User’s Manual

μPD172×× Subseries

4-bit Single-chip Microcontroller

Common Functions

μPD17201A	μPD17227
μPD17203A	μPD17228
μPD17204	μPD17P203A
μPD17207	μPD17P204
μPD17225	μPD17P207
μPD17226	μPD17P218

Document No. U12795EJ4V0UM00
(Previous No. IEU-1317B)
Date Published June 1998 N CP(K)

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.

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

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

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

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

NEC devices are classified into the following three quality grades:

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

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

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

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

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

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

Anti-radioactive design is not implemented in this product.

M7 96.5

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC Electronics (Germany) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 02
Fax: 0211-65 03 490

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Italiana s.r.1.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (Germany) GmbH
Benelux Office
Eindhoven, The Netherlands
Tel: 040-2445845
Fax: 040-2444580

NEC Electronics (France) S.A.
Velizy-Villacoublay, France
Tel: 01-30-67 58 00
Fax: 01-30-67 58 99

NEC Electronics (France) S.A.
Spain Office
Madrid, Spain
Tel: 01-504-2787
Fax: 01-504-2860

NEC Electronics (Germany) GmbH
Scandinavia Office
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
United Square, Singapore 1130
Tel: 65-253-8311
Fax: 65-250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-719-2377
Fax: 02-719-5951

NEC do Brasil S.A.
Cumbica-Guarulhos-SP, Brasil
Tel: 011-6465-6810
Fax: 011-6465-6829

J98. 2

MAJOR REVISIONS IN THIS EDITION

Page	Contents
Throughout	μPD17225, 17226, 17227, and 17228 added
Throughout	μPD17202A, 17215, 17216, 17217, 17218 and 17P202A deleted
Throughout	Assembler changed (AS17K → RA17K)
p. 20	Changes in **1.1 List of Functions**
p. 137	Extension instruction added to the table in **15.4 Assembler (RA17K) Macro instruction**
p. 201	**A.1 Hardware List** modified
p. 202	**A.2 Software List** modified

The mark ★ shows the major revised points.

INTRODUCTION

Targeted reader

This manual is intended for the users who understand the functions of the μPD172×× subseries microcontrollers and design application systems using these microcontrollers.

Objective

This manual describes the functions common to all the models in the μPD172×× subseries, and will serve as a reference manual when you develop a program for a μPD172×× subseries microcontrollers.

How to read this manual

It is assumed that the readers of this manual possess general knowledge about electric engineering, logic circuits, and microcomputers.

- To understand the overall functions of the μPD172×× subseries,
  → Read this manual using the Contents.

- To understand the function of an instruction whose mnemonic is known,
  → Use the **APPENDIX C INSTRUCTION INDEX**.

- To understand the function of the instruction whose mnemonic is not known but whose function is known,
  → Refer to **15.3 Instruction List** by referring to **15.5 Instruction Functions**.

- To learn the electrical specifications of the μPD172×× subseries,
  → Refer to the Data Sheet for the respective models.

Legend

Data significance	: Higher digit on left, lower digit on right
Active low	: $\overline{\times\times\times}$ (bar over pin and signal names)
Memory map address	: Top-low, bottom-high
Note	: Description of **Note** in the text.
Caution	: Information requiring particular attention
Remark	: Supplementary explanation
Number	: Binary ... ×××× or ××××B Decimal number ... ×××× or ××××D Hexadecimal number ... ××××H

Related Documents Refer to the following documents (the numbers in this table indicate the document number).

- **4-bit single-chip microcontrollers (μPD172×× subseries) (1/2)**

Part Number Item	μPD17201A	μPD17207	μPD17P207	μPD17203A	μPD17P203A	μPD17204	μPD17P204
Data sheet	U11778J [U11778E]		U11777J [U11777E]	IC-8089 [U10334E]	IC-8303 [IC-2851]	IC-8089 [U10334E]	IC-8303 [IC-2851]
Instruction table	IEM-5537 [IEM-1213]			IEM-5544			
User’s manual	U12795J [U12795E](This manual)						
Application note	IEA-707 [IEA-1285]			—	—	—	—
	IEA-757 [IEA-1306] (Floating-point arithmetic package)			—	—	—	—
SE board user’s manual	EEU-763 [EEU-1372]			EEU-762 [EEU-1371]			
Device file user’s manual	EEU-736 [EEU-1373]	EEU-746 [EEU-1360]		EEU-738 [EEU-1350]		EEU-751 [EEU-1361]	
Microcontrollers for remote controllers selection guide	X10088J [X10088E]						
17K series/DTS standard models selection guide	U10317J [U10317E]						
RA17K user’s manual	U10305J [U10305E]						
IE-17K/IE-17K-ET CLICE/CLICE-ET user’s manual	U10063J [U10063E]						
SIMPLEHOST™ user’s manual	EEU-723: Introduction [EEU-1336] EEU-724: Reference [EEU-1337]						
SIMPLEHOST *emIC-17K*™/RA17K compatible user’s manual	EEU-5009: Introduction [U10445E] EEU-5007: Reference [U10496E]						
Project manager user’s manual	U12810J [EEU-1527]						
MAKE/CNV17K user’s manual	U10596J [U10596E]						
emIC-17K user’s manual	EEU-876 [EEU-1511]						
LK17K user’s manual	U12518J [U12518E]						
DOC17K user’s manual	EEU-5006 [EEU-1536]						

Remark The number inside [ ] indicates document number for English version.

- **4-bit single-chip microcontrollers (μPD172×× subseries) (2/2)**

Part Number Item	μPD17225	μPD17226	μPD17227	μPD17228	μPD17P218
Data sheet	U12643J [U12643E]				U12217J [IC-3252]
Instruction table	—	—	—	—	—
User's manual	U12795J [U12795E] (This manual)				
Application note	—	—	—	—	—
SE board user's manual	U12372J (U12372E)				—
Device file user's manual	U12136J (U12136E)				EEU-925 [EEU-1461]
Microcontrollers for remote controllers selection guide	X10088J [X10088E]				
17K series/DTS standard models selection guide	U10317J [U10317E]				
RA17K user's manual	U10305J [U10305E]				
IE-17K/IE-17K-ET CLICE/CLICE-ET user's manual	U10063J [U10063E]				
SIMPLEHOST user's manual	EEU-723: Introduction [EEU-1336] EEU-724: Reference [EEU-1337]				
SIMPLEHOST *emIC-17K*/RA17K compatible user's manual	EEU-5009: Introduction [U10445E] EEU-5007: Reference [U10496E]				
Project manager user's manual	U12810J [EEU-1527]				
MAKE/CNV17K user's manual	U10596J [U10596E]				
emIC-17K user's manual	EEU-876 [EEU-1511]				
LK17K user's manual	U12518J [U12518E]				
DOC17K user's manual	EEU-5006 [EEU-1536]				

Remark The number inside [ ] indicates document number for English version.

[MEMO]

TABLE OF CONTENTS

CHAPTER 1 GENERAL 19

1.1 List of Functions 20

CHAPTER 2 PROGRAM COUNTER (PC) 23

2.1 Program Counter Configuration 23
2.2 Program Counter Operations 24
2.2.1 On reset 24
2.2.2 On branch instruction (BR) execution 25
2.2.3 On subroutine call instruction (CALL) execution 26
2.2.4 On return instruction (RET, RETSK, or RETI) execution 26
2.2.5 On table reference instruction (MOVT) execution 27
2.2.6 On skip instruction (SKE, SKGE, SKLT, SKNE, SKT, or SKF) execution 27
2.2.7 On accepting interrupt 27

CHAPTER 3 PROGRAM MEMORY (ROM) 29

3.1 Program Memory Configuration 30
3.2 Program Memory Function 31
3.2.1 Storing program 31
3.2.2 Table reference 37

CHAPTER 4 DATA MEMORY (RAM) 41

4.1 Data Memory Configuration 42
4.1.1 System register configuration 43
4.1.2 Data buffer (DBF) 43
4.1.3 General register 44
4.1.4 Port register 44

CHAPTER 5 STACK 45

5.1 Configuration of Stack 45
5.2 Function of Stack 46
5.3 Address Stack Registers 46
5.4 Interrupt Stack Register 46
5.5 Stack Pointer (SP) and Interrupt Stack Register 47
5.6 Stack Operation When Subroutine or Table Reference Instruction Is Executed or When Interrupt Is Accepted 48
5.6.1 When subroutine call (CALL) or return (RET, RETSK) instruction is executed 48
5.6.2 When table reference instruction (MOVT DBF, @AR) is executed 48
5.6.3 When interrupt is accepted or return instruction (RETI) is executed 49
5.7 Nesting Level of Stack, and PUSH and POP Instructions 49

CHAPTER 6 SYSTEM REGISTERS (SYSREG) 51

6.1 Configuration of System Registers 51
6.2 Address Register (AR) 53
6.2.1 Address register configuration 53
6.2.2 Address register function 54
6.3 Window Register (WR) 55
6.3.1 Window register configuration 55
6.3.2 Window register function 55
6.4 Bank Register (BANK) 56
6.5 Index Register (IX) 57
6.5.1 Index register and data memory row address pointer functions 58
6.5.2 When MPE = 0, IXE = 0 (no data memory modification) 60
6.5.3 When MPE = 1, IXE = 0 (diagonal indirect transfer) 62
6.5.4 When MPE = 0, IXE = 1 (data memory address index modification) 64
6.6 General Register Pointer (RP) 69
6.6.1 General register pointer configuration 69
6.6.2 General register pointer function 69
6.7 Program Status Word (PSWORD) 71
6.7.1 Program status word configuration 71
6.7.2 Program status word function 72
6.7.3 Index enable flag (IXE) 73
6.7.4 Zero (Z) and compare (CMP) flags 73
6.7.5 Carry flag (CY) 74
6.7.6 Binary coded decimal flag (BCD) 74
6.7.7 Notes on executing arithmetic operation 74
6.8 Notes on Using System Registers 75
6.8.1 Reserved words of system registers 75
6.8.2 Handling system register fixed to “0” 77

CHAPTER 7 GENERAL REGISTERS (GR) 79

7.1 General Register Configuration 79
7.2 General Register Function 79

CHAPTER 8 REGISTER FILE (RF) 81

8.1 Register File Configuration 81
8.1.1 Register file configuration 81
8.1.2 Register file and data memory 81
8.2 Register File Function 82
8.2.1 Register file function 82
8.2.2 Control register function 83
8.2.3 Register file manipulation instructions 84
8.3 Control Registers 86
8.4 Notes on Using Register File 86
8.4.1 Notes on manipulating control registers (read-only and unused registers) 86
8.4.2 Symbol definition of register file and reserved words 87

CHAPTER 9 DATA BUFFER (DBF) 89

9.1 Data Buffer Configuration 89
9.2 Data Buffer Function 90
9.2.1 Data buffer and peripheral hardware 91
9.2.2 Data transfer with peripheral hardware 92
9.2.3 Table reference 93

CHAPTER 10 ARITHMETIC LOGIC UNIT (ALU) 95

10.1 ALU Block Configuration 95
10.2 ALU Block Function 95
10.2.1 ALU function 95
10.2.2 Functions of temporary registers A and B 100
10.2.3 Status flip-flop functions 100
10.2.4 Binary 4-bit operation 100
10.2.5 BCD operation 101
10.2.6 ALU block processing sequence 102
10.3 Arithmetic Operation (Binary 4-bit addition/subtraction and BCD addition/subtraction) 103
10.3.1 Addition/subtraction when CMP = 0, BCD = 0 104
10.3.2 Addition/subtraction when CMP = 1, BCD = 0 104
10.3.3 Addition/subtraction when CMP = 0, BCD = 1 104
10.3.4 Addition/subtraction when CMP = 1, BCD = 1 105
10.3.5 Notes on using arithmetic operation instruction 105
10.4 Logical Operation 105
10.5 Bit Testing 106
10.5.1 True bit (1) testing 107
10.5.2 False bit (0) testing 107
10.6 Compare 108
10.6.1 Comparison of “Equal to” 109
10.6.2 Comparison of “Not equal to” 109
10.6.3 Comparison of “Greater than” 110
10.6.4 Comparison of “Less than” 110
10.7 Rotation Processing 111
10.7.1 Right rotation processing 111
10.7.2 Left rotation processing 112

CHAPTER 11 INTERRUPT FUNCTION 113

11.1 Interrupt Control Circuit Configuration 113
11.1.1 Interrupt control (EI, DI) 113
11.1.2 Interrupt enable flag (IP×××) 113
11.1.3 Interrupt request flag (IRQ×××) 113
11.2 Interrupt Sequence 114
11.2.1 Accepting interrupt 114
11.2.2 Returning from interrupt routine 118

CHAPTER 12 STANDBY FUNCTION 121

12.1 Function Outline 121
12.2 Setting and Releasing STOP Mode 122
12.2.1 Setting STOP mode 122
12.2.2 Operation when STOP mode is released 122
12.3 Setting and Releasing HALT Mode 124
12.3.1 Setting HALT mode 124
12.3.2 Operation after releasing HALT mode 124

CHAPTER 13 RESET FUNCTION 125

13.1 Reset by $\overline{\text{RESET}}$ Pin 125
13.2 Watchdog Function ($\overline{\text{WDOUT}}$ output) 126
13.2.1 Reset by watchdog timer (connect $\overline{\text{RESET}}$ and $\overline{\text{WDOUT}}$ pins) 126
13.2.2 Reset by stack pointer (connect $\overline{\text{RESET}}$ and $\overline{\text{WDOUT}}$ pins) 126
13.3 Low Voltage Detection Circuit (connect $\overline{\text{RESET}}$ and $\overline{\text{WDOUT}}$ pins) 126
13.4 Notes on Using INT and $\overline{\text{RESET}}$ Pins 126

CHAPTER 14 WRITING AND VERIFYING ONE-TIME PROM 127

14.1 Differences between Mask ROM and One-Time PROM Models 127
14.2 Operation Modes for Writing/Verifying Program Memory 129
14.3 How to Write Program Memory 130
14.4 How to Read Program Memory 131

CHAPTER 15 INSTRUCTION SET 133

15.1 Instruction Set Outline 133
15.2 Legend 134
15.3 Instruction List 135
15.4 Assembler (RA17K) Macro instructions 137
15.5 Instruction Functions 138
15.5.1 Addition instructions 138
15.5.2 Subtraction instructions 150
15.5.3 Logical operation instructions 158
15.5.4 Test instructions 164
15.5.5 Compare instructions 166
15.5.6 Rotation instruction 169
15.5.7 Transfer instructions 170
15.5.8 Branch instructions 187
15.5.9 Subroutine instructions 192
15.5.10 Interrupt instructions 197
15.5.11 Other instructions 199

APPENDIX A DEVELOPMENT TOOLS 201

A.1 Hardware List 201
A.2 Software List 202
A.3 PROM Programmers 203

APPENDIX B HOW TO ORDER THE MASK ROM 205

APPENDIX C INSTRUCTION INDEX 207

C.1 Instruction Index (by function) 207
C.2 Instruction Index (by alphabetic order) 208

APPENDIX D REVISION HISTORY 209

LIST OF FIGURES (1/2)

Fig. No.	Title	Page
2-1	Program Counter	23
2-2	Program Counter Value After Instruction Execution	24
2-3	Program Counter Value on Reset	24
2-4	Program Counter Value on Direct Branch Instruction Execution	25
2-5	Program Counter Value on Indirect Branch Instruction Execution	25
2-6	Program Counter Value on Direct Subroutine Call Instruction Execution	26
2-7	Program Counter Value on Indirect Subroutine Call Instruction Execution	26
3-1	Program Memory Map	30
3-2	Direct Subroutine Call (CALL addr)	34
3-3	If First Subroutine Address is in Page 1	35
3-4	Indirect Subroutine Call (CALL @AR)	36
3-5	Table Referencing (MOVT DBF, @AR)	37
4-1	Configuration of Data Memory	42
4-2	Configuration of System Register	43
4-3	Data Buffer	43
4-4	General Register	44
4-5	Configuration of Port Register (μPD17225)	44
5-1	Configuration of Stack (μPD17207)	45
6-1	Location of System Registers on Data Memory	51
6-2	Configuration of System Registers (μPD17204)	52
6-3	Configuration of Address Register (μPD17226)	53
6-4	Address Register of μPD172×× Subseries	53
6-5	Address Register Used as Peripheral Circuit	54
6-6	Configuration of Window Register	55
6-7	Example of Manipulating Window Register	55
6-8	Configuration of Bank Register (μPD17207)	56
6-9	Bank Register of μPD172×× Subseries	56
6-10	Configuration of Index Register (μPD17201A)	57
6-11	Example of Operation When MPE = 0, IXE = 0	61
6-12	Example of Operation When MPE = 1, IXE = 0	63
6-13	Example of Operation When MPE = 0, IXE = 1	65
6-14	Example of General Register Indirect Transfer Operation When MPE = 0, IXE = 1	67
6-15	Example of Operation When MPE = 0, IXE = 1 (array processing)	68
6-16	Configuration of General Register Pointer (μPD17201A)	69
6-17	Configuration of General Register (μPD17201A)	70
6-18	Configuration of Program Status Word	71
6-19	Functions of Program Status Word	72
7-1	Configuration of General Register (μPD17201A)	80

LIST OF FIGURES (2/2)

Fig. No.	Title	Page
8-1	Configuration of Register File	81
8-2	Relations between Register File and Data Memory	82
8-3	Accessing Register File with PEEK or POKE Instruction	85
9-1	Location of Data Buffer	89
9-2	Configuration of Data Buffer	90
9-3	Data Buffer and Peripheral Hardware (μPD17201A)	90
10-1	Configuration of ALU Block	96
11-1	Accepting Interrupt	115
11-2	Interrupt Processing Sequence	118
11-3	Returning from Interrupt Processing	119
12-1	Operation after Releasing STOP Mode	123
12-2	Operation after Releasing HALT Mode	124
13-1	Reset Operation by $\overline{\text{RESET}}$ Input	125
14-1	Program Memory Writing Sequence	130
14-2	Program Memory Reading Sequence	131

LIST OF TABLES

Table No.	Title	Page
3-1	The Program Memory of the μPD172×× Subseries	29
3-2	Vector Table for μPD17201A, 17207, and 17P207	32
3-3	Vector Table for μPD17203A, 17P203A, 17204, and 17P204	32
3-4	Vector Table for μPD17225, 17226, 17227, 17228 and 17P218	32
4-1	μPD172×× Subseries Data Memory	41
5-1	Operation of Stack Pointer	47
5-2	Operation When Subroutine Call or Return Instruction Is Executed	48
5-3	Operation When Table Reference Instruction Is Executed	48
5-4	Operation of Stack When Interrupt Is Accepted and Return Instruction Is Executed (μPD17207)	49
5-5	Operations of PUSH and POP Instructions	49
6-1	Data Memory Address Modification by Index Register and Data Memory Row Address Pointer	59
6-2	Status of Compare Flag (CMP) and Set and Reset Conditions of Zero Flag (Z)	73
8-1	Peripheral Hardware of μPD172×× Subseries	83
9-1	Peripheral Hardware (μPD17201A)	91
10-1	ALU Processing Instructions	98
10-2	Results for Binary 4-bit and BCD Operations	101
10-3	Arithmetic Operation Instructions	103
10-4	Logical Operation Instructions	106
10-5	Logical Operation Truth Table	106
10-6	Bit Testing Instructions	106
10-7	Compare Instructions	108
12-1	Status in Standby Mode	121
14-1	Pins Used to Write and Verify Program Memory	127
14-2	One-Time PROM Models and Corresponding Mask ROM Models	127
14-3	Differences between μPD17P203A and μPD17203A	128
14-4	Differences between μPD17P204 and μPD17204	128
14-5	Differences between μPD17P207, μPD17201A, and μPD17207	128
14-6	Differences between μPD17P218, μPD17225, 17226, 17217 and 17228	128
14-7	Selecting Operation Modes	129

CHAPTER 1 GENERAL

★ The μPD17201A, 17203A, 17204, 17207, 17225, 17226, 17227, and 17228 are microcontrollers for infrared remote controllers, integrating a CPU, ROM, RAM, I/O ports, timer, and remote controller carrier generator on a single chip.

The μPD17P203A, 17P204, 17P207, and 17P218 are one-time PROM models suitable for evaluating programs at system development or for small-scale production.

★ 1.1 List of Functions

- **4-bit single-chip microcontrollers (µPD172×× subseries) (1/2)**

Item Part Number	µPD17201A	µPD17207	µPD17203A	µPD17204
ROM capacity	3072 × 16 bits	4096 × 16 bits	4096 × 16 bits	7936 × 16 bits
RAM capacity	336 × 4 bits		336 × 4 bits	
Static RAM	None		4096 × 4 bits	2048 × 4 bits
LCD controller/driver	136 segments MAX.		None	
Carrier generator circuit (REM) for infrared remote controller	Provided (LED output is active-high)		Provided (LED output is active-low)	
Infrared remote controller reception preamplifier	None		Provided	
Number of I/O ports	19		28	
External interrupt (INT)	1 (rising-edge detection)		1 (rising and falling-edge detection)	
A/D converter	4 channels (8-bit A/D)		None	
Timer	2 channels { 8-bit timer Watch timer		4 channels { 8-bit timer: 3 channels Watch timer	
Watchdog timer	Provided ($\overline{\text{WDOUT}}$ output)			
Low-voltage detection circuit	None			
Serial interface	1 channel		1 channel	
Stack level	5 levels			7 levels
Minimum instruction execution time	4 µs (at 4 MHz)			
Operating supply voltage	• 2.2 to 5.5 V (at 4 MHz) • 2.0 to 5.5 V (at 32 kHz)			
Package	80-pin plastic QFP (14 × 20 mm)		52-pin plastic QFP (14 × 14 mm)	
One-time PROM model	µPD17P207		µPD17P203A	µPD17P204

Caution To use the NEC transmission format, apply to NEC for the custom code.

- **4-bit single-chip microcontrollers (μPD172×× subseries) (2/2)**

Item Part Number	μPD17225	μPD17226	μPD17227	μPD17228
ROM capacity	2048 × 16 bits	4096 × 16 bits	6144 × 16 bits	8192 × 16 bits
RAM capacity	111 × 4 bits		223 × 4 bits	
Static RAM	None			
LCD controller/driver	None			
Carrier generator circuit (REM) for infrared remote controller	Provided (no LED)			
Infrared remote controller reception preamplifier	None			
Number of I/O ports	20			
External interrupt (INT)	1 (rising-edge and falling-edge detection)			
A/D converter	None			
Timer	2 channels 8-bit timer Basic interval timer			
Watchdog timer	Provided ($\overline{\text{WDOUT}}$ output)			
Low-voltage detection circuit[Note]	Provided ($\overline{\text{WDOUT}}$ output: mask option)			
Serial interface	None			
Stack level	5 levels			
Minimum instruction execution time	• 2 μs (at 8 MHz : high-speed mode) • 4 μs (at 8 MHz : normal mode)			
Operating supply voltage	• 2.2 to 3.6 V (at 8 MHz : high-speed mode) • 2.0 to 3.6 V (at 8 MHz : normal mode)			
Package	• 28-pin plastic SOP (375 mil) • 28-pin plastic shrink DIP (400 mil)			
One-time PROM model	μPD17P218			

Note Although the circuit configuration is the same, the electrical characteristics differ depending on the product.

Caution To use the NEC transmission format, apply to NEC for the custom code.

[MEMO]

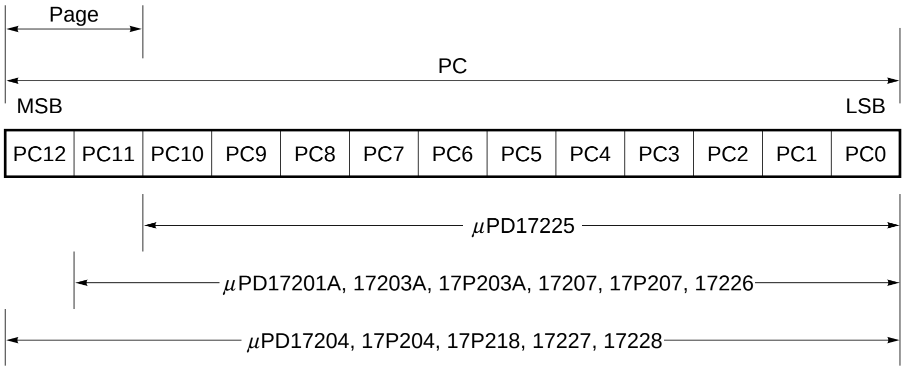
CHAPTER 2 PROGRAM COUNTER (PC)

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

2.1 Program Counter Configuration

The program counter is a binary counter consisting of up to 13 bits, as shown in Figure 2-1.
The contents of the 13-bit binary counter are incremented each time an instruction is executed.
The program counter transfers data with the address stack and address register in 16-bit units.
At this time, the bits in the program counter that exceed the program memory address range are fixed at 0.

★ **Figure 2-1. Program Counter**


Page
PC
MSB
LSB
PC12
PC11
PC10
PC9
PC8
PC7
PC6
PC5
PC4
PC3
PC2
PC1
PC0
μPD17225
μPD17201A, 17203A, 17P203A, 17207, 17P207, 17226
μPD17204, 17P204, 17P218, 17227, 17228

2.2 Program Counter Operations

Usually, the program counter contents are automatically incremented each time an instruction executed.

When the reset signal has been input, if a branch, subroutine call, return, or table reference instruction has been executed, and if an interrupt has been accepted, a specified value is set in the program counter.

The following **2.2.1** through **2.2.7** describe the program counter operations, when each of the above instructions has been executed.

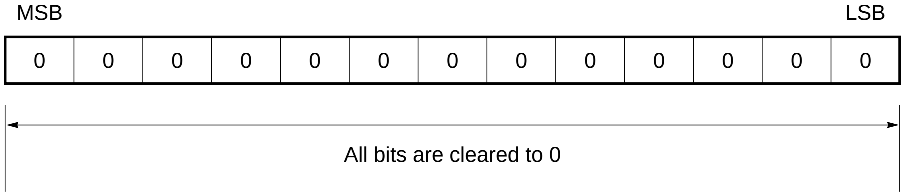
Figure 2-2. Program Counter Value After Instruction Execution

Bit of PC	Program Counter Value												
Instruction	PC12	PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
On reset	0	0	0	0	0	0	0	0	0	0	0	0	0
BR addr	Value specified by instruction												
CALL addr	0	0	Value specified by instruction										
BR @AR CALL @AR (MOVT DBF, @AR)	Address register contents												
RET RETSK RETI	Address stack register contents, specified by stack pointer (return address)												
On accepting interrupt	Interrupt vector address												

2.2.1 On reset

When the $\overline{\text{RESET}}$ pin is made low, the program counter contents are initialized to 0000H.

Figure 2-3. Program Counter Value on Reset



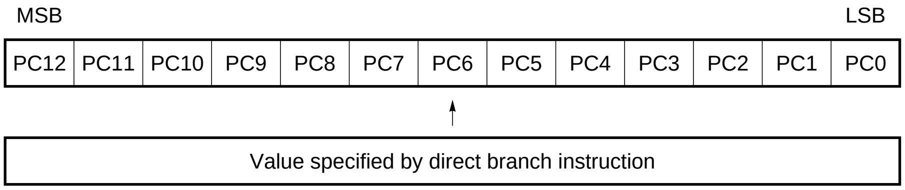
0	0	0	0	0	0	0	0	0	0	0	0	0

2.2.2 On branch instruction (BR) execution

Available branch instructions are a direct branch instruction (BR addr), whose branch destination is described as the operand of the instruction, and an indirect branch instruction (BR @AR), whose destination address is specified by the address register.

In Figure 2-4 below, an address specified for the direct branch instruction is set in the program counter.

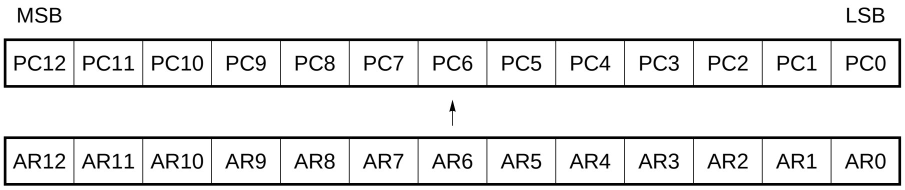
Figure 2-4. Program Counter Value on Direct Branch Instruction Execution



PC12	PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0

Figure 2-5 shows that the value of the address register is set by the indirect branch instruction in the program counter.

Figure 2-5. Program Counter Value on Indirect Branch Instruction Execution



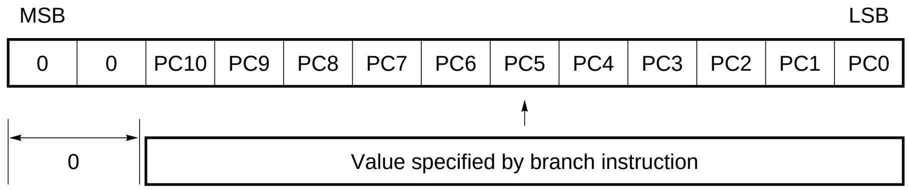
PC12	PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
AR12	AR11	AR10	AR9	AR8	AR7	AR6	AR5	AR4	AR3	AR2	AR1	AR0

2.2.3 On subroutine call instruction (CALL) execution

Available subroutine call instructions are a direct subroutine call instruction (CALL addr), whose destination is directly described as the operand, and an indirect subroutine call instruction (CALL @AR), whose destination is specified by the address register.

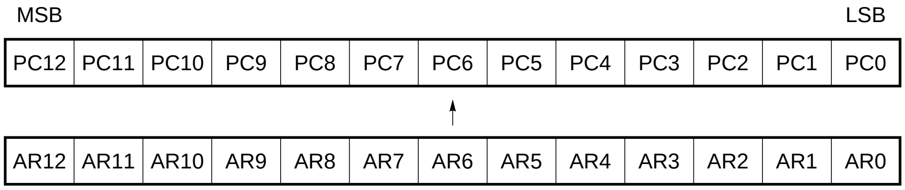
When the direct subroutine call instruction has been executed, the program counter value is pushed to the address stack, and the address, described as the operand of the instruction, is set in the program counter. The address range that can be specified by the direct subroutine call instruction is in page 0 of the program memory, i.e., from 0000H to 07FFH.

Figure 2-6. Program Counter Value on Direct Subroutine Call Instruction Execution



After the program counter value has been pushed to the stack address, by executing the indirect subroutine call instruction, the address register value is set in the program counter.

Figure 2-7. Program Counter Value on Indirect Subroutine Call Instruction Execution



2.2.4 On return instruction (RET, RETSK, or RETI) execution

The value (address) pushed to the address stack is restored to the program counter, by executing a return instruction (RET, RETSK, or RETI), and the instruction at that address is executed.

2.2.5 On table reference instruction (MOVT) execution

By executing the table reference instruction (MOVT DBF,@AR), the program counter value is pushed to the address stack, and the address register value is set in the program counter. The program memory contents addressed by this value are read to the data buffer (DBF). After the program memory contents have been read, the value pushed to the address stack is restored to the program counter.

Because of these operations, the table reference instruction consumes one level of the stack. Therefore, when executing this instruction, pay attention to the stack level nesting.

2.2.6 On skip instruction (SKE, SKGE, SKLT, SKNE, SKT, or SKF) execution

When the skip condition of the skip instruction (SKE, SKGE, SKLT, SKNE, SKT, or SKF) has been satisfied, the instruction next to the skip instruction is executed as a no-operation (NOP) instruction. Consequently, the number of instructions to be executed and the execution time do not vary, when the skip instruction has been executed, regardless of whether or not the skip condition is satisfied.

2.2.7 On accepting interrupt

When an interrupt has been accepted, the program counter value is pushed to the address stack, and the vector address corresponding to each interrupt is set in the program counter.

[MEMO]

CHAPTER 3 PROGRAM MEMORY (ROM)

The program memory configuration for the μPD172×× subseries is as follows:

★ **Table 3-1. The Program Memory of the μPD172×× Subseries**

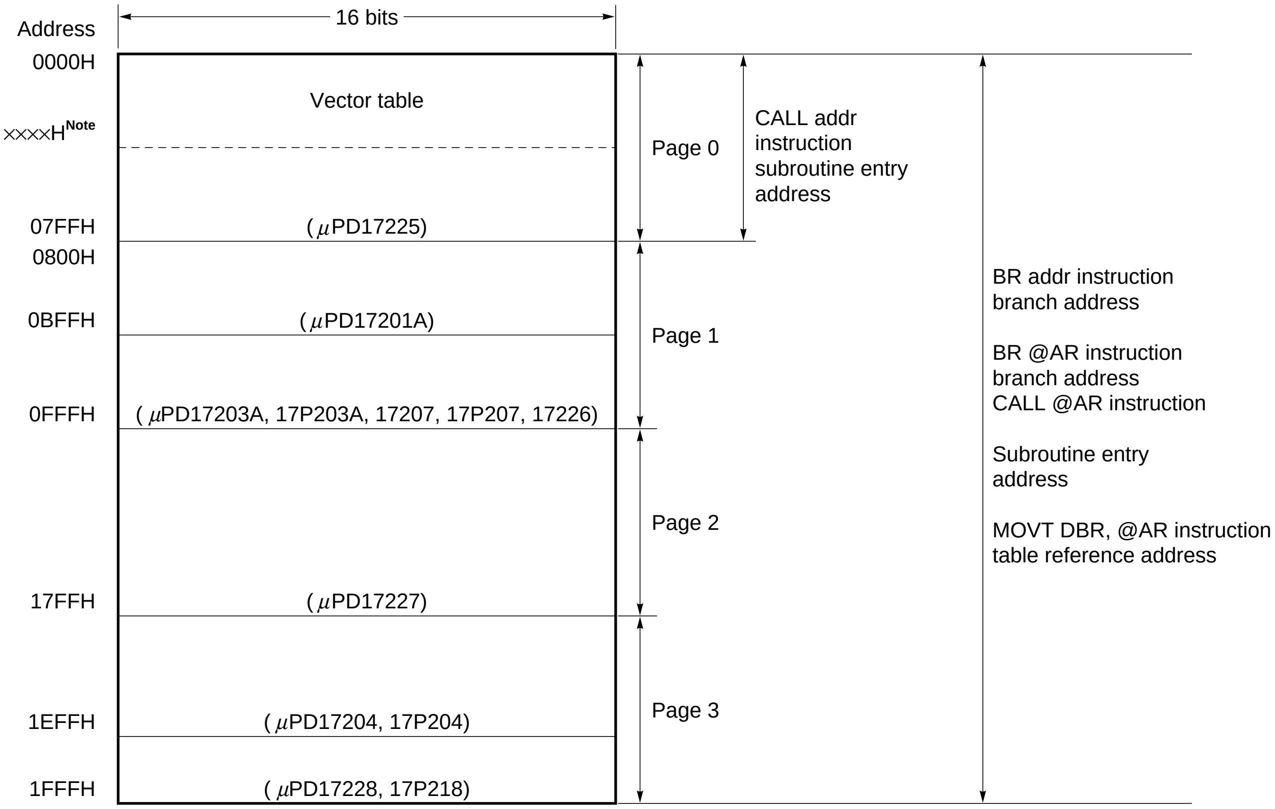
Part Number	ROM Capacity	ROM Address
μPD17225	2048 × 16 bits	0000H-07FFH
μPD17201A	3072 × 16 bits	0000H-0BFFH
μPD17203A	4096 × 16 bits	0000H-0FFFH
μPD17207		
μPD17226		
μPD17P203A		
μPD17P207		
μPD17227	6144 × 16 bits	0000H-17FFH
μPD17204	7936 × 16 bits	0000H-1EFFH
μPD17P204		
μPD17228	8192 × 16 bits	0000H-1FFFH
μPD17P218		

The program memory stores such things as the program, interrupt vector table, and fixed data table. The program memory is addressed by the program counter.

3.1 Program Memory Configuration

Figure 3-1 shows the program memory map. All the addresses of the program memory can be specified by the BR addr, BR @AR, CALL @AR, MOVT DBF, and @AR instructions. However, only addresses 0000H-07FFH can be specified as the subroutine entry address of the CALL addr instruction.

★ **Figure 3-1. Program Memory Map**



Note The vector address differ depending on the product.

3.2 Program Memory Function

The program memory has the following two major functions:

(1) Storing the program
(2) Storing constant data

A program is a collection of “instructions” that directs the central processing unit (CPU) on how to operate. The CPU sequentially performs processing in accordance with the instructions described in a program. In other words, the CPU reads instructions from the program stored in the program memory, and executes processing in accordance with the instructions.

All instructions are 16 bits, 1 word long. Therefore, one instruction can be stored in one address in a 16-bit program memory.

Constant data is predetermined data, such as a display pattern. The constant data can read the contents of the program memory to the data buffer (DBF) on the data memory by using the MOVT instruction, which is exclusively used for reading constant data. Reading constant data from a program memory like this is called “table reference”.

Because the program memory is a read-only memory (ROM), its contents cannot be rewritten by instructions.

3.2.1 Storing program

The program stored in the program memory is usually executed on an address-by-address basis starting from address 0000H. However, to execute another program when a certain condition is satisfied, the program execution flow must be changed. To do this, a branch instruction (BR) is used.

If it is necessary to execute the same program repeatedly, and if the same program is described each time that program is to be executed, it will be inefficient. In this case, therefore, only one program is described. It is called by a CALL instruction whenever necessary. This program is known as a “subroutine”. As opposed to the subroutine, the program normally executed is called the “main routine”.

If there is a program that is to be executed independently from the program execution flow, when a certain condition is satisfied, a function called an interrupt is used. By using the interrupt function, execution can be branched to a predetermined address (called a vector address) independently from the program execution flow, when a certain condition has been satisfied.

(1) Vector table

The addresses to which the program execution are to branched (vector address), when the reset signal has been input or an interrupt has occurred, are listed in **Tables 3-2** through **3-5**.

Table 3-2. Vector Table for μPD17201A, 17207, and 17P207

Vector Address	Interrupt Source
0000H	Reset
0001H	Serial interface interrupt
0002H	Watch timer interrupt
0003H	External input (INT) interrupt
0004H	8-bit timer interrupt

Table 3-3. Vector Table for μPD17203A, 17P203A, 17204, and 17P204

Vector Address	Interrupt Source
0000H	Reset
0001H	XRAM address interrupt
0002H	Serial interface interrupt
0003H	Watch timer interrupt
0004H	External input (INT) interrupt
0005H	10-bit timer interrupt (TM1)
0006H	8-bit timer interrupt (TM0)
0007H	16-bit timer interrupt (TM2)
0008H	Envelope circuit output interrupt

★

Table 3-4. Vector Table for μPD17225, 17226, 17227, 17228 and 17P218

Vector Address	Interrupt Source
0000H	Reset
0001H	Basic interval timer interrupt
0002H	External input (INT) interrupt
0003H	8-bit timer interrupt

(2) Direct branch

The direct branch instruction (BR addr) specifies a program memory address, to which the execution is to be branched, using the low-order 2 bits in the op code for the instruction and 11 bits of the operand. Therefore, the direct branch instruction can branch the execution to any of the program memory addresses.

(3) Indirect branch

The indirect branch instruction (BR @AR) branches the execution to an address indicated by the address register (AR). Therefore, this instruction can branch the execution to any of the program memory addresses. Also, refer to **6.2 Address Register**.

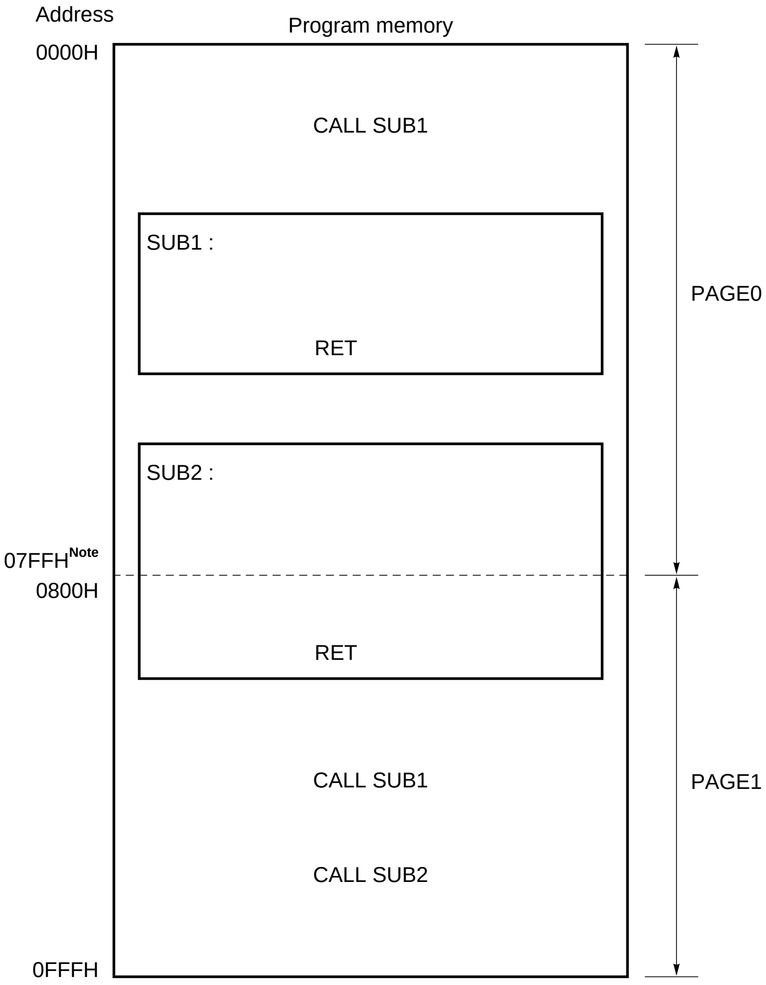
(4) Direct subroutine call

The direct subroutine call instruction (CALL addr) specifies the program memory address from which a subroutine is to be called, using the 11 operand bits for the instruction. Therefore, to use the direct subroutine call instruction, the address to be called, i.e., the first address in the subroutine to be called, must be in page 0 (addresses 0000H-07FFH). A subroutine in the other pages (addresses 0800H-1FFFH) cannot be called. However, the direct subroutine call instruction itself, or a subroutine return instruction (RET or RETSK), can be in a page other than page 0.

Example 1 If the first address in a subroutine is in page 0

As long as the first address in a subroutine is in page 0, as shown in Figure 3-2, the call instruction that calls this subroutine and the return instruction can be in any page. As long as the first address in the subroutine is in page 0, the CALL instruction can be used regardless of the page.
However, if the first address in the subroutine cannot be placed in page 0, use the technique shown in **Example 2**.

Figure 3-2. Direct Subroutine Call (CALL addr)

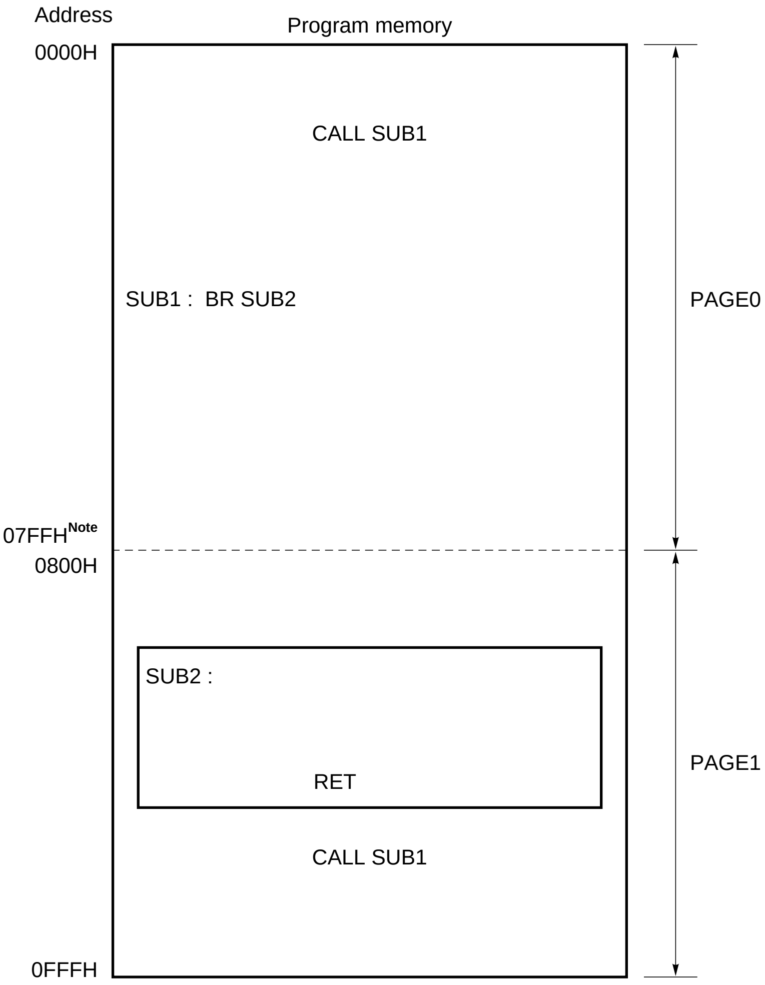


★ **Note** The program memory of the μPD17225 consists of addresses 0000H-07FFH.

Example 2 If the first address in a subroutine is in page 1

If the first address in the subroutine is in page 1, a branch instruction (BR) should be placed in page 0, through which the subroutine (SUB1) in page 1 is to be called, as shown in Figure 3-3.

Figure 3-3. If First Subroutine Address is in Page 1



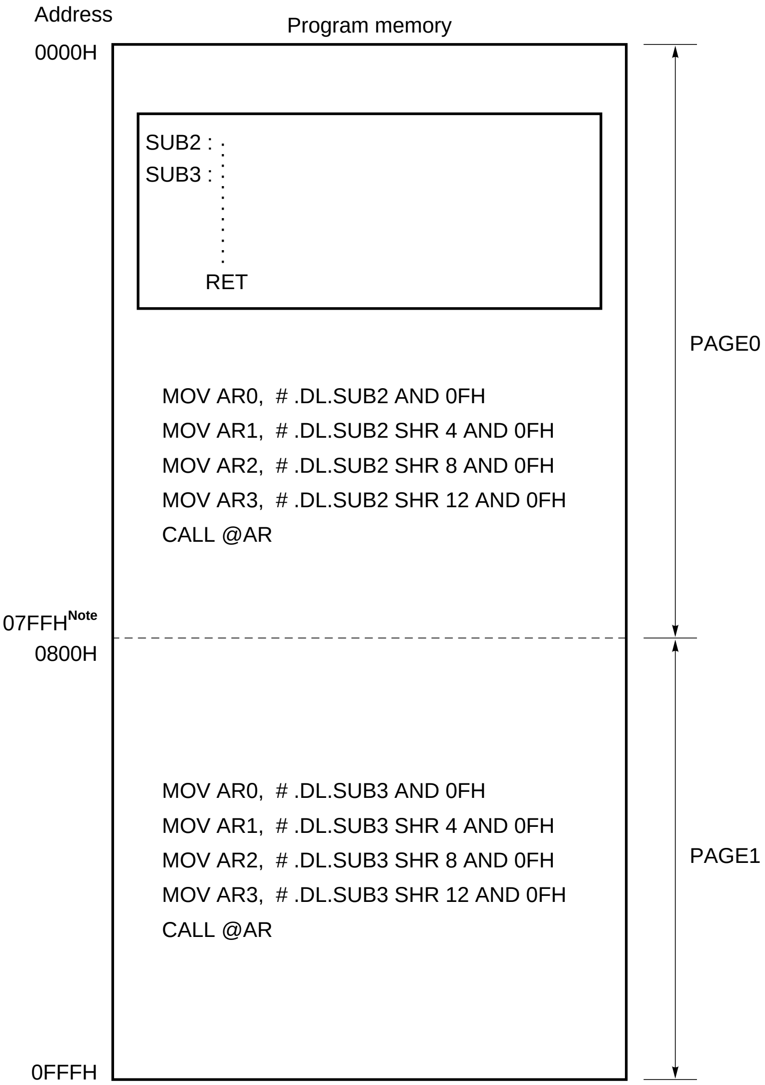
★ **Note** The program memory of the μPD17225 consists of addresses 0000H-07FFH.

(5) Indirect subroutine call

The indirect subroutine call (CALL @AR) instruction uses the address register (AR) value as the address from which a subroutine is to be called. Therefore, this instruction can call a subroutine from any of the program memory addresses.

Also refer to **6.2 Address Register (AR)**.

Figure 3-4. Indirect Subroutine Call (CALL @AR)


Address
Program memory
0000H
SUB2 :
SUB3 :
RET
PAGE0
MOV AR0, # .DL.SUB2 AND 0FH
MOV AR1, # .DL.SUB2 SHR 4 AND 0FH
MOV AR2, # .DL.SUB2 SHR 8 AND 0FH
MOV AR3, # .DL.SUB2 SHR 12 AND 0FH
CALL @AR
07FFHNote
0800H
MOV AR0, # .DL.SUB3 AND 0FH
MOV AR1, # .DL.SUB3 SHR 4 AND 0FH
MOV AR2, # .DL.SUB3 SHR 8 AND 0FH
MOV AR3, # .DL.SUB3 SHR 12 AND 0FH
CALL @AR
PAGE1
0FFFH

★ **Note** The program memory of the μPD17225 consists of addresses 0000H-07FFH.

3.2.2 Table reference

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

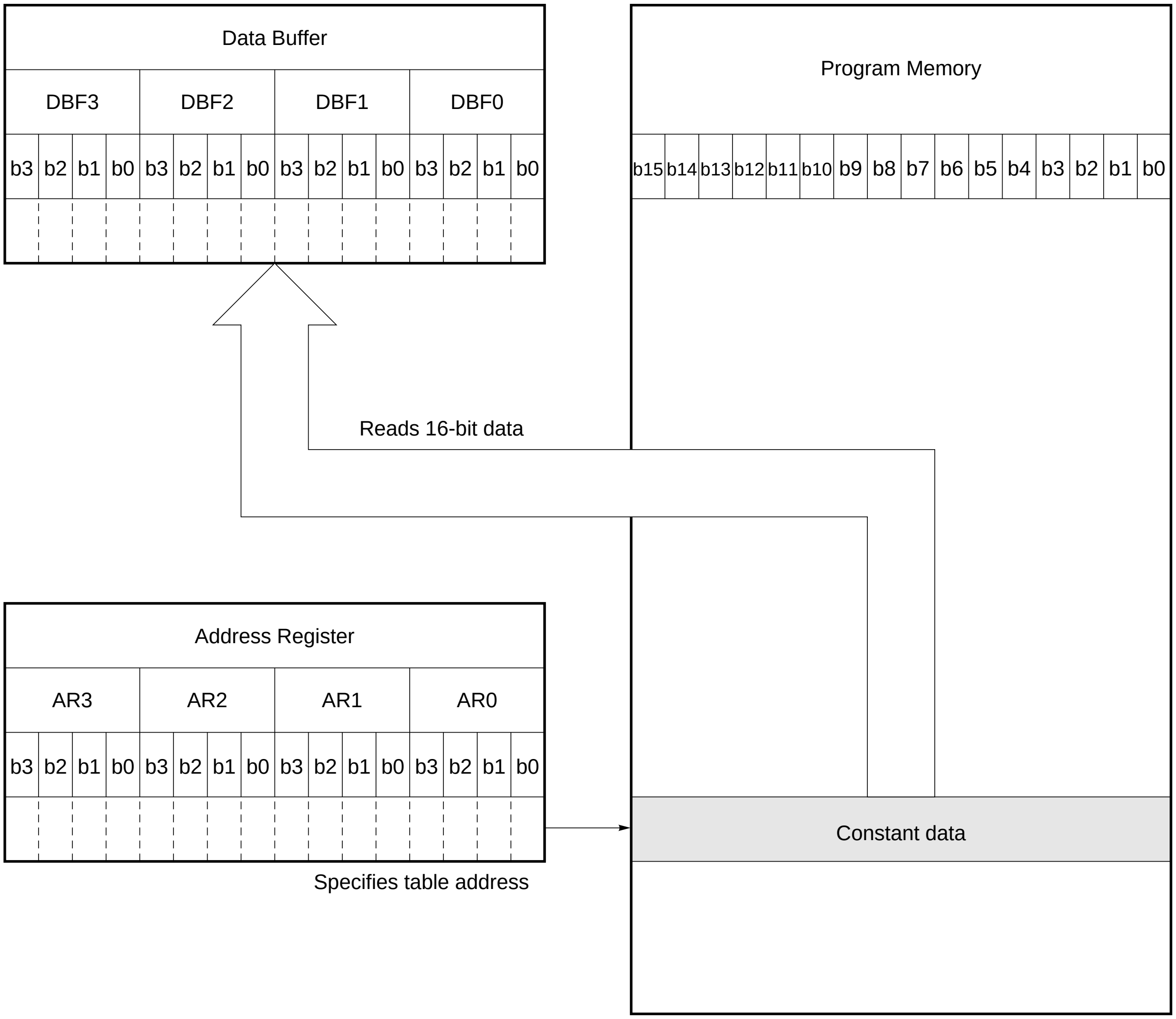
When the table reference instruction (MOVT DBF, @AR) is executed, the contents of the program memory address specified by the address register are stored in the data buffer.

Since the contents of a program memory address are 16 bits wide, constant data that is stored in the data buffer by the MOVT instruction is also 16 bits wide. By using the address register, any program memory address can be table-referenced.

When table referencing is performed, one stack level is temporarily used. Therefore, pay attention to the stack level.

Also refer to **6.2 Address Register (AR)** and **4.1.2 Data buffer (DBF)**.

Figure 3-5. Table Referencing (MOVT DBF, @AR)


Data Buffer
DBF3
DBF2
DBF1
DBF0
b3 b2 b1 b0 b3 b2 b1 b0 b3 b2 b1 b0 b3 b2 b1 b0
Program Memory
b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0
Reads 16-bit data
Address Register
AR3
AR2
AR1
AR0
b3 b2 b1 b0 b3 b2 b1 b0 b3 b2 b1 b0 b3 b2 b1 b0
Constant data
Specifies table address

(1) Constant table

The following is an example of a table reference program that references a constant table:

Example Program that reads the address OFFSET value of the constant data table .

```
OFFSET   MEM      0.00H            ; Stores offset address
         BANK0
         MOV      RPH, #0          ; Sets row address 7 in register pointer
         MOV      RPL, #7 SHL 1
ROMREF :
         BANK0                     ; Sets first address in table
         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 offset address
         ADDC     AR1, #0
         ADDC     AR2, #0
         ADDC     AR3, #0
         MOVT     DBF, @AR         ; Executes table referencing

           ≀      Program

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 destination table

The following shows an example of a table reference program for the branch destination table:

Example Program that branches to the address indicated by the address OFFSET value

```
OFFSET   MEM     0.00H            ; Stores offset address
         BANK0
         MOV     RPH, #0          ; Sets row address 7 in register pointer
         MOV     RPL, #7 SHL 1
ROMREF :
         BANK0                    ; Sets first address for table
         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 offset address
         ADDC    AR1, #0
         ADDC    AR2, #0
         ADDC    AR3, #0
         MOVT    DBF, @AR         ; Executes table referencing
         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
         DW      0400H
         DW      0800H
         DW      1000H
         DW      2000H
         DW      4000H
         DW      8000H
         END
```

[MEMO]

CHAPTER 4 DATA MEMORY (RAM)

The data memory stores data for arithmetic and control operations. The data can be written to or read from this memory by an instruction.

The data memory capacity list for the μPD172×× subseries is as follows:

★ **Table 4-1. μPD172×× Subseries Data Memory**

Part Number	RAM Capacity
μPD17225	111 × 4 bits (BANK0)
μPD17226	
μPD17227	223 × 4 bits (BANK0, BANK1)
μPD17228	
μPD17P218	
μPD17201A	336 × 4 bits (BANK0-BANK2)
μPD17203A	
μPD17P203A	
μPD17204	
μPD17P204	
μPD17207	
μPD17P207	

4.1 Data Memory Configuration

Figure 4-1 shows the configuration of the data memory.

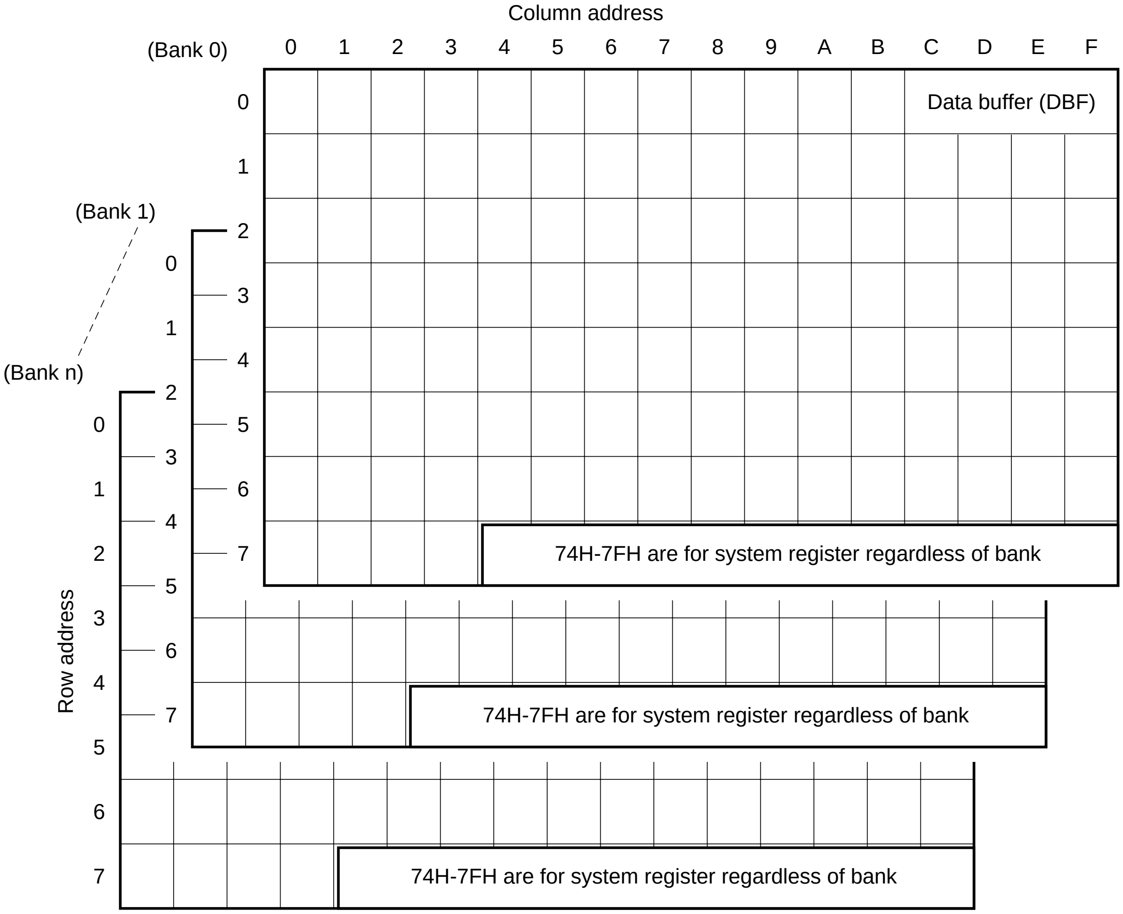
The data memory is managed with a conception called “bank”. The μPD1712× and 1713× have only bank 0.

Each bank of the data memory is assigned addresses with 4 bits, or a “nibble” of the memory corresponding to one address.

A data memory address is represented by 7 bits with the high-order 3 bits called a “row address” and the low-order 4 bits a “column address”. For example, address 1AH (0011010B) consists of a row address of 1H (001B) and a column address of AH (1010B).

The data memory is divided by function into the blocks described in **4.1.1** through **4.1.4**.

Figure 4-1. Configuration of Data Memory


Column address
(Bank 0)
0 1 2 3 4 5 6 7 8 9 A B C D E F
(Bank 1)
(Bank n)
Row address
0 1 2 3 4 5 6 7
Data buffer (DBF)
74H-7FH are for system register regardless of bank
74H-7FH are for system register regardless of bank
74H-7FH are for system register regardless of bank

4.1.1 System register configuration

The system register (SYSREG) consists of 12 nibbles assigned to addresses 74H-7FH of the data memory. SYSREG is assigned independently of the bank; therefore, the same system register exists at addresses 74H through 7FH of any bank.

Figure 4-2 shows the configuration of the system register.

Figure 4-2. Configuration of System Register

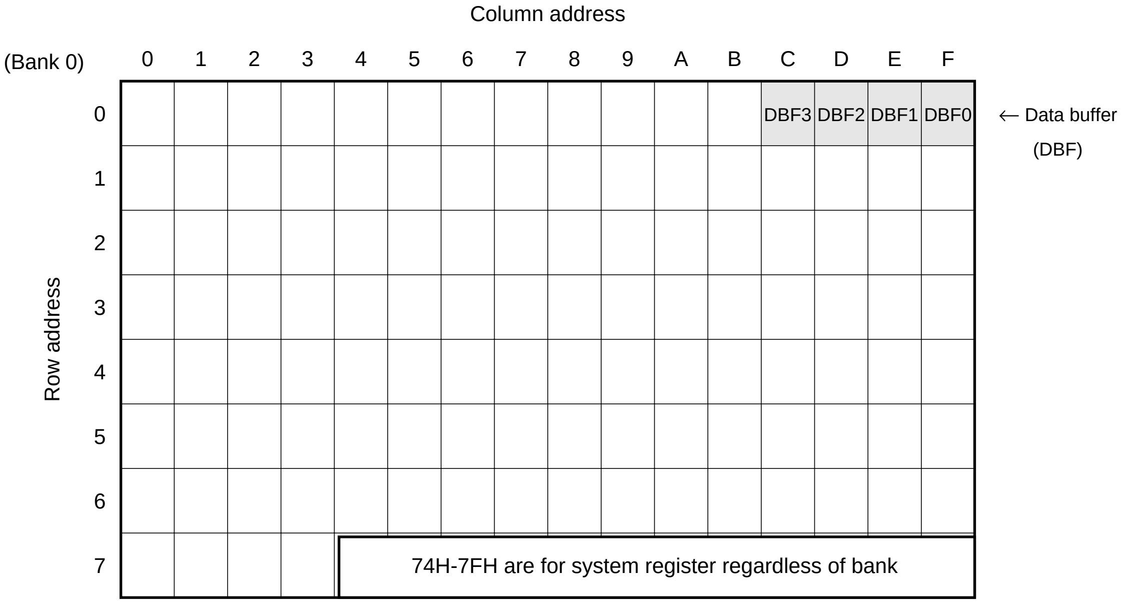
System register (SYSREG)												
Address	74H	75H	76H	77H	78H	79H	7AH	7BH	7CH	7DH	7EH	7FH
Symbol	Address register (AR)				Window register (WR)	Bank register (BANK)	Index register (IX) Data memory row pointer address (MP)			General register pointer (RP)	Program status word (PSWORD)	

4.1.2 Data buffer (DBF)

Data buffer (DBF) is assigned to addresses 0CH-0FH in the bank 0 in the data memory.

The data buffer is used for transferring data with peripheral hardware (by the PUT and GET instructions) and for table referencing (MOVT instruction).

Figure 4-3. Data Buffer



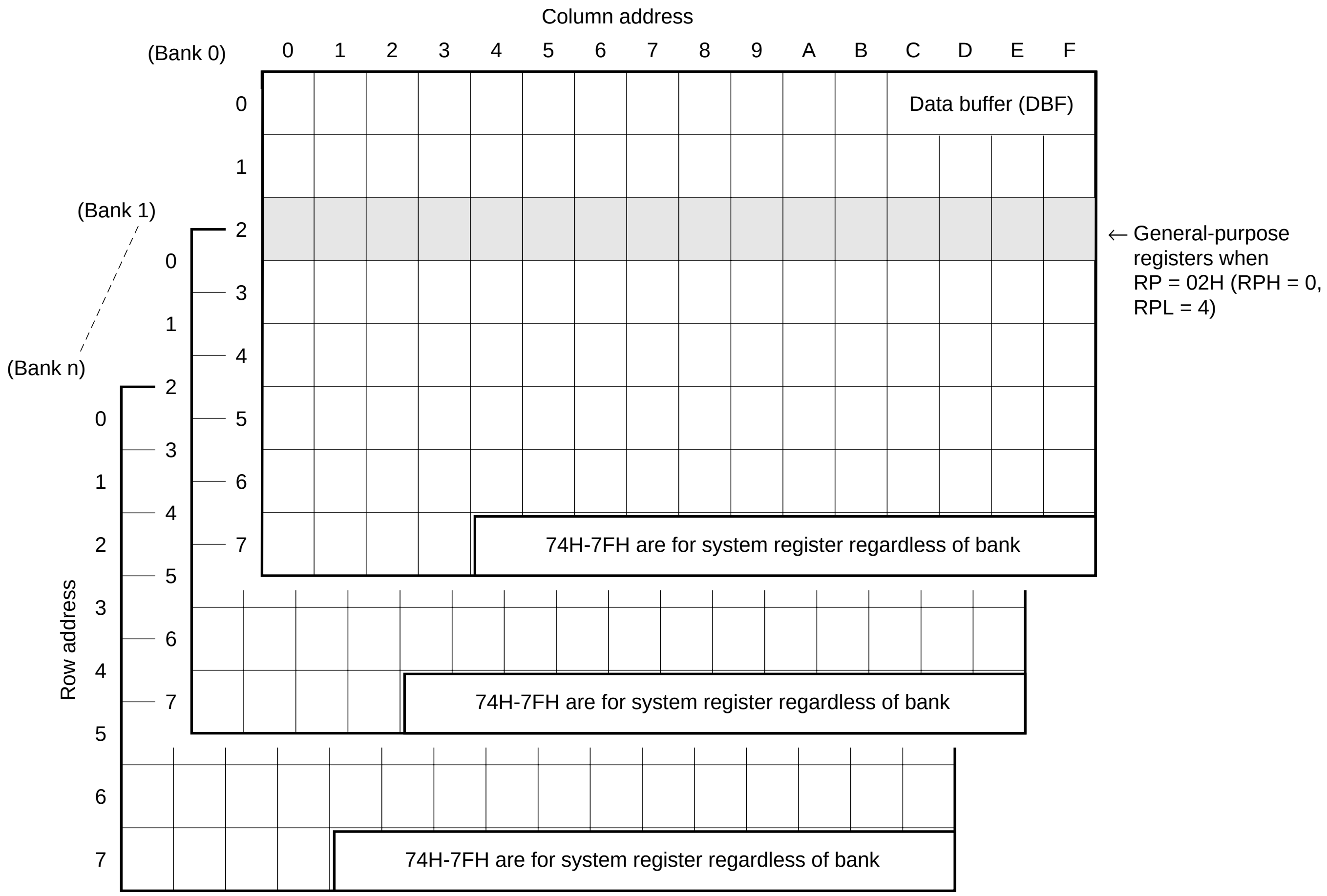
4.1.3 General register

The general register (GR) consists of 16 nibbles specified by any row address of the data memory.

The row address is specified by the general register pointer (RP) in the system register (SYSREG).

Figure 4-4 shows the configuration of the general register.

Figure 4-4. General Register


Column address
(Bank 0)
0 1 2 3 4 5 6 7 8 9 A B C D E F
Data buffer (DBF)
← General-purpose registers when RP = 02H (RPH = 0, RPL = 4)
(Bank 1)
(Bank n)
Row address
0 1 2 3 4 5 6 7
74H-7FH are for system register regardless of bank
74H-7FH are for system register regardless of bank
74H-7FH are for system register regardless of bank

4.1.4 Port register

The port register consists of 5 nibbles (5 × 4 bits) assigned to addresses 6FH through 73H of each bank of the data memory.

Figure 4-5 shows the configuration of the port register.

Figure 4-5. Configuration of Port Register (μPD17225)

Address	6FH				70H				71H				72H				73H			
Symbol	P0E				P0A				P0B				P0C				P0D			
	P0E3	P0E2	P0E1	P0E0	P0A3	P0A2	P0A1	P0A0	P0B3	P0B2	P0B1	P0B0	P0C3	P0C2	P0C1	P0C0	P0D3	P0D2	P0D1	P0D0

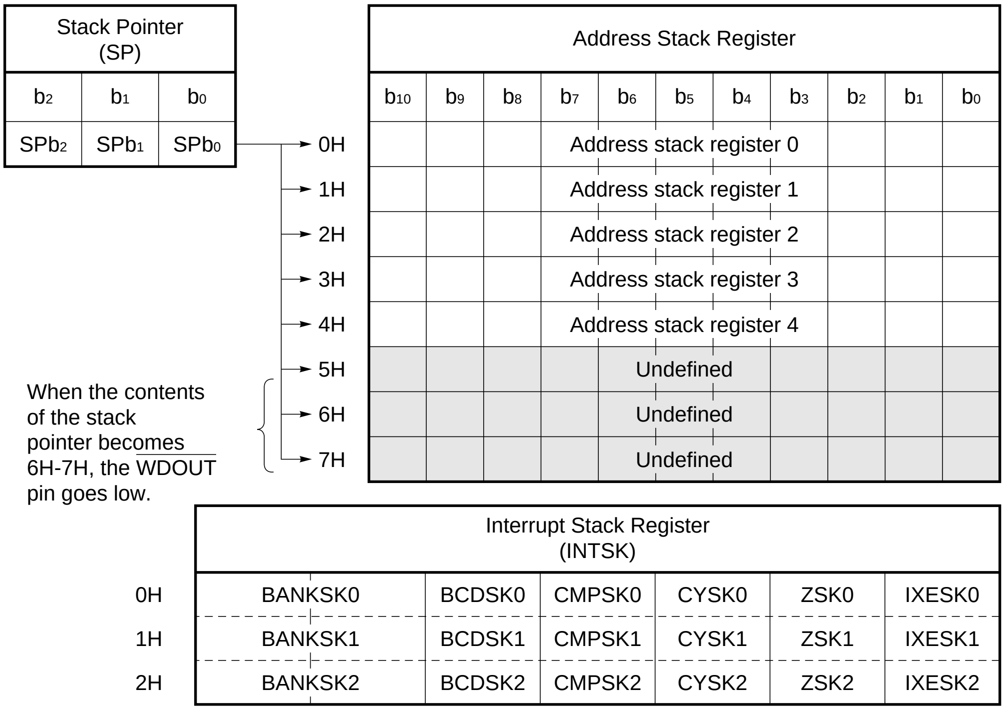
CHAPTER 5 STACK

Two types of stacks are available: an address stack that stores the contents of the program counter, and an interrupt stack that stores the contents of the system register. The address stack is a register to which the return address of the program is saved when a subroutine is called or when an interrupt is accepted. The interrupt stack is a register that saves part or all of the contents of the system register when an interrupt is accepted.

5.1 Configuration of Stack

Figure 5-1 shows the configuration of the stack of the μPD17207 as an example. This stack consists of a stack pointer (SP), which is a 3-bit binary counter, five 11-bit address stack registers, and three 7-bit interrupt stack registers.

Figure 5-1. Configuration of Stack (μPD17207)



Stack Pointer (SP)

b_2	b_1	b_0
SPb_2	SPb_1	SPb_0

Address Stack Register

	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0H				Address stack register 0							
1H				Address stack register 1							
2H				Address stack register 2							
3H				Address stack register 3							
4H				Address stack register 4							
5H				Undefined							
6H				Undefined							
7H				Undefined							

When the contents of the stack pointer becomes 6H-7H, the $\overline{WDOUT}$ pin goes low.

Interrupt Stack Register (INTSK)

0H	BANKSK0	BCDSK0	CMPSK0	CYSK0	ZSK0	IXESK0
1H	BANKSK1	BCDSK1	CMPSK1	CYSK1	ZSK1	IXESK1
2H	BANKSK2	BCDSK2	CMPSK2	CYSK2	ZSK2	IXESK2

Remark μPD17204 and 17P204 have seven address stack registers, 0H-6H.

5.2 Function of Stack

The stack is used to save the return address when a subroutine call instruction or table reference instruction is executed. When an interrupt is accepted, the return address of the program and the contents of the program status word (PSWORD) are automatically saved to the stack.

5.3 Address Stack Registers

The address stack registers save the value of the program counter (PC) plus 1, i.e., the return address when the first instruction cycle of a subroutine call (CALL addr, CALL @AR) or table reference (MOVT DBF, @AR) instruction is executed. When a stack manipulation instruction (PUSH AR) is executed, the contents of the address register (AR) are saved to an address stack register. The address stack register that stores data is specified by the value of the stack pointer (SP) minus 1 when any of the above instructions is executed.

When the second instruction cycle of a subroutine return (RET, RETSK), interrupt return (RETI), or table reference (MOVT DBF, @AR) instruction is executed, the contents of the address stack register specified by the stack pointer are restored to the program counter, and the value of the stack pointer is incremented by 1. When a stack manipulation instruction (POP AR) is executed, the value of the address stack register specified by the stack pointer is transferred to the address register, and the value of the stack pointer is incremented by 1.

If a subroutine call or interrupt is executed exceeding 5 levels[Note], **the $\overline{\text{WDOUT}}$ pin goes low**.

Note In case of μPD17204 and 17P204, 7 levels

5.4 Interrupt Stack Register

This section describes the interrupt stack register of the μPD17207 as an example.

The interrupt stack register consists of 3 × 7 bits as shown in Figure 5-1.

When an interrupt is accepted 7 bits, o.e., five flags (BCD, CMP, CY, Z, and IXE) of the program status word (PSWORD) in the system register (SYSREG) and bank registers are saved to this register. When an interrupt return instruction (RETI) is later executed, the contents of the interrupt stack register are restored to the program status word.

The interrupt stack register saves data each time an interrupt is accepted.

If an interrupt is accepted exceeding 3 levels, the first data saved to the interrupt stack register is lost.

Remark All the bits of PSWORD and BANK are automatically cleared to “0” after the contents of PSWORD and BANK have been saved to the interrupt stack register.

5.5 Stack Pointer (SP) and Interrupt Stack Register

The stack pointer (SP) is a 3-bit binary counter that specifies the addresses of five address stack registers as shown in Figure 5-1. The stack pointer is located at address 01H of the register file.

The value of the stack pointer is decremented by 1 when the first instruction cycle of a subroutine call (CALL addr, CALL @AR) or table reference (MOVT DBF, @AR) instruction is executed, when a stack manipulation instruction (PUSH AR) is executed, or when an interrupt is accepted, as indicated in Table 5-1; it is incremented by 1 when the second instruction cycle of a subroutine return (RET, RETSK) or table reference (MOVT DBF, @AR) instruction is executed, or when a stack manipulation instruction (POP AR) or interrupt return instruction (RETI) is executed. When an interrupt is accepted, the value of the interrupt stack register is also decremented by 1. The value of the interrupt stack register is incremented by 1 only when the interrupt return (RETI) instruction is executed.

Table 5-1. Operation of Stack Pointer

Instruction	Stack Pointer (SP) Value	Counter of Interrupt Stack Register
CALL addr CALL @AR MOVT DBF, @AR (1st instruction cycle) PUSH AR	–1	0
Accepting interrupt	–1	–1
RET RETSK MOVT DBF, @AR (2nd instruction cycle) POP AR	+1	0
RETI	+1	+1

Because the stack pointer is a 3-bit binary counter, as described above, it can take eight values: 0H through 7H. However, because only five address stack registers are available, the $\overline{\text{WDOUT}}$ pin goes low if the value of the stack pointer is 6 or more, or if the stack pointer value becomes –1 to –7 due to execution of a CALL or MOVT instruction, or due to acknowledgement of an interrupt, when the stack pointer value is 0.

Since the stack pointer is located on the register file, its value can be directly read or data can be written to the stack pointer by manipulating the register file with the PEEK or POKE instruction. Although the value of the stack pointer is changed at this time, the value of the address stack register is not affected.

The stack pointer is set to 5 on reset.

Remark µPD17204 and 17P204 have seven address stack registers. The stack pointers are set to 7 on reset.

5.6 Stack Operation When Subroutine or Table Reference Instruction Is Executed or When Interrupt Is Accepted

The following Paragraphs **5.6.1** through **5.6.3** describe the operations of the stack.

5.6.1 When subroutine call (CALL) or return (RET, RETSK) instruction is executed

Table 5-2 shows the operations of the stack pointer (SP), address stack registers, and program counter (PC) when the subroutine call or return instruction is executed.

Table 5-2. Operation When Subroutine Call or Return Instruction Is Executed

Instruction	Operation
CALL addr	<1> Increments value of program counter (PC) by 1 <2> Decrements value of stack pointer (SP) by 1 <3> Saves value of program counter (PC) to address stack register specified by stack pointer (SP) <4> Transfers value specified by operand (addr) of instruction to program counter
RET RETSK	<1> Restores value of address stack register specified by stack pointer (SP) to program counter (PC) <2> Increments value of stack pointer (SP) by 1

When the RETSK instruction is executed, the first instruction after restoration is treated as a no-operation (NOP) instruction.

5.6.2 When table reference instruction (MOVT DBF, @AR) is executed

Table 5-3 shows the operations when the table reference instruction is executed.

Table 5-3. Operation When Table Reference Instruction Is Executed

Instruction	Cycle	Operation
MOVT DBF, @AR	First	〈1〉 Increments value of program counter (PC) by 1 〈2〉 Decrements value of stack pointer (SP) by 1 〈3〉 Saves value of program counter (PC) to address stack register specified by stack pointer (SP) 〈4〉 Transfers value of address register (AR) to program counter (PC)
	Second	〈5〉 Transfers contents of program memory (ROM) specified by program counter (PC) to data buffer (DBF) 〈6〉 Restores value of address stack register specified by stack pointer (SP) to program counter (PC) 〈7〉 Increments value of stack pointer (SP) by 1

5.6.3 When interrupt is accepted or return instruction (RETI) is executed

Table 5-4 shows the operation of the stack of the μPD17207 when an interrupt is accepted and when the return instruction is executed.

Table 5-4. Operation of Stack When Interrupt Is Accepted and Return Instruction Is Executed (μPD17207)

Instruction	Operation
When interrupt is accepted	<1> Increments value of program counter (PC) by 1 However, if branch (BR) or subroutine call (CALL) instruction is executed when interrupt is accepted, address of program memory (ROM) to which execution branches or from which subroutine is called is loaded to PC <2> Decrements value of stack pointer (SP) by 1 <3> Saves value of program counter (PC) to address stack register specified by stack pointer (SP) <4> Saves BCD, CMP, CY, Z, and IXE flags of PSWORD and BANK to interrupt stack register <5> Transfers vector address to program counter (PC)
RETI	<1> Restores value of interrupt stack register to BCD, CMP, CY, Z, and IXE flags of PSWORD and BANK <2> Restores value of address stack register specified by stack pointer (SP) to program counter (PC) <3> Increments stack pointer (SP) by 1

5.7 Nesting Level of Stack, and PUSH and POP Instructions

The stack pointer (SP) operates as a 3-bit counter whose value is incremented or decremented by 1 when the subroutine call or return instruction is executed. Therefore, if the CALL or MOVT instruction is executed or an interrupt is accepted while the value of the stack pointer is 0H, and the value of the stack pointer is decremented by 1 to 7H as a result, the microcontroller judges that the program is not executed normally, and makes the $\overline{\text{WDOUT}}$ pin low.

To prevent this, the contents of the address stack registers should be saved by using the PUSH or POP instructions if the address stack registers are frequently used.

Table 5-5 shows the operations of the PUSH and POP instructions.

Table 5-5. Operations of PUSH and POP Instructions

Instruction	Operation
POP	<1> Transfers value of address stack register specified by stack pointer (SP) to address register (AR) <2> Increments value of stack pointer (SP) by 1
PUSH	<1> Decrements value of stack pointer (SP) by 1 <2> Transfers value of address register (AR) to address stack register specified by stack pointer (SP)

[MEMO]

CHAPTER 6 SYSTEM REGISTERS (SYSREG)

The system registers (SYSREG) are a group of registers that directly control the CPU and are located on the data memory.

6.1 Configuration of System Registers

Figure 6-1 shows the location of the system registers on the data memory. As shown, the system registers are located at addresses 74H through 7FH of the data memory, independently of the banks. This means that each bank has the same system register at addresses 74H through 7FH.

Because the system registers are located on the data memory, they can be manipulated by all data memory manipulation instructions. Therefore, the system registers can also be specified as general registers.

Figure 6-1. Location of System Registers on Data Memory

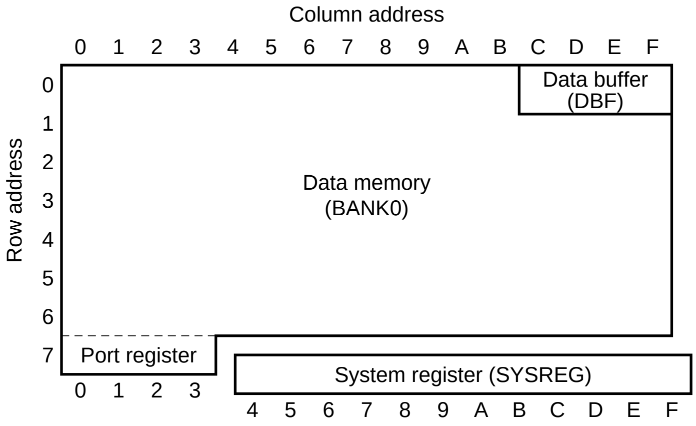


Figure 6-2 shows the configuration of the system registers. As shown in this figure, the system registers consist of the following seven registers:

• 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 6-2. Configuration of System Registers (μPD17204)

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	b3	b2	b1	b0	b3	b2	b1	b0	b3	b2	b1	b0	b3	b2	b1	b0	b3	b2	b1	b0	b3	b2	b1	b0	b3	b2	b1	b0	b3	b2	b1	b0	b3	b2	b1	b0	b3	b2	b1	b0	b3	b2	b1	b0	b3	b2	b1	b0
Data	0	0	0																		0	0			MPE	0	0										0	0						BCD	CMP	CY	Z	IXE
	(AR)																				(BANK)					(IX) (MP)											(RP)											
Initial value on reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Un-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

6.2 Address Register (AR)

6.2.1 Address register configuration

As an example, Figure 6-3 shows the configuration of the address register of the μPD17226.

As shown, this address register consists of 16 bits of the system registers, 74H through 77H (AR3 through AR0). However, the address register actually consists of 12 bits because the high-order 4 bits are always fixed to 0. On reset, all the 16 bits are reset to 0.

Figure 6-3. Configuration of Address Register (μPD17226)

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												
On reset	0				0				0				0			

★ **Figure 6-4. Address Register of μPD172×× Subseries**

Part Number	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
μPD17225	←						(AR)						→
	0	0											
μPD17201A μPD17203A μPD17207 μPD17226	←						(AR)						→
	0												
μPD17204 μPD17227 μPD17228	←						(AR)						→

6.2.2 Address register function

The address register specifies an address of the program memory when the indirect branch (BR @AR), indirect subroutine call (CALL @AR), or table reference (MOVT DBF, @AR) instruction has been executed. The address register value can also be pushed to or popped from the stack by the stack manipulation instructions (PUSH AR and POP AR).

The following Paragraphs (1) through (4) describe the address register operations, when each of these instruction has been executed.

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

(1) Table reference instruction (MOVT DBF, @AR)

By executing the MOVT DBF, @AR instruction, the value of the program memory (16-bit data) addressed by the value of the address register can be read to the data buffer (0CH-0FH of BANK0).

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

The PUSH AR instruction decrements the stack pointer (SP) contents and then stores the address register contents to the address stack specified by the stack pointer.

The POP AR instruction transfers the address stack contents, specified by the stack pointer to the address register, and then increments the stack pointer contents.

Also refer to **CHAPTER 5 STACK**.

(3) Indirect branch instruction (BR @AR)

By executing the BR @AR instruction, the address register value can be branched to a program memory address.

(4) Indirect subroutine call instruction (CALL @AR)

By executing the CALL @AR instruction, a subroutine can be called from a program memory address indicated by the address register value.

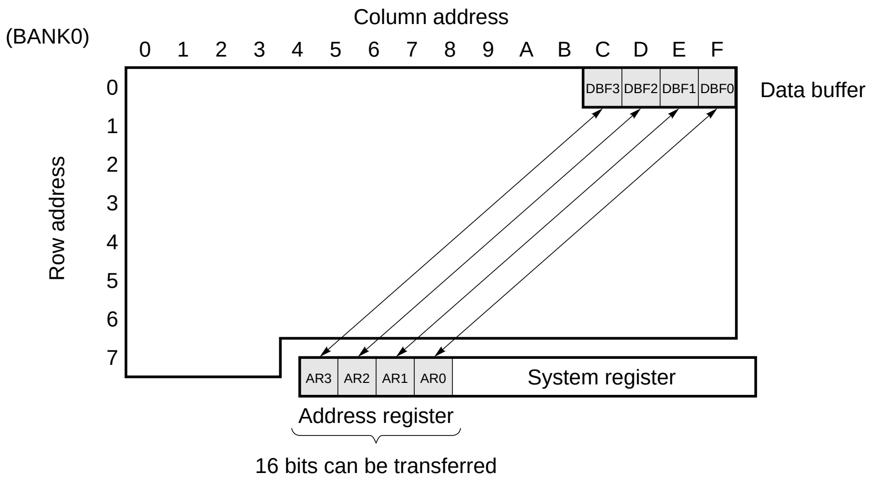
(5) Address register used as peripheral hardware

The address register can be manipulated in 4-bit units by the data memory manipulation instruction.

It is also possible to treat the address register as a peripheral hardware register to transfer 16-bit data with the data buffer. Therefore, the address register can transfer 16-bit data with the data buffer, by using the PUT AR, DBF and GET DBF, AR instructions.

The data buffer is assigned to addresses 0CH through 0FH in BANK0 of the data memory.

Figure 6-5. Address Register Used as Peripheral Circuit



6.3 Window Register (WR)

6.3.1 Window register configuration

Figure 6-6 shows the configuration of the window register (WR).

As shown, the window register consists of 4 bits at address 78H of the system registers, and its contents are undefined on reset. If the HALT or STOP mode has been released by using the $\overline{\text{RESET}}$ pin, the window register retains the previous contents.

Figure 6-6. Configuration of Window Register

Address	78H			
Name	Window register			
Symbol	WR			
Bit	b_3	b_2	b_1	b_0
Data	↔			
On reset	Undefined			

6.3.2 Window register function

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

The dedicated instructions PEEK WR, rf and POKE rf, WR are used to transfer data.

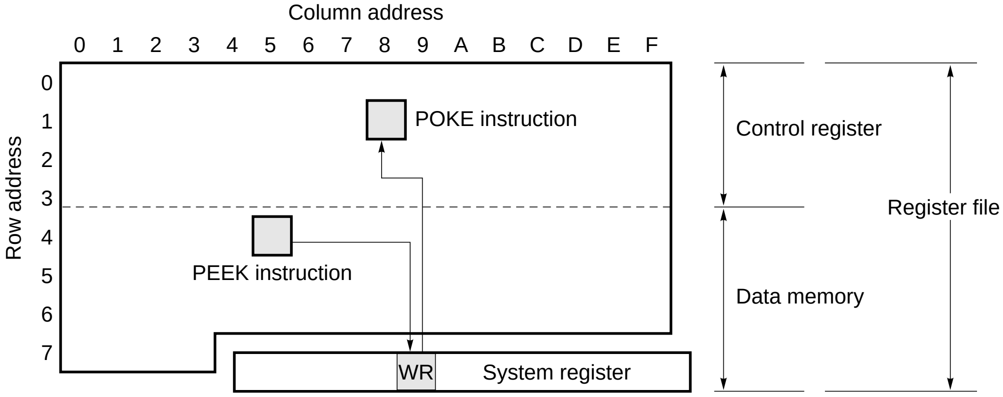
(1) PEEK WR, rf instruction

When this instruction has been executed, the register file contents, specified by rf, are transferred to the window register, as shown in Figure 6-7.

(2) POKE rf, WR instruction

When this instruction has been executed, the window register contents are transferred to the register file specified by rf, as shown in Figure 6-7.

Figure 6-7. Example of Manipulating Window Register



6.4 Bank Register (BANK)

Figure 6-8 shows the configuration of the bank register of the μPD17207.

The bank register consists of 4 bits at address 79H (BANK) of the system registers.

The bank register is used to select a bank of the RAM. Since the μPD17207 is provided with three banks, the high-order 2 bits of its bank register are fixed to 0.

When an interrupt is accepted, the contents of the bank register are saved to the interrupt stack register. After the bank register contents have been saved, BANK is cleared to “0”.

Figure 6-8. Configuration of Bank Register (μPD17207)

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

★ **Figure 6-9. Bank Register of μPD172×× Subseries**

Part number	b_3	b_2	b_1	b_0
μPD17225 μPD17226	(BANK)			
	0	0	0	0
μPD17201A μPD17203A μPD17P203A μPD17204 μPD17P204 μPD17207 μPD17P207	0	0		
	(BANK)			
μPD17227 μPD17228 μPD17P218	0	0	0	
	(BANK)			

6.5 Index Register (IX)

The index register is used to modify an address of the data memory when a data memory manipulation instruction is used.

Figure 6-10 shows the configuration of the index register of the μPD17201A as an example.

As shown in this figure, the index register consists of a total of 12 bits of the system register: IXH (7AH), IXM (7BH), and IXL (7CH). Of these bits, b_2 and b_1 at address 7AH are fixed to 0. The most significant bit of 7AH is a memory pointer enable flag (MPE).

The memory pointer enable flag is used to modify the address of a register specified by the operand @r of the MOV @r, m instruction with the contents of the data memory row address pointer (MP: memory pointer (low-order 3 bits of MPH and 4 bits of MPL)).

An index enable flag (IXE) is assigned as the least significant bit of the PSW. This flag is used to modify the address of the data memory addressed by the operand m of the ADD r,m instruction by ORing the data memory address with the contents of the index register (IX). When MPE = 0, the register address indicated by operand @r of a general register indirect transfer instruction (such as MOV @r, m) is also modified with the contents of IXH and IXM.

Figure 6-10. Configuration of Index Register (μPD17201A)

Address	74A				7BH				7CH				7FH			
Name	Index register (IX) Memory pointer (MP)															
Symbol	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_0	b_3	b_2	b_1	b_0
Flag	MPE															IXE
Data		← (IX) → ← (MP) → 0	0													
On reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

6.5.1 Index register and data memory row address pointer functions

The following paragraphs (1) and (2) describe the functions of the index register and data memory row address pointer:

(1) Index register

When a data memory manipulation instruction is executed, the index register modifies with its contents the bank and address of the data memory specified by the instruction.

However, the address modification by the index register is valid only when the index enable flag (IXE) is set.

To modify an address, the bank and address of the data memory are ORed with the contents of the index register, and the instruction is executed to the data memory at the address (called real address) specified by the result of the OR operation.

The index register modifies an address with all the data memory manipulation instructions.

The instructions that cannot be used for address modification are as follows:

```
MOVT   DBF, @AR     BR     addr       INC    AR     EI
PEEK   WR, rf       BR     @AR        INC    IX     DI
POKE   rf, WR       CALL   addr       RORC   r
GET    DBF, p       CALL   @AR        STOP   s
PUT    p, DBF       RET               HALT   h
PUSH   AR           RETSK             NOP
POP    AR           RETI
```

(2) Data memory row address pointer

The data memory row address pointer modifies with its contents the address at the indirect transfer destination when a general register indirect transfer instruction (MOV @r, m or MOV m, @r) is executed.

However, address modification by the data memory row address pointer is valid only when the data memory row address pointer enable flag (memory pointer enable flag: MPE) is set to 1.

In addition, the address specified by an instruction other than the general register indirect transfer instruction is not modified.

To modify an address, the bank and row address at the indirect transfer destination are replaced with the contents of the data memory row address pointer.

Figure 6-1 illustrates data memory address modification and indirect transfer address modification by the index register and data memory row address pointer.

Paragraphs **6.5.2** through **6.5.4** describe the operations to modify a data memory address by the index register and data memory row address pointer.

Table 6-1. Data Memory Address Modification by Index Register and Data Memory Row Address Pointer

IXE	MPE	General Register Address Specified by r			Data Memory Address Specified by m			Indirect Transfer Address Specified by @r		
		Bank	Row Address	Column Address	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	b_3 b_2 b_1 b_0	b_2 b_1 b_0	b_3 b_2 b_1 b_0
0	0	RP		r	BANK	m		BANK	m_R	(r)
0	1		ditto			ditto		MP		(r)
1	0		ditto		BANK	m Logical OR IX		BANK Logical OR IXH	m_R IXM	(r)
1	1	Setting prohibited								

Instructions modified				
Add/Sub	ADD ADDC SUB SUBC	r	m	
			m, #n4	
Logical	AND OR XOR	r	m	
			m, #n4	
Compare	SKE SKGE SKLT SKNE		m, #n4	
Judge-ment	SKT SKF		m, #n	
Transfer	LD ST	r	m	
	MOV		m, #n4	
		@r	m	Indirect transfer address

BANK	: Bank register
IX	: Index register
IXE	: Index enable flag
IXH	: Bits 10-8 of index register
IXM	: Bits 7-4 of index register
IXL	: Bits 3-0 of index register
m	: Data memory address specified by m_R, m_C
m_R	: Data memory row address (high)
MP	: Data memory row address pointer
MPE	: Memory pointer enable flag
r	: General register column address
RP	: General register pointer
(×)	: Contents addressed by ×
	× : Direct address such as m, r
	: Register such as BANK

6.5.2 When MPE = 0, IXE = 0 (no data memory modification)

As indicated in Table 6-1, the data memory address is not affected by the index register and data memory row address pointer.

(1) Data memory manipulation instruction

Examples 1. If general register is at row address 0

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

When the above instructions are executed, the contents of general register R003 and those of data memory M061 are added, and the result is stored to general register R003.

(2) General register indirect transfer

Examples 2. If general register is at row address 0

```
R005      MEM 0.05H
M034      MEM 0.34H
MOV       R005, #8        ; R005 ← 8
MOV     @R005, M034       ; Register indirect transfer
```

When the above instructions are executed, the contents of data memory M034 are transferred to address 38H of the data memory.

Therefore, the "MOV @r, m" instruction transfers the contents of the data memory specified by m to the data memory at an indirect address specified by @r of the same row address as m.

The indirect transfer address is the contents of the general register with a row address same as m (row address 3 in the above example) and a column address specified by r (8 in the above example). Therefore, it is 38H in the above example.

Examples 3. If general register is at row address 0

```
R00B    MEM 0.0BH
M034    MEM 0.34H
MOV     R00B, #0EH       ; R00B ← 0EH
MOV     M034, @R00B      ; Register indirect transfer
```

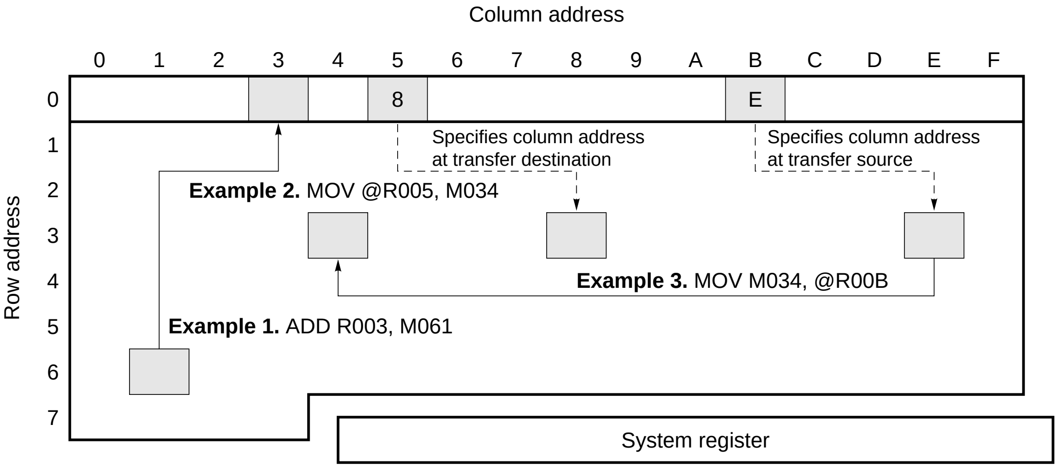
When the above instructions are executed, the contents of the data memory at address 3EH are transferred to data memory M034 as indicated in Figure 6-11.

Therefore, the “MOV m, @r” instruction transfers the contents of the data memory at an indirect address specified by @r of a row address same as m to the data memory addressed by m.

The indirect transfer address is the contents of the general register with a row address same as m (row address 3 in the above example) and a column address specified by r (0EH in the above example). Therefore, it is 3EH in the above example.

Comparing this with Example 2, the source address of the data memory whose contents are to be transferred and the destination address are exchanged.

Figure 6-11. Example of Operation When MPE = 0, IXE = 0


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

Generation of address 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

Generation of address 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		Content of R

6.5.3 When MPE = 1, IXE = 0 (diagonal indirect transfer)

As shown in Table 6-1, the bank and row address of the indirect transfer address specified by @r are the value of the data memory row address pointer only when a general register indirect transfer instruction (MOV @r, m or MOV m, @r) is executed.

Examples 1. If general register is at row address 0

```
R005    MEM 0.05H
M034    MEM 0.34H
MOV     MPL, #0110B    ; MP ← 6
MOV     MPH, #1000B    ; MPE ← 1
MOV     R005, #8       ; R005 ← 8
MOV     @R005, M034    ; Register indirect transfer
```

When the above instructions are executed, the contents of data memory M034 are transferred to data memory address 68H as shown in Figure 6-12.

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

At this time, the indirect address specified by @r is the contents of the general register with a bank and row address being the value of the data memory row address pointer (row address 6 in the above example) and a column address specified by r. It is, therefore, 68H in the above example.

When this is compared with Example 2 in **6.5.2**, the bank and row address of the indirect address specified by @r are specified by the data memory row address pointer in the above example, while the bank and row address of the indirect address in Example 2 in **6.5.2** are the same as m.

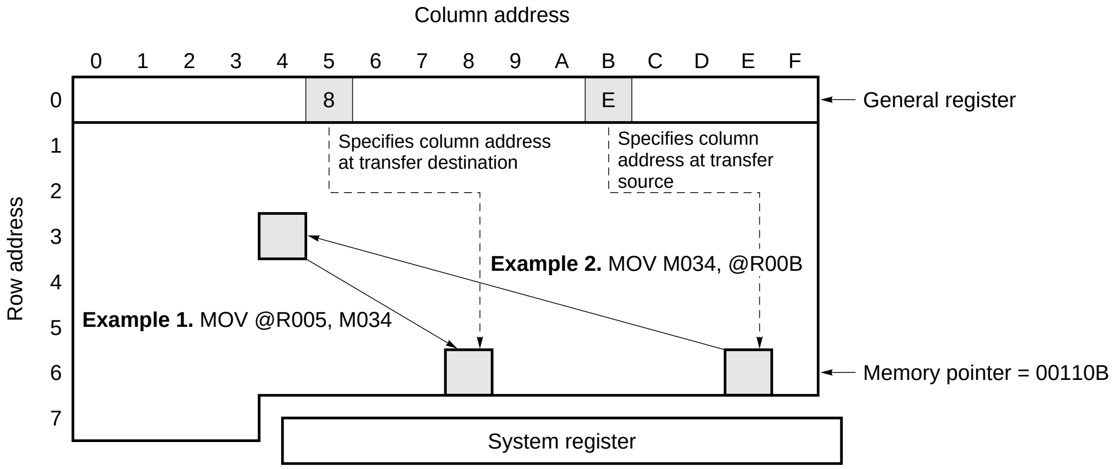
Therefore, general register indirect transfer can be diagonally carried out by setting MPE to 1.

Examples 2. If general register is at 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    ; Register indirect transfer
```

When the above instructions are executed, the contents of the data memory at address 6EH are transferred to data memory M034, as shown in Figure 6-12.

Figure 6-12. Example of Operation When MPE = 1, IXE = 0


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

Generation of address 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
	Content of MP		Content of R

Generation of address in Example 2

MOV M034, @R00B

	Bank	Row Address	Column Address
Data memory address M	0000	011	0100
General register address R	0000	000	0111
Indirect transfer address @R	0000	110	1110
	Content of MP		Content of R

6.5.4 When MPE = 0, IXE = 1 (data memory address index modification)

When a data memory manipulation instruction is executed as indicated in Table 6-1, all the banks and addresses of the data memory directly specified by the operand “m” of the instruction are modified by the index register.

When a general register indirect transfer instruction (MOV @r, m or MOV m, @r) is executed, the bank and row address of the indirect transfer address specified by @r are also modified by the index register.

To modify an address, the contents of the data memory address and those of the index register are ORed, and the instruction is executed to the data memory address (called an real address) specified by the result of the OR operation.

Here is an example:

Examples 1. If general register is at 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
```

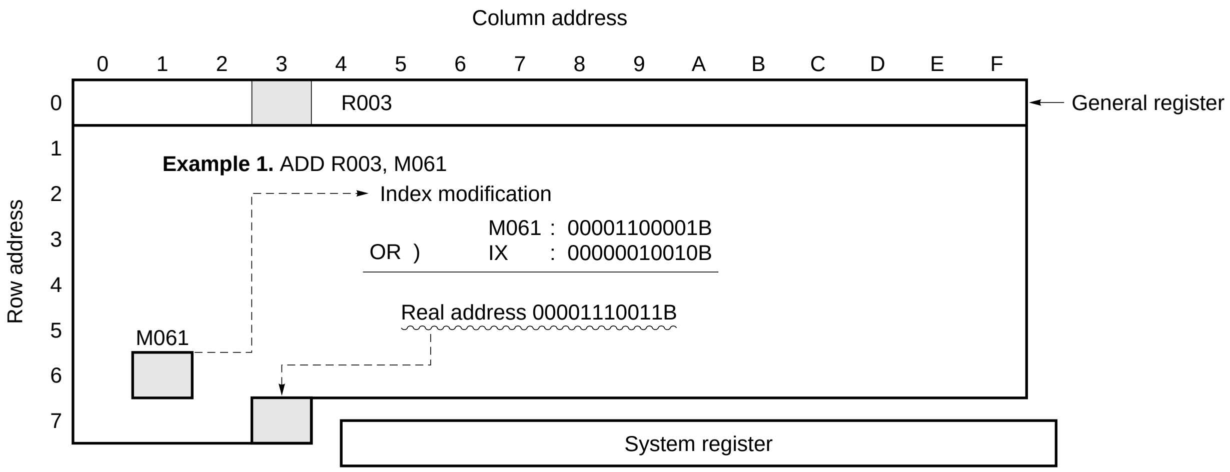
When the instructions in this example are executed, the contents of the data memory at address 73H (real address) and the contents of general register R003 (address 03H) are added, and the result is stored to general register R003 as indicated in Figure 6-13.

Therefore, when the “ADD r, m” instruction is executed, the data memory address specified by “m” (address 61H in the above example) is modified by the index register.

To modify the address, address 61H, which is the address of data memory M061 (00001100001B in binary), is ORed with the value of the index register (00000010010B in the above example), and the result 00001110011B is treated as the real address (address 73H), and the instruction is executed to this real address.

Comparing this with Example in **6.5.2** (when IXE = 0), the address of the data memory directly specified by the operand “m” of the instruction is modified (ORed) by the index register.

Figure 6-13. Example of Operation When MPE = 0, IXE = 1



Generation of address 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 addr. (ORed)	0000	111	0011

Instruction is executed to this address.

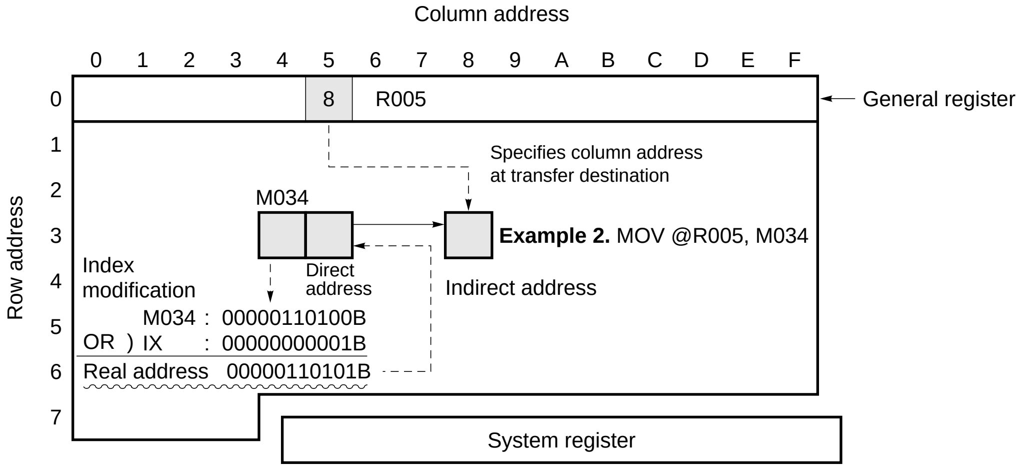
Examples 2. General register indirect transfer

If general register is in BANK0 at 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              ; Register indirect transfer
```

When the above instructions are executed, the contents of the data memory at address 35H are transferred to the address 38H of the data memory as shown in Figure 6-14.
Therefore, if the “MOV @r, m” instruction is executed when IXE = 1, the data memory address (direct address) specified by “m” is modified with the contents of the index register, and the bank and row address of the indirect address specified by “@r” are also modified by the index register. All the bank, row, and column address of the address specified by “m” are modified, and the bank and row address of the indirect address specified by “@r” are modified.
In the above example, therefore, the direct address is 35H and the indirect address is 38H.
When this is compared with Example 3 in **6.5.2** when IXE = 0, the bank, row, and column address of the direct address specified by “m” are modified by the index register and general register indirect transfer is executed to the row address same as the modified data memory address in the above example, while the direct address is not modified in Example 3 in **6.5.2**.

Figure 6-14. Example of General Register Indirect Transfer Operation When MPE = 0, IXE = 1



Examples 3. To clear contents of all data memory 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                ; Clears data memory specified by IX to 0
        INC     IX                      ; IX ← IX + 1
        AND     PSW, #1110B             ; IXE ← 0: Since IXE is
                                        ; at address 7FH, it is not modified by IX
        SKE     IXM, #0111B             ; Row address 7?
        BR      LOOP                    ; LOOP if not 7 (row address is not cleared)
```

Examples 4. Processing of array

Suppose 8-bit data A is defined one-dimensionally as shown in Figure 6-15. To execute the following operation, the instructions below should be executed:

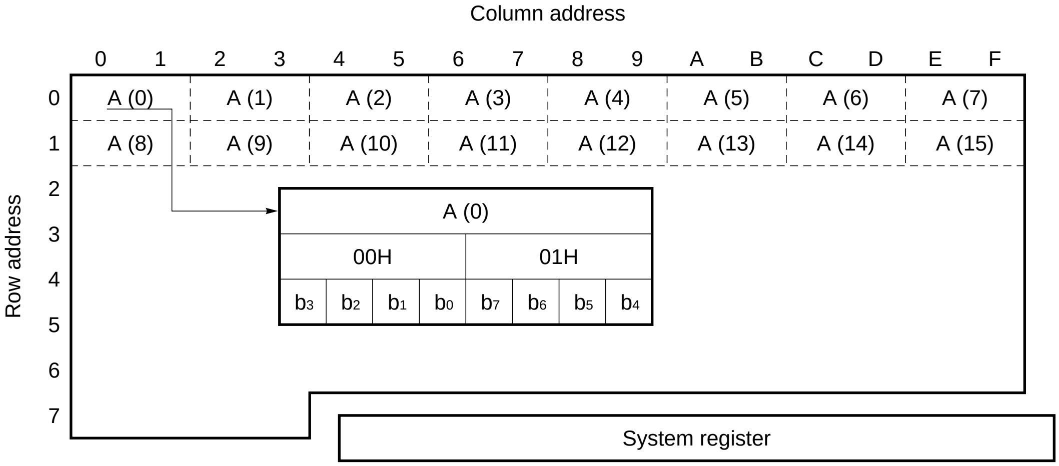
A (N) = A (N) + 4 (0 ≤ N ≤ 15)

Where general register is at row address 7

```
M000    MEM 0.00H
M001    MEM 0.01H
MOV     IXH, #0                ; IX ← 2N
MOV     IXM, #N SHR 3          ; Since array element is 8 bits, data memory address to
MOV     IXL, #N SHL 1 AND 0FH  ; be modified is shifted
OR      PSW, #.DF.IXE AND 0FH  ; IXE ← 1
ADD     M000, #4               ; Adds 4 to data memory M000
ADDC    M001, #0               ; and M001 that are modified by IX, i.e., adds 4 to 8-bit
                               ; array specified by A(N)
```

To specify N of array A(N) as indicated in the above example, specify a value 2 times that of N to the index register.

Figure 6-15. Example of Operation When MPE = 0, IXE = 1 (array processing)



6.6 General Register Pointer (RP)

6.6.1 General register pointer configuration

Figure 6-16 shows the configuration of the general register pointer of the μPD17201A as an example.

Figure 6-16. Configuration of General Register Pointer (μPD17201A)

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 (RP)	0						
On reset	0				0			

As shown in this figure, the general register pointer consists of 4 bits of address 7DH (RPH) of the system register and the high-order 3 bits of address 7EH (RPL). Actually, however, only the low-order 2 bits of address 7DH and the high-order 3 bits of address 7EH are valid because the high-order 3 bits of address 7DH are always fixed to 0.

On reset, all the bits are cleared to 0.

6.6.2 General register pointer function

This section describes the functions of the general register pointer of the μPD17201A as an example.

The general register pointer is used to specify a data memory area as a general register.

As a general register, 16 nibbles, which are at the same row address on the data memory, can be specified. Therefore, which row address is to be used is specified by using the general register pointer as shown in Figure 6-17.

Since the number of valid bits of the general register pointer is 5, the row addresses on the data memory that can be specified as general registers are 0H through 7H on BANK0 through 2. In other words, the entire data memory area can be specified as general registers.

When a data memory area is specified as a general register, an operation and data transfer can be executed between the general register and a data memory area.

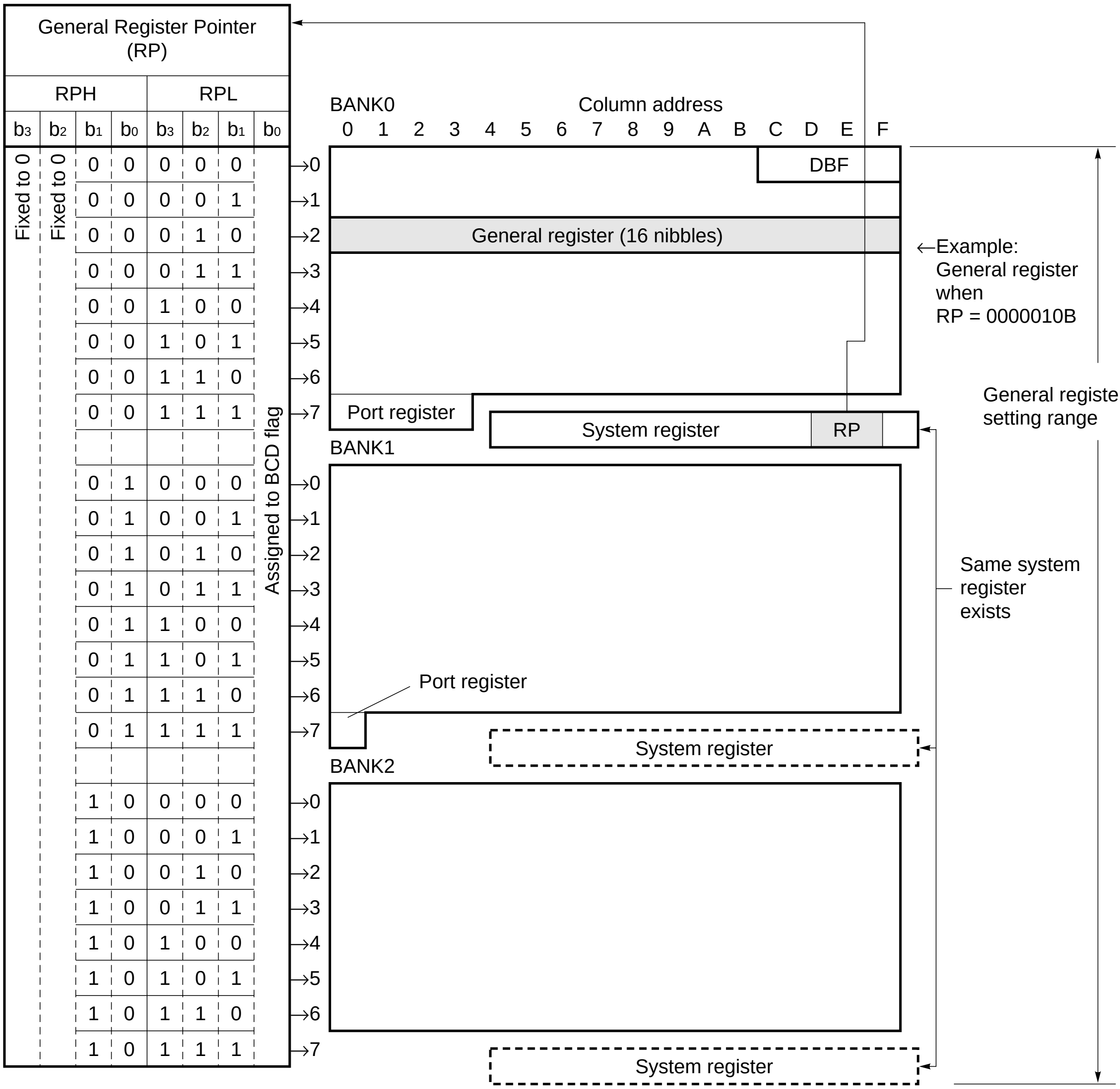
For example, suppose the following instruction is executed:

ADD r, m or LD r, m

Then addition or data transfer is executed between the general register addressed by operand “r” of the instruction and a data memory area addressed by “m”.

For details, refer to **CHAPTER 7 GENERAL REGISTER (GR)**.

Figure 6-17. Configuration of General Register (μPD17201A)



General Register Pointer (RP)
BANK0
Column address
0 1 2 3 4 5 6 7 8 9 A B C D E F
DBF
General register (16 nibbles)
←Example: General register when RP = 0000010B
Port register
System register
RP
General register setting range
BANK1
Same system register exists
Port register
System register
BANK2
System register

General Register Pointer (RP)								
RPH				RPL				
b3	b2	b1	b0	b3	b2	b1	b0	
Fixed to 0	Fixed to 0	0	0	0	0	0	Assigned to BCD flag	→0
		0	0	0	0	1		→1
		0	0	0	1	0		→2
		0	0	0	1	1		→3
		0	0	1	0	0		→4
		0	0	1	0	1		→5
		0	0	1	1	0		→6
		0	0	1	1	1		→7
		0	1	0	0	0		→0
		0	1	0	0	1		→1
		0	1	0	1	0		→2
		0	1	0	1	1		→3
		0	1	1	0	0		→4
		0	1	1	0	1		→5
		0	1	1	1	0		→6
		0	1	1	1	1		→7
		1	0	0	0	0		→0
		1	0	0	0	1		→1
		1	0	0	1	0		→2
		1	0	0	1	1		→3
		1	0	1	0	0		→4
		1	0	1	0	1		→5
		1	0	1	1	0		→6
		1	0	1	1	1		→7

6.7 Program Status Word (PSWORD)

6.7.1 Program status word configuration

Figure 6-18 shows the program status word configuration.

Figure 6-18. Configuration of Program Status Word

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
Flag				BCD	CMP	CY	Z	IXE
On reset	0				0			

As shown in this figure, the program status word consists of a total of 5 bits of the system register: the least significant bit of RPL (7EH) and PSW (7FH).

Each of these bits functions as a binary coded decimal flag (BCD), compare flag (CMP), carry flag (CY), zero flag (Z), and index enable flag (IXE).

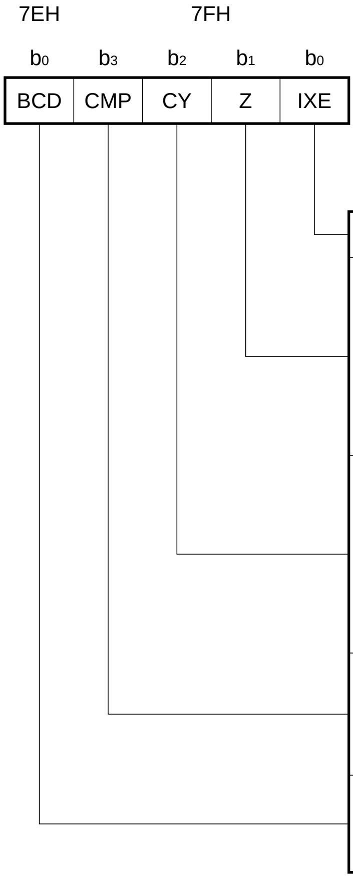
All the flags are cleared to 0 on reset.

When an interrupt is accepted, the contents of PSWORD are saved to the interrupt stack register. After the PSWORD contents have been saved, all the bits of PSWORD are cleared to “0”.

6.7.2 Program status word function

Each flag of the program status word sets the condition for an arithmetic operation or transfer instruction, or to indicate an operation result. Figure 6-19 shows the program status word functions.

Figure 6-19. Functions of Program Status Word



Index enable flag	Index modification is enabled when this flag is set.
Zero flag	Reset if result of arithmetic operation is other than "0". Set condition differs depending on contents of CMP flag. (1) When CMP = 0 Set if operation result is "0" (2) When CMP = 1 Set if Z = 1 and operation result is "0"
Carry flag	Set if carry occurs as result of executing addition instruction, or if borrow occurs as result of executing subtraction instruction. Remains reset if neither carry nor borrow occurs. Also set if least significant bit of general register is "1" when RORC instruction is executed, and reset if LSB is "0".
Compare flag	Result of arithmetic operation is not stored in data memory while this flag is set. CMP flag is reset automatically when SKT or SKF instruction is executed.
BCD flag	All arithmetic operations are performed in decimal (BCD) when this flag is set, and in binary when this flag is reset.

6.7.3 Index enable flag (IXE)

The IXE flag is used to modify an address of the data memory when a data memory manipulation instruction is executed.

When this flag is set to 1, the contents of the data memory address specified by the instruction are ORed with the contents of the index register (IX), and the instruction is executed to the data memory addressed by the result of the OR operation (real address).

For details, refer to **6.5 Index Register (IX)**.

6.7.4 Zero (Z) and compare (CMP) flags

The Z flag indicates that the result of an arithmetic operation executed is 0, and the CMP flag made setting so that the result of an arithmetic operation is not stored in the data memory or general register.

The conditions under which the Z flag is set or reset differ depending on the status of the CMP flag, as shown in Table 6-2.

Table 6-2. Status of Compare Flag (CMP) and Set and Reset Conditions of Zero Flag (Z)

Condition	Status of Z Flag	
	When CMP Is 0	When CMP Is 1
On reset	Reset	Reset with CMP
When “0” is directly written to Z flag by data memory manipulation instruction	Reset	Reset
When “1” is directly written to Z flag by data memory manipulation instruction	Set	Set
If result of arithmetic operation is other than“0”	Reset	Reset
If result of arithmetic operation is “0”	Set	Retains previous status of Z flag

The Z and CMP flags are used to compare the contents of a general register with those of the data memory. The status of the Z flag is not changed by an operation other than an arithmetic operation, and the status of the CMP flag is not changed by an operation other than bit testing.

6.7.5 Carry flag (CY)

The CY flag indicates occurrence of a carry or borrow after an addition or subtraction instruction is executed.

The CY flag is set to 1 if a carry or borrow occurs as a result of the arithmetic operation; it is reset to 0 if neither a carry nor a borrow occurs.

When the “RORC r” instruction, which shifts the contents of a general register specified by r 1 bit to the right, is executed, the value of the CY flag immediately before the instruction is executed is shifted to the most significant bit position of the general register, and the least significant bit is shifted to the CY flag.

The CY flag is convenient for skipping the next instruction if a carry or borrow occurs.

The status of this flag is not changed by an operation other than arithmetic operation or rotation processing.

6.7.6 Binary coded decimal flag (BCD)

The BCD flag is used to execute a BCD operation.

When this flag is set to 1, all arithmetic operations are executed in BCD format. When it is reset to 0, the operations are executed in binary and 4-bit units.

This flag does not affect the logical operation, bit judgment, comparison, and rotation processing.

6.7.7 Notes on executing arithmetic operation

When executing an arithmetic operation (addition or subtraction) to the program status word (PSWORD), note that the “result” of the arithmetic operation is stored in the PSWORD, as indicated by the following example:

Example

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

When the above instructions are executed, a carry occurs. Consequently, the CY flag, which is bit 2 of the PSW, would be set to 1. Actually, however, 0000B is stored to the PSW because the result of the operation is 0000B.

6.8 Notes on Using System Registers

6.8.1 Reserved words of system registers

★ Because the system registers are located on the data memory, all the data memory manipulation instructions can be used to manipulate the system registers. When using the 17K series assembler (RA17K), however, a data memory address must be defined as a symbol in advance because a data memory address cannot be directly written as the operand of an instruction.

★ Although the system registers are part of the data memory, they are defined as symbols as “reserved words” by the assembler (RA17K) because they have dedicated functions, unlike the ordinary data memory areas.

The reserved words of the system registers are assigned to addresses 74H through 7FH, and are defined by symbols (such as AR3, AR2, and PSW) shown in **Figure 6-2 Configuration of System Registers (μPD17204)**.

When these reserved words are used, it is not necessary to define a symbol, as shown in the following Example 2.

Examples 1.

```
MOV    34H, #0101B      ; If data memory address 34H or 76H is
MOV    76H, #1010B      ; written as operand, error occurs.
M037   MEM 0.37H        ; Data memory address of general-purpose
MOV    M037, #0101B     ; data memory must be defined as symbol by MEM directive
```

2.

```
MOV    AR1, #1010B      ; Symbol needs not to be defined if reserved word AR1 (address 6H)
                        ; is used.
                        ; Reserved word AR1 is defined in device file as “AR1 MEM 0.76H”
```

★ When the assembler (RA17K) is used, the following macro instructions are embedded in the assembler as flag type symbol manipulation instructions:

SETn	: Sets flag to “1”
CLRn	: Resets flag to “0”
SKTn	: Skips if all flags are “1”
SKFn	: Skips if all flags are “0”
NOTn	: Inverts flag
INITFLG	: Initializes flag

Therefore, by using these macro instructions, the data memory can be manipulated as flags as shown in **Example 3** below.

Since each bit (flag) of the program status word and memory pointer enable flag has its own function, a reserved word (MPE, BCD, CMP, CY, Z, or IXE) is defined for each bit.

By using this flag type reserved word, therefore, an embedded macro instruction can be used as is as shown in **Example 4**.

Examples 3.

```
F0003 FLG 0.00.3        ; Flag type symbol definition
      SET1 F0003        ; Embedded macro
```

Macro expansion

```
OR     .MF.F0003 SHR 4, #.DF.F0003 AND 0FH
                   ; Sets bit 3 at address 00H in BANK0
```

4.

```
SET1 BCD          ; Embedded macro
```

Macro expansion

```
OR     .MF.BCD SHR 4, #.DF.BCD AND 0FH
                   ; Sets BCD flag
                   ; BCD is defined by “BCD FLG 0.7EH.0”
```

```
CLR2 Z, CY        ; Flag of same address
```

Macro expansion

```
AND    .MF.Z SHR 4, #.DF. (NOT (Z OR CY) AND 0FH)
```

```
CLR2 Z, BCD       ; Flag of different addresses
```

Macro expansion

```
AND    .MF.Z SHR 4, #.DF. (NOT Z AND 0FH)
AND    .MF.BCD SHR 4, #.DF. (NOT BCD AND 0FH)
```

6.8.2 Handling system register fixed to “0”

Data of the system registers fixed to “0” (refer to **Figure 6-2. Configuration of System Registers (μPD17204)** calls for your attention when the device, emulator, or assembler operates, as described in (1), (2), and (3) below.

(1) When device operates

The data fixed to “0” is not affected even when a write instruction is executed to it. When this data is read, “0” is always read.

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

An error occurs if an instruction that writes “1” is executed to the data fixed to “0”.
Therefore, if the following instructions are executed, an error occurs on the in-circuit emulator:

```
Examples 1.  MOV     BANK, #0100B   ; Writes 1 to bit 3 fixed to 0

         2.  MOV     IXL, #1111B    ;
             MOV     IXM, #1111B    ;
             MOV     IXH, #0001B    ;
             ADD     IXL, #1        ;
             ADDC    IXM, #0        ;
             ADDC    IXH, #0        ;
```

However, an error does not occur even if the “INC AR” or “INC IX” instruction is executed when all the valid bits are “1” as shown in Example 2. This is because the “INC” instruction, which is executed when all the valid bits of the address register and index register are “1”, clears all the valid bits to “0”.
Even if “1” is written to the data fixed to “0” of the address register as shown in Examples 1 and 2 above, an error does not occur.

★ **(3) When using 17K series assembler (RA17K)**

An error is not output even if there is an instruction that writes “1” to data fixed to “0”. Therefore, when “MOV BANK, #0100B” instruction shown in Example 1 is used, the assembler does not cause an error, but an emulator error occurs when the instruction is executed on the in-circuit emulator.
★ The assembler (RA17K) does not causes an error because it cannot detect the data memory address subject to manipulation by an instruction while register indirect transfer is executed.
The assembler causes an error only on the following occasion:

When value greater than 1 is used as “n” of embedded macro instruction “BANKn”

This is because it is judged that the bank register of the system registers is to be explicitly manipulated when the “BANKn” instruction is used.

[MEMO]

CHAPTER 7 GENERAL REGISTERS (GR)

The general registers (GR) are registers located on the data memory and are used for direct operation or data transfer with the data memory.

7.1 General Register Configuration

Figure 7-1. shows the configuration of the general register of the μPD17201A as an example.

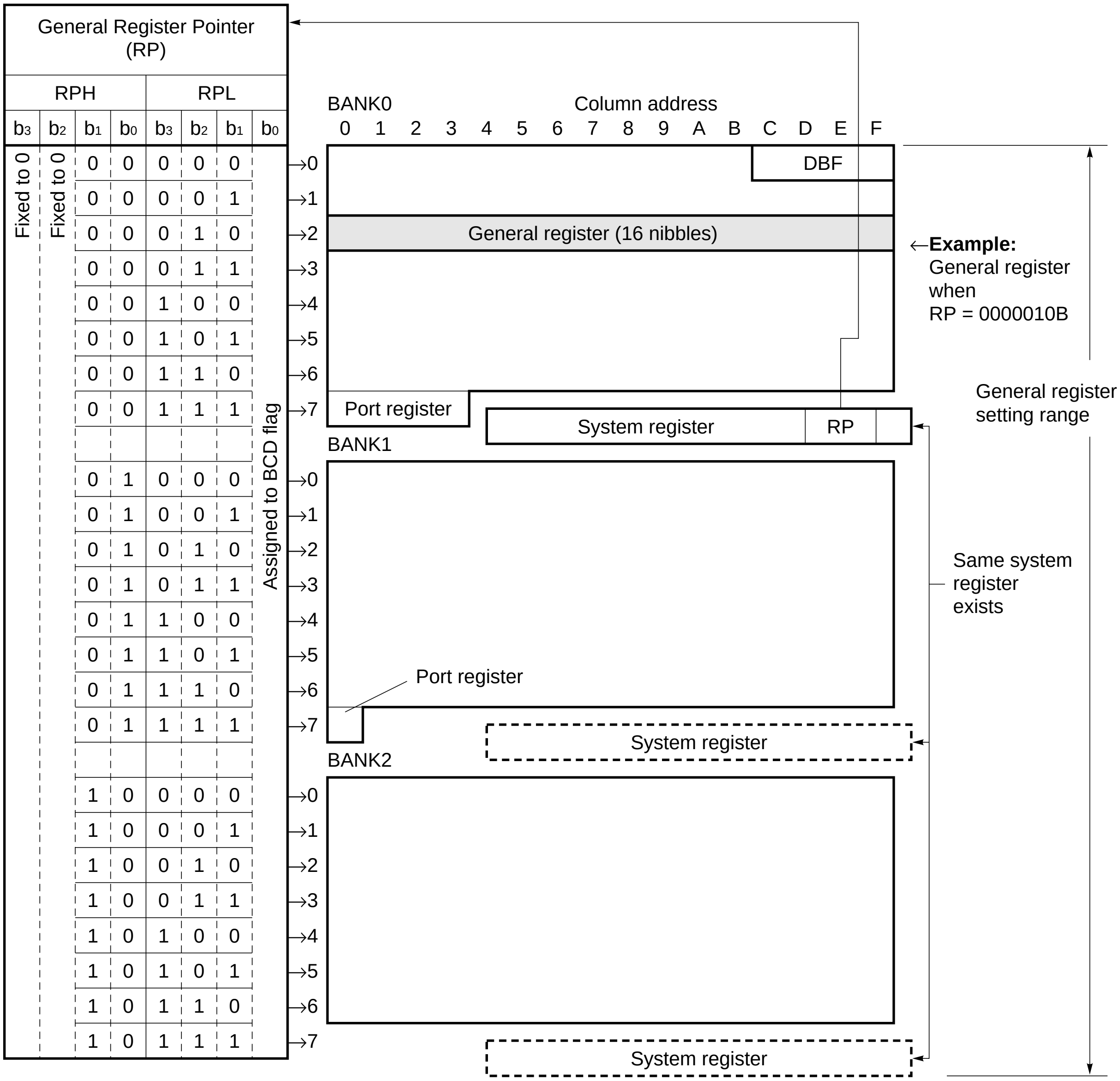
As shown in this figure, 16 nibbles (16 × 4 bits) at the same row address on the data memory can be used as a general register.

Which row address is to be used is specified by the general register pointer (RP) of the system registers. Since the RP has 5 valid bits, the range of the data memory in which general registers can be specified is row addresses 0H to 7H.

7.2 General Register Function

By using a general register, an operation or transfer between the data memory and the general register can be executed with a single instruction. In other words, an operation or transfer can be executed between two data memory areas with a single instruction because a general register is a part of the data memory. In addition, a general register can also be manipulated by a data memory manipulation instruction in the same manner as the other data memory areas because it is on the data memory.

Figure 7-1. Configuration of General Register (μPD17201A)



CHAPTER 8 REGISTER FILE (RF)

The register file is a group of registers that mainly set the conditions of the peripheral hardware.

8.1 Register File Configuration

8.1.1 Register file configuration

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

As shown in this figure, the register file consists of 128 nibbles (128 × 4 bits).

The register file is assigned addresses in 4-bit units, like the data memory, with row addresses 0H through 7H and column addresses 0H through 0FH.

Addresses 00H through 3FH are called control registers.

8.1.2 Register file and data memory

Figure 8-2 shows the relations between the register file and data memory.

As shown in this figure, addresses 40H through 7FH of the register file overlap the data memory.

Therefore, the same memory addresses 40H through 7FH of the bank selected at that time exist at addresses 40H through 7FH of the register file.

Figure 8-1. Configuration of Register File

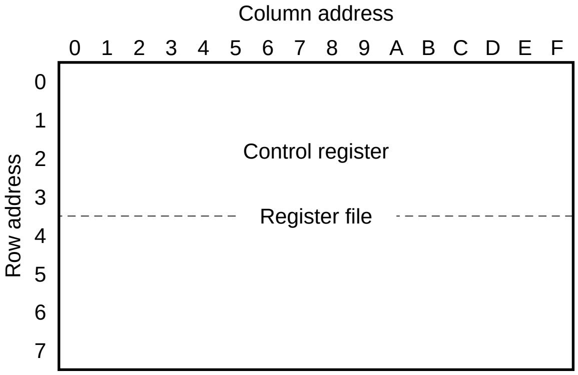
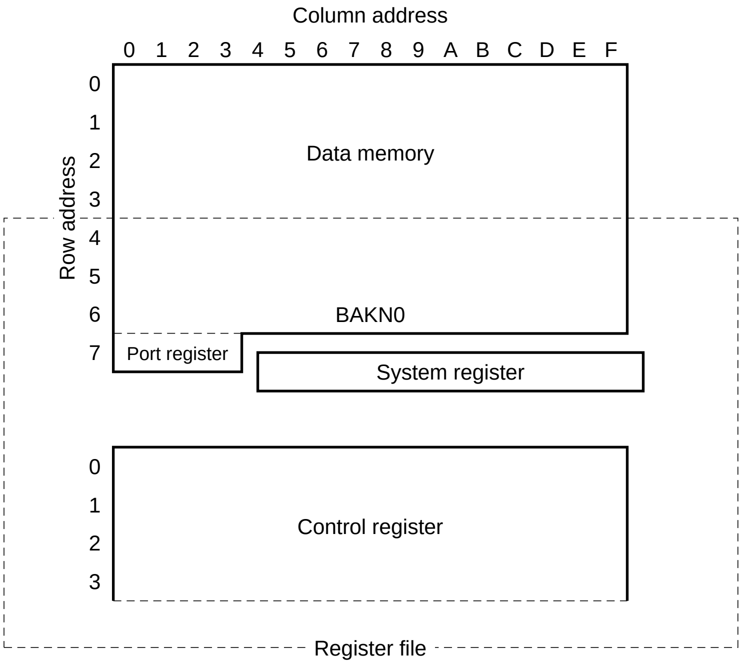


Figure 8-2. Relations between Register File and Data Memory



8.2 Register File Function

8.2.1 Register file function

The register file is a group of control registers that mainly set the conditions of the peripheral hardware.

These control registers are located at addresses 00H through 3FH of the register file.

The other addresses of the register file (40H through 7FH) overlap the data memory. Therefore, these addresses can be used in the same manner as the data memory except that they can be manipulated by register file manipulation instructions “PEEK” and “POKE” as described in **8.2.3**.

8.2.2 Control register function

The peripheral hardware whose conditions are set by the control registers is shown in Table 8-1.

For details on the peripheral hardware and control registers, refer to the description of each peripheral hardware.

★ **Table 8-1. Peripheral Hardware of μPD172×× Subseries**

Part Number Peripheral Hardware	17201A 17207	17203A 17204	17225 17226 17227 17228
Stack	○	○	○
Timer	○	○	○
Interrupt	○	○	○
Carrier generator	○	○	○
Remote controller reception amplifier		○	
General-purpose port	○	○	○
A/D converter	○		
Serial interface	○	○	
LCD driver	○		

Caution Some peripheral hardware transfers data via the data buffer (DBF) (refer to CHAPTER 9 DATA BUFFER (DBF)).

8.2.3 Register file manipulation instructions

Data is written to or read from the register file via the window register of the system registers (WR: address 78H). To write or read data, the following dedicated instructions are used:

PEEK WR, rf: Reads data of register file addressed by rf to WR
POKE rf, WR: Writes data of WR to register file addressed by rf

```
Example       M030   MEM 0.30H    ; Uses address 30H of data memory as WR saving area
              M032   MEM 0.32H    ; Uses address 32H of data memory as WR manipulation area
              RF11   MEM 0.91H    ; Symbol definition
              RF33   MEM 0.B3H    ; Symbols at addresses 00H-3FH of register file must be defined as
              RF70   MEM 0.70H    ; 80H-BFH of BANK0. For details, refer to 8.4 Notes on Using Register
              RF73   MEM 0.73H    ; File
              BANK0
          <1> PEEK   WR, RF11     ;
              CLR1   MPE          ; Indicates example for saving contents of WR to general-purpose data
              CLR1   IXE          ; memory (addresses 00H-3FH). As example, saving data to data memory
              OR     RPL, #0110B  ; address 30H without address modification is indicated.
          <2> LD     M030, WR     ;
          <3> POKE   RF73, WR     ; Data can be directly transferred between data memory at addresses
          <4> PEEK   WR, RF70     ; 40H-7FH and control register by WR, PEEK, and POKE instructions
          <5> POKE   RF33, WR     ;
          <6> ST     WR, M032     ;
```

Figure 8-3 shows an example of operation.

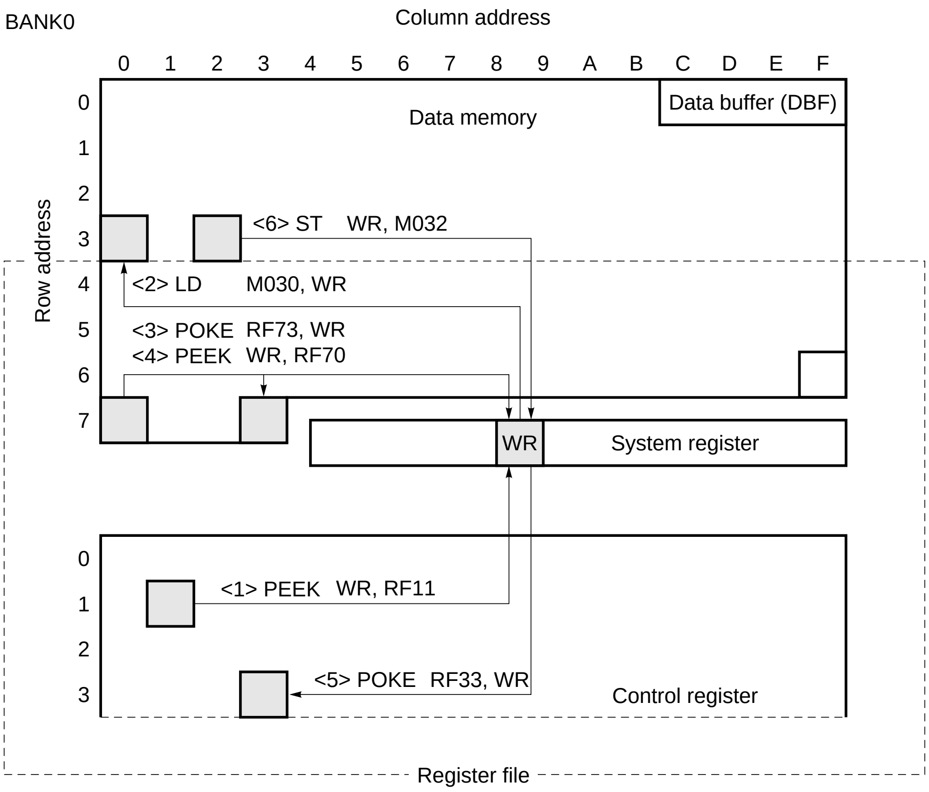
As shown in this figure, the control register (addresses 00H-3FH) reads or writes the contents of the register file addressed by “rf” from or to the window register when the “PEEK WR, rf” or “POKE rf, WR” instruction is executed.

Since addresses 40H through 7FH of the register file overlap the data memory, the “PEEK WR, rf” or “POKE rf, WR” instruction is executed to data memory address “rf” in the bank selected at that time.

Addresses 40H through 7FH of the register file can also be manipulated by a memory manipulation instruction.

The control register can be manipulated in 1-bit units by using a macro instruction (refer to **8.4.2 Symbol definition of register file and reserved word**).

Figure 8-3. Accessing Register File with PEEK or POKE Instruction



8.3 Control Registers

The control registers consist of 64 nibbles (64 × 4 bits) of addresses 00H through 3FH of the register file.

Each control register has an attribute of 1 nibble and serves as a read/write (R/W) or a read-only (R) register.

Some read/write flags, however, are always “0” when they are read.

Of the 4-bit data in 1 nibble, the bits fixed to “0” are always “0” when they are read, and retain “0” even when data is written to them.

When an unused register is read, its value is undefined. Nothing is changed even when data is written to this register.

8.4 Notes on Using Register File

8.4.1 Notes on manipulating control registers (read-only and unused registers)

When you manipulate the read-only (R) and unused registers of the control registers (addresses 00H through 3FH of the register file), you must pay attention when the device operates, as described in (1), (2), and (3) below when ★ you use the 17K Series assembler (RA17K) and the in-circuit emulators (IE-17K, IE-17K-ET).

(1) When device operates

Nothing is changed even when data is written to a read-only register.

If an unused register is read, an “undefined value” is read. Nothing is changed even when data is written to this register.

★ **(2) When using assembler (RA17K)**

An “error” occurs when an instruction that writes data is executed to access a read-only register.

An “error” also occurs when an instruction that reads or writes data is executed to an unused register.

(3) When using an 17K series in-circuit emulator (IE-17K or IE-17K-ET) (patch processing, etc.)

An “error” does not occur even when data is written to a read-only register.

When an unused register is read, an “undefined value” is read, and nothing is changed even when data is written to this register, but an “error” does not occur.

8.4.2 Symbol definition of register file and reserved words

★ If a register file address is directly written in numeric value as operand “rf” of the “PEEK WR, rf” or “POKE rf, WR” instruction when the 17K series assembler (RA17K) is used, an “error” occurs.

It is therefore necessary to define the address of the register file as a symbol as shown in Example 1 below.

Examples 1. Error occurs

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

Error does not occur

```
RF71   MEM0.71H      ; Symbol definition
PEEK   WR, RF71      ;
```

At this time, pay attention to the following point:

- To define a control register as a symbol of data memory address type, it must be defined as the addresses 80H through BFH of BANK0.

This is because the control register is manipulated via the window register, and an error must occur when the control register is manipulated by an instruction other than “PEEK” and “POKE”.

However, the register file (addresses 40H through 7FH) that overlap the data memory can be defined as a symbol without changing the address.

Here is an example:

Examples 2.

```
RF71     MEM1.71H        ; Register file overlapping data memory
RF02     MEM0.82H        ; Control register

PEEK     WR, RF71        ; RF71 is data memory at address “71H”
PEEK     WR, RF02        ; RF02 is control register at address 02H
```

★ When the assembler (RA17K) is used, the following macro instructions are included in the assembler as flag type symbol manipulation instructions:

SETn	: Sets flag to “1”
CLRn	: Clears flag to “0”
SKTn	: Skips if all flags are “1”
SKFn	: Skips if all flags are “0”
NOTn	: Inverts flag
INITFLG	: Initializes flag
★ INITFLGX	: Initializes flag

Therefore, by using these macro instructions, the contents of the register file can be manipulated in 1-bit units, as shown in the following Example 3.

Because many flags of the control registers are manipulated in 1-bit units, “reserved words” are defined on the ★ assembler (RA17K) as flag type symbols.

However, no flag type reserved word is available for the stack pointer. The reserved word for the stack pointer is defined as data memory type, “SP”. Therefore, the flag manipulation instruction cannot be used with a reserved word.

Examples 3.

```
INITFLG  WDTRES          ; Initialize
(SET1    WDTRES          ; Sets flag)
```

Macro expansion

```
PEEK     WR, .MF.WDTRES SHR4
OR       WR, #.DF.WDTRES AND 0FH
POKE     .MF.WDTRES SHR4, WR
```

CHAPTER 9 DATA BUFFER (DBF)

The data buffer consists of 4 nibbles assigned to address 0CH through 0FH of BANK0 of the data memory.

This area is a data storage area that is used to transfer data between the CPU and peripheral circuit (address register, serial interface, and timer) by using the GET and PUT instructions. Moreover, the constants data on the program memory can be read to the data buffer by the MOVT DBF, @AR instruction.

9.1 Data Buffer Configuration

Figure 9-1 shows the location of the data buffer on the data memory.

As shown, the data buffer is assigned addresses 0CH through 0FH of BANK0, and consists of a total of 16 bits (= 4 × 4 bits).

Figure 9-1. Location of Data Buffer

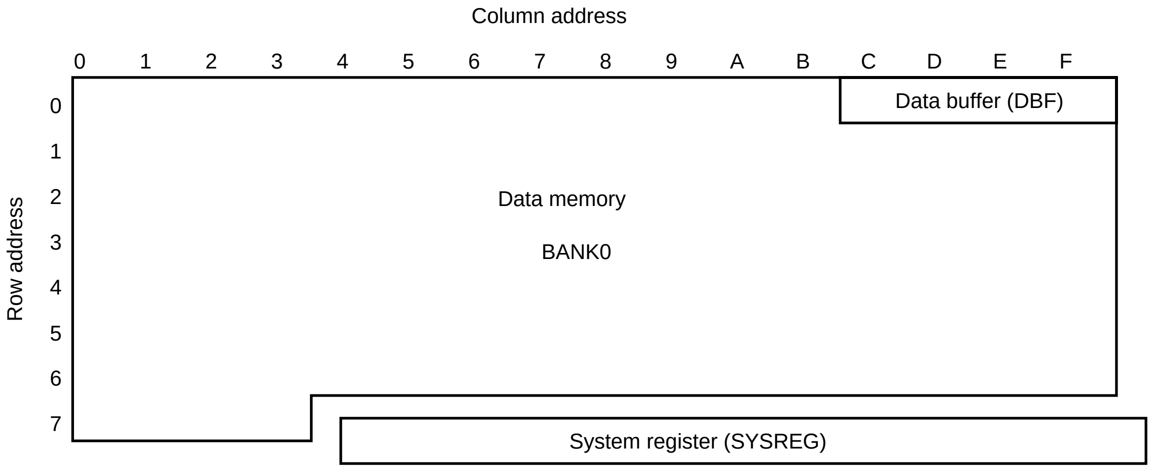


Figure 9-2 shows the configuration of the data buffer. As shown in this figure, the data buffer consists of 16 bits with the bit 0 at address 0FH of the data memory as the LSB and the bit 3 at address 0CH as the MSB.

Figure 9-2. Configuration of Data Buffer

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

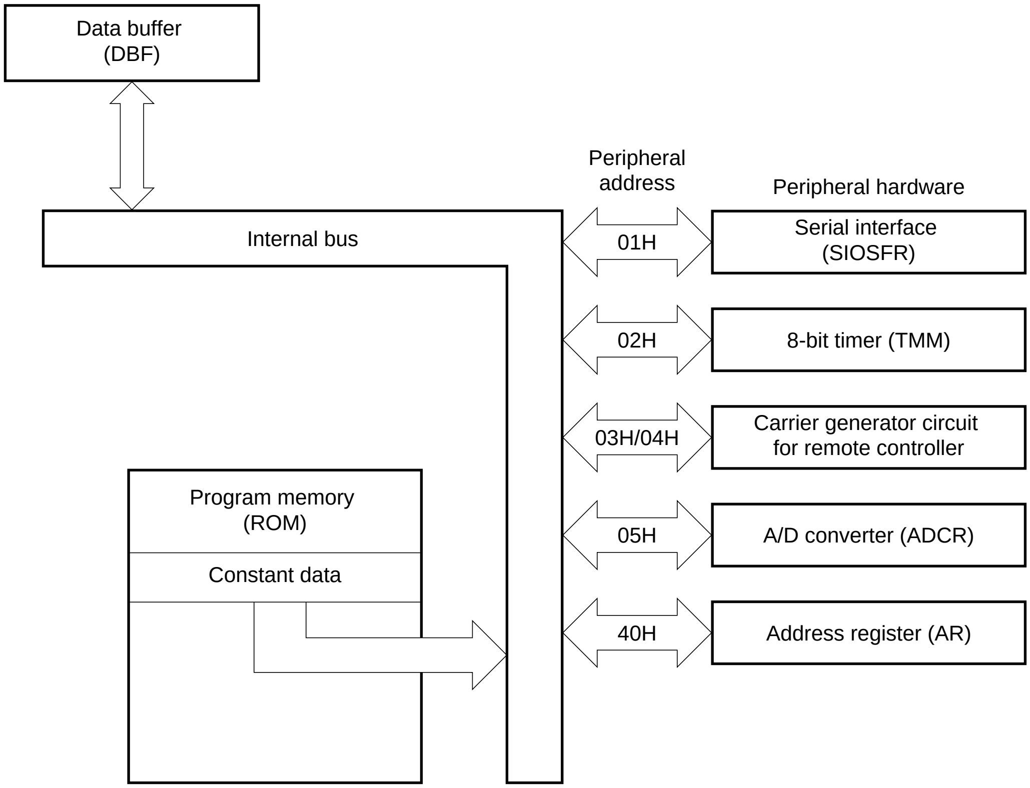
Because the data buffer is located on the data memory, it can be manipulated by all the data memory manipulation instructions.

9.2 Data Buffer Function

The data buffer has two major functions.

One is a function to transfer data with the peripheral hardware, and the other is a function to read the constants data on the program memory (table reference). Figure 9-3 shows the relations between the data buffer and the peripheral hardware of the μPD17201A as an example.

Figure 9-3. Data Buffer and Peripheral Hardware (μPD17201A)



9.2.1 Data buffer and peripheral hardware

Table 9-1 shows the peripheral hardware devices of the μPD17201A that transfer data via the data buffer.

Each of these peripheral hardware devices is assigned an address (called a peripheral address). By executing the dedicated instruction GET or PUT to these peripheral addresses, data can be transferred between the data buffer and peripheral hardware devices.

GET	DBF, p	: Reads data of peripheral hardware addressed by p to data buffer (DBF)
PUT	p, DBF	: Sets data of data buffer (DBF) to peripheral hardware addressed by p

Some peripheral hardware devices are write/read (PUT/GET), some are write-only (PUT), and the others are read-only (GET).

If the GET and PUT instructions are executed to the write-only and read-only devices, respectively, the operations are as follows:

- If GET is executed to write-only (PUT) peripheral hardware
  An undefined value is read.
- If PUT is executed to read-only (GET) peripheral hardware
  Nothing is affected (same as NOP).

Table 9-1. Peripheral Hardware (μPD17201A)

(1) Peripheral hardware that inputs/outputs in 8-bit units

Peripheral Address	Name	Peripheral Hardware	Data Direction		Valid Bit Length
			PUT	GET	
01H	SIOSFR	Serial interface	○	○	8 bits
02H	TMM	8-bit timer	○	○	8 bits
03H	NRZLTMM	NRZ modulo register (low level)	○	○	6 bits
04H	NRZHTMM	NRZ modulo register (high level)	○	○	6 bits
05H	ADCR	A/D converter	○	○	8 bits

(2) Peripheral hardware that inputs/outputs in 16-bit units

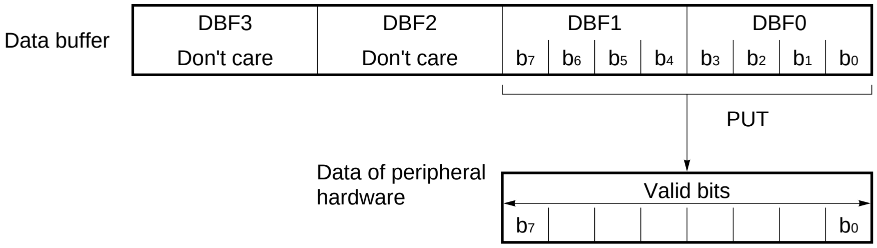
Peripheral Address	Name	Peripheral Hardware	Data Direction		Valid Bit Length
			PUT	GET	
40H	AR	Address register	○	○	12 bits

9.2.2 Data transfer with peripheral hardware

Data is transferred between the data buffer and peripheral hardware in 8-bit or 16-bit units.

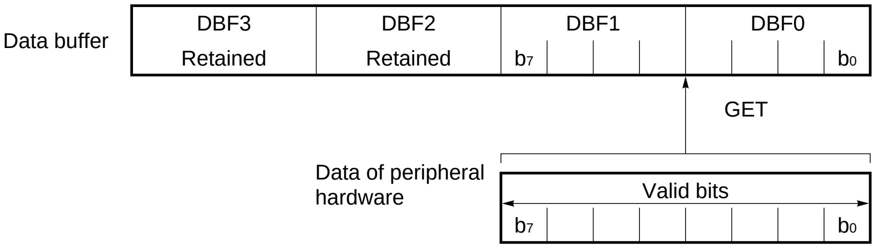
At this time, the PUT and GET instructions can be executed in one instruction execution time, regardless of whether the data length is 16 bits.

Examples 1. When executing PUT (when valid bit length of peripheral hardware is 8 bits (bits 0 through 7))


Data buffer
DBF3
Don't care
DBF2
Don't care
DBF1
DBF0
b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0
PUT
Data of peripheral hardware
Valid bits
b_7
b_0

To write 8-bit data, the high-order 8 bits of the data buffer (DBF3 and DBF2) are “don’t care” bits (that can take any value).

Examples 2. When executing GET



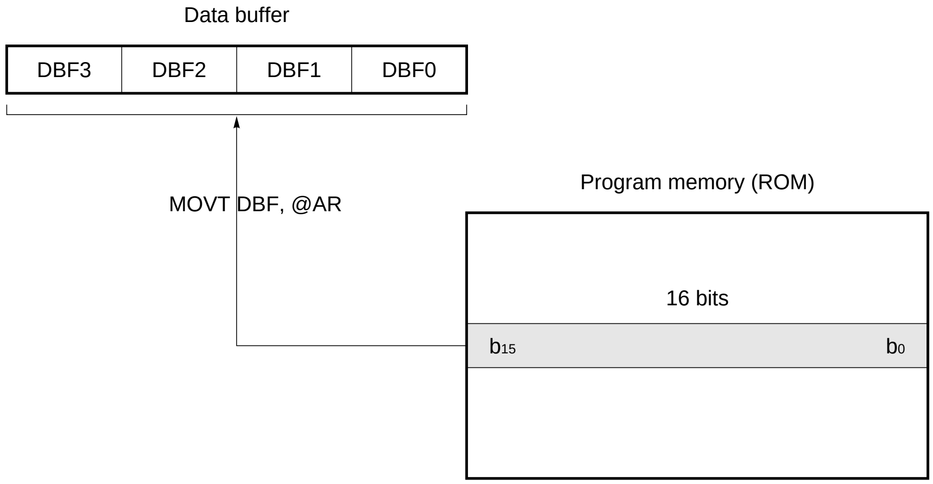
When 8-bit data is read, the high-order 8 bits of the data buffer (DBF3 and DBF2) are not affected.

9.2.3 Table reference

The constant data on the program memory can be read to the data buffer by using the MOVT instruction.

The function of the MOVT instruction is as follows:

MOVT DBF, @AR : Reads contents of program memory addressed by contents of address register (AR) to data buffer (DBF)



Caution Note that one stack level is temporarily used when table reference is executed.

[MEMO]

CHAPTER 10 ARITHMETIC LOGIC UNIT (ALU)

The ALU performs arithmetic operations, logical operations, bit testings, compare, and rotations of 4-bit data.

10.1 ALU Block Configuration

Figure 10-1 shows the configuration of the ALU block.

As shown, the ALU block consists of an ALU, which processes 4-bit data, temporary registers A and B, which are peripheral circuits of the ALU, status flip-flops controlling the status of the ALU, and a decimal correction circuit that is used when a BCD operation is performed.

The status flip-flops include a zero flag FF, carry flag FF, compare flag FF, and BCD flag FF, as shown in Figure 10-1.

The status flip-flops correspond to the zero (Z), carry (CY), compare (CMP), and BCD (BCD) flags of the program status word (PSWORD: addresses 7EH and 7FH) of the system registers on a one-to-one basis.

10.2 ALU Block Function

The ALU performs arithmetic operation, logical operation, bit testing, compare, or rotation processing, depending on the instructions written to the program. Table 10-1 lists the operation, testing, and rotation instructions.

By executing each of the instructions listed in this table, operation in 4-bit units, testing, rotation processing, or 1-digit decimal operation can be executed with a single instruction.

10.2.1 ALU function

Arithmetic operations include addition and subtraction. An arithmetic operation can be executed between the contents of a general register and those of the data memory, or between the contents of the data memory and immediate data. In addition, an arithmetic operation can be executed in binary number in 4-bit units, or in decimal number in 1-digit units (BCD operation).

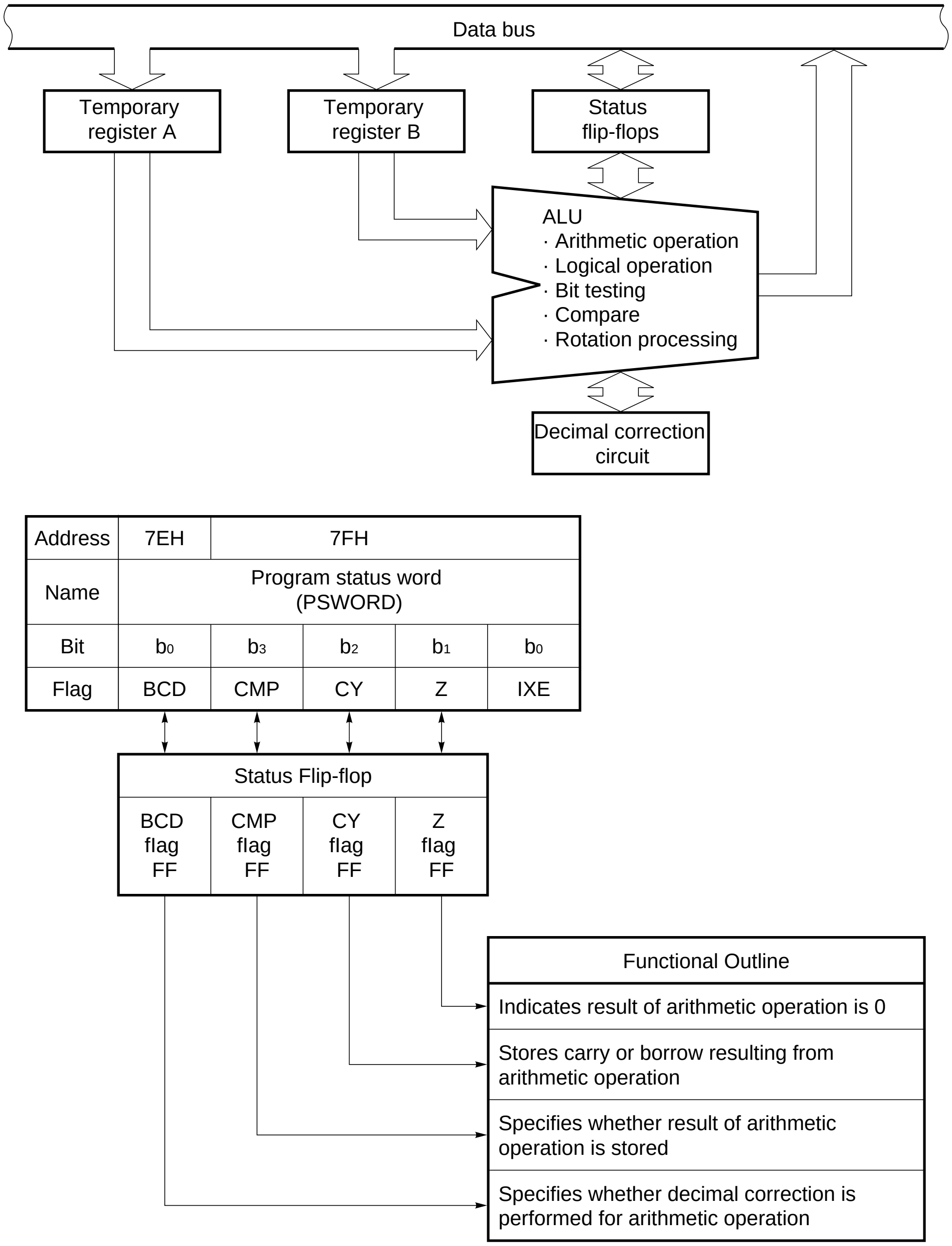
Logical operations include logical product (AND), logical sum (OR), and exclusive logical sum (XOR). A logical operation can be executed between the contents of a general register and those of the data memory, or between the contents of the data memory and immediate data.

Bit testing is to test whether one of the bits of the 4-bit data in the data memory is “0” or “1”.

Comparison is to compare the contents of the data memory with immediate data to judge whether one data is “equal to”, “not equal to”, “greater than”, or “less than” the other.

Rotation processing is to shift the 4-bit data of a general register 1 bit toward the least significant bit direction (rotation to the right).

Figure 10-1. Configuration of ALU Block



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

Status Flip-flop			
BCD flag FF	CMP flag FF	CY flag FF	Z flag FF

Functional Outline
Indicates result of arithmetic operation is 0
Stores carry or borrow resulting from arithmetic operation
Specifies whether result of arithmetic operation is stored
Specifies whether decimal correction is performed for arithmetic operation

[MEMO]

Table 10-1. ALU Processing Instructions (1/2)

ALU Function		Instruction	Operation	Remarks
Arith-metic	Addi-tion	ADD r, m	(r) ← (r) + (m)	Adds general register and data memory contents, and stores result to general register
		ADD m, #n4	(m) ← (m) + n4	Adds data memory and immediate data contents, and stores result to data memory
		ADDC r, m	(r) ← (r) + (m) + CY	Adds general register and data memory contents with CY flag, and stores result to general register
		ADDC m, #n4	(m) ← (m) + n4 + CY	Adds data memory and immediate data contents with CY flag, and stores result to data memory
	Sub-traction	SUB r, m	(r) ← (r) – (m)	Subtracts data memory contents from general register contents, and stores result to general register
		SUB m, #n4	(m) ← (m) – n4	Subtracts immediate data from data memory contents, and stores result to data memory
		SUBC r, m	(r) ← (r) – (m) – CY	Subtracts data memory contents from general register contents with CY flag, and stores result to general register
		SUBC m, #n4	(m) ← (m) – n4 – CY	Subtracts immediate data and CY flag from data memory contents, and stores result to data memory
Logical	OR	OR r, m	(r) ← (r) ∨ (m)	ORs general register and data memory contents, and stores result to general register
		OR m, #n4	(m) ← (m) ∨ n4	ORs data memory contents and immediate data, and stores result to data memory
	AND	AND r, m	(r) ← (r) ∧ (m)	ANDs general register and data memory contents, and stores result to general register
		AND m, #n4	(m) ← (m) ∧ n4	ANDs data memory contents and immediate data, and stores result to data memory
	XOR	XOR r, m	(r) ← (r) ⊻ (m)	XORs general register and data memory contents, and stores result to general register
		XOR m, #n4	(m) ← (m) ⊻ n4	XORs data memory contents and immediate data, and stores result to data memory
Bit testing	True	SKT m, #n	CMP ← 0, if (m) ∧ n = n, then skip	Skips if all bits of data memory contents specified by n are True (1). Result is not stored
	False	SKF m, #n	CMP ← 0, if (m) ∧ n = 0, then skip	Skips if all bits of data memory contents specified by n are False (0). Result is not stored
Compare	Equal to	SKE m, #n4	(m) – n4, skip if zero	Skips if data memory contents are equal to immediate data. Result is not stored
	Not equal to	SKNE m, #n4	(m) – n4, skip if not zero	Skips if data memory contents are not equal to immediate data. Result is not stored
	Greater than	SKGE m, #n4	(m) – n4, skip if not borrow	Skips if data memory contents are greater than immediate data. Result is not stored
	Less than	SKLT m, #n4	(m) – n4, skip if borrow	Skips if data memory contents are less than immediate data. Result is not stored
Rotation	Right rotation	RORC r	$\rightarrow \mathrm{CY} \rightarrow (r)_{b3} \rightarrow (r)_{b1} \rightarrow (r)_{b2} \rightarrow (r)_{b0}$	Rotates general register contents to right with CY flag, and stores result to general register

Table 10-1. ALU Processing Instructions (2/2)

ALU Function	Difference in Operation Because of Program Status Word (PSWORD)					
	Value of BCD Flag	Value of CMP Flag	Operation	CY Flag	Z Flag	Modification when IXE = 1
Arithmetic operation	0	0	Binary operation. Result is stored.	Set when carry or borrow occurs; otherwise, reset	Set if operation result is 0000B; otherwise, reset	Executed
	0	1	Binary operation. Result is not stored.		Retains status if operation result is 0000B; otherwise, reset	
	1	0	BCD operation. Result is stored.		Set if operation result is 0000B; otherwise, reset	
	1	1	BCD operation. Result is not stored.		Retains status if operation result is 0000B; otherwise, reset	
Logical operation	Don't care (retained)	Don't care (retained)	Not affected	Don't care (retained)	Don't care (retained)	Executed
Bit testing	Don't care (retained)	Reset	Not affected	Don't care (retained)	Don't care (retained)	Executed
Comparison	Don't care (retained)	Don't care (retained)	Not affected	Don't care (retained)	Don't care (retained)	Executed
Rotation	Don't care (retained)	Don't care (retained)	Not affected	Value of b_0 of general register	Don't care (retained)	Executed

10.2.2 Functions of temporary registers A and B

The temporary registers A and B are necessary for processing 4-bit data at one time, and temporarily store data to be processed and data processing.

10.2.3 Status flip-flop functions

The status flip-flops control the operations of the ALU and store the status of the processed data. Since these flip-flops correspond to the flags of the program status word (PSWORD) on a one-to-one basis, they can be manipulated by manipulating the system register. Each flag of the program status word has the following functions:

(1) Z flag

This flag is set to 1 if the result of an arithmetic operation is 0000B; otherwise, it is reset to 0.
However, the condition under which this flag is set to 1 differs depending on the status of the CMP flag, as follows:

(i) When CMP flag = 0

The Z flag is set to 1 if the result of an operation is 0000B; otherwise, it is reset to 0.

(ii) When CMP flag = 1

The Z flag retains the previous status if the result of an operation is 0000B; otherwise, it is reset to 0.
The flag is not affected by an operation other than arithmetic operations.

(2) CY flag

This flag is set to 1 if a carry or borrow occurs as a result of an arithmetic operation; otherwise, it is reset to 0.
If an arithmetic operation executed involves a carry or borrow, the content of the CY flag is reflected on the least significant bit of the execution result.
When rotation processing (RORC instruction) is executed, the content of the CY flag at that time is loaded to the most significant bit (b3) position of a general register, and the content of the least significant bit of the general register is loaded to the CY flag.
The CY flag is not affected by any operation other than arithmetic operation and rotation processing.

(3) CMP flag

The result of an arithmetic operation executed when the CMP flag is set to 1 is not stored in a general register or data memory.
If a bit judgment instruction is executed, the CMP flag is reset to 0.
This flag does not affect the compare and logical operations, and rotation processing.

(4) BCD flag

When the BCD flag is set to 1, the results of all the arithmetic operations executed are corrected to decimal.
When this flag is reset to 0, operation is performed in binary 4-bit.
The BCD flag does not affect the logical operation, bit testing, compare, and rotation processing.

The values of these flags can be changed by directly manipulating the program status word. At this time, the value of the corresponding status flip-flop is changed accordingly.

10.2.4 Binary 4-bit operation

An arithmetic operation is executed in binary and in 4-bit units when the BCD flag is 0.

10.2.5 BCD operation

When the BCD flag is 1, the arithmetic operation is performed in decimal format. The differences between the binary 4-bit operation and BCD operation are shown in Table 10-2. If the result of a decimal correction operation is more than 20, of if the result of a decimal subtraction is other than –10 to +9, data for more than 1010B (0AH) is stored in the data memory (shaded part in Table 10-2).

Table 10-2. Results for Binary 4-bit and BCD Operations

Result	Binary 4-bit Addition		BCD Addition	
	CY	Result	CY	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

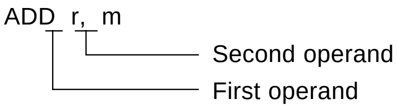
Result	Binary 4-bit Addition		BCD Addition	
	CY	Result	CY	Result
0	0	0000	0	0000
1	0	0001	0	0001
2	0	0010	0	0010
3	0	0011	0	0011
4	0	0100	0	0100
5	0	0101	0	0101
6	0	0110	0	0110
7	0	0111	0	0111
8	0	1000	0	1000
9	0	1001	0	1001
10	0	1010	1	1100
11	0	1011	1	1101
12	0	1100	1	1110
13	0	1101	1	1111
14	0	1110	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

10.2.6 ALU block processing sequence

When an arithmetic operation, logical operation, bit testing, compare, or rotation processing instruction is executed on the program, the data to be operated, tested, or processed and processing data are temporarily stored in temporary registers A and B.

The data to be processed is the contents of a general register or data memory addressed by the first operand of the instruction, and is 4-bit data. The processing data is the contents of the data memory addressed by the second or immediate data directly specified by the second operand, and is 4-bit data.

Take the following instruction for example:



The data to be processed is the contents of a general register addressed by r, and the processing data is the contents of the data memory addressed by m.

ADD m, #n4

The data to be processed by this instruction is the contents of the data memory addressed by m, and the processing data is immediate data specified by #n4.

RORC r

With the following rotation processing instruction, only the data to be processed is necessary because the processing method is determined, and the data to be processed is the contents of a general register addressed by r:

The data stored in temporary registers A and B are operated arithmetically or logically, tested, compared, or rotated according to the instruction executed. If an arithmetic operation, logical operation, or rotation processing instruction has been executed, the data processed by the ALU is stored in a general register or the data memory addressed by the first operand of the instruction, and the operation is finished. If a bit testing or compare instruction is executed, the next instruction on the program is skipped (i.e., executed as an NOP instruction) depending on the result of the processing performed by the ALU, and the operation is finished.

Bear in mind the following points when using the ALU block:

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

(2) Logical operations are not affected by the CMP and BCD flags of the program status word, and do not affect the Z and CY flags.

(3) The bit testing instruction resets the CMP flag of the program status word.

(4) Arithmetic and logical operations, bit testing, compare, and rotation processing are modified by the index register if the IXE flag of the program status word is set to 1.

10.3 Arithmetic Operation (Binary 4-bit addition/subtraction and BCD addition/subtraction)

As shown in Table 10-3, the arithmetic operations are broadly classified into four types: addition, subtraction, addition with carry, and subtraction with borrow. These operations are performed by “ADD”, “ADDC”, “SUB”, and “SUBC” instructions, respectively.

These instructions are also classified into addition or subtraction between a general register and data memory, and that between the data memory and immediate data. Whether the operation is executed between a general register and data memory, or between the data memory and immediate data is determined by the value written as the operand of the instruction. If the operand is “r, m”, addition or subtraction is executed between a general register and the data memory; if the operand is “m, #n4”, the operation is between the data memory and immediate data.

The arithmetic operation instruction is affected by the status flip-flops, that is, the program status word (PSWORD) of the system registers. The BCD flag of the program status word specifies whether the operation is executed in binary and 4-bit units or in BCD, and the CMP flag specifies that the result of the operation is not stored anywhere.

10.3.1 through **10.3.4** describe the relations between each arithmetic operation instruction and the program status word.

Table 10-3. Arithmetic Operation Instructions

Arithmetic operation	Add	Without carry ADD	General register and data memory	ADD r, m
			Data memory and immediate data	ADD m, #n4
		Add w/carry ADDC	General register and data memory	ADDC r, m
			Data memory and immediate data	ADDC m, #n4
	Subtract	Without borrow SUB	General register and data memory	SUB r, m
			Data memory and immediate data	SUB m, #n4
		Subtract w/borrow SUBC	General register and data memory	SUBC r, m
			Data memory and immediate data	SUBC m, #n4

10.3.1 Addition/subtraction when CMP = 0, BCD = 0

Addition or subtraction is executed in binary and 4-bit units, and the result is stored in a specified general register or data memory address.

The CY flag is set to 1 if the result of the operation exceeds 1111B (if a carry occurs) or is less than 0000B (a borrow occurs); otherwise, it is reset to 0.

If the result of the operation is 0000B, the Z flag is set to 1, regardless of whether a carry or borrow occurs; if the result is other than 0000B, the Z flag is reset to 0.

10.3.2 Addition/subtraction when CMP = 1, BCD = 0

Addition or subtraction is executed in binary and 4-bit units.

However, the result of the operation is not stored in a general register or data memory address because the CMP flag is set to 1.

If a carry or borrow occurs as a result of the operation, the CY flag is set to 1; otherwise, the flag is reset to 0.

The Z flag retains the previous status if the result of the operation is 0000B; otherwise, it is reset to 0.

10.3.3 Addition/subtraction when CMP = 0, BCD = 1

A BCD operation is executed.

The result of the operation is stored in a specified general register or data memory address. The CY flag is set to 1 if the result exceeds 1001B (9D) or is less than 0000B (0D), and is reset to 0 if the result is in the range of 0000B (0D) to 1001B (9D).

The Z flag is set to 1 if the result is 0000B (0D); otherwise, it is reset to 0.

The BCD operation is executed by converting the result of an operation executed in binary into decimal number by using the decimal correction circuit. For details on this binary-to-decimal conversion, refer to **Table 10-2 Results for Binary 4-bit and BCD Operations**.

To execute a BCD operation correctly, therefore, keep in mind the following points:

(1) The result of addition must be 0D to 19D.

(2) The result of subtraction must be 0D to 9D or –10 to –1D.

The value range of 0D to 19D is determined by giving consideration to the CY flag, and is in binary:

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

Likewise, the range of –10D to –1D is:

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

If a BCD operation is executed without the above conditions (1) and (2) satisfied, the CY flag is set to 1, and data greater than 1010B (0AH) is output as a result.

10.3.4 Addition/subtraction when CMP = 1, BCD = 1

A BCD operation is performed.

The result of the operation is not stored in a general register or data memory address.

Therefore, the operation to be performed when the CMP flag is 1 and that performed when the BCD flag is 1 are performed at the same time.

Example

```
MOV     RPL,    #0001B ; Sets BCD flag to (1)
MOV     PSW,    #1010B ; Sets CMP and Z flags to 1 and resets CY flag to (0)
SUB     M1,     #0001B ; ⟨1⟩
SUBC    M2,     #0010B ; ⟨2⟩
SUBC    M3,     #0011B ; ⟨3⟩
```

At this time, the contents of the 12 bits of M3, M2, and M1 can be compared with immediate data 321 in decimal number.

10.3.5 Notes on using arithmetic operation instruction

When an arithmetic operation is executed to the program status word (PSWORD), note that the result of the operation is stored in the program status word.

The CY and Z flags of the program status word are usually set or reset as a result of an arithmetic operation. If an arithmetic operation is executed to the program status word, however, the result is stored to the program status word, making it impossible to test occurrence of a carry or borrow, or whether the result is zero.

When the CMP flag is set to 1, however, the result is not stored in the program status word, and the CY and Z flags are set or reset as usual.

10.4 Logical Operation

As logical operations, logical sum (OR), logical product (AND), and exclusive logical OR (XOR) can be executed as shown in Table 10-4.

The logical operations are classified into these three types and are implemented by the “OR”, “AND”, and “XOR” instructions.

These instructions are also classified into an operation executed between a general register and data memory, and that between the data memory and immediate data. Whether the operation is executed between a general register and data memory, or between the data memory and immediate data is determined depending on the value written as the operand of the instruction, i.e., whether “r, m” or “m, #n4” is described as the operand, like the arithmetic operation instruction.

The logical operation is not affected by the BCD and CMP flags of the program status word (PSWORD). It does not affect the CY and Z flags. However, the operation is subject to modification by the index register if the index enable flag (IXE) is set to 1.

Table 10-4. Logical Operation Instructions

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

Table 10-5. Logical Operation Truth Table

Logical product C = A AND B			Logical sum C = A OR B			Exclusive logical sum 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

10.5 Bit Testing

As shown in Table 10-6, bit testing can be classified into True bit (1) testing and False bit (0) testing.

These judgments are made respectively by the “SKT” and “SKF” instructions.

These instructions can be executed to only the data memory.

Bit testing is not affected by the BCD flag of the program status word (PSWORD). It does not affect the CY and Z flags. However, the CMP flag is reset to 0 when the “SKT” or “SKF” instruction is executed. Modification is made by the index register if the instruction is executed while the index enable flag (IXE) is set to 1. For details on modification by the index register, refer to **CHAPTER 6 SYSTEM REGISTER (SYSREG)**.

10.5.1 and **10.5.2** describe True bit (1) testing and False bit (0) testing, respectively.

Table 10-6. Bit Testing Instructions

Bit testing	True bit (1) testing SKT m, #n
	False bit (0) testing SKF m, #n

10.5.1 True bit (1) testing

The True bit (1) testing instruction, “SKT m, #n”, tests whether bit(s) specified by n of the 4 bits of a data memory address is “True (1)”. If all the bits specified by n is “True (1)”, the next instruction is skipped.

Example

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

In <1>, execution branches to B because all the bits 3, 1, and 0 of M1 are True (1).

In <2>, the bits 3, 2, and 0 of M1 are tested, and execution branches to C because bit 2 is False (0).

10.5.2 False bit (0) testing

The False bit (0) testing instruction, “SKF m, #n”, tests whether bit(s) specified by n of the 4 bits of a data memory address is “False (0)”. If all the bits specified by n is “False (0)”, the next instruction is skipped.

Example

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

In <1>, execution branches to B because both the bits 2 and 1 of M1 are False (0).

In <2>, the bits 3, 2, and 0 of M1 are tested, and execution branches to C because bit 3 of M1 is True (1).

10.6 Compare

As shown in Table 10-7, the compare operations are divided into four types: “equal to”, “not equal to”, “greater than”, and “less than”.

To make these comparisons, the “SKE”, “SKNE”, “SKGE”, and “SKLT” instructions are used.

These instructions can be used only to compare the contents of a data memory address with immediate data. To compare the contents of a general register and those of a data memory address, use a subtraction instruction with the CMP and Z flags of the program status word (PSWORD) (refer to **10.3 Arithmetic Operation (Binary 4-bit addition/subtraction and BCD addition/subtraction)**).

Comparison is not affected by the BCD and CMP flags of the program status word. It does not affect the CY and Z flags.

10.6.1 through **10.6.4** describe comparison of “equal to”, “not equal to”, “greater than”, and “less than”, respectively.

Table 10-7. Compare Instructions

Compare	Equal to SKE m, #n4
	Not equal to SKNE m, #n4
	Greater than SKGE m, #n4
	Less than SKLT m, #n4

10.6.1 Comparison of “Equal to”

The “SKE m, #n4” instruction tests whether the contents of a specified data memory address are “equal to” specified immediate data.

If the data memory contents are “equal to” the immediate data, the instruction next to this instruction is skipped.

Example

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

In <1>, execution branches to B because the contents of M1 are equal to immediate data 1010B. In <2>, however, execution branches to C because the contents of M1 are not equal to immediate data 1000B.

10.6.2 Comparison of “Not equal to”

The “SKNE m, #n4” instruction tests whether the contents of a specified data memory address are “not equal to” specified immediate data.

If the data memory contents are “not equal to” the immediate data, the instruction next to this instruction is skipped.

Example

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

In <1>, execution branches to B because the contents of M1 are not equal to immediate data 1000B. In <2>, however, execution branches to C because the contents of M1 are equal to immediate data 1010B.

10.6.3 Comparison of “Greater than”

The “SKGE m, #n4” instruction tests whether the contents of a specified data memory address are “greater than” specified immediate data.

If the data memory contents are “greater than” or “equal to” the immediate data, the instruction next to this instruction is 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
```

Because the contents of M1 are 1000B, <1> is judged to be “Greater than”, <2>, “Equal to”, and <3>, “Less than”, and execution branches to B, D, and E, respectively.

10.6.4 Comparison of “Less than”

The “SKLT m, #n4” instruction tests whether the contents of a specified data memory are “less than” specified immediate data.

If the data memory contents are “less than” the immediate data, the instruction next to this instruction is 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
```

Because the contents of M1 are 1000B, <1> is judged to be “Less than”, <2>, “Equal to”, and <3>, “Greater than”, and execution branches to B, C, and E, respectively.

10.7 Rotation Processing

Rotation processing can be classified into right rotation and left rotation.

To execute the right rotation processing, the “RORC” instruction is used.

This instruction can be executed only to a general register.

The rotation processing by the “RORC” instruction is not affected by the BCD and CMP flags of the program status word (PSWORD). It does not affect the Z flag.

10.7.1 and **10.7.2** below describe the respective rotation processing.

10.7.1 Right rotation processing

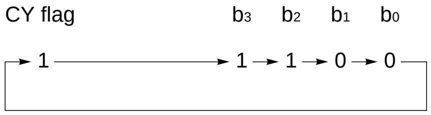
The right rotation processing instruction “RORC r” rotates the contents of a specified general register 1 bit toward the least significant bit direction.

At this time, the content of the CY flag is written to the most significant bit (bit 3) position of the general register, and the content of the least significant bit (bit 0) is written to the CY flag.

Examples 1.

```
MOV     PSW,    #0100B ; Sets CY flag to 1
MOV     R1,     #1001B
RORC    R1
```

At this time, the processing is performed as follows:



Therefore, right rotation is executed from the CY flag as shown above.

Examples 2.

```
MOV     PSW,    #0000B ; Resets CY flag to 0
MOV     R1,     #1000B ; MSB
MOV     R2,     #0100B
MOV     R3,     #0010B ; LSB
RORC    R1
RORC    R2
RORC    R3
```

The above program rotates the 13-bit data of R1, R2, and R3 to the right.

10.7.2 Left rotation processing

The left rotation processing can be performed by using the addition instruction “ADDC r, m” as follows:

Example

```
MOV     PSW,    #0000B  ; Resets CY flag to 0
MOV     R1,     #1000B  ; MSB
MOV     R2,     #0100B
MOV     R3,     #0010B  ; LSB
ADDC    R3, R3
ADDC    R2, R2
ADDC    R1, R1
SKF     CY
OR      R3,     #0001B
```

The above program rotates the 13-bit data of R1, R2, and R3 to the left.

CHAPTER 11 INTERRUPT FUNCTION

The interrupt control circuit for the μPD172×× subseries has the following features. It can perform high-speed interrupt processing:

(1) Accepting each interrupt can be controlled by the interrupt enable flag (INTEF) and interrupt enable flag (IP×××).
(2) Any interrupt processing start address can be set by a interrupt vector table.
(3) Nesting is enabled.
(4) The interrupt request flag (IRQ×××) can be read/written.
(5) The standby mode (STOP or HALT) can be released by an interrupt request (the releasing condition can also be selected by the interrupt flag).

Caution Only the BCD, CMP, CY, Z, and IXE flags, and BANK are automatically saved to the stack, up to three levels, by the hardware durig interrupt processing. When a peripheral hardware unit (such as the timer or serial interface) is accessed during interrupt processing, the contents of DBF and WR are not saved by the hardware. It is therefore recommended that DBF and WR be saved to RAM at the beginning of interrupt processing, and that they be restored immediately before the end of the interrupt processing.

11.1 Interrupt Control Circuit Configuration

11.1.1 Interrupt control (EI, DI)

To perform interrupt processing in response to an interrupt request, the interrupt must be enabled in advance by the EI instruction (INTE flag set).

If the interrupt is disabled by executing the DI instruction (INTE flag clear), all the interrupts are kept pending.

11.1.2 Interrupt enable flag (IP×××)

The interrupt enable flag (IP×××) corresponds to an individual interrupt request flag on a one-to-one basis. When the interrupt enable flag is set, the corresponding interrupt is enabled. When the interrupt enable flag is reset, the interrupt is kept pending.

The pending interrupt is set, when the interrupt request has been read. By resetting the set interrupt request, the interrupt can be released.

11.1.3 Interrupt request flag (IRQ×××)

The interrupt request flag (IRQ×××) is set, when an interrupt request has been generated. It is automatically reset, when the interrupt processing is executed.

If the interrupt request flag is reset by an instruction, a vectored interrupt is executed (software interrupt) in the same manner as when an interrupt has been generated, even if the interrupt has not been generated.

11.2 Interrupt Sequence

11.2.1 Accepting interrupt

Figure 11-1 shows the timing chart illustrating how an interrupt is accepted.

Figure 11-1 (1) is the timing chart for one type of interrupt.

An interrupt is accepted when all the interrupt request flags (IRQ×××), INTE flag, and interrupt enable flags (IP×××) are set.

(a) in (1) in the figure is the timing chart if the interrupt request flag (IRQ×××) is set to 1 last, and (b) is the timing chart if the interrupt enable flag (IP×××) is set to 1 last.

If the last flag is set in the first instruction cycle of the MOVT DBF, @AR instruction or by an instruction that satisfies the skip condition, the interrupt is accepted in the second cycle of the MOVT DBF, @AR instruction or after the instruction (NOP) that is skipped has been executed.

The INTE flag is set in the instruction cycle next to the one in which the EI instruction has been executed.

(2) in Figure 11-1 is the timing chart by two or more interrupts.

When two or more interrupts are used, the interrupt assigned the highest priority by hardware is accepted first if all the interrupt enable flags (IP×××) are set, but the hardware priority can be changed by manipulating the interrupt enable flags by program.

“Interrupt cycle” in Figure 11-1 is a special cycle in which the interrupt request flag is reset to (0), a vector address is specified, and the contents of the program counter are saved after the interrupt has been accepted. The interrupt cycle lasts for the execution time of one instruction.

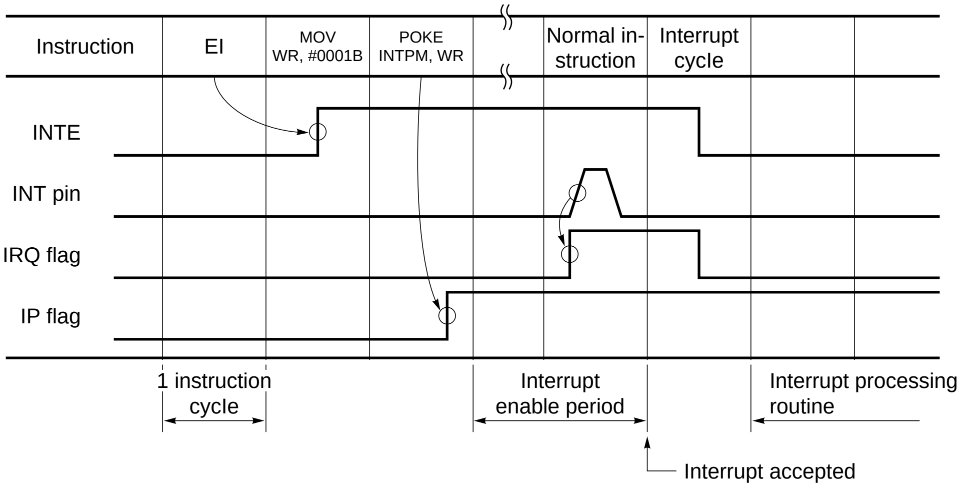
When interrupt processing is started, one level of the address stack register, which is used to store the return address of the program, is consumed, and one level of the interrupt stack register, which saves the PSWORD and BANK of the system registers, is also consumed.

Figure 11-1. Accepting Interrupt (1/3)

(1) When one type of interrupt (e.g., rising edge of INT pin) is used

(a) When interrupt request flag (IRQ×××) is set last

<1> When normal instruction is executed on accepting interrupt



<2> When MOVT instruction of instruction satisfying skip condition is executed on accepting interrupt

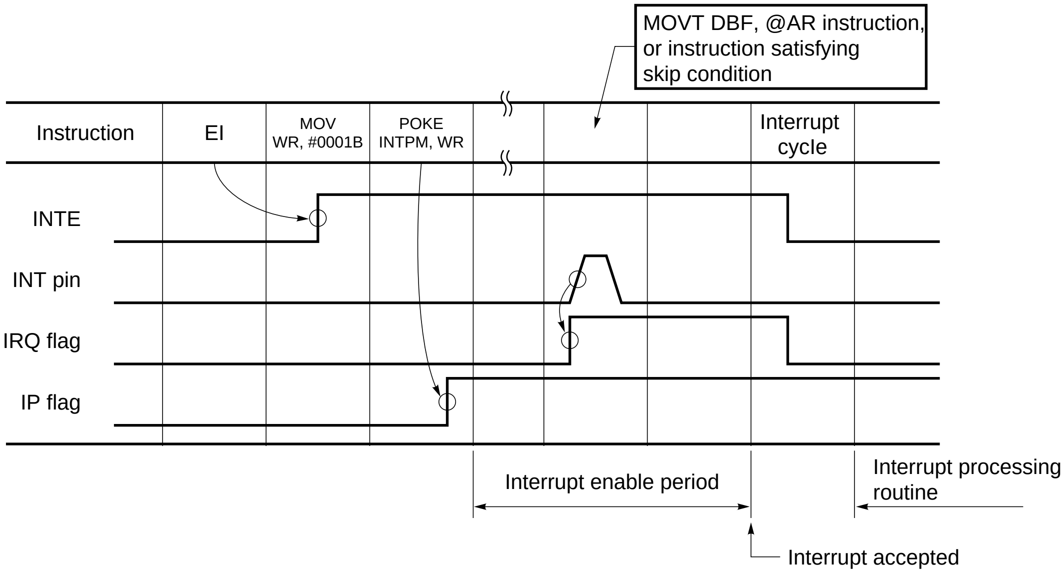


Figure 11-1. Accepting Interrupt (2/3)

(b) When interrupt enable flag (IP×××) is set last

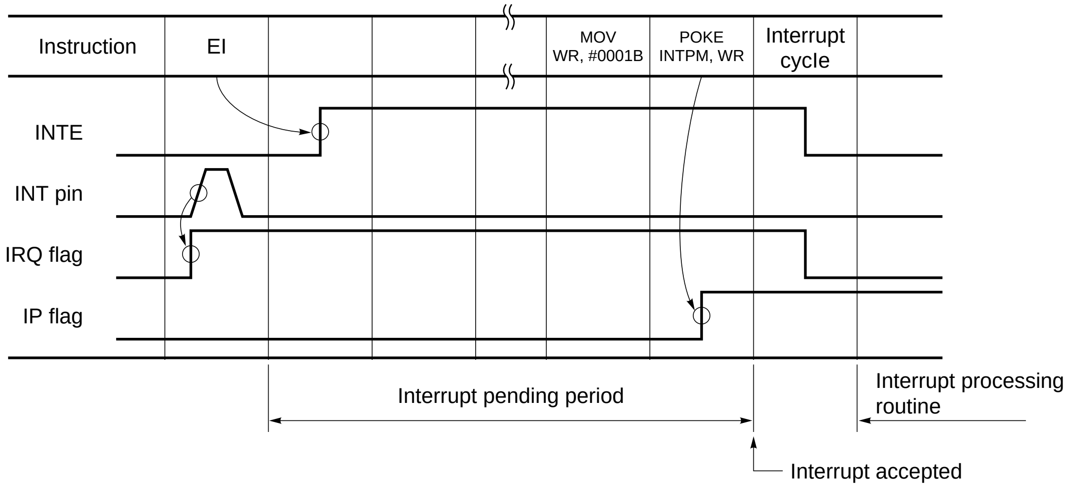
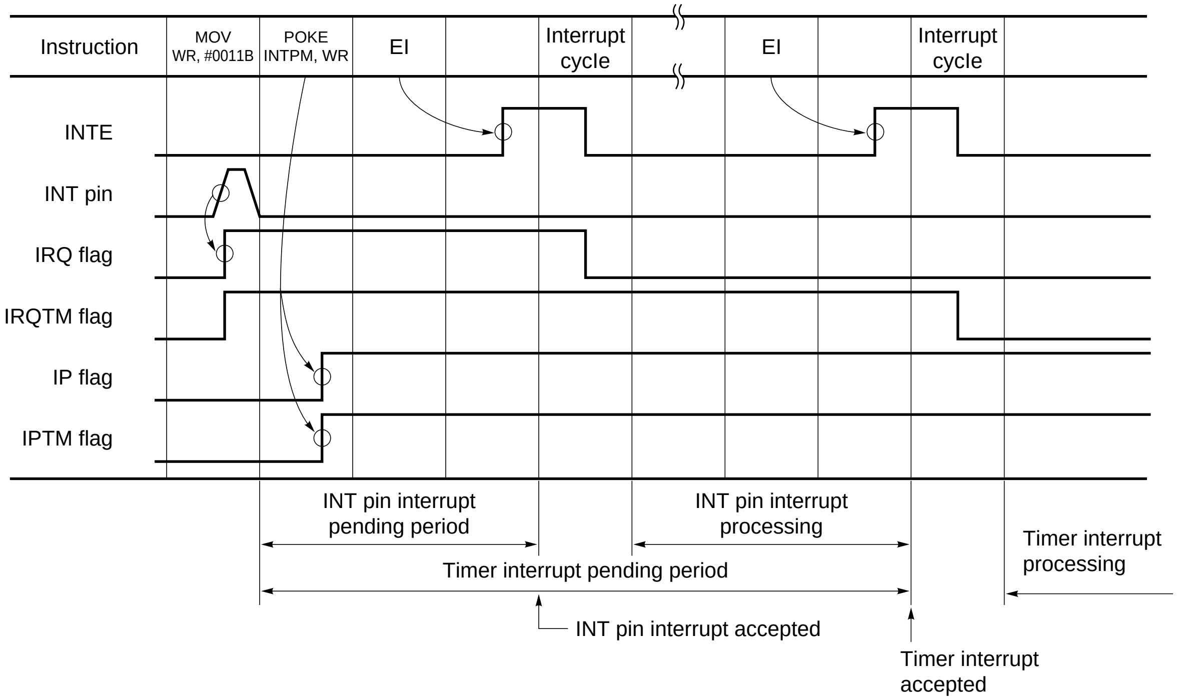


Figure 11-1. Accepting Interrupt (3/3)

(2) When two or more interrupts (e.g., INT pin and timer) are used

(a) Hardware priority


Instruction
MOV WR, #0011B
POKE INTPM, WR
EI
Interrupt cycle
EI
Interrupt cycle
INTE
INT pin
IRQ flag
IRQTM flag
IP flag
IPTM flag
INT pin interrupt pending period
INT pin interrupt processing
Timer interrupt processing
Timer interrupt pending period
INT pin interrupt accepted
Timer interrupt accepted

(b) Software priority

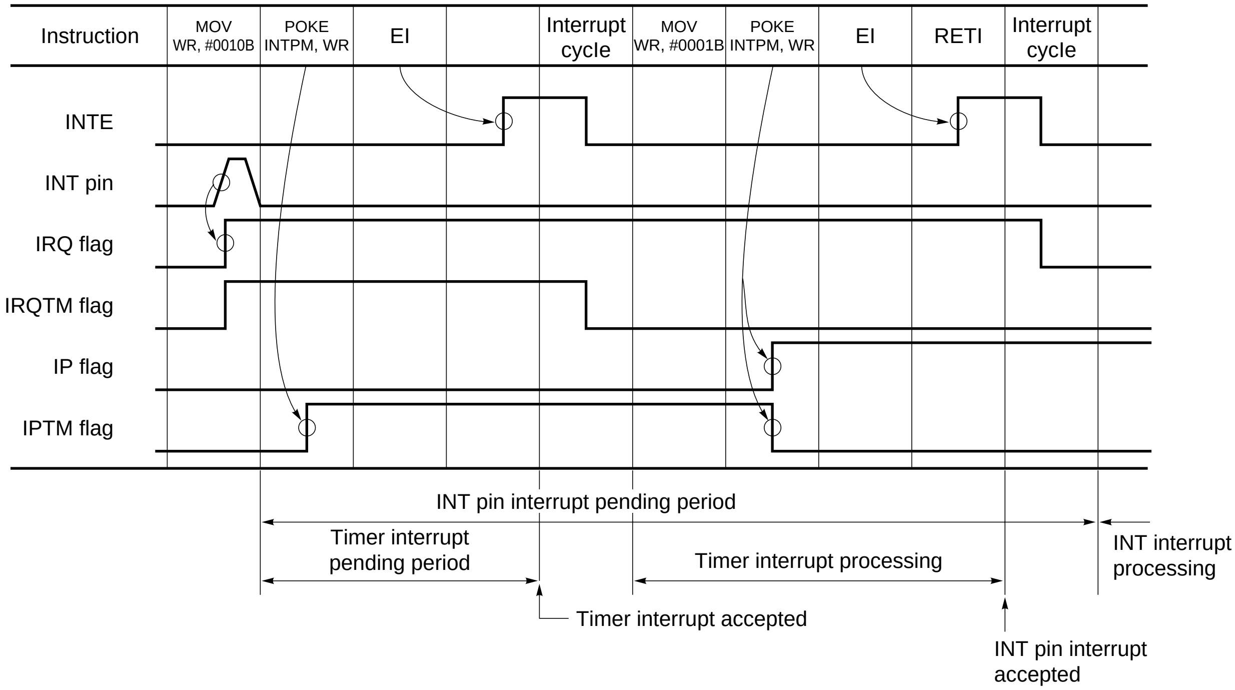
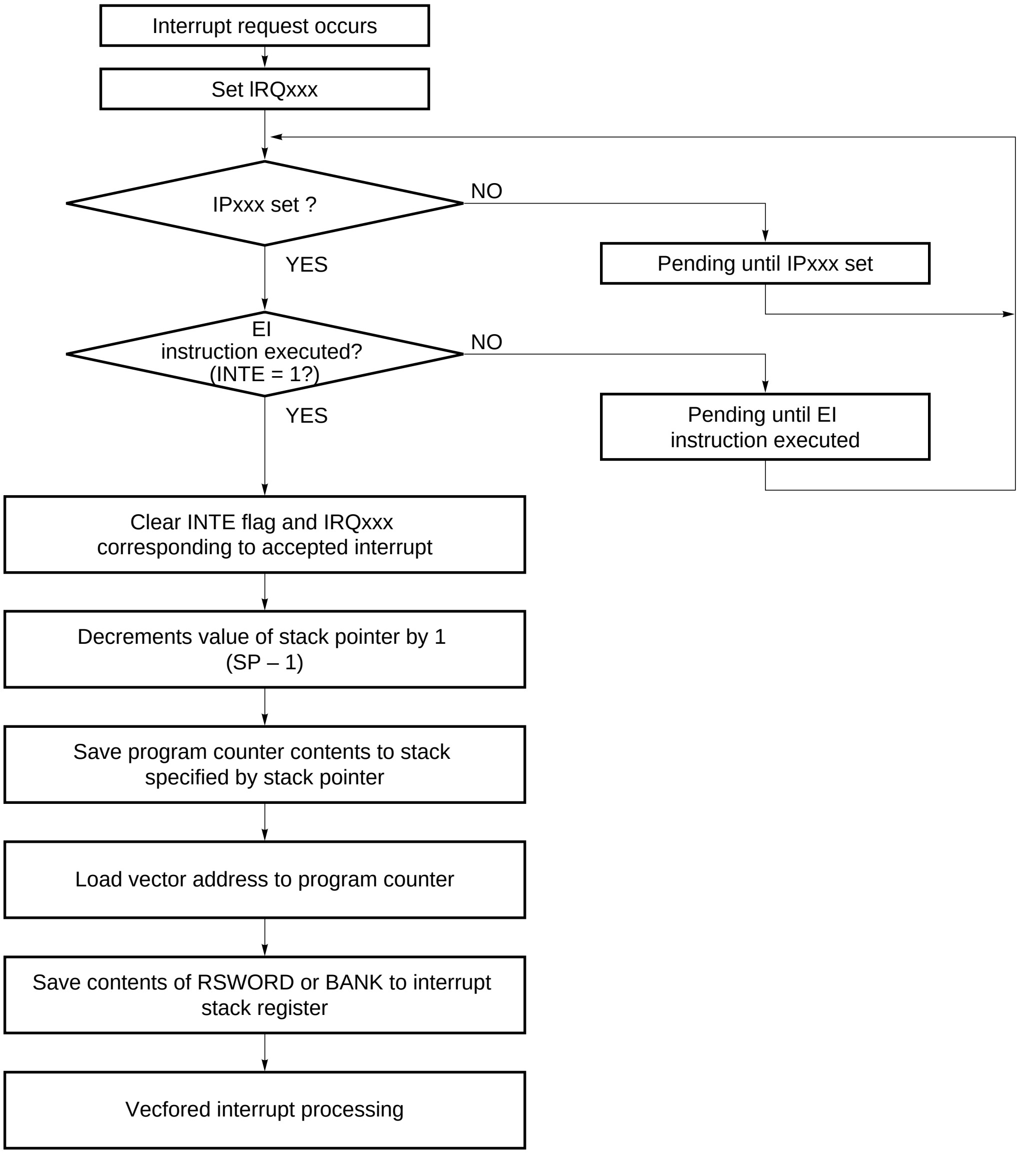


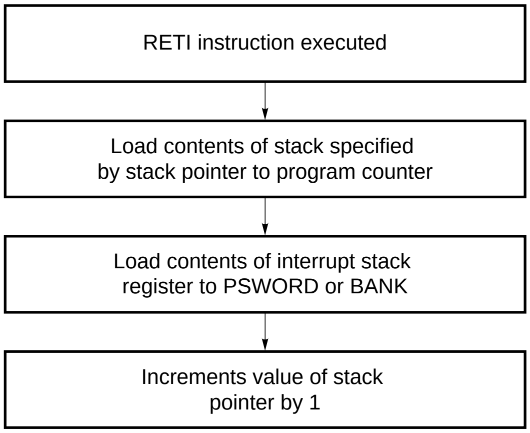
Figure 11-2. Interrupt Processing Sequence


Interrupt request occurs
Set IRQxxx
IPxxx set ?
NO
YES
Pending until IPxxx set
EI
instruction executed?
(INTE = 1?)
NO
YES
Pending until EI
instruction executed
Clear INTE flag and IRQxxx
corresponding to accepted interrupt
Decrements value of stack pointer by 1
(SP – 1)
Save program counter contents to stack
specified by stack pointer
Load vector address to program counter
Save contents of RSWORD or BANK to interrupt
stack register
Vecfored interrupt processing

11.2.2 Returning from interrupt routine

To return execution from the interrupt routine, the RETI instruction is used. In the instruction cycle of this instruction, the following processing is performed:

Figure 11-3. Returning from Interrupt Processing

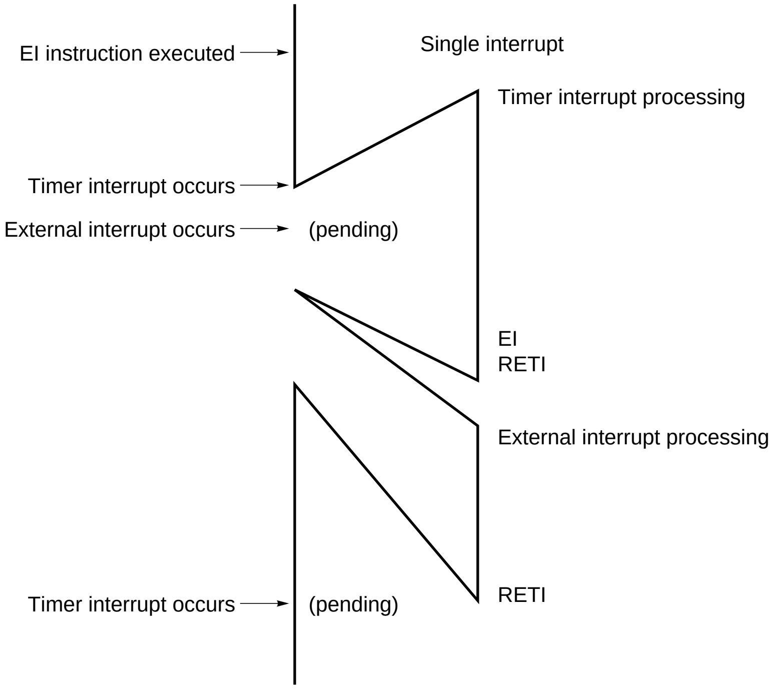

RETI instruction executed
Load contents of stack specified by stack pointer to program counter
Load contents of interrupt stack register to PSWORD or BANK
Increments value of stack pointer by 1

Caution The INTE flag is not set by the RETI instruction.

To process a pending interrupt after finishing certain interrupt processing, execute the EI instruction immediately before the RETI instruction to set the INTE flag to (1).

If the RETI instruction is executed following the EI instruction, no interrupt is accepted in between the EI and RETI instructions. This is because the INTE flag is designed to be set after the execution of the instruction that follows has been completed.

Example



[MEMO]

CHAPTER 12 STANDBY FUNCTION

The μPD172×× subseries is provided with a standby function that reduces the power consumption further from the level peculiar to the CMOS process.

12.1 Function Outline

The μPD172×× subseries standby function is implemented in two modes: STOP and HALT modes.

In the STOP mode, the main clock oscillator circuit is stopped. In this mode, the CPU only consumes leakage current. Therefore, this mode is useful for retention of the data memory contents, without operating the CPU.

In the HALT mode, the main clock oscillator circuit continues oscillating, but the system clock supply is stopped. Consequently, the CPU operation is stopped. The power consumption in the HALT mode is not so low as in the STOP mode, but the HALT mode is useful in applications, where the ordinary operation mode must be restored immediately, when an interrupt request has been issued.

The contents are retained for the data memory, registers, and the output latches of the output ports, immediately before the standby mode, regardless of whether it is the STOP or HALT mode. Therefore, set the I/O ports statuses in advance, in such a manner that the power consumption for the overall system can be minimized.

Table 12-1. Status in Standby Mode

	STOP Mode	HALT Mode
Instruction	STOP	HALT
Main clock oscillator circuit	Stops	Continues
Subclock oscillator circuit	Continues	
Watch timer	Can operate if subclock oscillates	Can operate
Serial interface	Can operate if external $\overline{\text{SCK}}$ is specified as serial clock	Can operate
Timer/counter	Can operate if external clock input is specified	Can operate
CPU	Stops	
Data memory	Retains data	
Control register	Retains data	
Output port	Retains output data	

Caution The STOP instruction is invalid in a system that operates only on the subsystem clock.

12.2 Setting and Releasing STOP Mode

12.2.1 Setting STOP mode

To set the STOP mode, use the STOP instruction. The STOP instruction can be executed only when the main clock is used as the system clock. If the STOP instruction is executed, when the subclock is used as the system clock, the STOP instruction is treated as a NOP instruction, and the STOP mode is not set.

The STOP instruction is also treated as an NOP instruction if executed when the STOP mode releasing condition has been satisfied, and the STOP mode is not set.

The STOP mode releasing condition can be specified by the STOP instruction operand. For the relations between the STOP instruction operand and releasing condition, refer to the Data Sheet for each device.

12.2.2 Operation when STOP mode is released

When the standby releasing condition, specified by the STOP instruction operand, has been satisfied, the following operations are performed after the mode has been released:

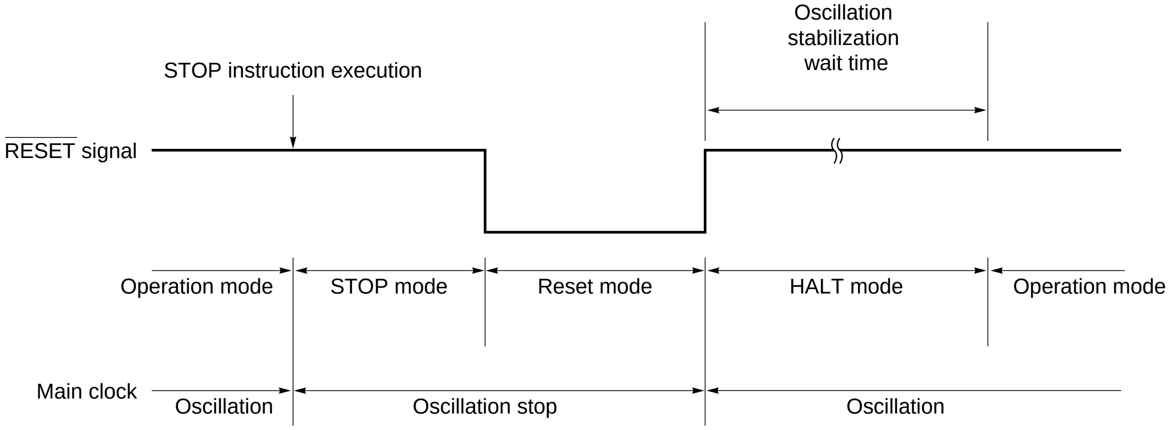
<1> IRQTM is reset.
<2> The basic interval timer (or watch timer counter) and watchdog timer are started (not reset).
<3> The 8-bit timer/counter is reset and started.
<4> The instruction next to “STOP 8H” or the interrupt vector address branch instruction is executed when the value of the 8-bit counter coincides with the value of the modulo register (IRQTM is set).

The oscillation circuit is stopped when the STOP instruction is executed (i.e., when the STOP mode is set), and oscillation is not resumed until the STOP mode is released. After the STOP mode has been released, the HALT mode is set. Set the time at which the HALT mode is to be released by using the timer with modulo function.

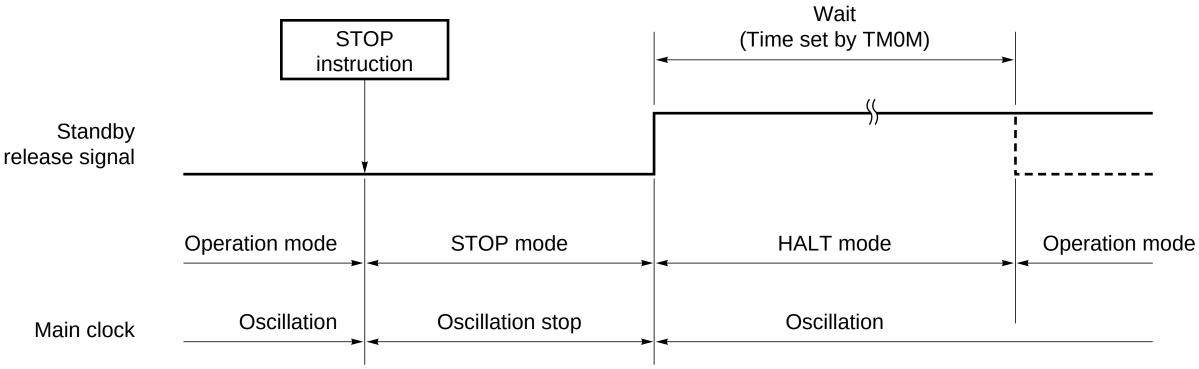
Caution Set the 8-bit modulo register before executing the STOP instruction.

Figure 12-1. Operation after Releasing STOP Mode

(1) If released by $\overline{\text{RESET}}$ input



(2) If released by other than $\overline{\text{RESET}}$



Remark The broken line indicates the case that the interrupt request is accepted after releasing the standby.

12.3 Setting and Releasing HALT Mode

12.3.1 Setting HALT mode

To set the HALT mode, use the HALT instruction.

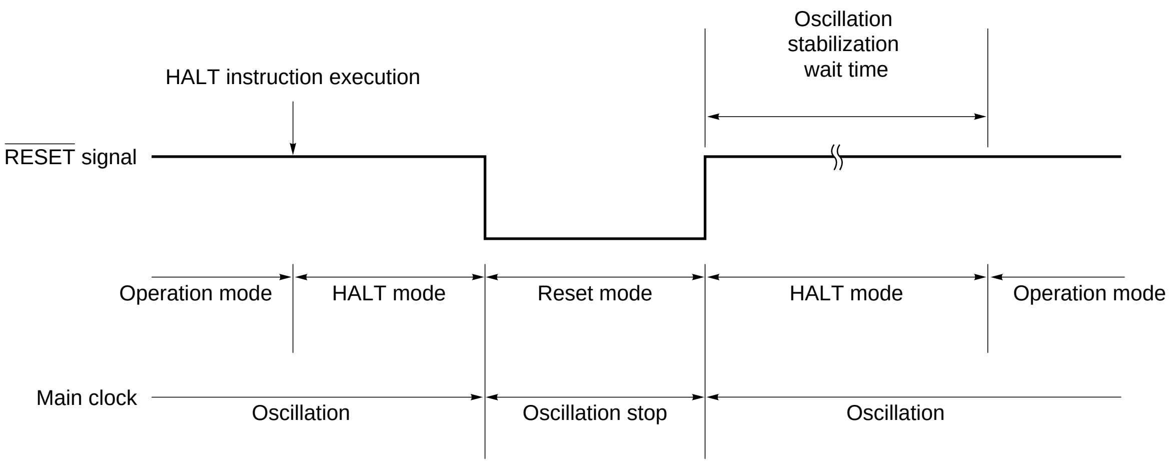
The HALT mode releasing condition can be specified by the HALT instruction operand. For the relations between the operand of the HALT instruction and the releasing conditions, refer to the Data Sheet for each device.

12.3.2 Operation after releasing HALT mode

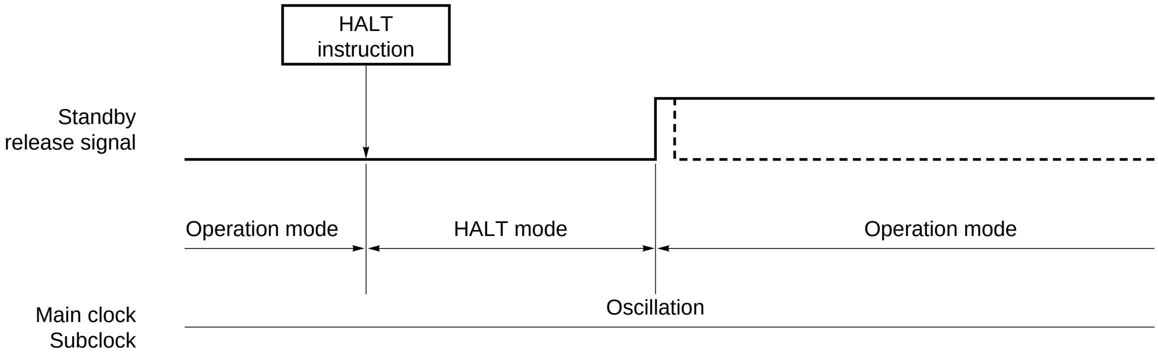
The following operations are performed, when the standby mode releasing condition, specified by the HALT instruction operand, has been satisfied:

Figure 12-2. Operation after Releasing HALT Mode

(1) If released by $\overline{\text{RESET}}$ input



(2) If released by other than $\overline{\text{RESET}}$ input



Remark The broken line indicates the case that the interrupt request is accepted after releasing the standby.

CHAPTER 13 RESET FUNCTION

13.1 Reset by $\overline{\text{RESET}}$ Pin

When a low-level signal has been input to the $\overline{\text{RESET}}$ pin, the system is reset.

Be sure to reset the system at least once after you turn on the power, because the operation of the internal circuits will be undefined.

When the system has been reset, the following circuits are initialized:

(1) Program counter is reset to 0000H.

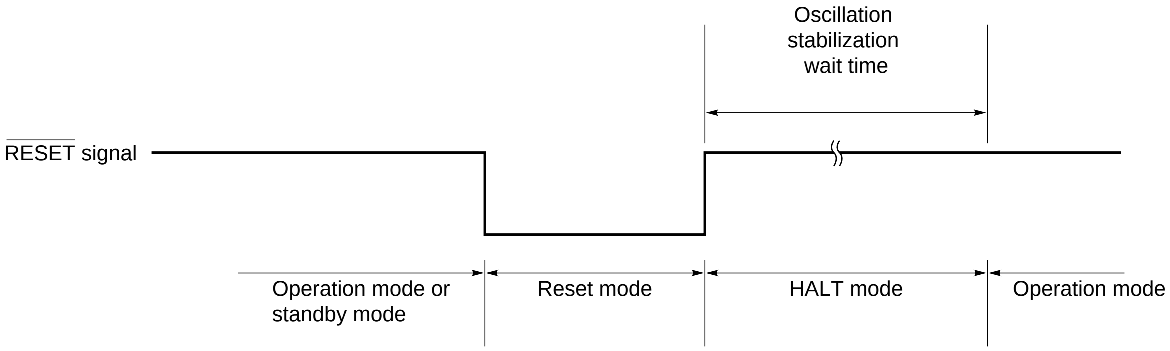
(2) Control registers are initialized.
The initial values of the control registers differ, depending on the device involved. Refer to the Data Sheet for each device.

(3) Data buffer (DBF) is initialized.

(4) Peripheral hardware is initialized.

When the $\overline{\text{RESET}}$ pin is made high, the main clock starts oscillating. After the wait time for oscillation stabilization, the program execution is started from address 0.

Figure 13-1. Reset Operation by $\overline{\text{RESET}}$ Input


Oscillation stabilization wait time
RESET signal
Operation mode or standby mode
Reset mode
HALT mode
Operation mode

13.2 Watchdog Function ($\overline{\text{WDOUT}}$ output)

The µPD172×× subseries microcomputers can check the watchdog timer and stack level when the $\overline{\text{RESET}}$ pin and $\overline{\text{WDOUT}}$ pin are connected, as a watchdog function that prevents program hang up. Therefore, be sure to use the microcomputers with the $\overline{\text{RESET}}$ and $\overline{\text{WDOUT}}$ pins connected.

13.2.1 Reset by watchdog timer (connect $\overline{\text{RESET}}$ and $\overline{\text{WDOUT}}$ pins)

If the watchdog timer is activated while the program is executed, a low level is output to the $\overline{\text{WDOUT}}$ pin, and the program counter is reset to 0.

If the watchdog timer is not reset for a fixed period, therefore, the program can be executed starting from address 0H.

When developing a program, reset the watchdog timer (i.e., set the WDTRES flag) at intervals of within 340 ms (at f_X = 4 MHz).

13.2.2 Reset by stack pointer (connect $\overline{\text{RESET}}$ and $\overline{\text{WDOUT}}$ pins)

If the address stack value reaches a value at which no stack pointer is mounted during the program execution, a low level is output to the $\overline{\text{WDOUT}}$ pin, and the program counter is reset to 0000H.

13.3 Low Voltage Detection Circuit (connect $\overline{\text{RESET}}$ and $\overline{\text{WDOUT}}$ pins)

The low voltage detection circuit outputs a low level from the $\overline{\text{WDOUT}}$ pin to initialize (reset) the system to prevent program hang up that may take place when the battery is exchanged, if the circuit detects a low voltage.

★ With the µPD17225, 17226, 17227 and 17228, a low voltage detection circuit can be set arbitrarily by mask option. For details, refer to the Data Sheets of the respective models.

13.4 Notes on Using INT and $\overline{\text{RESET}}$ Pins

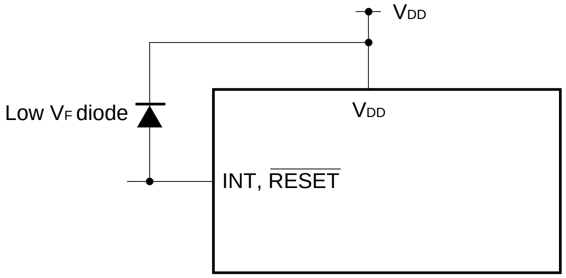
The INT and $\overline{\text{RESET}}$ pins have a function to set a test mode in which the internal operations of the microcontroller is tested (for IC test only), in addition to the pin function.

If a voltage exceeding V_{DD} is applied to either of these pins, the test mode is set. This means that if a noise exceeding V_{DD} is applied to either of these pins while the microcomputer is operating normally, the test mode is accidentally set, hindering the normal operation.

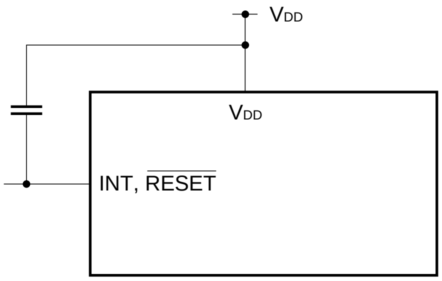
This may take place especially when the wiring length of the INT and $\overline{\text{RESET}}$ pins is long, because such a long wiring is susceptible to noise.

Therefore, keep the wiring length as short as possible to prevent the noise. Taking preventive measures against noise by using external components as shown below is also recommended.

- **Connect a diode having a low V_F across INT and RESET, and V_{DD}**



- **Connect a capacitor across INT and RESET, and V_{DD}**



CHAPTER 14 WRITING AND VERIFYING ONE-TIME PROM

Five one-time PROM models in the μPD172×× subseries are available: μPD17P203A, 17P204, 17P207, and 17P218. The program memories for these models are one-time PROMs, whose contents can be electrically written.

These one-time PROMs use the pins shown in Table 14-1 to write and verify data. Note that there is no address pin. Instead, the address is mPDated by clock input from the CLK pin.

Caution The INT/V_{PP} pin is used as the V_{PP} pin in the program write/verify mode. If a voltage of V_{DD}+0.3 V or higher is applied to the INT/V_{PP} pin in the normal operation mode, the microcontroller may malfunction. Take care to protect this pin from damage.

Table 14-1. Pins Used to Write and Verify Program Memory

Pin	Function
V_{PP}	Applies program voltage (Apply +12.5 V)
V_{DD}	Supplies voltage (Supply +6 V)
CLK	Inputs clock that updates address. Updates program memory address when pulse is input to this pin four times
MD_0-MD_3	Selects operation mode
D_0-D_7	Inputs/outputs 8-bit data

14.1 Differences between Mask ROM and One-Time PROM Models

The μPD17P2×× is a product replacing the program memory of the mask ROM model μPD172×× subseries with a one-time PROM.

Tables 14-3 through 14-7 show the differences between the mask ROM models and one-time PROM models.

Each model differs from the others in terms of ROM and RAM capacities, and whether mask options can be specified, but the CPU function and internal peripheral hardware are the same. The one-time PROM models in the μPD172×× subseries and the corresponding mask ROM models are listed in Table 14-2.

Note that models with a subclock oscillation circuit cannot be used with only the main clock oscillation circuit. Be sure to use the subclock oscillation circuit also.

★ **Table 14-2. One-Time PROM Models and Corresponding Mask ROM Models**

One-time PROM Model	Mask ROM Model
μPD17P203A	μPD17203A
μPD17P204	μPD17204
μPD17P207	μPD17201A, 17207
μPD17P218	μPD17225, 17226, 17227, 17228

Table 14-3. Differences between µPD17P203A and µPD17203A

Parameter	µPD17P203A-001	µPD17P203A-002	µPD17P203A-003	µPD17203A
ROM	One-time PROM			Mask ROM
	4096 × 16 bits			
Pull-up resistor of $\overline{\text{RESET}}$ pin	Provided	None	None	Mask option
Pull-up resistor of P0A, P0B pin		Provided		
Main clock oscillation circuit				
Subclock oscillation circuit		None	Provided	
V_{PP}, PROM program pin	Provided			None

Table 14-4. Differences between µPD17P204 and µPD17204

Parameter	µPD17P204-001	µPD17P204-002	µPD17P204-003	µPD17204
ROM	One-time PROM			Mask ROM
	7936 × 16 bits			
Pull-up resistor of $\overline{\text{RESET}}$ pin	Provided	None	None	Mask option
Pull-up resistor of P0A, P0B pin		Provided		
Main clock oscillation circuit				
Subclock oscillation circuit		None	Provided	
V_{PP}, PROM program pin	Provided			None

Table 14-5. Differences between µPD17P207, µPD17201A, and µPD17207

Parameter	µPD17P207-001	µPD17P207-002	µPD17P207-003	µPD17207	µPD17201A
ROM	One-time PROM			Mask ROM	
	4096 × 16 bits				3072 × 16 bits
Pull-up resistor of $\overline{\text{RESET}}$ pin	Provided	None	None	Mask option	
Main clock oscillation circuit		Provided			
Subclock oscillation circuit		None	Provided		
V_{PP}, PROM program pin	Provided			None	

★

Table 14-6. Differences between µPD17P218, µPD17225, 17226, 17227 and 17228

Parameter	µPD17P218	µPD17225	µPD17226	µPD17227	µPD17228
ROM	One-time PROM	Mask ROM			
	8192 × 16 bits	2048 × 16 bits	4096 × 16 bits	6144 × 16 bits	8192 × 16 bits
Pull-up resistor of $\overline{\text{RESET}}$ pin	Provided	Mask option			
Low-voltage detection circuit					
V_{PP}, PROM program pin	Provided	None			

14.2 Operation Modes for Writing/Verifying Program Memory

The μPD17P2×× is set in a mode in which the program memory can be written or verified after the microcontroller has been reset (V_{DD} = 5 V, $\overline{RESET}$ = 0 V) and then +6 V is applied to the V_{DD} pin and +12.5 V is applied to the V_{PP} pin. This mode is selected as shown in Table 14-8 by the signals input to the MD_0-MD_3 pins. Connect the pins not used for writing/verifying the program memory to GND via a pull-down resistor (470 Ω).

Table 14-7. Selecting Operation Modes

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

Remark ×: don’t care (L or H)

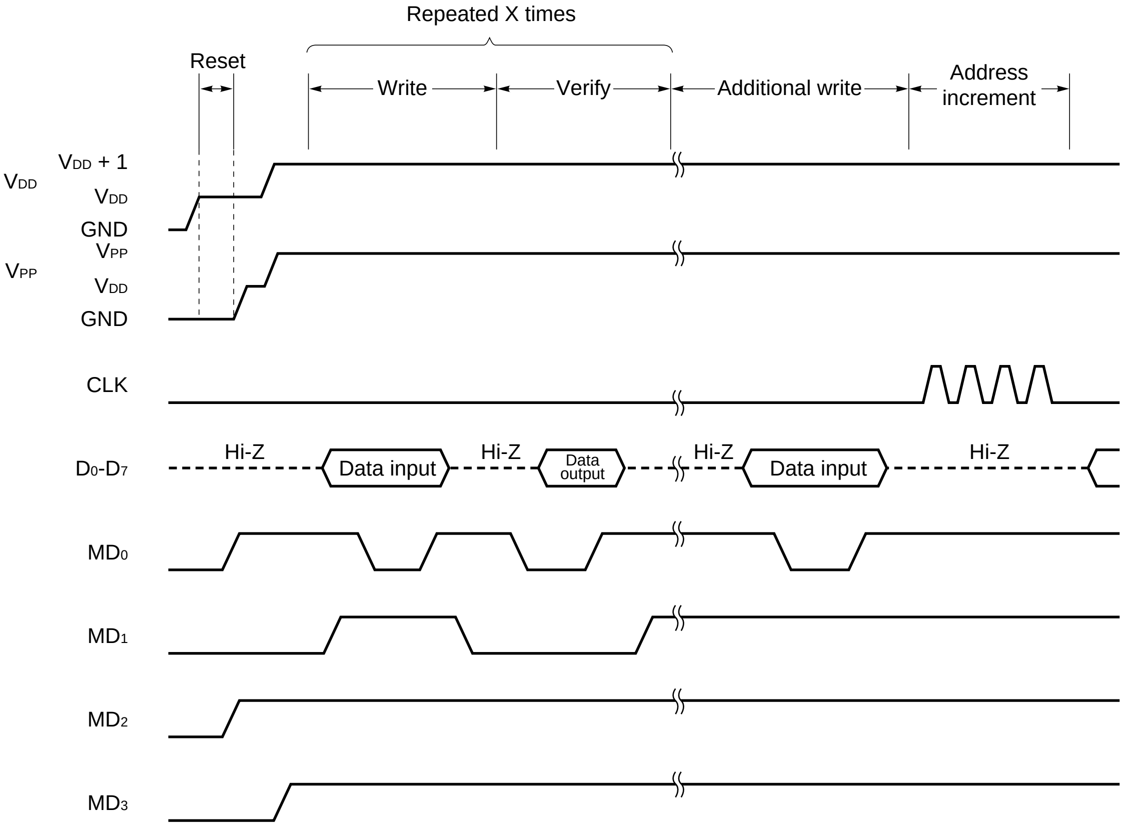
14.3 How to Write Program Memory

Data is written to the program memory in the following sequence. The program memory can be written at high speeds.

(1) Pull down the unused pins through resistors to GND (X_{OUT} pin is open). Make the CLK pin low.
(2) Apply +5 V to the V_{DD} pin. Make the V_{PP} low.
(3) Wait for 10 μs. Then, apply +5 V to the V_{PP} pin.
(4) Set the mode setting pins to the 0 clear mode for the program memory address.
(5) Apply +6 V to the V_{DD} pin and +12.5 V to the V_{PP} pin.
(6) Set the program inhibit mode.
(7) Write data in the 1-ms write mode.
(8) Set the program inhibit mode.
(9) Set the verify mode. If the data has been correctly written, proceed to Step (10). If not, repeat (7) through (9).
(10) Write data again the same number of times (X) as the data has been written during (7) through (9) above × 1 ms.
(11) Set the program inhibit mode.
(12) Input a pulse four times to the CLK pin to increment the program memory address by 1.
(13) Repeat (7) through (12) up to the last address.
(14) Set the 0 clear mode for the program memory address.
(15) Change the voltages of V_{DD} pin and V_{PP} pin into 5 V.
(16) Turn off all the power sources.

Steps (2) through (12) above are illustrated in Figure 14-1.

Figure 14-1. Program Memory Writing Sequence



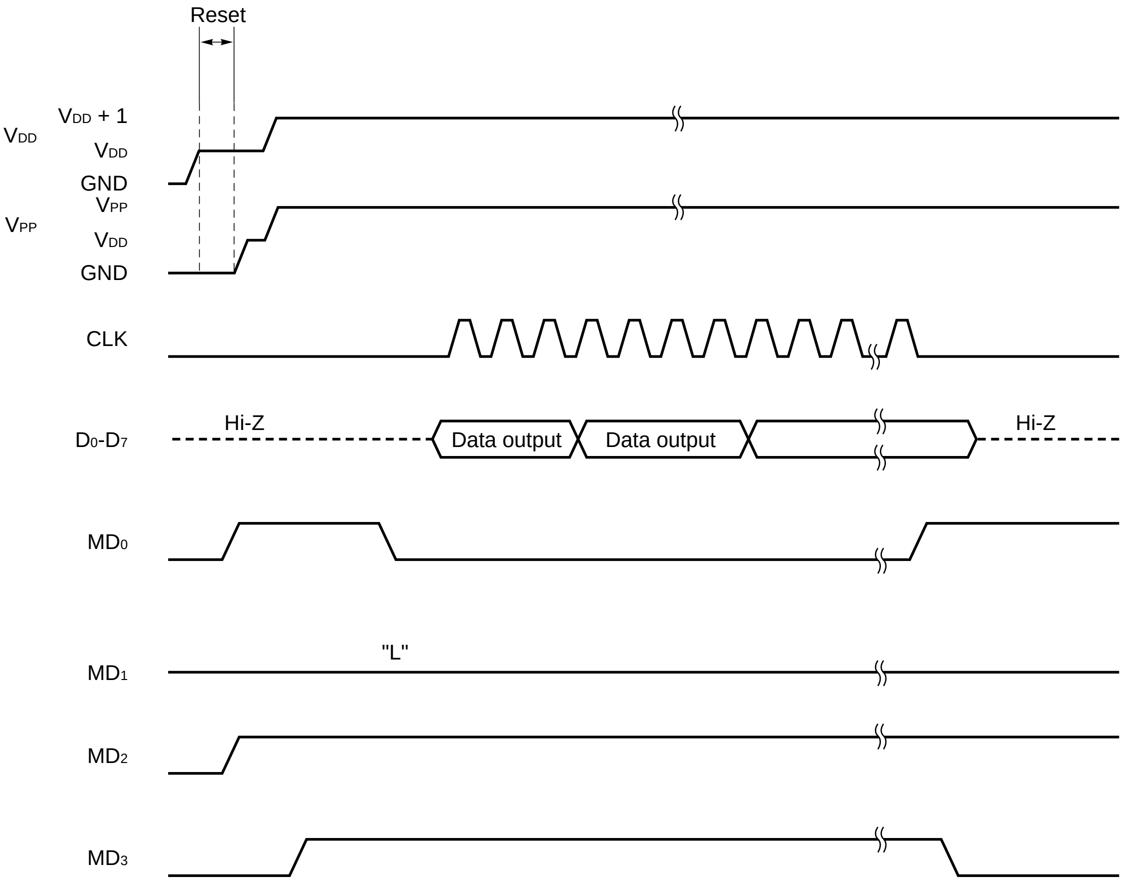
14.4 How to Read Program Memory

The program memory is read in the following sequence:

(1) Pull down the unused pins through resistors to GND. Make the CLK pin low.
(2) Apply +5 V to the V_{DD} pin. Make the V_{PP} pin low.
(3) Wait for 10 μs. Then, apply +5 V to the V_{PP} pin.
(4) Set the mode setting pins to the 0 clear mode for the program memory address.
(5) Apply +6 V to the V_{DD} pin and +12.5 V to the V_{PP} pin.
(6) Set the program inhibit mode.
(7) Set the verify mode. One data address is output, each time a pulse has been input four times to the CLK pin.
(8) Set the program inhibit mode.
(9) Set the 0 clear mode for the program memory address.
(10) Change the voltages of V_{DD} and V_{PP} pins into 5 V.
(11) Turn off all the power sources.

Steps (2) through (9) above are illustrated in Figure 14-2.

Figure 14-2. Program Memory Reading Sequence



[MEMO]

CHAPTER 15 INSTRUCTION SET

15.1 Instruction Set Outline

b_{14} – b_{11} \ b_{15}		0		1	
BIN	HEX				
0 0 0 0	0	ADD	r, m	ADD	m, #n4
0 0 0 1	1	SUB	r, m	SUB	m, #n4
0 0 1 0	2	ADDC	r, m	ADDC	m, #n4
0 0 1 1	3	SUBC	r, m	SUBC	m, #n4
0 1 0 0	4	AND	r, m	AND	m, #n4
0 1 0 1	5	XOR	r, m	XOR	m, #n4
0 1 1 0	6	OR	r, m	OR	m, #n4
0 1 1 1	7	INC INC MOVT BR CALL RET RETSK EI DI RETI PUSH POP GET PUT PEEK POKE RORC STOP HALT NOP	AR IX DBF, @AR @AR @AR AR AR DBF, p p, DBF WR, rf rf, WR r s h 		
1 0 0 0	8	LD	r, m	ST	m, r
1 0 0 1	9	SKE	m, #n4	SKGE	m, #n4
1 0 1 0	A	MOV	@r, m	MOV	m, @r
1 0 1 1	B	SKNE	m, #n4	SKLT	m, #n4
1 1 0 0	C	BR	addr (page 0)	CALL	addr
1 1 0 1	D	BR	addr (page 1)	MOV	m, #n4
1 1 1 0	E			SKT	m, #n
1 1 1 1	F			SKF	m, #n

15.2 Legend

AR	: Address register
ASR	: Address stack register indicated by stack pointer
addr	: Program memory address (11 bits, with highest bit 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 automatically saved 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, m_C
m_R	: Data memory row address (high)
m_C	: Data memory column address (low)
n	: Bit position (4 bits)
n4	: Immediate data (4 bits)
PC	: Program counter
p	: Peripheral address
p_H	: Peripheral address (high-order 3 bits)
p_L	: Peripheral address (low-order 4 bits)
r	: General register column address
rf	: Register file address
rf_R	: Register file row address (high-order 3 bits)
rf_C	: Register file column address (low-order 4 bits)
SP	: Stack pointer
s	: Stop release condition
WR	: Window register
(×)	: Contents addressed by ×

15.3 Instruction List

Instruction	Mnemonic	Operand	Operation	Machine Code			
				Op Code	Operand		
Addition	ADD	r, m	(r) ← (r) + (m)	00000	m_R	m_C	r
		m, #n4	(m) ← (m) + n4	10000	m_R	m_C	n4
	ADDC	r, m	(r) ← (r) + (m) + CY	00010	m_R	m_C	r
		m, #n4	(m) ← (m) + n4 + CY	10010	m_R	m_C	n4
	INC	AR	AR ← AR + 1	00111	000	1001	0000
		IX	IX ← IX + 1	00111	000	1000	0000
Subtraction	SUB	r, m	(r) ← (r) – (m)	00001	m_R	m_C	r
		m, #n4	(m) ← (m) – n4	10001	m_R	m_C	n4
	SUBC	r, m	(r) ← (r) – (m) – CY	00011	m_R	m_C	r
		m, #n4	(m) ← (m) – n4 – CY	10011	m_R	m_C	n4
Logical operation	OR	r, m	(r) ← (r) $\vee$ (m)	00110	m_R	m_C	r
		m, #n4	(m) ← (m) $\vee$ n4	10110	m_R	m_C	n4
	AND	r, m	(r) ← (r) $\wedge$ (m)	00100	m_R	m_C	r
		m, #n4	(m) ← (m) $\wedge$ n4	10100	m_R	m_C	n4
	XOR	r, m	(r) ← (r) $\veebar$ (m)	00101	m_R	m_C	r
		m, #n4	(m) ← (m) $\veebar$ n4	10101	m_R	m_C	n4
Test	SKT	m, #n	CMP ←0, if (m) $\wedge$ n = n, then skip	11110	m_R	m_C	n
	SKF	m, #n	CMP ← 0, if (m) $\wedge$ n = 0, then skip	11111	m_R	m_C	n
Compare	SKE	m, #n4	(m) – n4, skip if zero	01001	m_R	m_C	n4
	SKNE	m, #n4	(m) – n4, skip if not zero	01011	m_R	m_C	n4
	SKGE	m, #n4	(m) – n4, skip if not borrow	11001	m_R	m_C	n4
	SKLT	m, #n4	(m) – n4, skip if borrow	11011	m_R	m_C	n4
Rotate	RORC	r	$\rightarrow CY \rightarrow (r)_{b3} \rightarrow (r)_{b2} \rightarrow (r)_{b1} \rightarrow (r)_{b0}$	00111	000	0111	r
Transfer	LD	r, m	(r) ← (m)	01000	m_R	m_C	r
	ST	m, r	(m) ← (r)	11000	m_R	m_C	r

	MOV	@r, m	if MPE = 1 : (MP, (r)) ← (m)
			if MPE = 0 : (BANK, m_R, (r)) ← (m)
				01010	m_R	m_C	r

		m, @r	if MPE = 1 : (m) ← (MP, (r))
			if MPE = 0 : (m) ← (BANK, m_R, (r))
				11010	m_R	m_C	r

		m, #n4	(m) ← n4	11101	m_R	m_C	n4
	MOVT^Note	DBF, @AR	SP ← SP – 1, ASR ← PC, PC ← AR, DBF ← (PC), PC ← ASR, SP ← SP +1	00111	000	0001	0000
	PUSH	AR	SP ← SP – 1, ASR ← AR	00111	000	1101	0000
	POP	AR	AR ← ASR, SP ← SP + 1	00111	000	1100	0000
	PEEK	WR, rf	WR ← (rf)	00111	rf_R	0011	rf_C
	POKE	rf, WR	(rf) ← WR	00111	rf_R	0010	rf_C

Note As an exception, two machine cycles are necessary for executing the MOVT instruction.

Instruction	Mnemonic	Operand	Operation	Op Code			
				Op Code	Operand		
Transfer	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	**Note**		addr		
		@AR	PC ← AR	00111	000	0100	0000
Subroutine	CALL	addr	SP ← SP – 1, ASR ← PC, PC_{10-0} ← addr, PAGE ← 0	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
Other operations	STOP	s	STOP	00111	010	1111	s
	HALT	h	HALT	00111	011	1111	h
	NOP		No operation	00111	100	1111	0000

Note The operation and op code of “BR addr” differs depending on the ROM size of each model, as follows:

★ **(a) *μ*PD17225**

Operand	Operation	Op Code
addr	PC_{10-0} ← addr	01100

★ **(b) *μ*PD17201A, 17203A, 17P203A, 17207, 17P207, and 17226**

Operand	Operation	Op Code
addr	PC_{10-0} ← addr, PAGE ← 0	01100
	PC_{10-0} ← addr, PAGE ← 1	01101

★ **(c) *μ*PD17227**

Operand	Operation	Op Code
addr	PC_{10-0} ← addr, PAGE ← 0	01100
	PC_{10-0} ← addr, PAGE ← 1	01101
	PC_{10-0} ← addr, PAGE ← 2	01110

★ **(d) *μ*PD17204, 17P218 and 17228**

Operand	Operation	Op Code
addr	PC_{10-0} ← addr, PAGE ← 0	01100
	PC_{10-0} ← addr, PAGE ← 1	01101
	PC_{10-0} ← addr, PAGE ← 2	01110
	PC_{10-0} ← addr, PAGE ← 3	01111

★

15.4 Assembler (RA17K) Macro instructions

Legend

flag n	: FLG type symbol
< >	: Can be omitted

	Mnemonic	Operand	Operation	n
Embedded macro	SKTn	flag 1, ... flag n	if (flag 1) to (flag n)=all "1", then skip	1 ≤ n ≤ 4
	SKFn	flag 1, ... flag n	if (flag 1) to (flag n)=all "0", then skip	1 ≤ n ≤ 4
	SETn	flag 1, ... flag n	(flag 1) to (flag n) ← 1	1 ≤ n ≤ 4
	CLRn	flag 1, ... flag n	(flag 1) to (flag n) ← 0	1 ≤ n ≤ 4

	NOTn	flag 1, ... flag n	if (flag n)="0", then (flag n) ← 1
			if (flag n)="1", then (flag n) ← 0
				1 ≤ n ≤ 4

	INITFLG	<NOT> flag 1,
		... <<NOT> flag n>
			if description=NOT flag n, then (flag n) ← 0
			if description=flag n, then (flag n) ← 1
				1 ≤ n ≤ 4

	BANKn		(BANK) ← n	**Note**
Extension	BRX	Label	Jump Label	—
	CALLX	function-name	CALL sub-routine	—

	INITFLGX	<NOT/INV> flag 1,
		... <NOT/INV> flag n
			if description = NOT (or INV) ← 0
			flag, (flag) ← 0
			if description = flag, (flag) ← 1
				n ≤ 4

Note n = 0 : *μ*PD17225, *μ*PD17226

n = 0, 1 : *μ*PD17P218, *μ*PD17227, *μ*PD17228

n = 0 - 2: *μ*PD17201A, *μ*PD17203A, *μ*PD17P203A, *μ*PD17204, *μ*PD17P204, *μ*PD17207, *μ*PD17P207

15.5 Instruction Functions

15.5.1 Addition instructions

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

<1> OP code

	10 8	7 4	3 0
00000	m_R	m_C	r

<2> Function

When CMP = 0 (r) ← (r) + (m)

Adds the contents of a specified data memory address to the contents of a specified general register, and stores the result in the general register.

When CMP = 1 (r) + (m)

The result is not stored in the register, and the carry flag (CY) and Zero flag (Z) are affected according to the result.

If a carry has occurred as a result of the addition, the carry flag (CY) is set. If not, the carry flag is reset.
If the result of the addition is other than zero, the zero flag (Z) is reset, regardless of the compare flag (CMP).
If the result of the addition becomes zero, with the compare flag reset (CMP = 0), the zero flag (Z) is set.
If the result of the addition becomes zero, with the compare flag set (CMP = 1), the zero flag (Z) is not changed.
Addition can be executed in binary or BCD, which can be selected by the BCD flag (BCD) of the PSWORD.

<3> Example 1

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

```
          (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

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

```
                (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

<table>
<tr><td rowspan="2">Register</td><td colspan="8">RP</td></tr>
<tr><td colspan="4">RPH</td><td colspan="4">RPL</td></tr>
<tr><td>Bit</td><td>b3</td><td>b2</td><td>b1</td><td>b0</td><td>b3</td><td>b2</td><td>b1</td><td>b0</td></tr>
<tr><td rowspan="2">Data</td><td colspan="4">Bank</td><td colspan="3" rowspan="2">Row address</td><td rowspan="2">B
C
D</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

The assignment of RP (general register pointer) in the system register is 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 arithmetic operations to be performed thereafter are carried out in binary and 4-bit units, because the BCD (binary coded decimal) flag is reset.

Example 3

To add the contents of address 0.6FH to those of address 0.03H and store the result in address 0.03H: If IXE = 1, IXH = 0, IXM = 4, and IXL = 0, i.e., if IX = 0.40H, data memory address 0.6FH can be specified by specifying address 2FH.

```
             (0.03H) ← (0.03H) + (0.6FH)
                                   └──Address obtained by ORing index register contents 0.40H with
                                      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

To add the contents of address 0.3FH to those for address 0.03H and store the result in address 0.03H: If IXE = 1, IXH = 0, IXM = 1, and IXL = 0, i.e., if IX = 0.10H, data memory address 0.3FH can be specified by specifying address 2FH.

```
(0.03H) ← (0.03H) + (0.3FH)
                      └──Address obtained by ORing index register contents 0.10H with
                         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

Register	IX											
	IXH				IXM				IXL			
Bit	b3	b2	b1	b0	b3	b2	b1	b0	b3	b2	b1	b0
Data	M P E	0	Bank 0	0	0	Row address			Column address			

The IX (index pointer) assignment in the system register is as shown above.

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

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

<4> Precaution

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

```
MEM013   MEM   0.13H
MEM02F   MEM   0.2FH
         ADD   MEM013, MEM02F
               └──── Means column address of general register.
                     Low-order 4 bits (03H in this case) are valid.
```

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

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

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

<1> OP code

```
          10      8 7      4 3      0
 | 10000  |  mR    |  mC    |  n4   |
```

<2> Function

When CMP = 0 (m) ← (m) + n4

Adds the immediate data to the contents of a specified data memory address and stores the results in the data memory.

When CMP = 1 (m) + n4

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

If a carry has occurred as a result of the addition, the carry flag (CY) is set. If not, the carry flag is reset.
If the result of the addition is other than zero, the zero flag (Z) is reset, regardless of the compare flag (CMP).
If the result of the addition becomes zero, with the compare flag reset (CMP = 0), the zero flag (Z) is set.
If the result of the addition becomes zero, with the compare flag set (CMP = 1), the zero flag (Z) is not changed.
Addition can be executed in binary 4-bit units or BCD, which can be selected by the BCD flag (BCD) of the PSWORD.

<3> Example 1

To add 5 to the contents of address 0.2FH and store the result in address 0.2FH:

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

Example 2

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

```
          (0.6FH) ← (0.6FH) + 05H
                     └─Address obtained by ORing index register contents 0.40H with 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

To add 5 to the contents of address 0.2FH and store the result in address 0.2FH: If IXE = 1, IXH = 0, IXM = 0, and IXL = 0, i.e., if IX = 0.00H, data memory address 0.2FH can be specified by specifying address 2FH.

$(2.2FH) \leftarrow (\underline{0.2FH}) + 05H$

Address obtained by ORing index register contents 0.00H with 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> Precaution

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

When BCD flag = 1, the result of a BCD operation is stored.

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

<1> OP code

	10	8 7	4 3	0
00010	m_R	m_C	r	

<2> Function

When CMP = 0 (r) ← (r) + (m) + CY

Adds the contents of a specified data memory address and the carry flag CY value to the contents of a general register, and stores the result in the general register specified by r.

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

The result is not stored in the register, and the carry flag (CY) and zero flag (Z) are affected by the result.

You can use this ADDC instruction to easily add, two or more words.
If a carry has occurred as a result of the addition, the carry flag (CY) is set. If not, the carry flag is reset.
If the result of the addition is other than zero, the zero flag (Z) is reset regardless of the compare flag (CMP).
If the addition results in zero, with the compare flag reset (CMP = 0), the zero flag (Z)is set.
If the result of the addition results in zero, with the compare flag set (CMP = 1), the zero flag (Z) is not affected.
You can perform addition in binary and 4-bit units or BCD, which you can select by the BCD flag of the PSWORD.

<3> Example 1

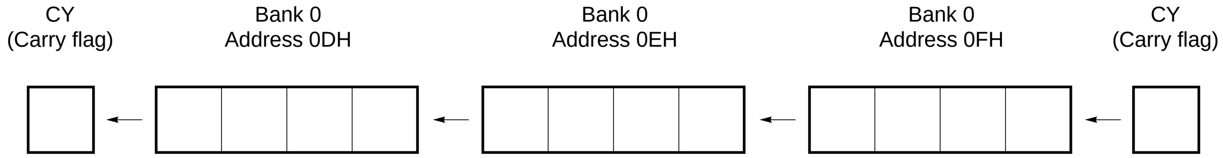
To add the contents of 12-bit addresses 0.2DH through 0.2FH to the 12-bit contents of addresses 0.0DH through 0.0FH and store the result in the 12 bits of addresses 0.0DH through 0.0FH when row address 0 in bank 0 (0.00H-0.0FH) is specified as a general register:

0.0FH ← ($\underline{\text{0.0FH}}$) + (0.2FH)
0.0EH ← ($\underline{\text{0.0EH}}$) + (0.2EH) + CY
0.0DH ← ($\underline{\text{0.0DH}}$) + (0.2DH) + CY

```
MEM00D  MEM   0.0DH
MEM00E  MEM   0.0EH
MEM00F  MEM   0.0FH
MEM02D  MEM   0.2DH
MEM02E  MEM   0.2EH
MEM02F  MEM   0.2FH
        MOV   BANK, #00H          ; 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

To shift the 12-bit contents of addresses 0.2DH through 0.2FH 1 bit to the left with the carry flag when row address 2 (0.20H-0.2FH) of bank 0 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

To add the contents of addresses 0.40H through 0.4FH to the contents of address 0.0FH and store 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

To add the 12-bit contents of addresses 0.40H through 0.42H to the 12-bit contents of addresses 0.0DH through 0.0FH and store the result in the 12 bits of 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

10 87 43 0

10010	m_R	m_C	n4

<2> Function

When CMP = 0 (m) ← (m) + n4 + CY

Adds the immediate data to the contents of a specified data memory address, including the carry flag (CY), and stores the results in the data memory address.

When CMP = 1 (m) + n4 + CY

The result is not stored in the data memory, and the carry flag (CY) and zero flag (Z) are affected by the result.

If a carry has occurred as a result of the addition, the carry flag (CY) is set. If not, the carry flag is reset.
If the result of the addition is other than zero, the zero flag (Z) is reset, regardless of the compare flag (CMP).
If the result of the addition becomes zero, with the compare flag reset (CMP = 0), the zero flag is set.
If the result of the addition becomes zero, with the compare flag set (CMP = 1), the zero flag is not affected.
You can perform addition in binary or BCD, which you can select by the BCD flag of the PSWORD.

<3> Example 1

To add 5 to the 12-bit contents of addresses 0.0DH through 0.0FH and store 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

To add 5 to the 12-bit contents of 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
                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

```
         10     8 7     4 3     0
+-------+-------+-------+-------+
| 00111 |  000  | 1001  | 0000  |
+-------+-------+-------+-------+
```

<2> Function

AR ← AR + 1

Increments the contents of the address register (AR).

<3> Example 1

To add 1 to the 16-bit contents of AR3 through AR0 (address registers) in the system register and store the result in AR3 through AR0:

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

This instruction effect can also be implemented by an addition instruction, as follows:

```
    ADD   AR0,  #01H
    ADDC  AR1,  #00H
    ADDC  AR2,  #00H
    ADDC  AR3,  #00H
```

Example 2

To transfer table data in 16-bit units (1 address) to DBF (data buffer) by using the table reference instruction (for details, refer to **9.2.3 Table reference**):

```
; Address           Table data
010H      DW        0F3FFH
011H      DW        0A123H
012H      DW        0FFF1H
013H      DW        0FFF5H
014H      DW        0FF11H
          :
          :
          MOV       AR3, #0H     ; Table data address
          MOV       AR2, #0H     ; Sets 0010H in address register
          MOV       AR1, #1H
          MOV       AR0, #0H
LOOP:
          MOVT       @AR         ; Reads table data to DBF
          :
          :
          :                      ; Processing referencing table data
          INC       AR           ; register by 1
          BR        LOOP
```

<4> Precaution

The number of bits of the address registers (AR3, AR2, AR1, and AR0) that can be used differs according to the model. For details, refer to the Data Sheet for your device.

(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 contents of the index register (IX).

<3> Example 1

To add 1 to the 12-bit contents of IXH, IXM, and IXL (index registers) in the system register and store the result in IXH, IXM, and IXL:

```
; IXL ← IXL + 1
; IXM ← IXM + CY
; IXH ← IXH + CY
; INC IX
```

You can also execute this instruction by an addition instruction, as follows:

```
ADD   IXL,  #01H
ADDC  IXM,  #00H
ADDC  IXH,  #00H
```

Example 2

To clear all the contents of data memory addresses 0.00H through 0.73H to 0 by using the index register:

```
        MOV     IXH,    #00H       ; Sets index register contents to 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 if index register contents are 73H for bank 0
        SUBC    IXM, #07H          ;
        SUBC    IXH, #00H          ;
        SKT1    Z                  ; Loops until index register contents become 73H for bank 0
        BR      RAM clear          ;
```

15.5.2 Subtraction instructions

(1) SUB r, m **Subtract data memory from general register**

<1> OP code

	10 8	7 4	3 0
00001	m_R	m_C	r

<2> Function

When CMP = 0 (r) ← (r) – (m)

Subtracts the contents of a specified data memory address from the contents of a specified general register, and stores the result in the general register.

When CMP = 1 (r) – (m)

The result is not stored in the register, and the carry flag (CY) and zero flag (Z) are affected by the result.

If a borrow has occurred as a result of the subtraction, the carry flag (CY) is set. If not, the carry flag is reset.
If the result of the subtraction is other than zero, the zero flag (Z) is reset, regardless of the compare flag (CMP).
If the subtraction results in zero, with the compare flag reset (CMP = 0), the zero flag (Z) is set.
If the subtraction results in zero, with the compare flag set (CMP = 1), the zero flag (Z) is not affected.
You can perform subtraction in binary and 4-bit units or BCD, which you can select by the BCD flag of the PSWORD.

<3> Example 1

To subtract the contents of address 0.2FH from those of address 0.03H and store the result in address 0.03H when the row address 0 (0.00H-0.0FH) of 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

To subtract the contents of address 0.2FH from those of address 0.23H and store the result in address 0.23H when row address 2 (0.20H-0.2FH) of bank 0 is specified as a general register (RPH=0, RPL=4):

```
          (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

To subtract the contents of address 0.6FH from those of address 0.03H, and store the result in address 0.03H: If IXE = 1, IXH = 0, IXM = 4, and IXL = 0, i.e., if IX = 0.40H, data memory address 0.6FH can be specified by specifying address 2FH.

```
                (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

To subtract the contents of address 0.3FH from those of address 0.03H and store the result in address 0.03H: If IXE = 1, IXH = 0, IXM = 1, and IXL = 0, i.e., if IX = 0.10H, data memory address 0.3FH can be specified by specifying address 2FH.

```
                (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> Precaution

The first operand of the SUB r, m instruction must be a general register address. Therefore, if you make the following description, address 03H is specified as a register.

```
    MEM013      MEM   0.13H
    MEM02F      MEM   0.2FH
                SUB   MEM013, MEM02F
                          General register address must be within the 00H-0FH range (set register
                          pointer to other than row address 1).
```

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

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

(2) SUB m, #n4 Subtract immediate data from data memory

<1> OP code

```
           10        8 7       4 3        0
  +---------+---------+---------+---------+
  |  10001  |   mR    |   mC    |   n4    |
  +---------+---------+---------+---------+
```

<2> Function

When CMP = 0 (m) ← (m) – n4

Subtracts specified immediate data from the contents of a specified data memory address, and stores the result in the data memory address.

When CMP = 1 (m) – n4

The result is not stored in the data memory, and the carry flag (CY) and zero flag (Z) are affected by the result.

If a borrow has occurred as a result of the subtraction, the carry flag (CY) is set. If not, the carry flag is reset.
If the result of the subtraction is other than zero, the zero flag (Z) is reset regardless of the compare flag (CMP).
If the subtraction results in zero when the compare flag is reset (CMP = 0), the zero flag is set.
If the subtraction results in zero when the compare flag is set (CMP = 1), the zero flag is not affected.
You can perform subtraction in binary and 4-bit units and BCD, which you can select by the BCD flag for the PSWORD.

<3> Example 1

To subtract 5 from the address 0.2FH contents and store 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 contents of address 0.6FH and store the result in address 0.6FH: At this time, if IXE = 1, IXH = 0, IXM = 4, and IXL = 0, i.e., if IX = 0.40H, data memory address 0.6FH can be specified by specifying address 2FH.

```
            0.6FH ← (0.6FH) – 5
                      └─Address obtained by ORing index register contents 0.40H with 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

To subtract 5 from the contents of address 0.2FH and store the result in address 0.2FH: If IXE = 1, IXH = 0, IXM = 0, and IXL = 0, i.e., if IX = 0.00H, data memory address 0.2FH can be specified by specifying address 2FH.

$(0.2FH) \leftarrow (\underline{0.2FH}) - 5$

└ Address obtained by ORing index register contents 0.00H with 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> Precaution

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

When BCD flag = 1, the result of the BCD format operation is stored.

(3) SUBC r, m — Subtract data memory from general register with carry flag

<1> OP code

10 8 7 4 3 0

00000	m_R	m_C	r

<2> Function

When CMP = 0 (r) ← (r) – (m) – CY

Subtracts the contents of a specified data memory, including the carry flag (CY), from the contents of a specified general register, and stores the result in the general register. By using this SUBC instruction, subtraction of two or more words can be easily carried out.

When CMP = 1 (r) – (m) – CY

The result is not stored in the register, and the carry flag (CY) and zero flag (Z) are affected by the result.

If a borrow has occurred, as a result of the subtraction, the carry flag (CY) is set. If not, the carry flag is reset.
If the result of the subtraction is other than zero, the zero flag (Z) is reset, regardless of the compare flag (CMP).
If the subtraction results in zero when the compare flag is reset (CMP = 0), the zero flag is reset.
If the subtraction results in zero when the compare flag set (CMP = 1), the zero flag is not changed.
You can perform subtraction in binary and 4-bit units or BCD, which you can select by the BCD flag of the PSWORD.

<3> Example 1

To subtract the 12-bit contents of addresses 0.2DH through 0.2FH from the 12-bit contents of addresses 0.0DH through 0.0FH and store the result in the 12 bits of addresses 0.0DH through 0.0FH when row address 0 of bank 0 (0.00H-0.0FH) 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

To subtract the 12-bit contents of addresses 0.40H through 0.42H from the 12-bit contents of addresses 0.0DH through 0.0FH and store the result in the 12 bits of 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

To compare the 12-bit contents of addresses 0.00H through 0.03H with the 12-bit contents of addresses 0.0CH through 0.0FH and jump to LAB1, if both the 12-bit contents are the same. If not, jump 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 ; 0.00H-0.03H, because CMP flag is set
                SUBC  MEM001, MEM00D ; Address contents are not affected
                SUBC  MEM002, MEM00E ;
                SUBC  MEM003, MEM00F ;
                SKF1  Z                   ; Z flag = 1, if result is the same.
                BR    LAB1                ; Z flag = 0, if result is not the same.
                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 specified immediate data from the contents of a specified data memory, including the carry flag, and stores the result in the data memory address.

When CMP = 1 (m) – n4 – CY

The result is not stored in the data memory, and the carry flag (CY) and zero flag (Z) are affected by the result.

If a borrow has occurred as a result of the subtraction, the carry flag (CY) is set. If not, the carry flag is reset.
If the result of the subtraction is other than zero, the zero flag (Z) is reset, regardless of the compare flag (CMP).
If the subtraction results in zero when the compare flag is reset (CMP = 0), the zero flag is set.
If the subtraction results in zero when the compare flag is set (CMP = 1), the zero flag is not changed.
You can perform subtraction in binary and 4-bit units or BCD, which can be selected by the BCD flag of the PSWORD.

<3> Example 1

To subtract 5 from the 12-bit contents of addresses 0.0DH through 0.0FH and store 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 of 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

To compare the 12-bit contents of addresses 0.00H through 0.03H with immediate data 0A3FH and jump to LAB1 when both the 12-bit contents are the same. If not, jump 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       ; 0.00H-0.03H, because CMP flag is set
                SUBC  MEM001, #0AH      ; Address contents are not affected
                SUBC  MEM002, #3H       ;
                SUBC  MEM003, #0FH      ;
                SKF1  Z                 ; Z flag = 1, if result is the same.
                BR    LAB1              ; Z flag = 0, if result is not the same.
                BR    LAB2
                      :
          LAB1:       :
                      :
          LAB2:       :
                      :
                      :
```

15.5.3 Logical operation instructions

(1) OR r, m OR between general register and data memory

<1> OP code

```
      10      8 7      4 3      0
| 00110 |  mR   |  mC   |   r   |
```

<2> Function

(r) ← (r) ∨ (m)

ORs the contents of a specified data memory address with the contents of a specified general register, and stores the result in the general register.

<3> Example

To OR the contents of address 0.03H (1010B) with the contents of address 0.2FH (0111B) and store the result (1111B) in address 0.03H.

(0.03H) ← (0.03H) ∨ (0.2FH)

```
| 1 | 0 | 1 | 0 |   Address 03H
       OR
| 0 | 1 | 1 | 1 |   Address 2FH
        ↓
| 1 | 1 | 1 | 1 |   Address 03H
```

```
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

10 87 43 0

10110	m_R	m_C	n4

<2> Function

(m) ← (m) ∨ n4

ORs the contents of a specified data memory address with specified immediate data and stores the result in the data memory address.

<3> Example 1

To set bit 3 (MSB) of address 0.03H.

(0.03H) ← (0.03H) ∨ 1000B

Address 0.03H

1	×	×	×

× : don't care

```
MEM003   MEM   0.03H
         OR    MEM003, #1000B
```

Example 2

To set all the bits of 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

	10 8	7 4	3 0
00100	m_R	m_C	r

<2> Function

(r) ← (r) ∧ (m)

ANDs the contents of a specified data memory address with the contents of a specified general register, and stores the result in the general register.

<3> Example

To AND the contents of address 0.03H (1010B) with the contents of address 0.2FH (0110B) and store the result (0010B) in address 0.03H.

(0.03H) ← (0.03H) ∧ (0.2FH)

1	0	1	0	Address 03H
AND				
0	1	1	0	Address 2FH
↓				
0	0	1	0	Address 03H

```
MEM003    MEM 0.03H
MEM02F    MEM 0.2FH
          MOV   MEM003, #1010B
          MOV   MEM02F, #0110B
          AND   MEM003, MEM02F
```

(4) AND m, #n4 AND between data memory and immediate data

<1> OP code

	10 8	7 4	3 0
00000	m_R	m_C	n4

<2> Function

(m) ← (m) ∧ n4

ANDs the contents of a specified data memory address with specified immediate data, and stores the result in the data memory address.

<3> Example 1

To reset bit 3 (MSB) of address 0.03H.

(0.03H) ← (0.03H) ∧ 0111B

Address 0.03H

0	×	×	×

×: don't care

```
MEM003    MEM   0.03H
          AND   MEM003, #0111B
```

Example 2

To reset all the bits of 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

	10 8	7 4	3 0
00101	m_R	m_C	r

<2> Function

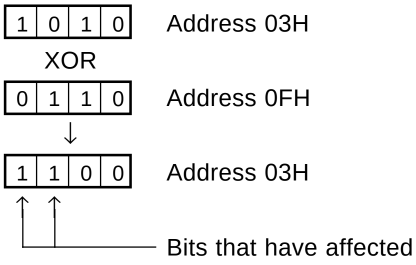
$(r) \leftarrow (r) \veebar (m)$

Exclusive-ORs the contents of a specified data memory address with the contents of a specified general register, and stores the result in the general register.

<3> Example 1

To compare the contents of address 0.03H with those of address 0.0FH, set and store in address 0.03H bits not in agreement. If all the bits of address 0.03H are reset (i.e., if the address 0.03H contents are the same as those of address 0.0FH), jump to LBL1; otherwise, to jump to LBL2.

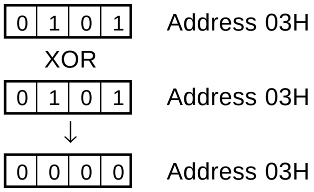
This example compares the status of an alternate switch (address 0.03H contents) with the internal status (address 0.0FH contents) and to branch to the processing of the switch that has affected.



```
MEM003  MEM   0.03H
MEM00F  MEM   0.0FH
        XOR   MEM003, MEM00F
        SKNE  MEM003, #00H
        BR    LBL1
        BR    LBL2
```

Example 2

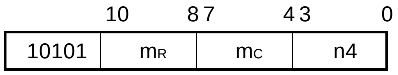
To clear the address 0.03H contents



```
MEM003  MEM   0.03H
        XOR   MEM003, MEM003
```

(6) XOR m, #n4 Exclusive OR between data memory and immediate data

<1> OP code



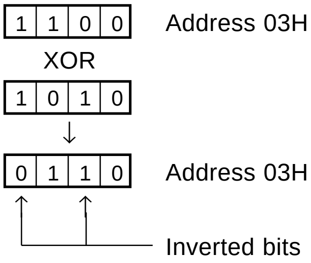
<2> Function

$(m) \leftarrow (m) \veebar n4$

Exclusive-ORs the contents of a specified data memory address with specified immediate data, and stores the result in the data memory address.

<3> Example

To invert bits 1 and 3 of address 0.03H and store the results in address 03H:



```
MEM003   MEM   0.03H
         XOR   MEM003, #1010B
```

15.5.4 Test instructions

(1) SKT m, #n **Skip next instruction if data memory bits are true**

<1> OP code

	10 8	7 4	3 0
11110	m_R	m_C	n

<2> Function

CMP ← 0, if (m) ∧ n = n, then skip

Skips the next one instruction if the result of ANDing the specified data memory contents with immediate data n is equal to n (Excecutes as NOP instruction).

<3> Example 1

To jump to AAA if bit 0 of address 03H is ‘1’; if it is ‘0’, to jump to BBB.

```
SKT     03H, #0001B
BR      BBB
BR      AAA
```

Example 2

To skip the next instruction if both bits 0 and 1 of 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 of 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) $\wedge$ n = 0, then skip

Skips the next one instruction if the result of ANDing the specified data memory contents with immediate data n is 0 (Executes as NOP instruction).

<3> Example 1

To store immediate data 00H in data memory address 0FH if bit 2 of address 13H is 0; if it is 1, to jump to ABC.

```
MEM013    MEM    0.13H
MEM00F    MEM    0.0FH
          SKF    MEM013, #0100B
          BR     ABC
          MOV    MEM00F, #00H
```

Example 2

To skip the next instruction if both bits 3 and 0 of address 29H are ‘0’.

```
SKF         29H, #1001B
```

Skip condition 29H

b_3	b_2	b_1	b_0
0	×	×	0

×: don’t care

Example 3

The results of executing the following two instructions are the same:

```
SKF         34H, #1111B
SKE         34H, #00H
```

15.5.5 Compare 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

Skips the next one instruction if the contents of a specified data memory address are equal to the value of the immediate data (Executes as NOP instruction).

<3> Example

To transfer 0FH to address 24H, if the address 24H contents are 0. If not, jump 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

10	8 7	4 3	0
01011	m_R	m_C	n4

<2> Function

(m) – n4, skip if not zero

Skips the next one instruction if the contents of a specified data memory address are not equal to the value of the immediate data (Executes as NOP instruction).

<3> Example

To jump to XYZ if the contents of address 1FH are 1 and if the address 1EH contents are 3; otherwise, jump to ABC.

To compare 8-bit data, this instruction is used in the following combination.

	3		1
IEH	0011	IFH	0001

```
MEM01E   MEM   0.1EH
MEM01F   MEM   0.1FH
         SKNE  MEM01F, #01H
         SKE   MEM01E, #03H
         BR    ABC
         BR    XYZ
```

The same operation can be performed by using the compare and zero flags as follows:

```
MEM01E   MEM   0.1EH
MEM01F   MEM   0.1FH
         SET2  CMP, Z              ; CMP flag ← 1, Z flag ← 1
         SUB   MEM01F, #01H
         SUBC  MEM01E, #03H
         SKT1  Z
         BR    ABC
         BR    XYZ
```

(3) SKGE m, #n4 Skip if 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 contents of a specified data memory address are greater than the value of the immediate data (Executes as NOP instruction).

<3> Example

To execute RET if the 8-bit data, stored in addresses 1FH (higher) and 2FH (lower) is greater than immediate data 17H; otherwise, execute RETSK.

```
MEM01F   MEM   0.1FH
MEM02F   MEM   0.2FH
         SKGE  MEM01F, #1
         RETSK
         SKNE  MEM01F, #1
         SKLT  MEM02F, #8        ; 7+1
         RET
         RETSK
```

(4) SKLT m, #n4 Skip if 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 contents of a specified data memory address are less than the value of the immediate data (Executes as NOP instruction).

<3> Example

To store 01H in address 0FH if the address 10H contents is greater than immediate data ‘6’; otherwise, to store 02H in address 0FH.

```
MEM00F   MEM   0.0FH
MEM010   MEM   0.10H
         MOV   MEM00F, #02H
         SKLT  MEM010, #06H
         MOV   MEM00F, #01H
```

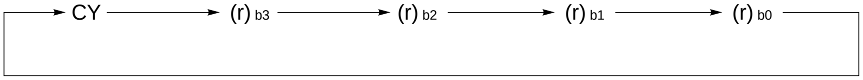
15.5.6 Rotation instruction

(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 a general register specified by r 1 bit to the right with the carry flag.

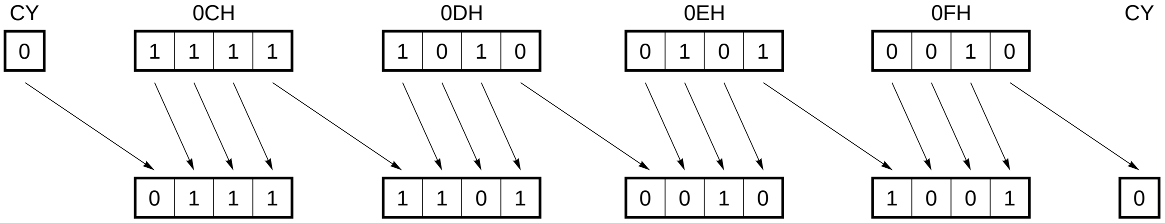
<3> Example 1

To rotate the value of address 0.00H (1000B) 1 bit to the right when row address 0 (0.00H-0.0FH) of bank 0 is specified as a general register (RPH = 0, RPL = 0). As a result, the value of the address becomes 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                ; Carry flag ← 0
                RORC  MEM000
```

Example 2

To rotate the value 0FA52H of the data buffer (DBF) 1 bit to the right when row address 0 (0.00H-0.0FH) of bank 0 is specified as a general register (RPH = 0, RPL = 0). As a result, the value of the data buffer becomes 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                ; Carry flag ← 0
                RORC  MEM00C
                RORC  MEM00D
                RORC  MEM00E
                RORC  MEM00F
```

15.5.7 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

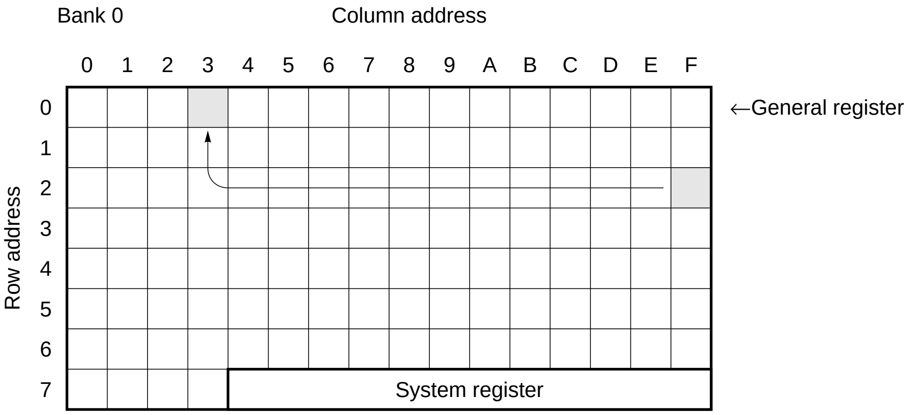
(r) ← (m)

Loads the contents of a specified data memory address to a specified general register.

<3> Example 1

To load the address 0.2FH contents to address 0.03H.

```
                (0.03H) ← (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
          LD    MEM003, MEM02F
```



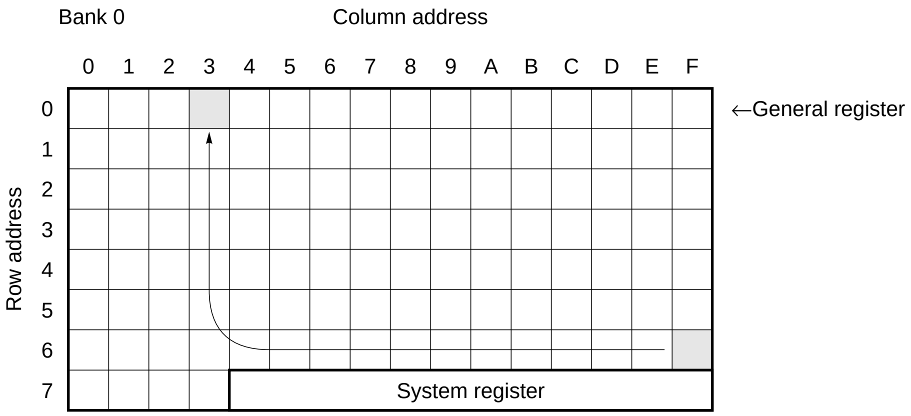
Example 2

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

IXH ← 00H
IXM ← 04H
IXL ← 00H
IXE flag ← 1
(0.03H) ← ($\underline{0.6FH}$)

Address obtained by ORing index register contents 040H with 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

```
          10       8 7       4 3       0
| 11000  |   mR    |   mC    |    r    |
```

<2> Function

(m) ← (r)

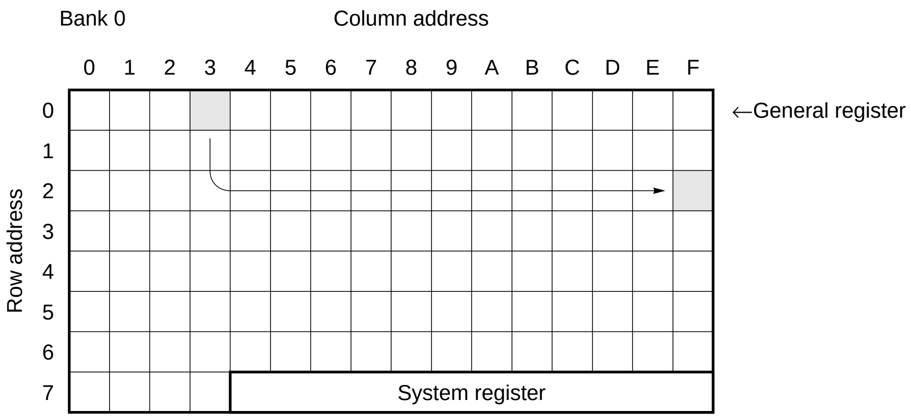
Stores the contents of a specified general register to a specified data memory address.

<3> Example 1

To store the contents of address 0.03H in address 0.2FH.

(0.2FH) ← (0.03H)

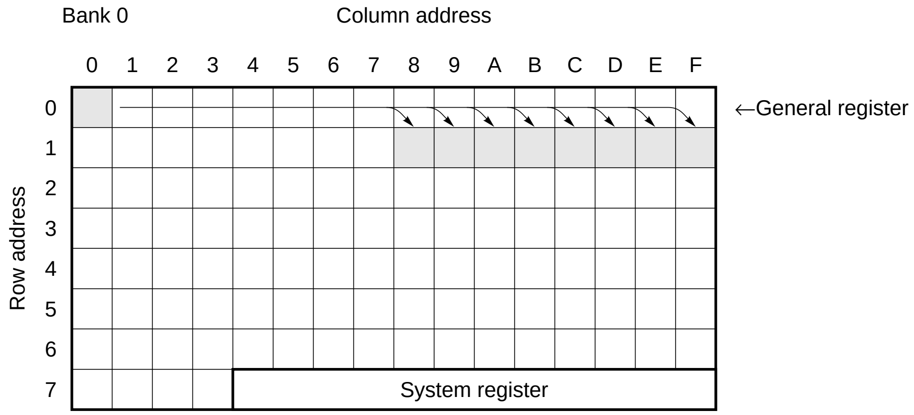
```
MOV   RPH, #00H      ; General register bank 0
MOV   RPL, #00H      ; General register row address 0
ST    2FH, 03H       ; Transfers general register contents to data memory
```



Example 2

To store the contents of 0.00H in addresses 0.18H through 0.1FH. Data memory (18H to 1FH) is addressed 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 address 0.00H in data memory.
MEM018    MEM   0.18H
MEM000    MEM   0.00H
  LOOP1:
          SET1  IXE               ; IXE flag ← 1
          ST    MEM018, MEM000    ; (0.1 x H) ← (0.00H)
          CLR1  IXE               ; IXE flag ← 0
          INC   IX                ; IX ← IX + 1
          SKGE  IXL, #08H
          BR    LOOP1
```



(3) MOV @r, m Move data memory to destination indirect

<1> OP code

10 87 43 0

01010	m_R	m_C	r

<2> unction

When MPE = 1

(MP,(r)) ← (m)

When MPE = 0

(BANK, m_R, (r)) ← (m)

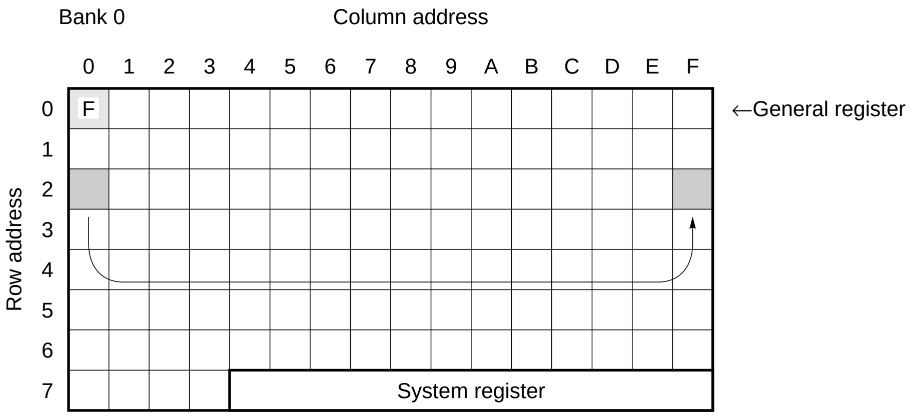
Stores the contents of a specified data memory address to the data memory addressed by the contents of a specified general register.

When MPE = 0, transfer is executed in the same row address of the same bank.

<3> Example 1

To store the contents of address 0.20H in address 0.2FH with the MPE flag cleared to 0. The destination data memory source address is the same row address as that of the transfer source, and the contents of the general register at address 0.00H are the column address.

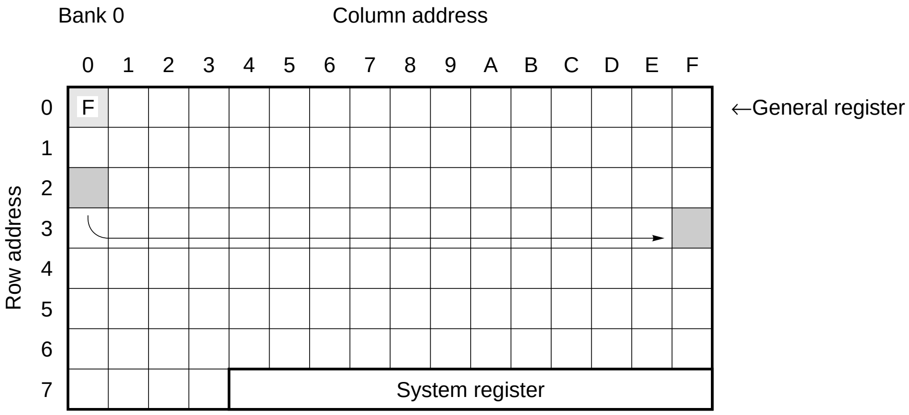
```
          (0.2FH) ← (0.20H)
MEM000    MEM    0.00H
MEM020    MEM    0.20H
          CLR1   MPE                 ; MPE flag   0
          MOV    MEM000,#0FH         ; Sets column address in general register
          MOV    @MEM000, MEM020     ; Stores
```



Example 2

To store the contents of address 0.20H in address 0.3FH with the MPE flag set to 1. The row address of the data memory at the transfer destination is specified is the contents of memory pointer MP, and the column address is the contents of the general register at address 0.00H.

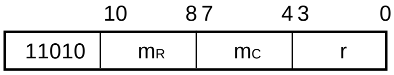
```
          (0.3FH) ← (0.20H)
MEM000    MEM   0.00H
MEM020    MEM   0.20H
          MOV   RPH, #00H          ; General register bank 0
          MOV   RPL, #00H          ; General register row address 0
          MOV   00H, #0FH          ; Sets column address in general register
          MOV   MPH, #00H          ; Sets row address in memory pointer
          MOV   MPL, #03H          ;
          SETI  MPE                ; MPE flag ← 1
          MOV   @MEM000, MEM020    ; Stores
```



(4) MOV m, @r

Move data memory to destination indirect

<1> OP code



<2> Function

When MPE = 1

(m) ← (MP,(r))

When MPE = 0

(m) ← (BANK,mR, (r))

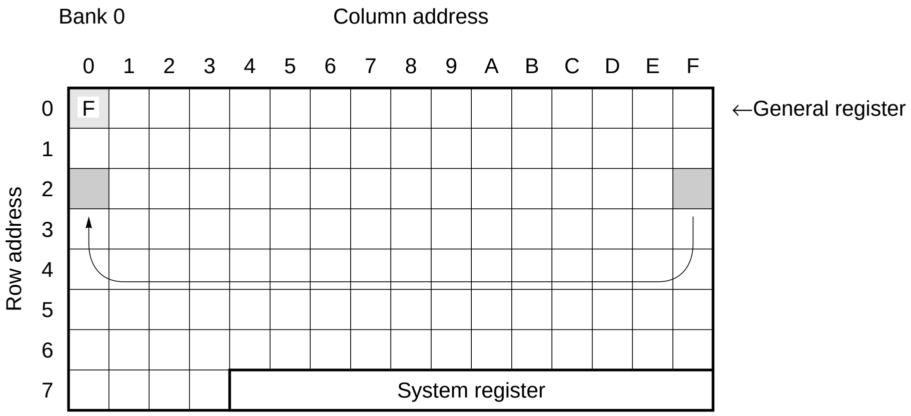
Stores the contents of the data memory addressed by the contents of a specified general register to another data memory address.

When MPE = 0, transfer is executed in the same row address of the same bank.

<3> Example 1

To store the contents of address 0.2FH in address 0.20H with the MPE flag cleared to 0. The data memory at the transfer source is at the same row address as the destination, and the column address is the contents of the general register at address 0.00H.

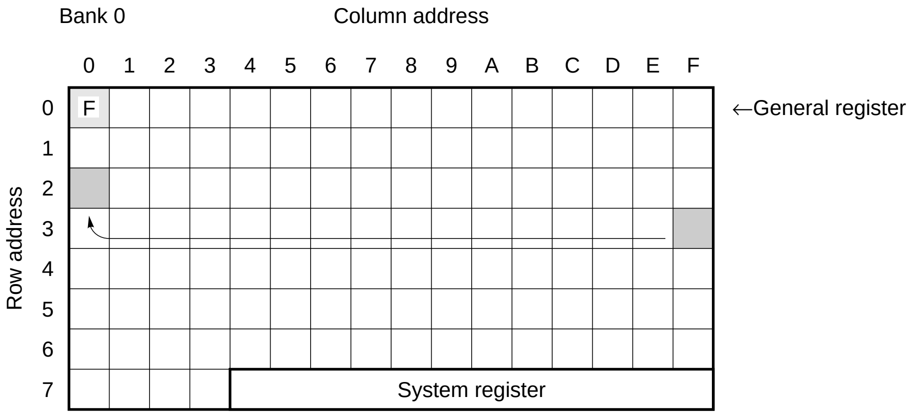
```
                (0.20H) ← (0.2FH)
    MEM000      MEM   0.00H
    MEM020      MEM   0.20H
                CLR1  MPE                ; MPE flag ← 0
                MOV   MEM000, #0FH       ; Sets column address in general register
                MOV   MEM020, @MEM000    ; Stores
```



Example 2

To store the contents of address 0.3FH in address 0.20H with the MPE flag set to 1. The row address of the data memory at the transfer source is the contents of the memory pointer, and the column address is the contents of the general register at address 0.00.

```
                (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          ;
                SETI  MPE                ; MPE flag ← 1
                MOV   MEM020, @MEM000    ; Stores
```



(5) MOV m, #n4 — Move immediate data to data memory

<1> OP code

	10 8	7 4	3 0
11001	m_R	m_C	n4

<2> Function

(m) ← n4

Stores immediate data in a specified data memory address.

<3> Example 1

To store immediate data 0AH in data memory address 0.50H.

```
          (0.50H) ← 0AH
MEM050    MEM  0.50H
          MOV  MEM050, #0AH
```

Example 2

To store immediate data 07H in address 0.32H when data memory address 0.00H is specified and if 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

	10 8	7 4	3 0
00111	000	0001	0000

<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, in data buffer DBF.

Because this instruction temporarily uses one level of stack, pay attention to the nesting of subroutines and interrupts.

<3> Example

To transfer 16 bits of table data to data buffers (DBF3, DBF2, DBF1, and DBF0) according to the values of the address registers (AR3, AR2, AR1, and AR0) in the system register.

```
            ; *
            ; **    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 data of address 0011H to DBF
```

In this case, the data is stored in DBF as follows:

DBF3 = 0AH
DBF2 = 0BH
DBF1 = 0CH
DBF0 = 0DH

(7) PUSH AR — Push address register

<1> OP code

```
               10       8 7       4 3      0
| 00111  |   000    |  1101   |  0000  |
```

<2> Function

SP ← SP–1,

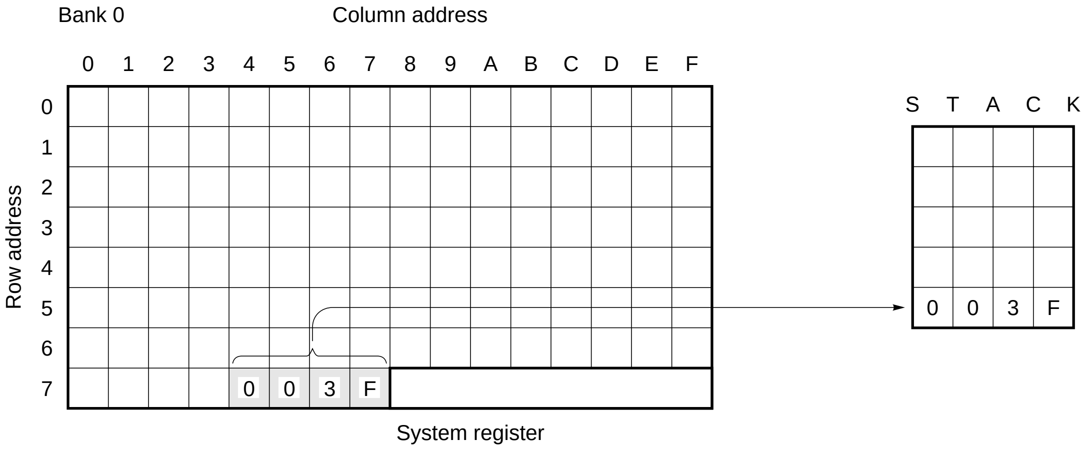
ASR ← AR

Decrements the stack pointer SP and stores the value of the address register AR in the address stack register specified by the stack pointer.

<3> Example 1

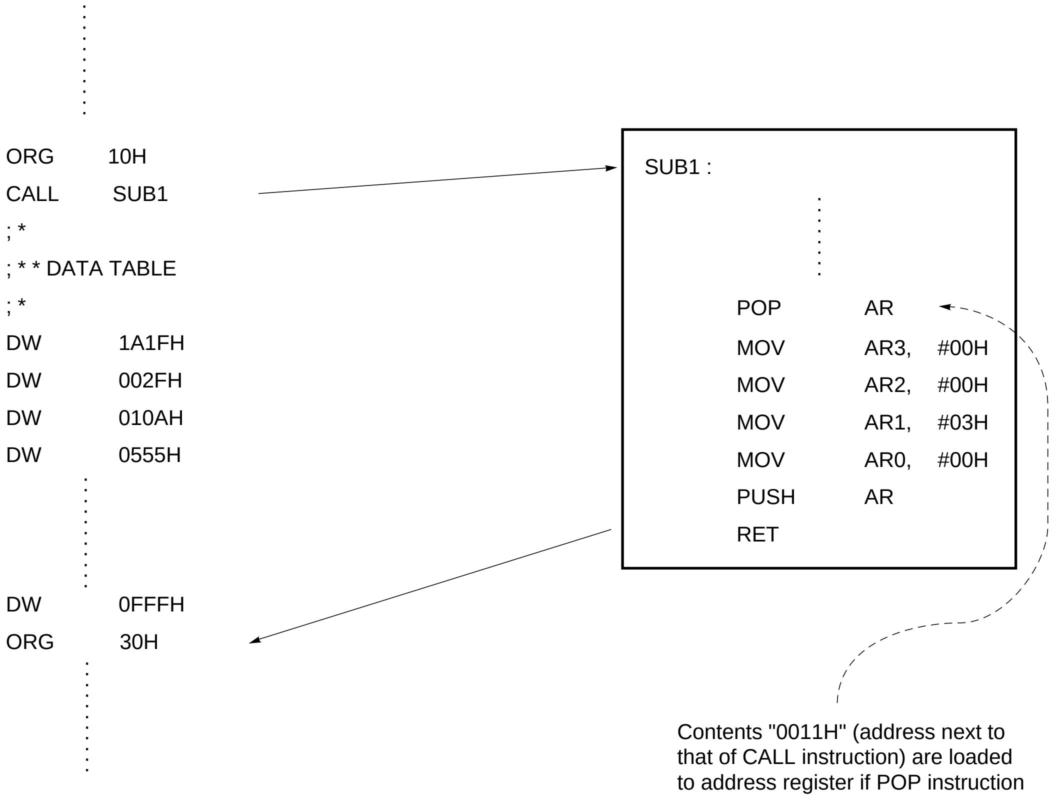
To set 003FH in the address register and store it in stack.

```
MOV     AR3, #00H
MOV     AR2, #00H
MOV     AR1, #03H
MOV     AR0, #0FH
PUSH    AR
```



Example 2

To set the return address of a subroutine in the address register and return if there is a data table following the CALL instruction.



```
        :
ORG     10H
CALL    SUB1
; *
; * * DATA TABLE
; *
DW      1A1FH
DW      002FH
DW      010AH
DW      0555H
        :
DW      0FFFH
ORG     30H
        :
```

```
SUB1 :
            :
      POP     AR
      MOV     AR3,  #00H
      MOV     AR2,  #00H
      MOV     AR1,  #03H
      MOV     AR0,  #00H
      PUSH    AR
      RET
```

Contents "0011H" (address next to that of CALL instruction) are loaded to address register if POP instruction is executed at this point.

(8) POP AR — Pop address register

<1> OP code

00111	000	1100	0000

<2> Function

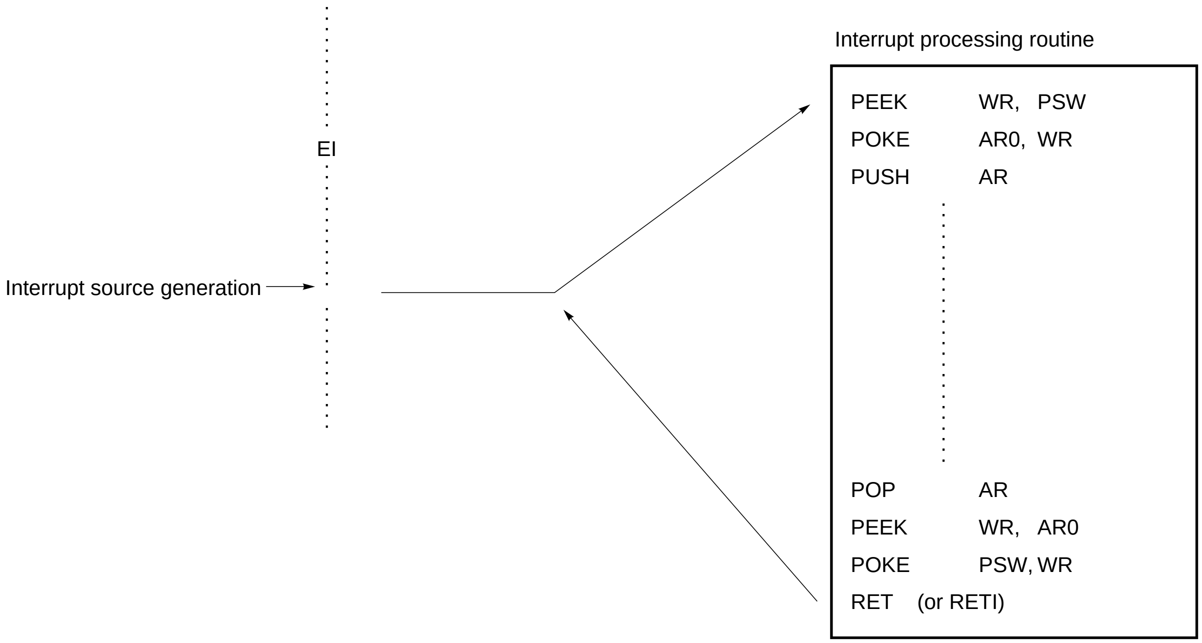
AR ← ASR

SP ← SP+1

Loads the contents of the address stack register specified by the stack pointer to address register AR, and then increments the value of stack pointer SP.

<3> Example

If the PSW contents have been changed in an interrupt processing routine when interrupt processing is performed and you want to transfer the PSW contents to the address register through WR, you should save them at the beginning of the interrupt processing, to the address stack register by the PUSH instruction. Then, restore them to the address register by the POP instruction before return, and transfer them to PSW through WR.



```
PEEK        WR,  PSW
POKE        AR0, WR
PUSH        AR
          :
          :
POP         AR
PEEK        WR,  AR0
POKE        PSW, WR
RET   (or RETI)
```

(9) PEEK WR, rf — Peek register file to window register

<1> OP code

00111	rf_R	0011	rf_C

<2> Function

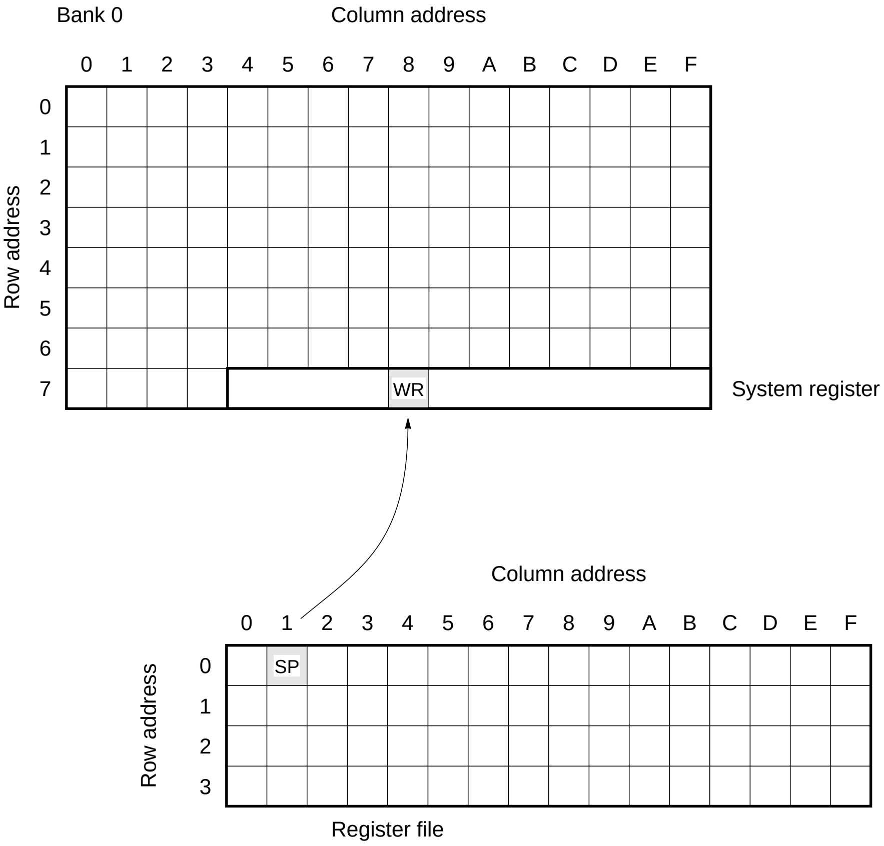
WR ← (rf)

Stores the register file contents to window register WR.

<3> Example

To store the stack pointer contents (SP) 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 |  rfR  |  0010  |  rfC  |
```

<2> Function

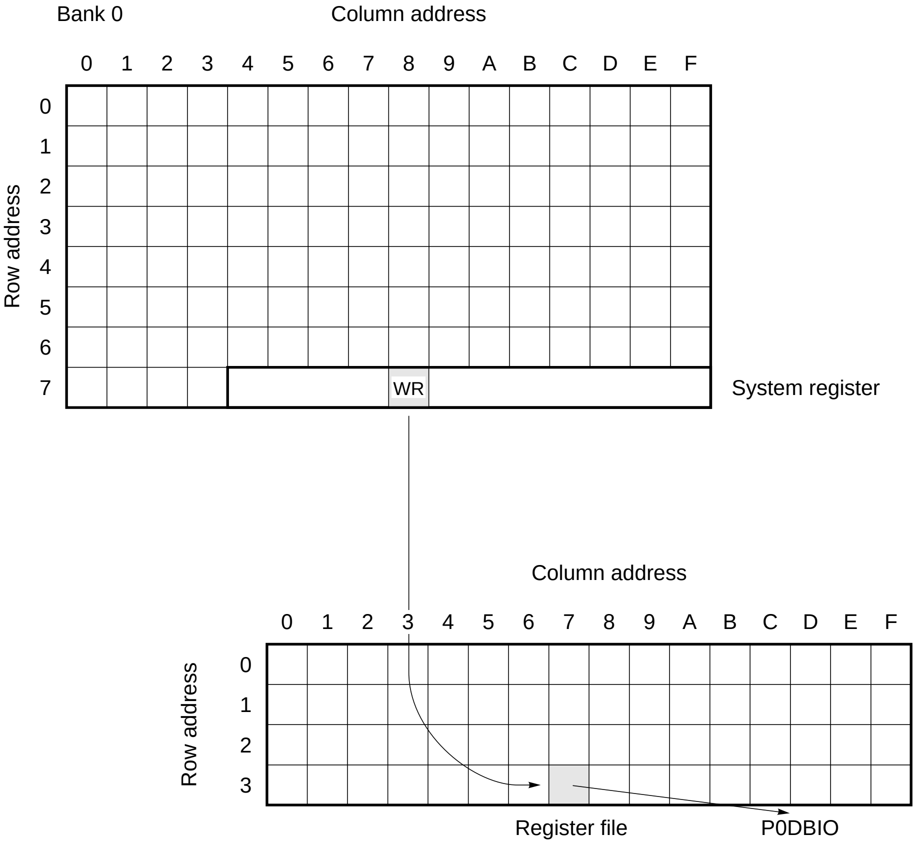
(rf) ← WR

Stores the window register WR contents to the register file.

<3> Example

To store immediate data 0FH in P0DBIO of the register file through the window register.

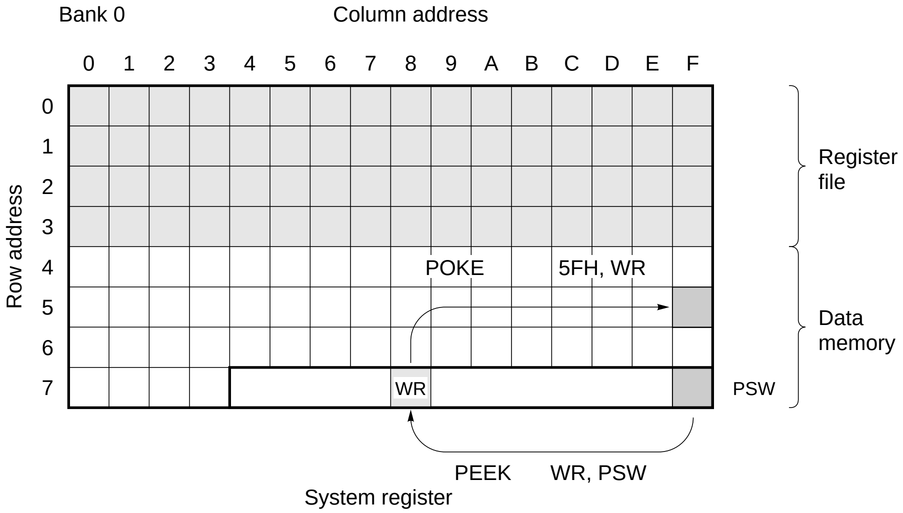
```
MOV     WR, #0FH
POKE    P0DBIO, WR          ; Sets all P0D0, P0D1, P0D2, and P0D3 in output mode
```



<4> Precaution

Addresses 40H through 7FH in the register file appear in data memory in a program. Consequently, the PEEK and POKE instructions can access addresses 40H through 7FH of each bank of the data memory in addition to the register file. For example, these instructions can also be used as follows:

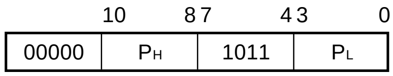
```
MEM05F    MEM   0.5FH
          PEEK  WR, PSW          ; Stores PSW (7FH) contents in system register to WR
          POKE  MEM05F, WR       ; Stores WR contents to data memory at address 5FH
```



(11) GET DBF, p

Get peripheral data to data buffer

<1> OP code



<2> Function

DBF ← (p)

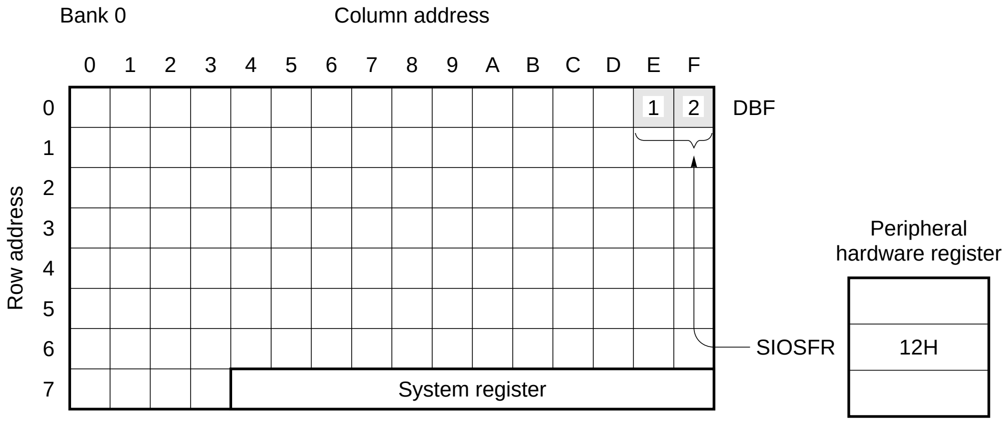
Stores the peripheral circuit contents to data buffer DBF.

DBF is a 16-bit area of addresses 0CH through 0FH of BANK0 of the data memory regardless of the value of the bank register.

<3> Example 1

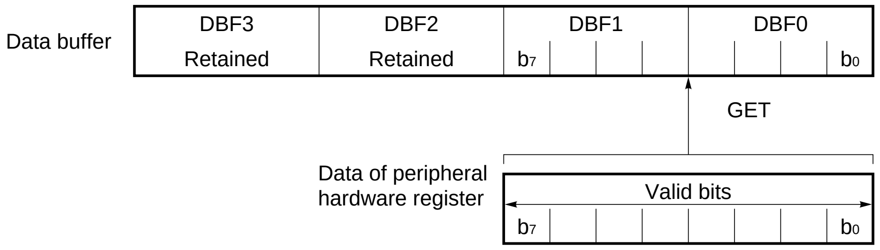
To store the 8-bit contents of the shift register SIOSFR of the serial interface in data buffers DBF0 and DBF1.

GET DBF, SIOSFR



<4> Precaution

The data buffer is 16 bits wide. The number of bits differs depending on the peripheral hardware to be accessed. For example, when the GET instruction is executed to a peripheral hardware register whose valid bit length is 8 bits, data is stored in the low-order 8 bits of 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_M	1010	P_L

<2> Function

(p) ← DBF

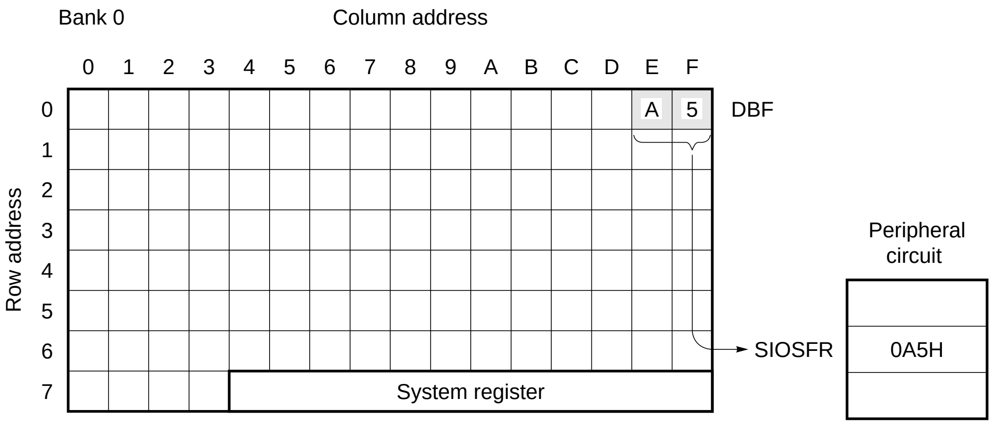
Stores the data buffer DBF contents in the peripheral register.

DBF is a 16-bit area of addresses 0CH through 0FH of BANK0 of the data memory regardless of the value of the bank register.

<3> Example 1

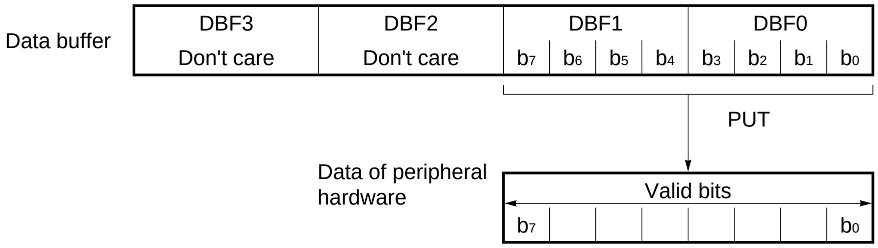
To set 0AH and 05H in data buffers DBF1 and DBF0, respectively, and transfer them to the shift register (SIOSFR) for serial interface.

```
MOV     BANK, #00H          ; Data memory bank 0
MOV     DBF0, #05H
MOV     DBF1, #0AH
PUT     SIOSFR, DBF
```



<4> Precaution

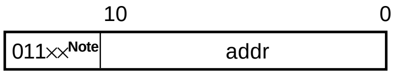
The data buffer is 16 bits wide. The number of bits differs depending on the peripheral hardware to be accessed. For example, when the PUT instruction is executed to the peripheral hardware register whose valid bit length is 8-bit, the low-order 8 bits data of data buffer DBF (DBF1, DBF0) is transferred to a peripheral hardware register (DBF3 and DBF2 data is not transferred).



15.5.8 Branch instructions

(1) BR addr **Branch to the address**

<1> OP code



Note Refer to **<4> Precaution**

<2> Function

$PC_{10-0} \leftarrow addr$

Branches to an address specified by addr.

<3> Example

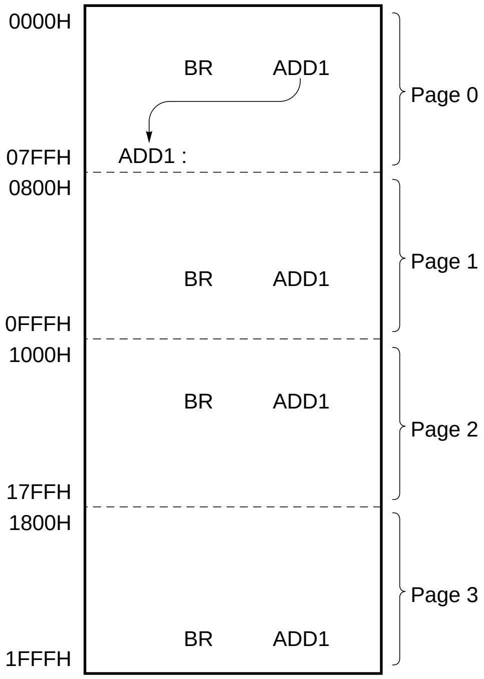
```
FLY         LAB    0FH     ; Defines FLY = 0FH
                :
                :
            BR     FLY     ; Jumps to address 0F
                :
                :
            BR     LOOP1   ; Jumps to LOOP1
                :
                :
            BR     $ + 2   ; Jumps to address two addresses  lower than the current address
                :
            BR     $ – 3   ; Jumps to address three addresses higher than the current address
                :
                :
LOOP1:
```

<4> Precation

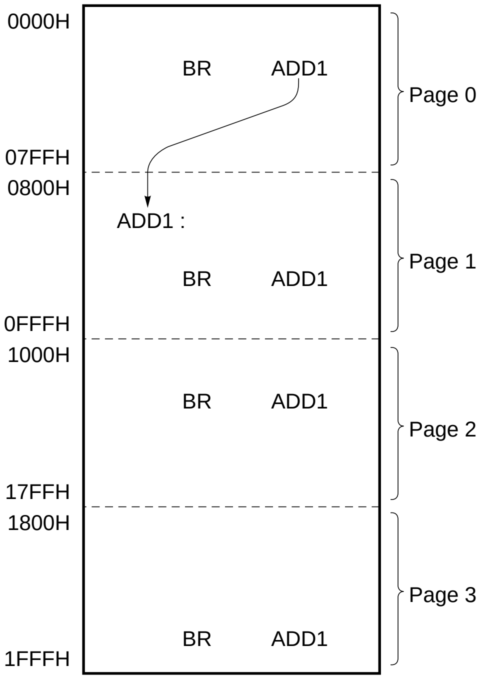
The BR instruction does not use the division of “page”, and the instruction can be written in the ROM addresses 0000H-1FFFH. However, the BR instruction branching within page 0 (addresses 0000H-07FFH) and the BR instruction branching in page 1 (07FFH-0FFFH), and the BR instruction branching in page 2 (1000H-17FFH) and the BR instruction branching in page 3 (17FFH-1FFFH) differ in OP code. The OP codes are 0C in page 0, 0D in page 1, 0E in page 2, and 0F in page 3.

If these instructions are assembled with the 17K-series assembler, the jump destination is automatically referenced.

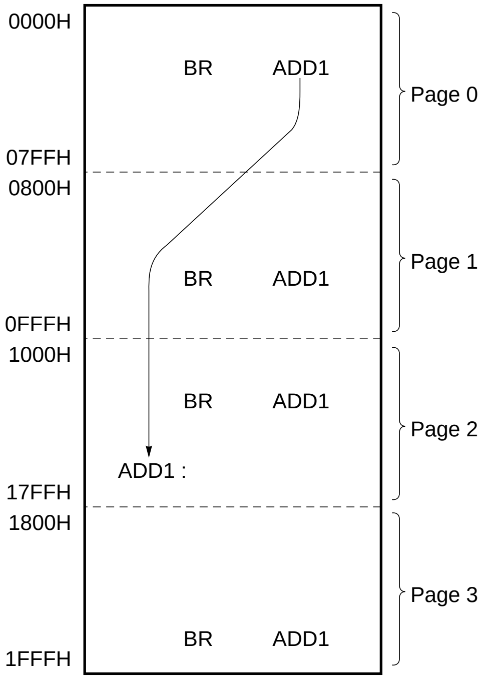
If OP code is 0C
(if jump destination address is in page 0)



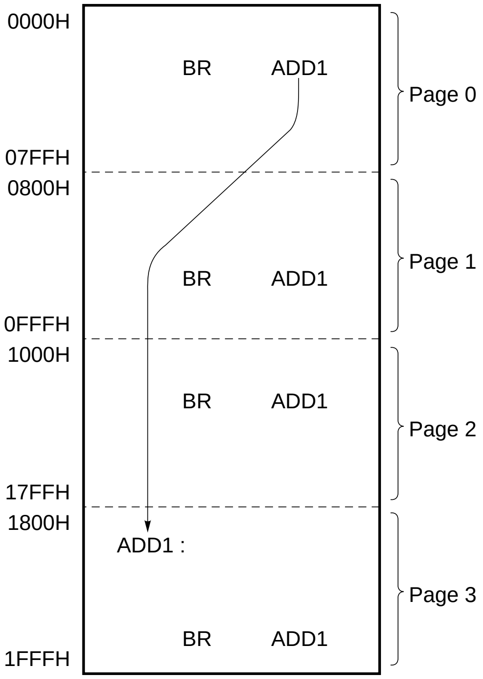
If OP code is 0D
(if jump destination address is in page 1)



If OP code is 0E
(if jump destination address is in page 2)

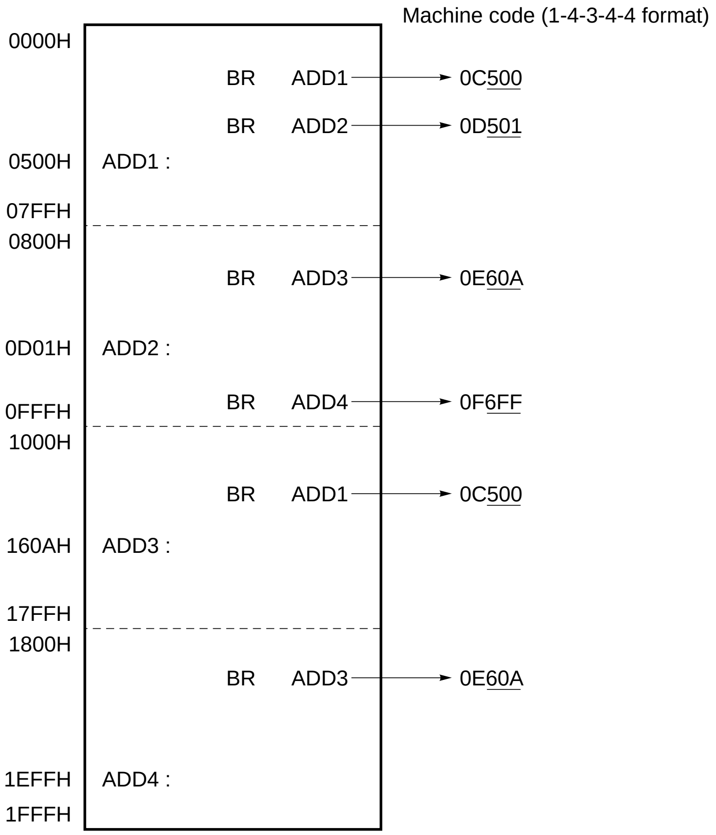


If OP code is 0F
(if jump destination address is in page 3)



To perform patch correction during debugging, it is necessary for the programmer to convert 0C, 0D, 0E, and 0F.

Address conversion is also necessary when the jump destination of the BR instructions are in addresses 0000H-07FFH, 0800H-0FFFH, 1000H-17FFH, and 1800H-1FFFH. In other words, addresses 0000H, 0800H, 1000H, and 1800H are treated as address 000H, starting from which the subsequent addresses are incremented by 1.



Caution The number of pages of each model in the μPD172×× subseries differs. For details, refer to the Data Sheet of your device.

(2) BR @AR — Branch to the address specified by address register

<1> OP code

00111	000	0100	0000

<2> Function

PC ← AR

Branches to a program address specified by address register AR.

<3> Example 1

To set 003FH in the address registers AR (AR0-AR3) and jump to address 003FH by 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

To change the branch destination as follows according to the contents of data memory address 0.10H.

```
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
;*
ORG     10H
BR      AAA
BR      BBB
BR      CCC
BR      DDD
BR      EEE
BR      FFF
BR      GGG
BR      HHH
BR      ZZZ
```

```
                        :
                        :
                        :
MEM010   MEM     0.10H
         MOV     AR3,  #00H      ; AR3 ← 00H  001 x H in AR
         MOV     AR2,  #00H      ; AR2 ← 00H
         MOV     AR1,  #01H      ; AR1 ← 01H
         MOV     RPH,  #00H      ; General register bank 0
         MOV     RPL,  #02H      ; General register row address 1
         ST      AR0,  MEM010    ; AR0 ← 0.10H
         SKLT    AR0,  #08H
         MOV     AR0,  #08H      ; Sets 08H in AR0 if AR0 contents are greater than 08H
         BR      @AR
```

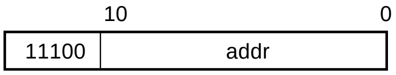
<4> Precaution

The number of bits of the address registers (AR3, AR2, AR1, and AR0) differs depending on the model. Refer to the Data Sheet of your device.

15.5.9 Subroutine instructions

(1) CALL addr **Call subroutine**

<1> OP code



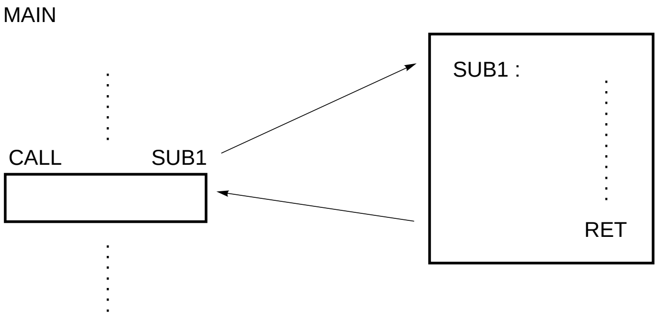
<2> Function

SP ← SP – 1, ASR ← PC,

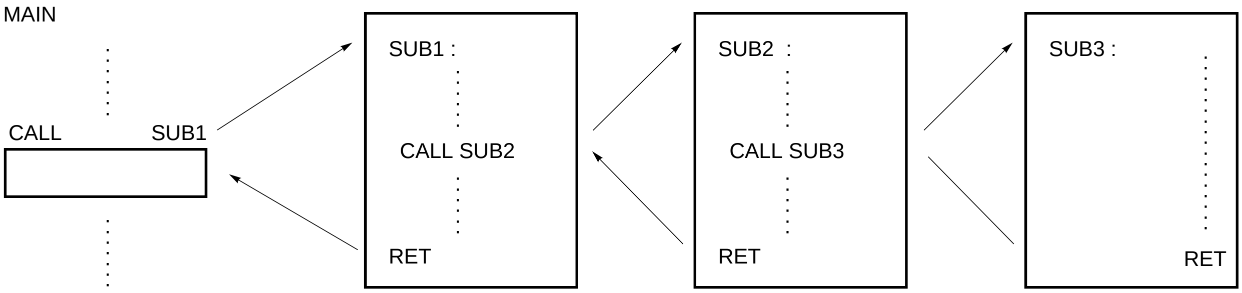
PC_{10-0} ← addr, PAGE ← 0

Increments the value of the program counter (PC), saves it in the stack, and branches to the subroutine whose address is 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 the value of the program counter (PC), saves it in the stack, and branches to the subroutine that starts from the address specified by the address register (AR).

<3> Example 1

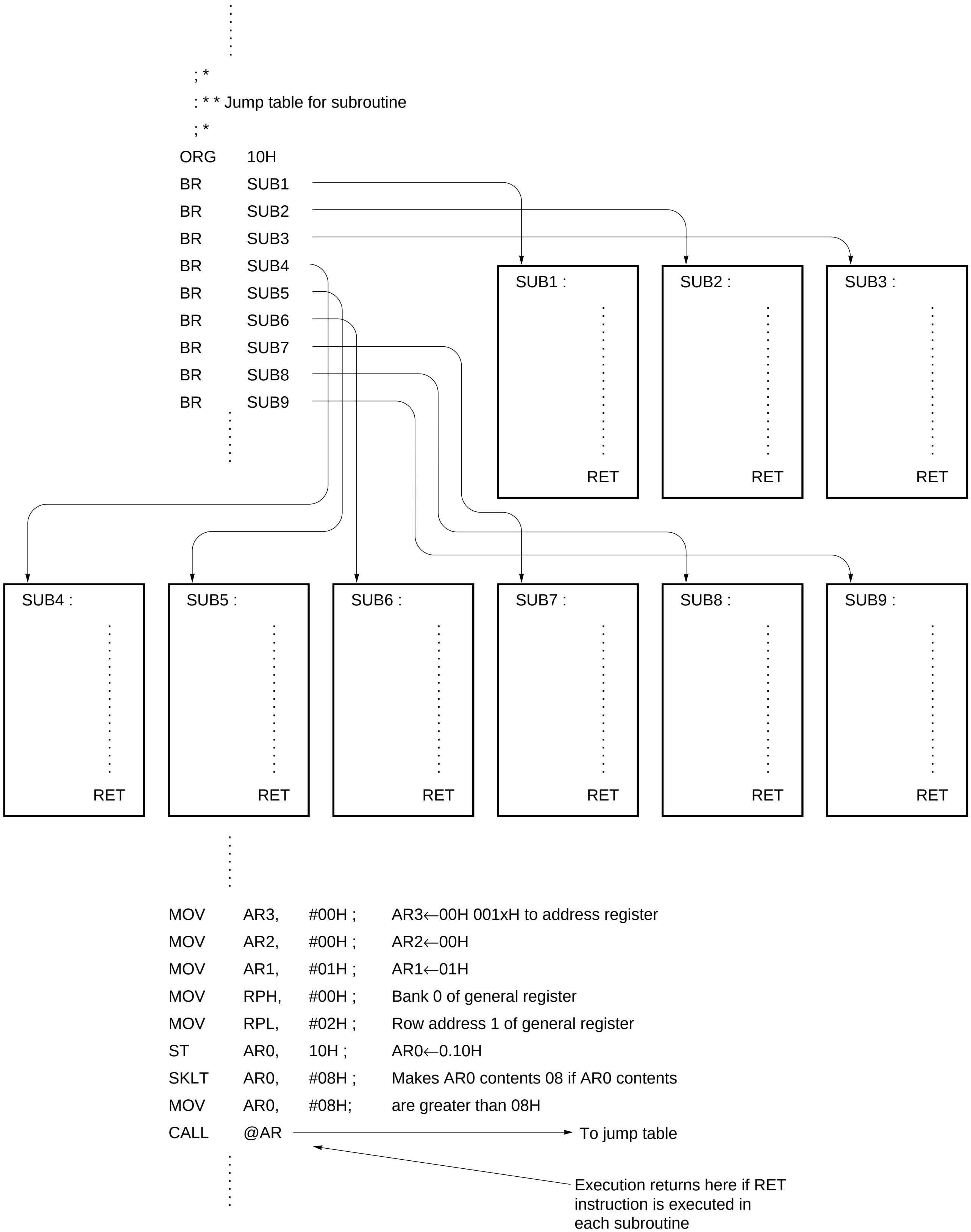
To set 0020H in address register AR (AR0-AR3) and call the subroutine at address 0020H by the CALL @AR instruction.

```
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

To call the following subroutine by the data memory address 0.10H contents:

0.10H contents		Subroutine name
00H	→	SUB1
01H	→	SUB2
02H	→	SUB3
03H	→	SUB4
04H	→	SUB5
05H	→	SUB6
06H	→	SUB7
07H	→	SUB8
08H-0FH	→	SUB9



```
          :
          :
    ; *
    : * * Jump table for subroutine
    ; *
ORG     10H
BR      SUB1
BR      SUB2
BR      SUB3
BR      SUB4
BR      SUB5
BR      SUB6
BR      SUB7
BR      SUB8
BR      SUB9
          :
          :
```

```
          :
          :
MOV     AR3,    #00H ;    AR3←00H 001xH to address register
MOV     AR2,    #00H ;    AR2←00H
MOV     AR1,    #01H ;    AR1←01H
MOV     RPH,    #00H ;    Bank 0 of general register
MOV     RPL,    #02H ;    Row address 1 of general register
ST      AR0,    10H ;     AR0←0.10H
SKLT    AR0,    #08H ;    Makes AR0 contents 08 if AR0 contents
MOV     AR0,    #08H;     are greater than 08H
CALL    @AR
          :
          :
```

To jump table

Execution returns here if RET
instruction is executed in
each subroutine

<4> Precaution

The number of bits of the address registers (AR3, AR2, AR1, and AR0) that can be used differs depending on the model of the device. For details, refer to the manual of your device.

(3) RET **Return to the main program from subroutine**

<1> OP code

10 8 7 4 3 0

00111	000	1110	0000

<2> Function

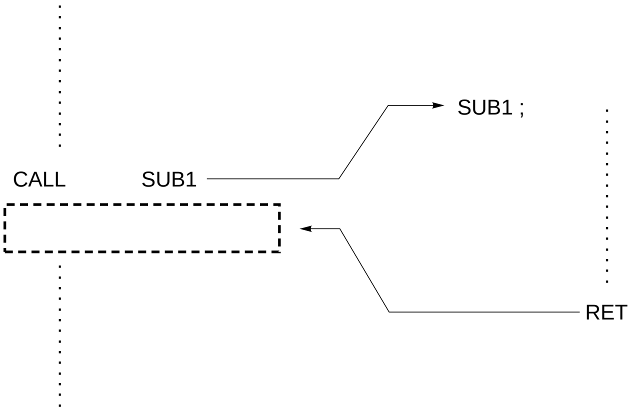
PC ← ASR,

SP ← SP + 1

Returns execution from a subroutine to the main program.

Restores the return address, saved by the CALL instruction to the stack, 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

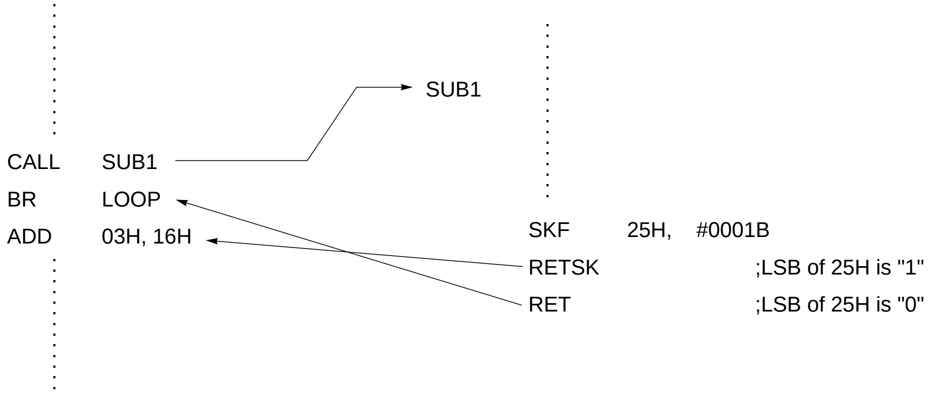
Returns execution from a subroutine to the main program.

Skips the instruction next to the CALL instruction (Executes as NOP instruction).

Restores the return address, saved by the CALL instruction to the stack, to the program counter PC, and then increments the program counter contents.

<3> Example

To execute the RET instruction and return the execution to the instruction next to the CALL instruction if the LSB (least significant bit) at address 25H of the data memory (RAM) is 0; if the LSB is 1, to execute the RETSK instruction to return the execution to the instruction after the next to the CALL instruction (ADD 03H, 16H in this example).



```
  :                                              :
  :                           SUB1               :
  :                                              :
CALL    SUB1                                     :
BR      LOOP
ADD     03H, 16H                              SKF      25H,   #0001B
  :                                           RETSK            ;LSB of 25H is "1"
  :                                           RET              ;LSB of 25H is "0"
  :
```

(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

Returns execution from an interrupt processing program to the main program.

Restores to the program counter the return address which was saved in the stack by a vectored interrupt.

A part of the system registers is also restored to the states before the occurrence of the vectored interrupt.

<3> Precaution 1

The contents of the system register are automatically saved by an interrupt (which can be restored by the RETI instruction) are the PSWORD.

Precaution 2

If the RETI instruction is used in the place of the RET instruction to return from an ordinary subroutine, the bank contents (which were saved when the interrupt has occurred) may be replaced with the contents of the interrupt stack. Consequently, probably the bank contents are unknown to the user. To avoid this, be sure to use the RET (or RETSK) instruction to return from a subroutine.

15.5.10 Interrupt instructions

(1) EI **Enable Interrupt**

<1> OP code

00111	000	1111	0000

<2> Function

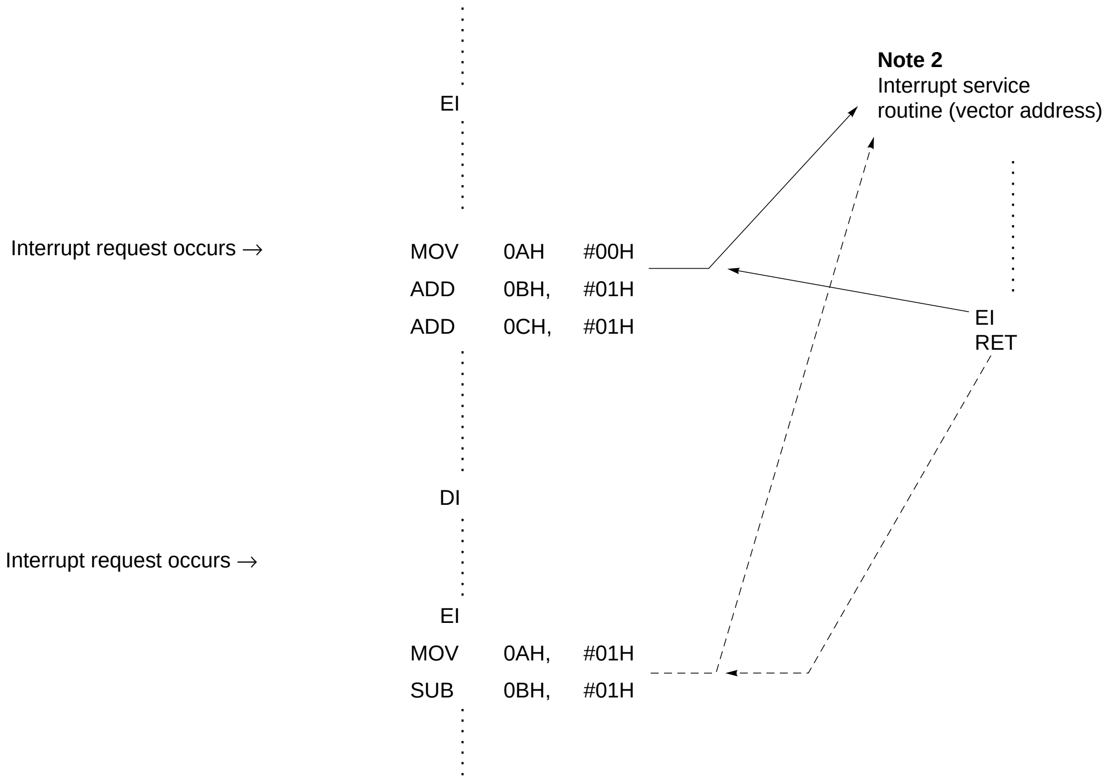
INTEF ← 1

Enables the 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 next instruction (except the instruction that manipulates the program counter) has been executed, and then the execution flow shifts to a vector address[Note1].

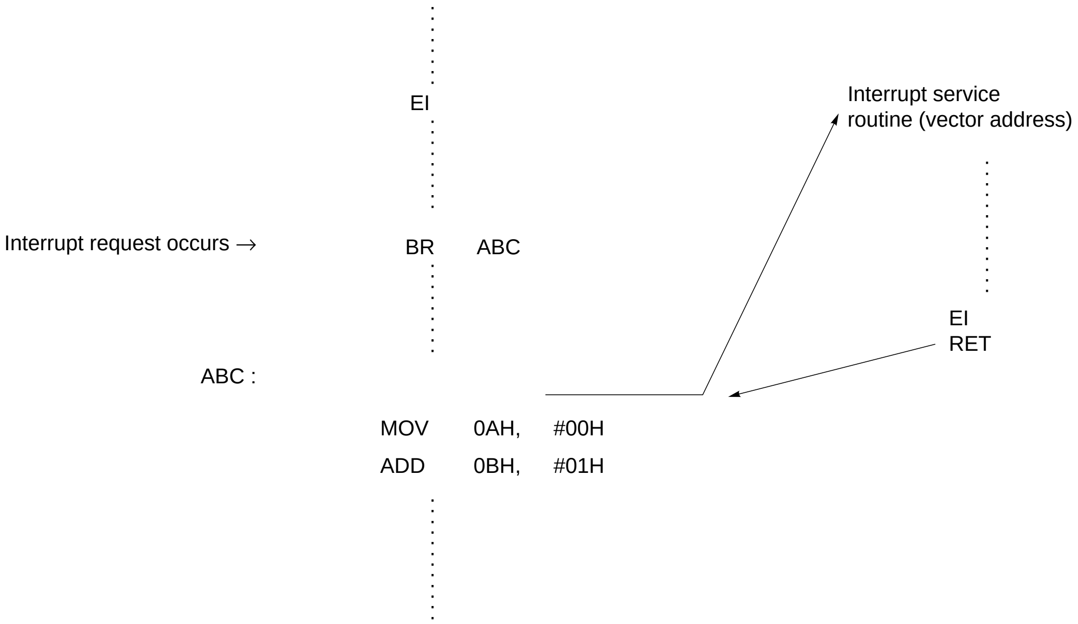


Notes 1. The vector address differs depending on the interrupt accepted.

2. The interrupt accepted (an interrupt request occurs after the execution of the EI instruction and the execution flow shifts to an interrupt service routine) is the interrupt whose interrupt enable flag (IP×××) is set. The flow of the program is not changed (i.e., the interrupt is not accepted) even if an interrupt request occurs after the EI instruction has been executed with the interrupt enable flag of each interrupt not set. However, the interrupt request flag (IRQ×××) is set. The interrupt is therefore accepted at the point where the interrupt enable flag is set.

Example 2

An example of an interrupt that is caused by an interrupt request that has been accepted while an instruction that manipulates the program counter is executed as follows.



(2) DI **Disable interrupt**

<1> OP code

00111	001	1111	0000

<2> Function

INTEF ← 0

Disables the vectored interrupt.

<3> Example

Refer to **Example 1** in (1) EI.

15.5.11 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 sets the device in the STOP mode.

By setting the device in the STOP mode, the current consumption of the device can be minimized.

The condition, under which the STOP mode is released, is specified by the operand (s).

For the condition, under which the STOP mode is released, refer to **12.2 Setting and Releasing STOP Mode**.

(2) HALT h **Halt CPU and release by condition h**

<1> OP code

			3　　0
00111	011	1111	h

<2> Function

Sets the device in the HALT mode.

By setting the device in the HALT mode, the current consumption of the device can be reduced.

The condition, under which the HALT mode is released, is specified by the operand (h).

For the condition, under which the HALT mode is released, refer to **12.3 Setting and Releasing HALT Mode**.

(3) NOP **No operation**

<1> OP code

00111	100	1111	0000

<2> Function

Executes nothing but consumes one machine cycle.

[MEMO]

APPENDIX A DEVELOPMENT TOOLS

★ A.1 Hardware List

Target Device		In-circuit Emulator	SE Board	Emulation Probe	Conversion Socket/ Adapter[Note 1]
Part Number	Package				
μPD17201AGF	80-pin QFP	IE-17K IE-17K-ET EMU-17K[Note 2]	SE-17207	EP-17201GF	EV-9200G-80
μPD17203AGC	52-pin QFP		SE-17204	EP-17203GC	EV-9200G-52
μPD17204GC	52-pin QFP				
μPD17207GF	80-pin QFP		SE-17207	EP-17201GF	EV-9200G-80
μPD17225CT μPD17225GT	28-pin SDIP 28-pin SOP		SE-17225	EP-17K28CT EP-17K28GT	EV-9500GT-28 (for EP-17K28GT)
μPD17226CT μPD17226GT					
μPD17227CT μPD17227GT					
μPD17228CT μPD17228GT					

Notes 1. EV-9200××–×× is a conversion socket, and EV-9500××–×× is a flexible board.

2. Manufactured by Naito Densei Machida Mfg. Co., Ltd. Host machine is compatible with PC-9800 series only. For details, contact Naito Densei (TEL 044-822-3813).

★ A.2 Software List

- **PC-9800 series (Japanese Windows$^{\text{TM}}$)**

Device	Assembler	C Compiler	Device File	*SIMPLEHOST*
17201A	μSAA13RA17K	μSAA13CC17K	μSAA13AS17201	μSAA13ID17K
17203A			μSAA13AS17203	
17204			μSAA13AS17204	
17207			μSAA13AS17207	
17225 17226 17227 17228			μSAA13AS17225	

- **IBM PC/AT$^{\text{TM}}$ (Japanese Windows$^{\text{TM}}$)**

Device	Assembler	C Compiler	Device File	*SIMPLEHOST*
17201A	μSAB13RA17K	μSAB13CC17K	μSAB13AS17201	μSAB13ID17K
17203A			μSAB13AS17203	
17204			μSAB13AS17204	
17207			μSAB13AS17207	
17225 17226 17227 17228			μSAB13AS17225	

- **IBM PC/AT (English Windows$^{\text{TM}}$)**

Device	Assembler	C Compiler	Device File	*SIMPLEHOST*
17201A	μSBB13RA17K	μSBB13CC17K	μSBB13AS17201	μSBB13ID17K
17203A			μSBB13AS17203	
17204			μSBB13AS17204	
17207			μSBB13AS17207	
17225 17226 17227 17228			μSBB13AS17225	

★ A.3 PROM Programmers

Target PROM	Program adapter	PROM Programmer
μPD17P203AGC	AF-9808B	AF-9703 AF-9704 AF-9705 AF-9706
μPD17P204GC		
μPD17P207GF	AF-9808A	
μPD17P218CT	AF-9808J	
μPD17P218GT	AF-9808H	

Remark The PROM programmer and program adapter for the 17K series are produced by Ando Electric Co.

[MEMO]

APPENDIX B HOW TO ORDER THE MASK ROM

After you have developed your program, place your order for a mask ROM as follows:

(1) Reservation for ordering mask ROM

Inform NEC in advance when you need the mask ROM; otherwise, the mask ROM may not be delivered in time to meet your needs.

(2) Creating ordering medium

The medium in which the mask ROM is ordered is a UV-EPROM.

★ First, create a hex file (with extension characters .PRO) for ordering the mask ROM by adding assemble option /PROM of the assembler (RA17K).

Next, write the hex file for ordering the mask ROM in the UV-EPROM.

When ordering with a UV-EPROM, create three UV-EPROM, all having identical contents.

Caution You cannot order a mask ROM by creating a hex file with .ICE.

(3) Creating necessary documents

Fill out the following forms, when ordering for the mask ROM:

- Mask ROM ordering sheet
- Mask ROM ordering check sheet

(4) Ordering

Submit the medium created in (2) and documents created in (3) to NEC by the deadline date for ordering.

★ **Remark** For details, refer to **ROM Code Ordering Procedure (IEM-1366)**.

[MEMO]

APPENDIX C INSTRUCTION INDEX

C.1 Instruction Index (by function)

[Addition]

ADD r, m 138
ADD m, #n4 141
ADDC r, m 143
ADDC m, #n4 146
INC AR 147
INC IX 149

[Subtraction]

SUB r, m 150
SUB m, #n4 152
SUBC r, m 154
SUBC m, #n4 156

[Logical]

OR r, m 158
OR m, #n4 159
AND r, m 160
AND m, #n4 161
XOR r, m 162
XOR m, #n4 163

[Test]

SKT m, #n 164
SKF m, #n 165

[Compare]

SKE m, #n 166
SKNE m, #n 167
SKGE m, #n 168
SKLT m, #n 168

[Rotate]

RORC r 169

[Transfer]

LD r, m 170
ST m, r 172
MOV @r, m 174
MOV m, @r 175
MOV m, #n4 177
MOVT DBF, @AR 177
PUSH AR 179
POP AR 181
PEEK WR, rf 182
POKE rf, WR 183
GET DBF, p 185
PUT p, DBF 186

[Branch]

BR addr 187
BR @AR 190

[Subroutine]

CALL addr 192
CALL @AR 193
RET 195
RETSK 195
RETI 196

[Interrupt]

EI 197
DI 198

[Others]

STOP s 199
HALT h 199
NOP 199

C.2 Instruction Index (by alphabetic order)

[A]

ADD m, #n4 141
ADD r, m 138
ADDC m, #n4 146
ADDC r, m 143
AND m, #n4 161
AND r, m 160

[B]

BR addr 187
BR @AR 190

[C]

CALL addr 192
CALL @AR 193

[D]

DI 198

[E]

EI 197

[G]

GET DBF, p 185

[H]

HALT h 199

[I]

INC AR 147
INC IX 149

[L]

LD r, m 170

[M]

MOV m, #n4 177
MOV m, @r 175
MOV @r, m 174
MOVT DBF, @AR 177

[N]

NOP 199

[O]

OR m, #n4 159
OR r, m 158

[P]

PEEK WR, rf 182
POKE rf, WR 183
POP AR 181
PUSH AR 179
PUT p, DBF 186

[R]

RET 195
RETI 196
RETSK 195
RORC r 169

[S]

SKE m, #n 166
SKF m, #n 165
SKGE m, #n4 168
SKLT m, #n4 168
SKNE m, #n4 167
SKT m, #n 164
ST m, r 172
STOP s 199
SUB m, #n4 152
SUB r, m 150
SUBC m, #n4 156
SUBC r, m 154

[X]

XOR m, #n4 162
XOR r, m 163

★

APPENDIX D REVISION HISTORY

A history of the revisions up to this edition is shown below. "Applied to:" indicates the chapters to which the revision was applied.

Edition	Major Revisions from the Previous Version	Applied to:
2nd	μPD17211, 17215, 17216, and 17P218 added	Throughout
3rd	μPD17217 and 17218 added μPD17211 deleted μPD17P218 under development → developed	Throughout
	9.2.3 Table reference Caution added to the table reference instruction	CHAPTER 9 DATA BUFFER (DBF)
	Caution added to CHAPTER 11 INTERRUPT FUNCTION	CHAPTER 11 INTERRUPT FUNCTION
	14.3 How to Write Program Memory and 14.4 How to Read Program Memory modified	CHAPTER 14 WRITING AND VERIFYING ONE-TIME PROM
4th	μPD17225, 17226, 17227, and 17228 added	Throughout
	μPD17202A, 17215, 17216, 17217, 17218 and 17P202A deleted	Throughout
	Assembler changed (AS17K → RA17K)	Throughout
	Changes in **1.1 List of Functions**	CHAPTER 1 GENERAL
	Extension instruction added to the table in **15.4 Assembler (RA17K) Macro** instruction	CHAPTER 15 INSTRUCTION SET
	A.1 Hardware List modified	APPENDIX A DEVELOPMENT TOOLS
	A.2 Software List modified	APPENDIX A DEVELOPMENT TOOLS

[MEMO]

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288
1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6465-6829

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: 02-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: 044-548-7900

I would like to report the following error/make the following suggestion:

Document title: ______________________

Document number: ______________________ Page number: ____________

____

____

____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❏	❏	❏	❏
Technical Accuracy	❏	❏	❏	❏
Organization	❏	❏	❏	❏