 0 0 0
		R/W					R/W
							0 IPSIO IPTM IP
							0 0 0 0
							R/W
					0 0 0 IRQSIO	0 0 0 IRQTM	0 0 0 IRQ
					0 0 0 0	0 0 0 1	0 0 0 0
					R/W	R/W	R/W

Note The INT flag differs depending on the INT pin state at the time.

19.2.2 List of Reserved Symbols (μPD17132, 17133, 17P132, 17P133)

System register (SYSREG)

Symbol Name	Attribute	Value	Read/Write	Description
AR3	MEM	0.74H	R	Address register bits 15 to 12
AR2	MEM	0.75H	R/W	Address register bits 11 to 8
AR1	MEM	0.76H	R/W	Address register bits 7 to 4
AR0	MEM	0.77H	R/W	Address register bits 3 to 0
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

Data buffer (DBF)

Symbol 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

Symbol Name	Attribute	Value	Read/Write	Description
P0E1	FLG	0.6FH.1	R/W	Port 0E bit 1
P0E0	FLG	0.6FH.0	R/W	Port 0E bit 0
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

Register file (control register)

(1/2)

Symbol Name	Attribute	Value	Read/Write	Description
SP	MEM	0.81H	R/W	Stack pointer
SIOEN	FLG	0.8AH.0	R	SIO enable flag
INT	FLG	0.8FH.0	R/W	INT pin status flag
PDRESEN	FLG	0.90H.0	R/W	Power-down reset enable flag
TMEN	FLG	0.91H.3	R/W	Timer enable flag
TMRES	FLG	0.91H.2	R/W	Timer reset flag
TMCK1	FLG	0.91H.1	R/W	Timer source clock selection flag bit 1
TMCK0	FLG	0.91H.0	R/W	Timer source clock selection flag bit 0
TMOSEL	FLG	0.92H.0	R/W	Timer output port/port selection flag
SIOTS	FLG	0.9AH.3	R/W	SIO start flag
SIOHIZ	FLG	0.9AH.2	R/W	SO pin status
SIOCK1	FLG	0.9AH.1	R/W	SIO source clock selection flag bit 1
SIOCK0	FLG	0.9AH.0	R/W	SIO source clock selection flag bit 0
CMPCH1	FLG	0.9CH.1	R/W	Comparator input channel selection flag bit 1
CMPCH0	FLG	0.9CH.0	R/W	Comparator input channel selection flag bit 0
CMPVREF3	FLG	0.9DH.3	R/W	Comparator reference voltage selection flag bit 3
CMPVREF2	FLG	0.9DH.2	R/W	Comparator reference voltage selection flag bit 2
CMPVREF1	FLG	0.9DH.1	R/W	Comparator reference voltage selection flag bit 1
CMPVREF0	FLG	0.9DH.0	R/W	Comparator reference voltage selection flag bit 0
CMPSTRT	FLG	0.9EH.1	R	Comparator start flag
CMPRSLT	FLG	0.9EH.0	R/W	Comparator comparison result flag
IEGMD1	FLG	0.9FH.1	R/W	INT pin edge detection selection flag bit 1
IEGMD0	FLG	0.9FH.0	R/W	INT pin edge detection selection flag bit 0
P0C3IDI	FLG	0.A3H.3	R/W	$P0C_3$ input port disable flag ($P0C_3$/Cin_3 selection)
P0C2IDI	FLG	0.A3H.2	R/W	$P0C_2$ input port disable flag ($P0C_2$/Cin_2 selection)
P0C1IDI	FLG	0.A3H.1	R/W	$P0C_1$ input port disable flag ($P0C_1$/Cin_1 selection)
P0C0IDI	FLG	0.A3H.0	R/W	$P0C_0$ input port disable flag ($P0C_0$/Cin_0 selection)
P0BGIO	FLG	0.A4H.0	R/W	P0B group input/output selection flag (1= all P0Es are output ports.)
IPSIO	FLG	0.AFH.2	R/W	SIO interrupt flag
IPTM	FLG	0.AFH.1	R/W	Timer interrupt enable flag
IP	FLG	0.AFH.0	R/W	INT pin interrupt enable flag
P0EBIO1	FLG	0.B2H.1	R/W	$P0E_1$ input/output selection flag (1=output port)
P0EBIO0	FLG	0.B2H.0	R/W	$P0E_0$ input/output selection flag (1=output port)
P0DBIO3	FLG	0.B3H.3	R/W	$P0D_3$ input/output selection flag (1=output port)
P0DBIO2	FLG	0.B3H.2	R/W	$P0D_2$ input/output selection flag (1=output port)
P0DBIO1	FLG	0.B3H.1	R/W	$P0D_1$ input/output selection flag (1=output port)
P0DBIO0	FLG	0.B3H.0	R/W	$P0D_0$ input/output selection flag (1=output port)

Register file (control register)

(2/2)

Symbol Name	Attribute	Value	Read/Write	Description
P0CBIO3	FLG	0.B4H.3	R/W	$P0C_3$ input/output selection flag (1=output port)
P0CBIO2	FLG	0.B4H.2	R/W	$P0C_2$ input/output selection flag (1=output port)
P0CBIO1	FLG	0.B4H.1	R/W	$P0C_1$ input/output selection flag (1=output port)
P0CBIO0	FLG	0.B4H.0	R/W	$P0C_0$ input/output selection flag (1=output port)
P0ABIO3	FLG	0.B5H.3	R/W	$P0A_3$ input/output selection flag (1=output port)
P0ABIO2	FLG	0.B5H.2	R/W	$P0A_2$ input/output selection flag (1=output port)
P0ABIO1	FLG	0.B5H.1	R/W	$P0A_1$ input/output selection flag (1=output port)
P0ABIO0	FLG	0.B5H.0	R/W	$P0A_0$ input/output selection flag (1=output port)
IRQSIO	FLG	0.BDH.0	R/W	SIO interrupt request flag
IRQTM	FLG	0.BEH.0	R/W	Timer interrupt request flag
IRQ	FLG	0.BFH.0	R/W	INT pin interrupt request flag

Peripheral hardware register

Symbol Name	Attribute	Value	Read/Write	Description
SIOSFR	DAT	01H	R/W	Peripheral address of the shift register
TMC	DAT	02H	R	Peripheral address of the timer count register
TMM	DAT	03H	W	Peripheral address of the timer modulo register
AR	DAT	40H	R/W	Peripheral address of the address register for GET, PUT, PUSH, CALL, BR, MOVT, and INC instructions

Others

Symbol Name	Attribute	Value	Description
DBF	DAT	0FH	Fix operand value of PUT, GET, MOVT instructions
IX	DAT	01H	Fix operand value of INC instruction

Figure 19-2. Configuration of Control Register (μPD17132, 17133, 17P132, 17P133) (1/2)

Column address / Row address	Item	0	1	2	3	4	5	6	7
0 (8)	Symbol		0 SP						
	At reset		0 1 0 1						
	Read/ Write		R/W						
1 (9)	Symbol	0 0 0 PDRESEN	TMEN TMRES TMCK1 TMCK0	0 0 0 TMOSEL					
	At reset	0 0 0 0	0 0 0 0	0 0 0 0					
	Read/ Write	R/W	R/W	R/W					
2 (A)	Symbol				P0C3IDI P0C2IDI P0C1IDI P0C0IDI	0 0 0 P0BGIO			
	At reset				0 0 0 0	0 0 0 0			
	Read/ Write				R/W	R/W			
3 (B)	Symbol			0 0 P0EBIO1 P0EBIO0	P0DBIO3 P0DBIO2 P0DBIO1 P0DBIO0	P0CBIO3 P0CBIO2 P0CBIO1 P0CBIO0	P0ABIO3 P0ABIO2 P0ABIO1 P0ABIO0		
	At reset			0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0		
	Read/ Write			R/W	R/W	R/W	R/W		

Remark ( ) means the address when using assembler (AS17K).
All flags of the control register are registered in device file as assembler reserved words. It is convenient for program design to use the reserved words.

Figure 19-2. Configuration of Control Register (μPD17132, 17133, 17P132, 17P133) (2/2)

8	9	A	B	C	D	E	F
		0 0 0 SIOEN					0 0 0 INT
		0 0 0 0					0 0 0 0
		R/W					R
		SIOTS SIOHIZ SIOCK1 SIOCK0		0 0 CMPCH1 CMPCH1	CMPVREF3 CMPVREF2 CMPVREF1 CMPVREF0	0 0 CMPSTRT CMPRSLT	0 0 IEGMD1 IEGMD0
		0 0 0 0		0 0 0 0	1 0 0 0	0 0 0 1	0 0 0 0
		R/W		R/W	R/W	R/W R	R/W
							0 IPSIO IPTM IP
							0 0 0 0
							R/W
					0 0 0 IRQSIO	0 0 0 IRQTM	0 0 0 IRQ
					0 0 0 0	0 0 0 1	0 0 0 0
					R/W	R/W	R/W

Note The INT flag differs depending on the INT pin state at the time.

[MEMO]

APPENDIX A DEVELOPMENT TOOLS

The following support tools are available for developing programs for the μPD17120 subseries.

Hardware

Name	Outline
In-circuit Emulator IE-17K IE-17K-ET**Note 1** EMU-17K**Note 1**	IE-17K, IE-17K-ET, and EMU-17K are the in-circuit emulators common to all 17K-series products. IE-17K and IE-17K-ET are used by connecting to the host machine PC-9800 series or IBM PC/AT™ through RS-232-C. EMU-17K is used by installing in the expansion slot of the host machine PC-9800 series. By using it in combination with the system evaluation board (SE board) dedicated to the relevant machine type, the emulator can perform operations compatible with it. An even more advanced debugging environment can be realized by using SIMPLEHOST, which is man-machine interface software. EMU-17K is equipped with the function of checking the contents of the data memory in a real-time environment.
SE Board (SE-17120)	The SE-17120 is an SE board for the μPD17120 subseries. The board is used for evaluation of single system units as well as for debugging by being combined with an in-circuit emulator.
Emulation Probe (EP-17120CS)	The EP-17120CS, which is the emulation probe for the μPD17120 subseries, connects between an SE board and a target system.
PROM Programmer AF-9703**Note 3** AF-9704**Note 3** AF-9705**Note 3** AF-9706**Note 3**	AF-9703, AF-9704, AF-9705, and AF-9706 are the PROM programmers compatible with the μPD17P132 and 17P133. By connecting them to the program adapter AF-9808M, the μPD17P132 and 17P133 are enabled for programming.
Program Adapter (AF-9808M**Note 3**)	The AF-9808M, which is an adapter for programming the μPD17P132CS, 17P132GT, 17P132CS, and 17P33GT, is used in combination with the AF-9703, AF9704, AF-9705, or AF-9706.

Notes
1. Low-price version: External-power type
2. This is a product of I.C Co., Ltd. For further details, please contact I.C Co. in Tokyo (Tel: 03-3447-3793).
3. This is the product of the Ando Electric, Ltd. For further details, please contact Ando Electric Co., Ltd. in Tokyo (Tel: 03-3733-1151).

Software

Name	Outline	Host Machine	OS		Supply Medium	Part Number
17K-Series Assembler (AS17K)	AS17K is an assembler which can be used in common for the 17K series. For program development of the μPD17120 series, AS17K and device files (AS17120, AS17121, AS17132, AS17133) are used together.	PC-9800 series	MS-DOS™		5-inch 2HD	μS5A10AS17K
					3.5-inch 2HD	μS5A13AS17K
		IBM PC/AT	PC DOS™		5-inch 2HC	μS7B10AS17K
					3.5-inch 2HC	μS7B13AS17K
Device Files [AS17120 AS17121 AS17132 AS17133]	AS17120, AS17121, AS17132, and AS17133 are the device files for the μPD17120 subseries. These are used in combination with the assembler (AS17K) common to the 17K series.	PC-9800 series	MS-DOS		5-inch 2HD	μS5A10AS17120**Note**
					3.5-inch 2HD	μS5A13AS17120**Note**
		IBM PC/AT	PC DOS		5-inch 2HC	μS7B10AS17120**Note**
					3.5-inch 2HC	μS7B13AS17120**Note**
Support Software (SIMPLE-HOST)	This software is used for machine interfacing on Windows™ when doing program development by means of an in-circuit emulator and a personal computer.	PC-9800 series	MS-DOS	Windows	5-inch 2HD	μS5A10IE17K
					3.5-inch 2HD	μS5A13IE17K
		IBM PC/AT	PC DOS		5-inch 2HC	μS7B10IE17K
					3.5-inch 2HC	μS7B13IE17K

Note μSxxxxAS17120 contains AS17120, AS17121, AS17132, and AS17133.

Remark Compatible OS versions include the following:

OS	Version
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 Vers. 5.00/5.00A and PC DOS Ver. 5.0 are equipped with the task swap function. However, this software is not.

APPENDIX B ORDERING MASK ROM

After developing the program, place an order for the mask ROM version, according to the following procedure:

(1) Make reservation when ordering mask ROM.

Advice NEC of the schedule for placing an order for the mask ROM. If NEC is not informed in advance, on-time delivery may not be possible.

(2) Create ordering medium.

Use UV-EPROM to place an order for the mask ROM.

Add/PROM as an assemble option of the Assembler (AS17K), and create a mask ROM ordering HEX file (with extender for .PRO). Next, write the mask ROM ordering HEX file into the UV-EPROM. Create three UV-EPROMs with the same contents.

(3) Prepare necessary documents.

Fill out the following forms to place an order for the mask ROM:

- Mask ROM ordering sheet
- Mask ROM ordering check sheet

(4) Ordering

Submit the media created in (2) and documents prepared in (3) to NEC by the specified date.

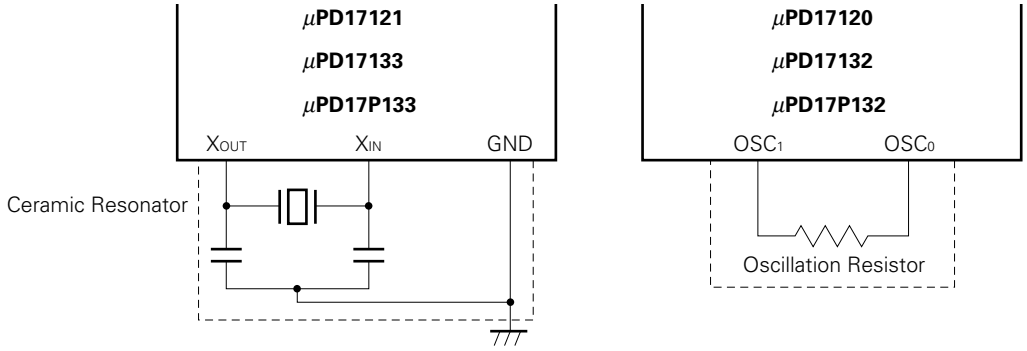
[MEMO]

APPENDIX C CAUTIONS TO TAKE IN SYSTEM CLOCK OSCILLATION CIRCUIT CONFIGURATIONS

The system clock oscillation circuit operates with a ceramic resonator connected to the X1 and X2 pins or with an oscillation resistor connected to the OSC_1 and OSC_0 pins.

Figure C-1 shows the externally installed system clock oscillation circuit.

Figure C-1. Externally Installed System Clock Oscillation Circuit



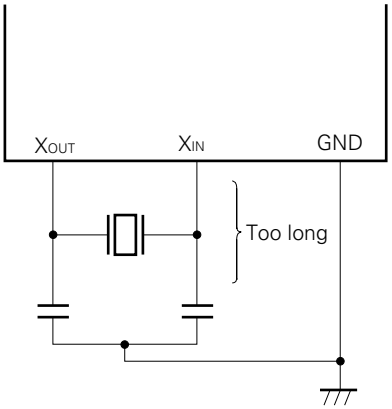
Caution Regarding the system clock oscillation circuit, make sure that its ground wire's resistance component and impedance component are minimized. Also, to avoid the effect of wiring capacity, etc., please wire the part encircled in the dotted line in Figure C-1 in the manner described below:

- **Make the wiring as short as possible.**
- **Do not allow it to intersect other signal conductors. Do not let it be near lines in which a large fluctuating current flows.**
- **Make sure that the grounding point of the oscillation circuit's capacitor is constantly at the same electric potential as V_{SS}. Do not let it be near a GND wire in which a large current flows.**
- **Do not extract signals from the oscillation circuit.**

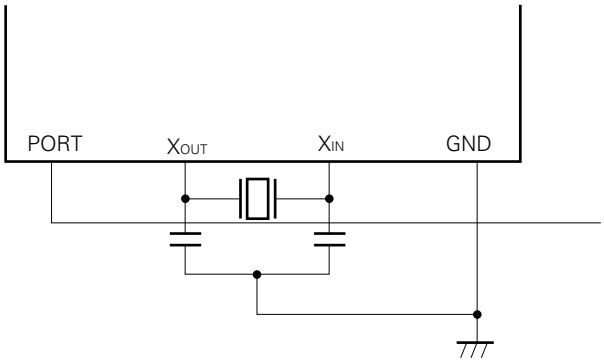
Figure C-2 shows unsatisfactory oscillation circuit examples.

Figure C-2. Unsatisfactory Oscillation Circuit Examples

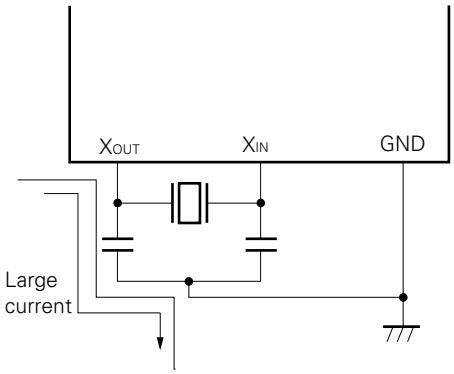
(a) Connecting circuit whose wiring is too long



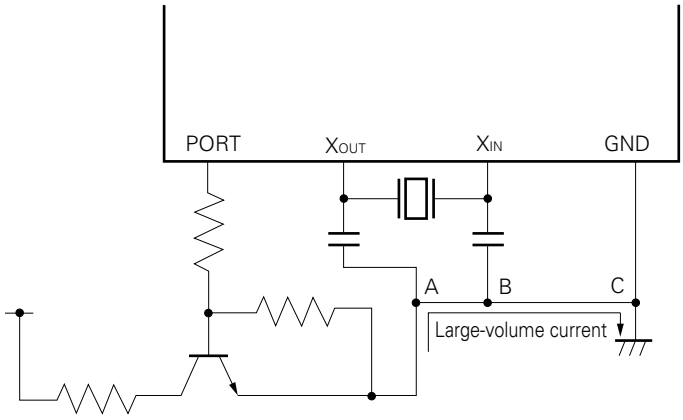
(b) Signal conductors are intersecting



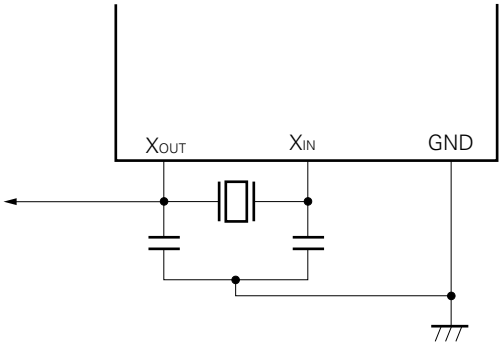
(c) Functuating large current located too close to the signal conductor



(d) Current flowing in the GND line of the oscillation circuit (Points A and B's potentials change as to point C.)



(e) Signals are being extracted



APPENDIX D INSTRUCTION LIST

[MEMO]