USER'S MANUAL

NEC

μPD17134A SUBSERIES

4-BIT SINGLE-CHIP MICROCONTROLLER

μPD17134A
μPD17135A
μPD17136A
μPD17137A
μPD17P136A
μPD17P137A

Document No. U11607EJ3V0UM00 (3rd edition)
Date Published December 1996 N
Printed in Japan

© NEC Corporation 1993

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.

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: 800-366-9782
Fax: 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: 253-8311
Fax: 250-3583

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

NEC do Brasil S.A.
Sao Paulo-SP, Brasil
Tel: 011-889-1680
Fax: 011-889-1689

***SIMPLEHOST* is a trademark of NEC Corp.**
MS-DOS and Windows are trademarks of Microsoft Corp.
PC/AT and PC DOS are trademarks of IBM Corp.

The export of this product from Japan is prohibited without governmental license. To export or re-export 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

Major Revisions in This Edition

Page	Description
Throughout	Change of name μPD1713XA to μPD17134A subseries
p. 5	Correction of **(2) Program memory write/verify mode** in **1.4 PIN CONFIGURATION**
p. 18	Change of **Figure 3-2 Value of Program Counter after Instruction** **Partial** correction of **3.2.2 On Execution of Branch Instruction (BR)**
p. 19	Partial correction of **3.2.3 On During Execution of Subroutine Call**
p. 23	Change of **CHAPTER 4 PROGRAM MEMORY (ROM)**
p. 31	Partial correction of **Figure 5-1 Data Memory Configuration**
p. 35	Change of **CHAPTER 6 STACK**
p. 43	Partial correction of **7.2.2 Address Register Functions**
p. 47	Change of **7.5 INDEX REGISTER (IX) AND DATA MEMORY ROW ADDRESS POINTER (MEMORY POINTER: MP)**
p. 58	Partial change of **7.6.2 Functions of General Register Pointer**
p. 59	Partial change of **7.7.1 Program Status Word Configuration**
p. 61	Change of **7.7.4 Zero Flag (Z) and Compare Flag (CMP)**
p. 61	Partial correction of **7.7.5 Carry Flag (CY)**
p. 71	Partial correction of **9.2.3 Register File Manipulation Instructions**
p. 111	Change of **CHAPTER 13 PERIPHERAL HARDWARE**
p. 149	Change of **CHAPTER 14 INTERRUPT FUNCTIONS**
p. 169	Change of **CHAPTER 16 STANDBY FUNCTION**
p. 179	Change of **CHAPTER 17 RESET**
p. 190	Partial change of **Table 18-2 Differences between Mask ROM Version and One-Time PROM Version**
p. 194	Partial change of **19.3 LIST OF THE INSTRUCTION SET**
p. 198	Partial change of **19.5 INSTRUCTIONS**
p. 255	Change of **CHAPTER 20 ASSEMBLER RESERVED WORDS**
p. 257	Partial change of **20.2 RESERVED SYMBOLS**
p. 261	Addition of **APPENDIX A DEVELOPMENT OF μPD171×× SUBSERIES**
p. 263	Addition of **APPENDIX B COMPARISON OF FUNCTIONS BETWEEN μPD17135A, 17137A, AND μPD17145 SUBSERIES**
p. 267	Addition of **APPENDIX D NOTES ON CONFIGURATION OF SYSTEM CLOCK OSCILLATION CIRCUIT**

The mark ★ shows major revisions made in this edition.

PREFACE

Target : This manual is intended for user engineers who understand the functions of each product in the μPD17134A subseries and try to design application systems using the μPD17134A subseries.

Purpose : The purpose of this manual is for the user to understand the hardware functions of the μPD17134A subseries.

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

- **To understand the functions of the μPD17134A subseries in a general way;**
  → Read the manual from **CONTENTS**.

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

- **To look up an instruction when you do not know its mnemonic but know outlines of the function;**
  → Refer to **19.3 LIST OF THE INSTRUCTION SET** for search for the mnemonic of the instruction, then see **19.5 INSTRUCTIONS** for the functions.

- **To learn the electrical specifications of the μPD17134A subseries**
  → Refer to the Data Sheet available separately.

- **To learn the application examples of the functions of the μPD17134A subseries**
  → Refer to the Application Note available separately.

Legend :

Data representation weight	:	High-order and low-order digits are indicated from left to right.
Active low representation	:	$\overline{\times\times\times}$ (pin or signal name is overlined)
Memory map address	:	Top: low-order, bottom: high-order
Note	:	Explanation of **Note** in the text
Caution	:	Caution to which you should pay attention
Remark	:	Supplementary explanation to the text
Number representation	:	Binary number ...×××× or ××××B
		Decimal number ...××××
		Hexadecimal number ...××××H

Related Documents : The following documents are provided for the μPD17134A subseries.
The numbers listed in the table are the document numbers.

Product name / Document name	μPD17134A	μPD17135A	μPD17136A	μPD17137A	μPD17P136A	μPD17P137A
Brochure	IF-1166	IF-1169	IF-1166	IF-1169	IF-1168	IF-1165
Data sheet	U10591E	U10592E	U10591E	U10592E	IC-2871	IC-2872
User's manual	IEU-1369					
Application note	IEA-1297 (Introduction), IEA-1293 (Rice cooker, thermos bottle)					
IE-17K (Ver. 1.6) user's manual	EEU-1467					
IE-17K-ET (Ver. 1.6) user's manual	EEU-1466					
SE board user's manual	EEU-1379					
SIMPLEHOST™ user's manual	EEU-1336 (Introduction), EEU-1337 (Reference)					
AS17K assembler user's manual	EEU-1287					
Device file user's manual	U10777E					

Pin name and symbol name should be read according to the system clock type.

System clock / Pin name, symbol name	RC oscillation (μPD17134A, μPD17136A, μPD17P136A)	Ceramic oscillation (μPD17135A, μPD17137A, μPD17P137A)
Pin for system clock oscillation	OSC_1	X_{IN}
	OSC_0	X_{OUT}
System clock	f_{CC}	f_X

TABLE OF CONTENTS

CHAPTER 1 GENERAL DESCRIPTION 1

1.1 FUNCTION LIST 2
1.2 ORDERING INFORMATION 3
1.3 BLOCK DIAGRAM 4
1.4 PIN CONFIGURATION (TOP VIEW) 5

CHAPTER 2 PIN FUNCTIONS 9

2.1 PIN FUNCTIONS 9
2.2 PIN INPUT/OUTPUT CIRCUIT 11
2.3 PROCESSING OF UNUSED PINS 14
2.4 NOTES ON USING $\overline{\text{RESET}}$ PIN AND P1B$_0$ PIN 15

CHAPTER 3 PROGRAM COUNTER (PC) 17

3.1 PROGRAM COUNTER CONFIGURATION 17
3.2 PROGRAM COUNTER OPERATION 17
3.2.1 At Reset 18
3.2.2 During Execution of the Branch Instruction (BR) 18
3.2.3 During Execution of Subroutine Calls (CALL) 19
3.2.4 During Execution of Return Instructions (RET, RETSK, RETI) 20
3.2.5 During Table Reference (MOVT) 20
3.2.6 During Execution of Skip Instructions (SKE, SKGE, SKLT, SKNE, SKT, SKF) 21
3.2.7 When an Interrupt Is Received 21

CHAPTER 4 PROGRAM MEMORY (ROM) 23

4.1 PROGRAM MEMORY CONFIGURATION 23
4.2 PROGRAM MEMORY USAGE 24
4.2.1 Flow of the Program 24
4.2.2 Table Reference 27

CHAPTER 5 DATA MEMORY (RAM) 31

5.1 DATA MEMORY CONFIGURATION 31
5.1.1 System Register (SYSREG) 32
5.1.2 Data Buffer (DBF) 32
5.1.3 General Register (GR) 33
5.1.4 Port Registers 33
5.1.5 General Data Memory 34
5.1.6 Unmounted Data Memory 34

CHAPTER 6 STACK **35**

6.1 STACK CONFIGURATION **35**
6.2 FUNCTIONS OF THE STACK **35**
6.3 ADDRESS STACK REGISTERS (ASRs) **36**
6.4 INTERRUPT STACK REGISTERS (INTSKs) **36**
6.5 STACK POINTER (SP) AND INTERRUPT STACK REGISTERS **37**
6.6 STACK OPERATION **38**
6.6.1 On Execution of Instructions CALL, RET, RETSK 38
6.6.2 Table Reference (MOVT DBF, @AR Instruction) 38
6.6.3 Operation on Execution of Interrupt Receipt and RETI Instruction 39
6.7 STACK NESTING LEVELS AND THE PUSH AND POP INSTRUCTIONS **39**

CHAPTER 7 SYSTEM REGISTER (SYSREG) **41**

7.1 SYSTEM REGISTER CONFIGURATION **41**
7.2 ADDRESS REGISTER (AR) **43**
7.2.1 Address Register Configuration 43
7.2.2 Address Register Functions 43
7.3 WINDOW REGISTER (WR) **45**
7.3.1 Window Register Configuration 45
7.3.2 Window Register Functions 45
7.4 BANK REGISTER (BANK) **46**
7.4.1 Bank Register Configuration 46
7.4.2 Functions of Bank Register 46
7.5 INDEX REGISTER (IX) AND DATA MEMORY ROW ADDRESS POINTER (MEMORY POINTER: MP) **47**
7.5.1 Index Register (IX) 47
7.5.2 Data Memory Row Address Pointer (Memory Pointer: MP) 47
7.5.3 IXE = 0 and MPE = 0 (No Data Memory Modification) 49
7.5.4 IXE = 0 and MPE = 1 (Diagonal Indirect Data Transfer) 51
7.5.5 IXE = 1 and MPE = 0 (Index Modification) 53
7.6 GENERAL REGISTER POINTER (RP) **57**
7.6.1 General Register Pointer Configuration 57
7.6.2 Functions of the General Register Pointer 58
7.7 PROGRAM STATUS WORD (PSWORD) **59**
7.7.1 Program Status Word Configuration 59
7.7.2 Functions of the Program Status Word 60
7.7.3 Index Enable Flag (IXE) 61
7.7.4 Zero Flag (Z) and Compare Flag (CMP) 61
7.7.5 Carry Flag (CY) 61
7.7.6 Binary-Coded Decimal Flag (BCD) 62
7.7.7 Notes Concerning Use of Arithmetic Operations 62
7.8 NOTES CONCERNING USE OF THE SYSTEM REGISTER **63**
7.8.1 Reserved Words for the System Register 63
7.8.2 Handling of System Register Addresses Fixed at 0 65

CHAPTER 8 GENERAL REGISTER (GR) 67

8.1 GENERAL REGISTER CONFIGURATION 67
8.2 FUNCTIONS OF THE GENERAL REGISTER 67

CHAPTER 9 REGISTER FILE (RF) 69

9.1 REGISTER FILE CONFIGURATION 69
9.1.1 Configuration of the Register File 69
9.1.2 Relationship between the Register File and Data Memory 69
9.2 FUNCTIONS OF THE REGISTER FILE 70
9.2.1 Functions of the Register File 70
9.2.2 Functions of Control Register 70
9.2.3 Register File Manipulation Instructions 71
9.3 CONTROL REGISTER 72
9.4 NOTES CONCERNING USE OF THE REGISTER FILE 73
9.4.1 Notes Concerning Operation of the Control Register (Read-Only and Unused Registers) ... 73
9.4.2 Register File Symbol Definitions and Reserved Words 73

CHAPTER 10 DATA BUFFER (DBF) 77

10.1 DATA BUFFER CONFIGURATION 77
10.2 FUNCTIONS OF THE DATA BUFFER 78
10.2.1 Data Buffer and Peripheral Hardware 79
10.2.2 Data Transfer with Peripheral Hardware 80
10.2.3 Table Reference 81

CHAPTER 11 ARITHMETIC AND LOGIC UNIT (ALU) 83

11.1 ALU BLOCK CONFIGURATION 83
11.2 FUNCTIONS OF THE ALU BLOCK 83
11.2.1 Functions of the ALU 83
11.2.2 Functions of Temporary Registers A and B 88
11.2.3 Functions of the Status Flip-flop 88
11.2.4 Operations in 4-Bit Binary 89
11.2.5 Operations in BCD 89
11.2.6 Operations in the ALU Block 90
11.3 ARITHMETIC OPERATIONS (ADDITION AND SUBTRACTION IN 4-BIT BINARY AND BCD) 91
11.3.1 Addition and Subtraction When CMP = 0 and BCD = 0 91
11.3.2 Addition and Subtraction When CMP = 1 and BCD = 0 91
11.3.3 Addition and Subtraction When CMP = 0 and BCD = 1 92
11.3.4 Addition and Subtraction When CMP = 1 and BCD = 1 92
11.3.5 Notes Concerning Use of Arithmetic Operations 92
11.4 LOGICAL OPERATIONS 93
11.5 BIT JUDGEMENTS 94
11.5.1 TRUE (1) Bit Judgement 94
11.5.2 FALSE (0) Bit Judgement 95

11.6 COMPARISON JUDGEMENTS 96
11.6.1 “Equal to” Judgement 96
11.6.2 “Not Equal to” Judgement 97
11.6.3 “Greater Than or Equal to” Judgement 97
11.6.4 “Less Than” Judgement 98
11.7 ROTATIONS 99
11.7.1 Rotation to the Right 99
11.7.2 Rotation to the Left 100

CHAPTER 12 PORTS 101

12.1 PORT 0A ($P0A_0$, $P0A_1$, $P0A_2$, $P0A_3$) 101
12.2 PORT 0B ($P0B_0$, $P0B_1$, $P0B_2$, $P0B_3$) 102
12.3 PORT 0C ($P0C_0/ADC_0$, $P0C_1/ADC_1$, $P0C_2/ADC_2$, $P0C_3/ADC_3$) 103
12.4 PORT 0D ($P0D_0/\overline{SCK}$, $P0D_1$/SO, $P0D_2$/SI, $P0D_3/\overline{TM0OUT}$) 104
12.5 PORT 1A ($P1A_0$, $P1A_1$, $P1A_2$, $P1A_3$) 105
12.6 PORT 1B ($P1B_0$) 105
12.7 PORT CONTROL REGISTER 106
12.7.1 Input/Output Switching by Group I/O 106
12.7.2 Input/Output Switching by Bit I/O 107
12.7.3 Specifying Pull-Up Resistor Incorporation Using Software 109

CHAPTER 13 PERIPHERAL HARDWARE 111

13.1 8-BIT TIMERS/COUNTERS (TM0 and TM1) 111
13.1.1 8-Bit Timers/Counters Configuration 111
13.1.2 Operation of 8-Bit Timers/Counters 115
13.1.3 Selecting Count Pulse 115
13.1.4 Setting Count Value to Modulo Register 116
13.1.5 Reading Value of Count Register 117
13.1.6 Setting of Interval Time 118
13.1.7 Error of Interval Time 119
13.1.8 Timer 0 Output 121
13.2 BASIC INTERVAL TIMER (BTM) 122
13.2.1 Basic Interval Timer Configuration 122
13.2.2 Registers Controlling Basic Interval Timer 123
13.2.3 Operation of Basic Interval Timer 124
13.2.4 Watchdog Timer Function 125
13.3 A/D CONVERTER 128
13.3.1 A/D Converter Configuration 128
13.3.2 Functions of A/D Converter 129
13.3.3 Setting Values in the 8-bit Data Register (ADCR) 132
13.3.4 Reading Values from the 8-bit Data Register (ADCR) 133
13.3.5 A/D Converter Operation 134
13.4 SERIAL INTERFACE (SIO) 141
13.4.1 Functions of the Serial Interface 141
13.4.2 3-wire Serial Interface Operation Modes 143
13.4.3 Setting Values in the Shift Register 147
13.4.4 Reading Values from the Shift Register 148

CHAPTER 14 INTERRUPT FUNCTIONS 149

14.1 INTERRUPT SOURCE TYPES AND VECTOR ADDRESSES 150
14.2 HARDWARE COMPONENTS OF THE INTERRUPT CONTROL CIRCUIT 151
14.3 INTERRUPT SEQUENCE 158
14.3.1 Receiving an Interrupt 158
14.3.2 Return from the Interrupt Routine 159
14.3.3 Interrupt Accepting Timing 160
14.4 MULTI-INTERRUPT 163
14.5 PROGRAM EXAMPLE OF INTERRUPT 164

CHAPTER 15 AC ZERO CROSS DETECTION 167

CHAPTER 16 STANDBY FUNCTION 169

16.1 OVERVIEW OF THE STANDBY FUNCTION 169
16.2 HALT MODE 170
16.2.1 Setting HALT Mode 170
16.2.2 Start Address after HALT Mode Is Released 170
16.2.3 HALT Mode Setting Conditions 172
16.3 STOP MODE 174
16.3.1 Setting of STOP Mode 174
16.3.2 Start Address after STOP Mode Is Released 174
16.3.3 STOP Mode Setting Conditions 176

CHAPTER 17 RESET 179

17.1 RESET FUNCTION 180
17.2 RESETTING 181
17.3 POWER-ON/POWER-DOWN RESET FUNCTION 182
17.3.1 Conditions Required to Enable the Power-On Reset Function 182
17.3.2 Power-On Reset Function and Operation 183
17.3.3 Condition Required for Use of the Power-Down Reset Function 185
17.3.4 Power-Down Reset Function and Operation 185

CHAPTER 18 ONE-TIME PROM WRITING/VERIFYING 189

18.1 DIFFERENCES BETWEEN MASK ROM VERSION AND ONE-TIME PROM MODEL 189
18.2 OPERATION MODE WHEN PROGRAM MEMORY IS WRITTEN/VERIFIED 190
18.3 WRITING PROCEDURE OF PROGRAM MEMORY 191
18.4 READING PROCEDURE OF PROGRAM MEMORY 192

CHAPTER 19 INSTRUCTION SET 193

19.1 OVERVIEW OF THE INSTRUCTION SET 193
19.2 LEGEND 194
19.3 LIST OF THE INSTRUCTION SET 195
19.4 ASSEMBLER (AS17K) EMBEDDED MACRO INSTRUCTIONS 197

19.5 INSTRUCTIONS 198
19.5.1 Addition Instructions 198
19.5.2 Subtraction Instructions 209
19.5.3 Logical Operation Instructions 216
19.5.4 Judgment Instructions 221
19.5.5 Comparison Instructions 223
19.5.6 Rotation Instructions 226
19.5.7 Transfer Instructions 227
19.5.8 Branch Instructions 243
19.5.9 Subroutine Instructions 246
19.5.10 Interrupt Instructions 251
19.5.11 Other Instructions 253

CHAPTER 20 ASSEMBLER RESERVED WORDS 255

20.1 MASK OPTION DIRECTIVE 255
20.1.1 Specifying Mask Option 255
20.2 RESERVED SYMBOLS 257

APPENDIX A DEVELOPMENT OF μPD171×× SUBSERIES 261

APPENDIX B COMPARISON OF FUNCTIONS BETWEEN μPD17135A, 17137A, AND μPD17145 SUBSERIES 263

APPENDIX C DEVELOPMENT TOOLS 265

APPENDIX D NOTES ON CONFIGURATION OF SYSTEM CLOCK OSCILLATION CIRCUIT 267

APPENDIX E INSTRUCTION LIST 269

E.1 INSTRUCTION LIST (by function) 269
E.2 IINSTRUCTION LIST (alphabetical order) 270

APPENDIX F ORDERING MASK ROM 271

LIST OF FIGURES (1/3)

Figure No.	Title	Page
3-1	Program Counter	17
3-2	Value of the Program Counter after Instruction Execution	18
3-3	Value in the Program Counter after Reset	18
3-4	Value in the Program Counter during Execution of a BR addr Instruction	18
3-5	Value in the Program Counter during Execution of an Indirect Branch Instruction	19
3-6	Value in the Program Counter during Execution of a CALL addr	19
3-7	Value in the Program Counter during Execution of an Indirect Subroutine Call	20
3-8	Value in the Program Counter during Execution of a Return Instruction	20
4-1	Program Memory Map for the μPD17134A Subseries	23
4-2	CALL addr Instruction	26
4-3	Table Reference (MOVT DBF, @AR)	27
5-1	Data Memory Configuration	31
5-2	System Register Configuration	32
5-3	Data Buffer Configuration	32
5-4	General Register (GR) Configuration	33
5-5	Port Register Configuration	33
6-1	Stack Configuration	35
7-1	Allocation of System Register in Data Memory	41
7-2	System Register Configuration	42
7-3	Address Register Configuration	43
7-4	Address Register Used as a Peripheral Circuit	44
7-5	Window Register Configuration	45
7-6	Example of Window Register Operation	45
7-7	Bank Register Configuration	46
7-8	Index Register Configuration	47
7-9	Modification of Data Memory Address by Index Register and Memory Pointer	48
7-10	Operation Example When IXE = 0 and MPE = 0	50
7-11	Operation Example When IXE = 0 and MPE = 1	52
7-12	Operation Example When IXE = 1 and MPE = 0	54
7-13	Operation Example When IXE = 1 and MPE = 0	55
7-14	Operation Example When IXE = 1 and MPE = 0 (Array Processing)	56
7-15	General Register Pointer Configuration	57
7-16	General Register Configuration	58
7-17	Program Status Word Configuration	59
7-18	Outline of Functions of the Program Status Word	60
8-1	General Register Configuration	68

LIST OF FIGURES (2/3)

Figure No.	Title	Page
9-1	Register File Configuration	69
9-2	Relationship Between the Register File and Data Memory	70
9-3	Accessing the Register File Using the PEEK and POKE Instructions	72
9-4	Control Register Configuration	75
10-1	Allocation of the Data Buffer	77
10-2	Data Buffer Configuration	77
10-3	Relationship Between the Data Buffer and Peripheral Hardware	78
11-1	ALU Configuration	84
12-1	Input/Output Switching by Group I/O	106
12-2	Port Control Register of Bit I/O	107
12-3	Specifying Pull-Up Resistor Incorporation Using Software	109
13-1	Configuration of the 8-Bit Timer Counters	112
13-2	Timer 0 Mode Register	113
13-3	Timer 1 Mode Register	114
13-4	Setting Count Value to Modulo Register	116
13-5	Reading Count Value of Count Register	117
13-6	Error When Count Register Is Cleared to 0 During Counting	119
13-7	Error When Counting Is Started from Count Stop Status	120
13-8	Timer 0 Output Setting Register	121
13-9	Basic Interval Timer Configuration	122
13-10	BTM Mode Register	123
13-11	Watchdog Timer Mode Register	124
13-12	Timing Chart of Watchdog Timer (with WDTRES Flag Used)	126
13-13	Block Diagram of the A/D Converter	128
13-14	A/D Converter Control Register	130
13-15	Setting a Value in the 8-Bit Data Register (ADCR)	132
13-16	Reading Values from the 8-bit Data Register (ADCR)	133
13-17	Relationship between the Analog Input Voltage and Digital Conversion Result	134
13-18	Using the Successive Mode for the A/D Converter	136
13-19	A/D Conversion Timing in the Continuous Mode	137
13-20	Using the Single Mode for the A/D Converter	139
13-21	Single Mode Operation (Comparison) Timing	140
13-22	Block Diagram of the Serial Interface	142
13-23	Timing of 8-Bit Transmission and Reception Mode (Simultaneous Transmission and Reception)	143
13-24	Timing of the Clock Synchronization 8-Bit Reception Mode (SO Pin Output High Impedance)	144
13-25	Serial Interface Control Register	145
13-26	Setting a Value in the Shift Register	147
13-27	Reading a Value from the Shift Register	148

LIST OF FIGURES (3/3)

Figure No.	Title	Page
14-1	Interrupt Control Register	152
14-2	Interrupt Processing Procedure	158
14-3	Return from Interrupt Processing	159
14-4	Interrupt Accepting Timing (When INTE = 1, IP××× = 1)	160
14-5	Example of Multi-interrupt	163
15-1	Block Diagram for the AC Zero Cross Detector	167
15-2	Zero Cross Detection Signal	168
16-1	Releasing HALT Mode	171
16-2	Releasing STOP Mode	175
17-1	Reset Block Configuration	181
17-2	Reset Operation	181
17-3	Example of the Power-On Reset Operation	184
17-4	Example of the Power-Down Reset Operation	186
17-5	Example of Reset Operation during the Period from Power-Down Reset to Power Recovery	187
18-1	Procedure of Program Memory Writing	191
18-2	Procedure of Program Memory Reading	192
D-1	External Circuit of System Clock Oscillation Circuit	267
D-2	Example of Incorrect Oscillation Circuits	268

LIST OF TABLES (1/2)

Table No.	Title	Page
2-1	Processing of Unused Pins	14
4-1	Program Memory Configuration	23
4-2	Vector Address for the μPD17134A Subseries	24
6-1	Operation of Stack Pointer	37
6-2	Operation of the Instructions CALL, RET, and RETSK	38
6-3	Stack Operation during Table Reference	38
6-4	Operation during Interrupt Receipt and RETI Instruction	39
6-5	Stack Operation during the PUSH and POP Instructions	39
7-1	Specifying the Bank in Data Memory	46
7-2	Instructions Subject to Address Modification	48
7-3	Zero Flag (Z) and Compare Flag (CMP)	61
10-1	Peripheral Hardware	79
11-1	List of ALU Instructions	86
11-2	Results of Arithmetic Operations Performed in 4-Bit Binary and BCD	89
11-3	Types of Arithmetic Operations	91
11-4	Logical Operations	93
11-5	Table of True Values for Logical Operations	93
11-6	Bit Judgement Instructions	94
11-7	Comparison Judgement Instructions	96
12-1	Writing into and Reading from the Port Register (0.70H)	101
12-2	Writing into and Reading from the Port Register (0.71H)	102
12-3	Switching the Port and A/D Converter	103
12-4	Register File Contents and Pin Functions	104
12-5	Contents Read from the Port Register (0.73H)	105
12-6	Writing into and Reading from the Port Register (1.70H)	105
13-1	Data Conversion Time for the A/D Converter	138
13-2	Serial Clock List	141
13-3	Operating Mode of the Serial Interface	143
14-1	Interrupt Source Types	150
14-2	Interrupt Request Flag and Interrupt Enable Flag	151
16-1	Status in Standby Mode	169
16-2	HALT Mode Release Condition	170
16-3	Start Address after HALT Mode Is Released	170
16-4	STOP Mode Release Condition	174
16-5	Start Address after STOP Mode Is Released	174

LIST OF TABLES (2/2)

Table No.	Title	Page
17-1	Hardware Status at Reset	180
18-1	Pins Used for Writing/Verifying Program Memory	189
18-2	Differences Between Mask ROM Version and One-Time PROM Version	190
18-3	Setting Operation Modes	190
20-1	Mask Option Definition Directive	256

[MEMO]

CHAPTER 1 GENERAL DESCRIPTION

The μPD17134A subseries is a 4-bit single-chip microcontroller employing the 17K architecture and containing an 8-bit A/D converter (4 channels), a timer (3 channels), an AC zero cross detector, a power-on reset circuit, and a serial interface.

The μPD17P136A and 17P137A are the one-time PROM version of the μPD17136A and 17137A, respectively, and are suitable for program evaluation at system development and for small-scale production.

The following are features of the μPD17134A subseries.

- 17K architecture: general-purpose register mode, instruction length: fixed to 16 bits
- Instruction execution time: 2 μs (f_X = 8 MHz, ceramic oscillation)
  8 μs (f_{CC} = 2 MHz, RC oscillation)
- Program memory: μPD17134A : 2K bytes (1024 × 16 bits)
  μPD17135A : 2K bytes (1024 × 16 bits)
  μPD17136A : 4K bytes (2048 × 16 bits)
  μPD17137A : 4K bytes (2048 × 16 bits)
  μPD17P136A : 4K bytes (2048 × 16 bits, one-time PROM)
  μPD17P137A : 4K bytes (2048 × 16 bits, one-time PROM)
- Data memory (RAM): 112 × 4 bits
- A/D converter: 4 channels (8-bit resolution, successive approximation type)
- Timer: 3 channels (8-bit timer/counter × 2 channels, basic interval timer[Note])
- Serial interface: 1 channel (clocked 3-wire mode)
- Supply voltage: V_{DD} = 4.5 to 5.5 V (f_X = 400 kHz to 8 MHz)
  V_{DD} = 2.7 to 5.5 V (f_X = 400 kHz to 4 MHz)
  V_{DD} = 2.7 to 5.5 V (f_{CC} = 400 kHz to 2 MHz) for μPD17134A and 17136A

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

These features of the μPD17134A subseries are suitable for use as a controller or a slave device in the following application fields;

- Electronic thermos bottle
- Rice cooker
- Audio equipment
- Battery charger
- Printer
- Plain Paper Copier

★ 1.1 FUNCTION LIST

Item	μPD17134A	μPD17135A	μPD17136A	μPD17137A	μPD17P136A	μPD17P137A
ROM configuration	Mask ROM				One-time PROM	
ROM capacity	2KB (1024 × 16 bits)		4KB (2048 × 16 bits)			
RAM capacity	112 × 4 bits					
Stack	Address stack × 5, interrupt stack × 3					
Number of I/O port	22 { • I/O : 20 • Input only : 1 • Sensor input[Note] : 1					
A/D converter	8-bit resolution × 4 channels (shared with port pin), absolute precision ± 1.5 LSB or less					
Timer	3 channels { • 8-bit timer counter : 2 channels (16-bit timer 1 channel applicable) • 7-bit basic interval timer : 1 channel (watchdog timer applicable)					
Serial interface	1 channel (3 wires)					
AC zero cross detection function	Provided (can be used in application circuit at V_{DD} = 5 V ± 10%)					
Interrupt	• Nesting by hardware (up to 3 levels) • External interrupts (INT) : 1 { Rising edge detection, Falling edge detection, Both rising and falling edges detection } Selectable • Internal interrupts : 1 { • Timer 0 (TM0) • Timer 1 (TM1) • Basic interval timer (BTM) • Serial interface (SIO)					
System clock	RC oscillation	Ceramic oscillation	RC oscillation	Ceramic oscillation	RC oscillation	Ceramic oscillation
Instruction execution time	8 μs at f_X = 2 MHz	2 μs at f_X = 8 MHz	8 μs at f_X = 2 MHz	2 μs at f_X = 8 MHz	8 μs at f_X = 2 MHz	2 μs at f_X = 8 MHz
Standby	HALT, STOP					
Power-on/ power-down reset	Available (effective only for application circuit with V_{DD} = 5 V ± 10 %, 400 kHz to 4 MHz)					
Supply voltage	V_{DD} = 2.7 to 5.5 V (5 V ±10 % when using A/D converter)					
Package	28-pin plastic shrink DIP, 28-pin plastic SOP					

Note The INT pin can be used as an input pin (sense input) when the external interrupt function is not used. The sense input function is to read the status of the pin by using the INT flag of a control register, instead of a port register.

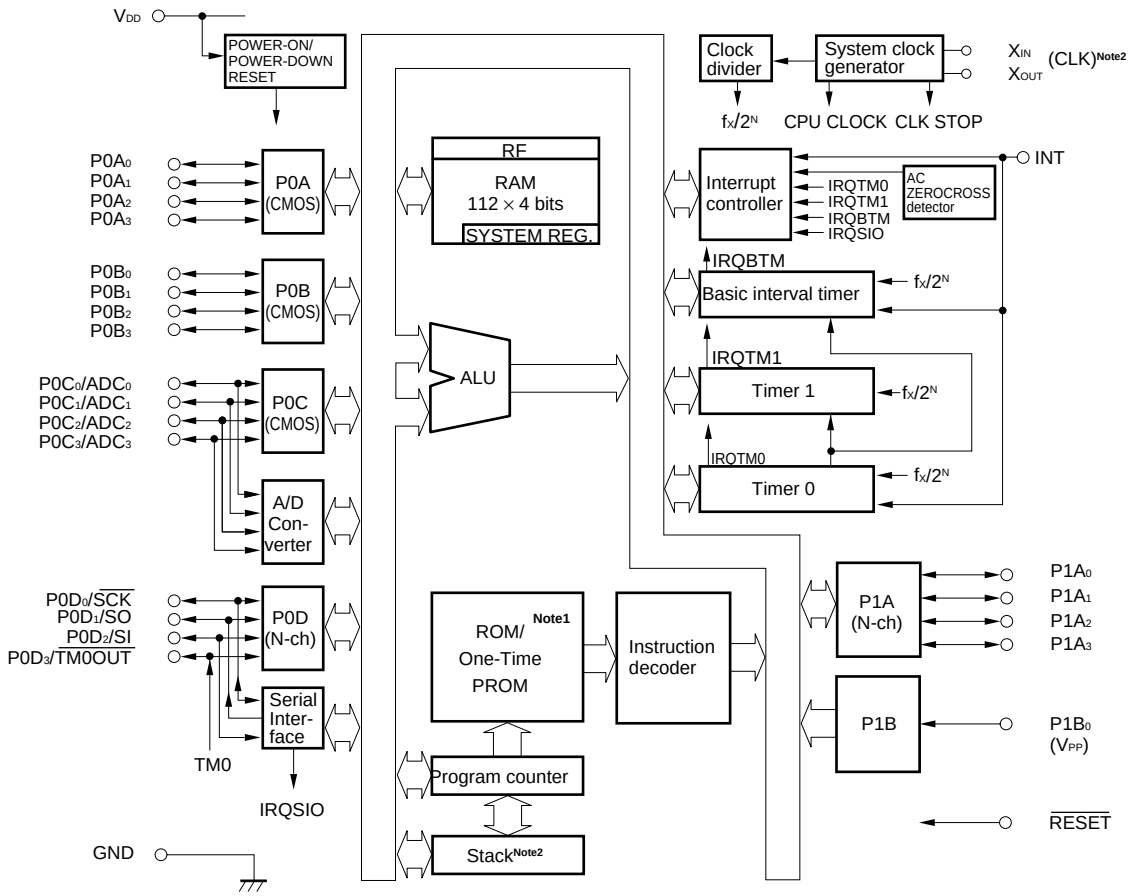
Caution The PROM model is highly compatible with the mask ROM model in terms of functions but its internal ROM circuit and electrical characteristics are partially different from those of the mask ROM model. To replace the PROM model with the mask ROM model, thoroughly evaluate the application by using a sample of the mask ROM model.

1.2 ORDERING INFORMATION

Part number	Package	Internal ROM
μPD17134ACT-xxx	28-pin plastic shrink DIP (400 mil)	Mask ROM
μPD17135ACT-xxx	28-pin plastic shrink DIP (400 mil)	Mask ROM
μPD17136ACT-xxx	28-pin plastic shrink DIP (400 mil)	Mask ROM
μPD17137ACT-xxx	28-pin plastic shrink DIP (400 mil)	Mask ROM
μPD17P136ACT	28-pin plastic shrink DIP (400 mil)	One-time PROM
μPD17P137ACT	28-pin plastic shrink DIP (400 mil)	One-time PROM
μPD17134AGT-xxx	28-pin plastic SOP (375 mil)	Mask ROM
μPD17135AGT-xxx	28-pin plastic SOP (375 mil)	Mask ROM
μPD17136AGT-xxx	28-pin plastic SOP (375 mil)	Mask ROM
μPD17137AGT-xxx	28-pin plastic SOP (375 mil)	Mask ROM
μPD17P136AGT	28-pin plastic SOP (375 mil)	One-time PROM
μPD17P137AGT	28-pin plastic SOP (375 mil)	One-time PROM

Remark xxx: ROM code number

1.3 BLOCK DIAGRAM



Remarks 1. The terms CMOS and N-ch in square brackets indicate the output form of the port.

CMOS : CMOS push-pull output

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

2. The devices in parentheses are effective only in the case of program memory write/verify mode of the μPD17P136A and μPD17P137A.

Notes 1. The ROM (or PROM) capacity of each product is as follows:

1024 × 16 bits : μPD17134A, 17135A

2048 × 16 bits : μPD17136A, 17137A, 17P136A, 17P137A

2. The stack capacity of each product is as follows:

5 × 10 bits : μPD17134A, 17135A

5 × 11 bits : μPD17136A, 17137A

1.4 PIN CONFIGURATION (TOP VIEW)

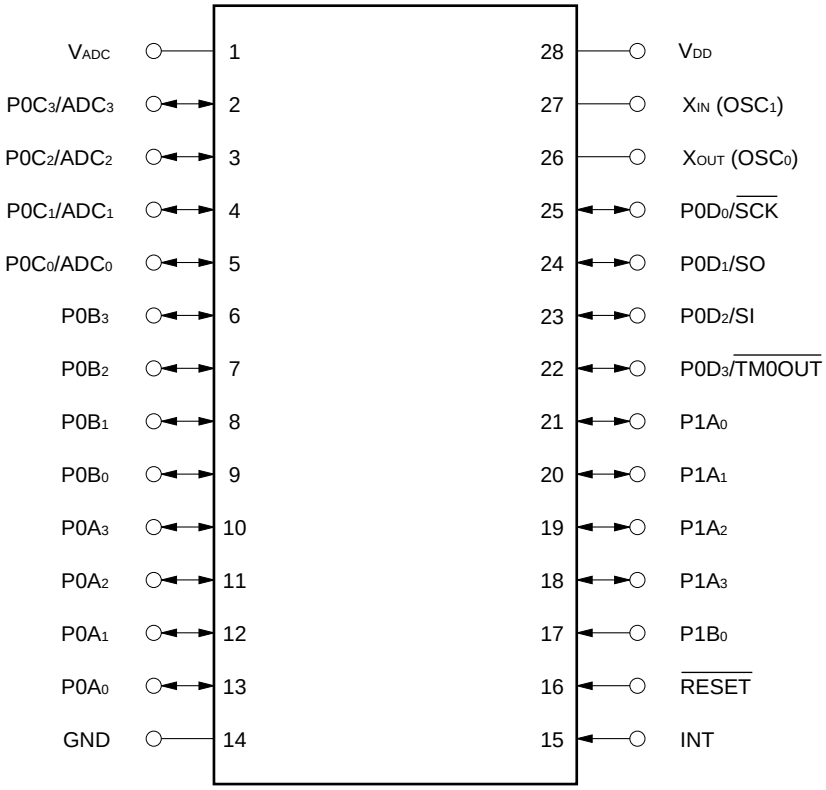
(1) Normal operating mode

28-pin plastic shrink DIP (400 mil)

μPD17134ACT-xxx, μPD17135ACT-xxx, μPD17136ACT-xxx, μPD17137ACT-xxx
μPD17P136ACT-xxx, μPD17P137ACT-xxx

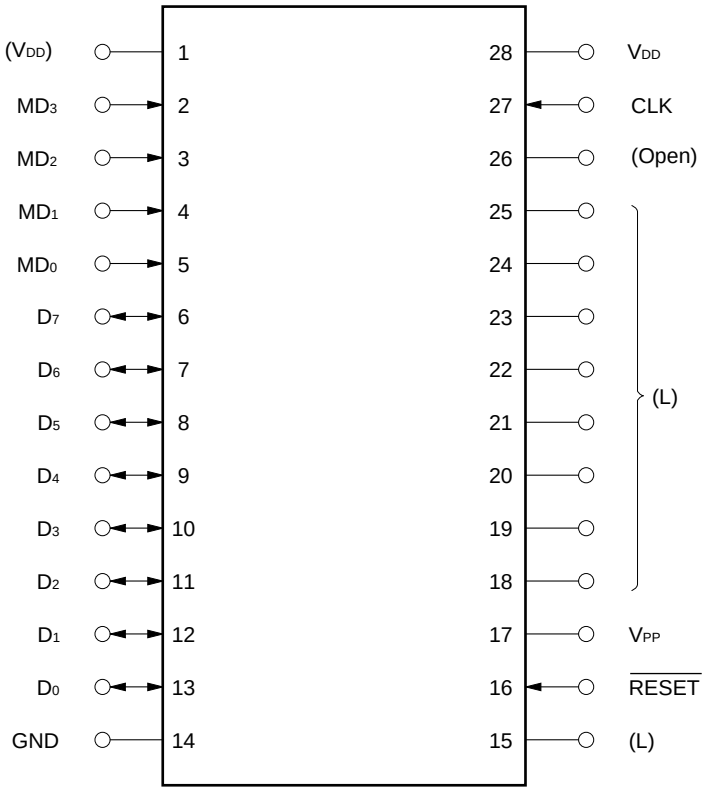
28-pin plastic SOP (375 mil)

μPD17134AGT-xxx, μPD17135AGT-xxx, μPD17136AGT-xxx, μPD17137AGT-xxx
μPD17P136AGT-xxx, μPD17P137AGT-xxx



ADC_0 to ADC_3	:	Analog input for the A/D converter
GND	:	Ground
INT	:	External interrupt input
OSC_0, OSC_1	:	System clock oscillation
$P0A_0$ to $P0A_3$	:	Port 0A
$P0B_0$ to $P0B_3$	:	Port 0B
$P0C_0$ to $P0C_3$	:	Port 0C
$P0D_0$ to $P0D_3$	:	Port 0D
$P1A_0$ to $P1A_3$	:	Port 1A
$P1B_0$	:	Port 1B
$\overline{RESET}$	:	Reset input
$\overline{SCK}$	:	Serial clock input/output
SI	:	Serial data input
SO	:	Serial data output
$\overline{TM0OUT}$	:	Timer 0 carry output
V_{ADC}	:	Analog power supply
V_{DD}	:	Power supply
X_{IN}, X_{OUT}	:	System clock oscillation

(2) Program memory write/verify mode
28-pin plastic shrink DIP (400 mil)
μPD17P136ACT, 17P137ACT
28-pin plastic SOP (375 mil)
μPD17P136AGT, 17P137AGT



Caution ( ) represents processing of the pins which are not used in program memory write/verify mode.

L : Connect to GND via pull-down resistor one by one.

$\overline{\text{RESET}}$: Set the same electric potential as V_{DD} in program memory write/verify mode. $\overline{\text{RESET}}$ pin is also used for system reset input before setting program memory write/verify mode. Therefore, $\overline{\text{RESET}}$ pin should be set to the same electric potential as V_{DD} 10 μs or later than that of V_{DD} pin (For details, refer to CHAPTER 18 ONE-TIME PROM WRITING/VERIFYING).

Open : Do not connect anything.

★ **V_{DD} : Connect to V_{DD} directly.**

CLK	: Clock input for address updating
D_0-D_7	: Data input/output
GND	: Ground
MD_0-MD_3	: Operation mode select
$\overline{RESET}$	: Reset input
V_{DD}	: Power supply
V_{PP}	: Program voltage application

[MEMO]

CHAPTER 2 PIN FUNCTIONS

2.1 PIN FUNCTIONS

Pin No.	Pin name	Function	Output	At reset
★ 1	V_{ADC}	Supplies power and reference voltage for the A/D converter	—	—
2 | 5	$P0C_3/ADC_3/MD_3$ Note1 | $P0C_0/ADC_0/MD_0$ Note1	Constitute port 0C, serve as analog input pins of A/D converter, or select operating mode when program memory is written or verified. ● $P0C_3$ to $P0C_0$ • 4-bit input/output port • Input/output setting in 1-bit unit ● ADC_3 to ADC_0 • Analog input for the A/D converter ● MD_3 to MD_0 • Available for the μPD17P136A and μPD17P137A only • Selects operating mode at program memory writing/ verification	CMOS push-pull	Input (P0C)
6 | 9	$P0B_3/D_7$ Note1 | $P0B_0/D_4$ Note1	Used as port 0B, or data input/output pins in program memory write/verify mode. ● $P0B_3$ to $P0B_0$ • 4-bit input/output port • Input/output setting in 4-bit unit • Software-selectable pull-up resistor ● D_7 to D_4 • Available for the μPD17P136A and μPD17P137A only • 8-bit data input/output at program memory writing/ verification	CMOS push-pull	Input (P0B)
10 | 13	$P0A_3/D_3$ Note1 | $P0A_0/D_0$ Note1	Used as port 0A, or data input/output pin in program memory write/verify mode. ● $P0A_3$ to $P0A_0$ • 4-bit input/output port • Input/output setting in 4-bit unit • Software-selectable pull-up resistor ● D_3 to D_0 • Available for the μPD17P136A and μPD17P137A only • 8-bit data input/output at program memory writing/ verification	CMOS push-pull	Input (P0A)
14	GND	Ground	—	—
15	INT	External interrupt request input or sensor signal input	—	Input
16	$\overline{RESET}$	System reset input pin A pull-up resistor can be internally connected by mask option Note2	—	Input

Notes 1. The MD_0-MD_3 and D_0-D_7 pins are valid with the μPD17P136A and 17P137A only.

2. The μPD17P136A and 17P137A do not have a pull-up resistor connected by mask option.

Pin No.	Pin name	Function	Output	At reset
17	$P1B_0/V_{PP}$ Note1	Used as port 1B, or programming voltage supply pin in program memory write/verify mode. ● $P1B_0$ • 1-bit input port • A pull-up resistor can be internally connected by mask option Note2 ● V_{PP} • Available for the μPD17P136A and μPD17P137A only • Applies programming voltage (+12.5 V) at program memory writing/verification	Input	Input
18 | 21	$P1A_3$ | $P1A_0$	Port 1A • 4-bit input/output port • Input/output setting in 4-bit unit • A pull-up resistor can be internally connected by mask option Note2	N-ch open drain	Input
22 23 24 25	$P0D_3/\overline{TM0OUT}$ $P0D_2/SI$ $P0D_1/SO$ $P0D_0/\overline{SCK}$	Used as port 0D, or timer 0 carry output, serial data input, serial data output, and serial clock input/output pins A pull-up resistor can be internally connected by mask option Note2 ● $P0D_3$ to $P0D_0$ • 4-bit input/output port • Input/output setting in 1 bit unit ● $\overline{TM0OUT}$ • Timer 0 carry output ● SI • Serial data input ● SO • Serial data output ● $\overline{SCK}$ • Serial clock input/output	N-ch open drain	Input
26 27	X_{OUT} X_{IN}/CLK Note3	In the case of the μPD17135A/17137A/17P137A ● X_{IN}, X_{OUT} • Connected to a resonator for system clock oscillation • The ceramic resonator is connected. ● CLK • Available for the μPD17P137A only • Clock input pin for address updating at program memory writing/verification	—	—
26 27	OSC_0 OSC_1/CLK Note3	In the case of the μPD17134A/17136A/17P136A ● OSC_0, OSC_1 • Connected to a resonator for system clock oscillation • Resistor is connected between OSC_0 and OSC_1. ● CLK • Available for the μPD17P136A only • Clock input pin for address updating at program memory writing/verification		
28	V_{DD}	Power supply In the program memory write/verify mode of the μPD17P136A/17P137A, +6 V is applied.	—	—

Notes 1. The V_{PP} pin is valid only with the μPD17P136A and 17P137A.

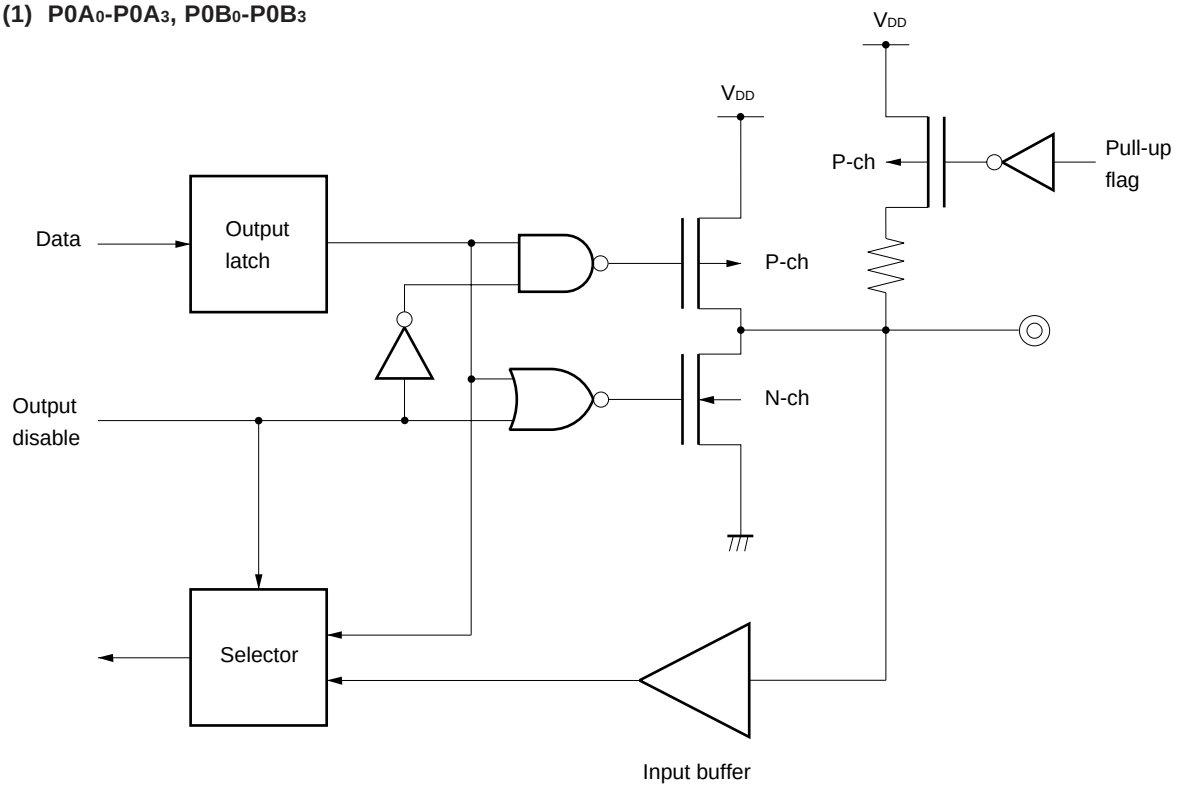
2. The μPD17P136A and 17P137A do not have a pull-up resistor connected by mask option.

3. The CLK pin is valid only with the μPD17P136A and 17P137A.

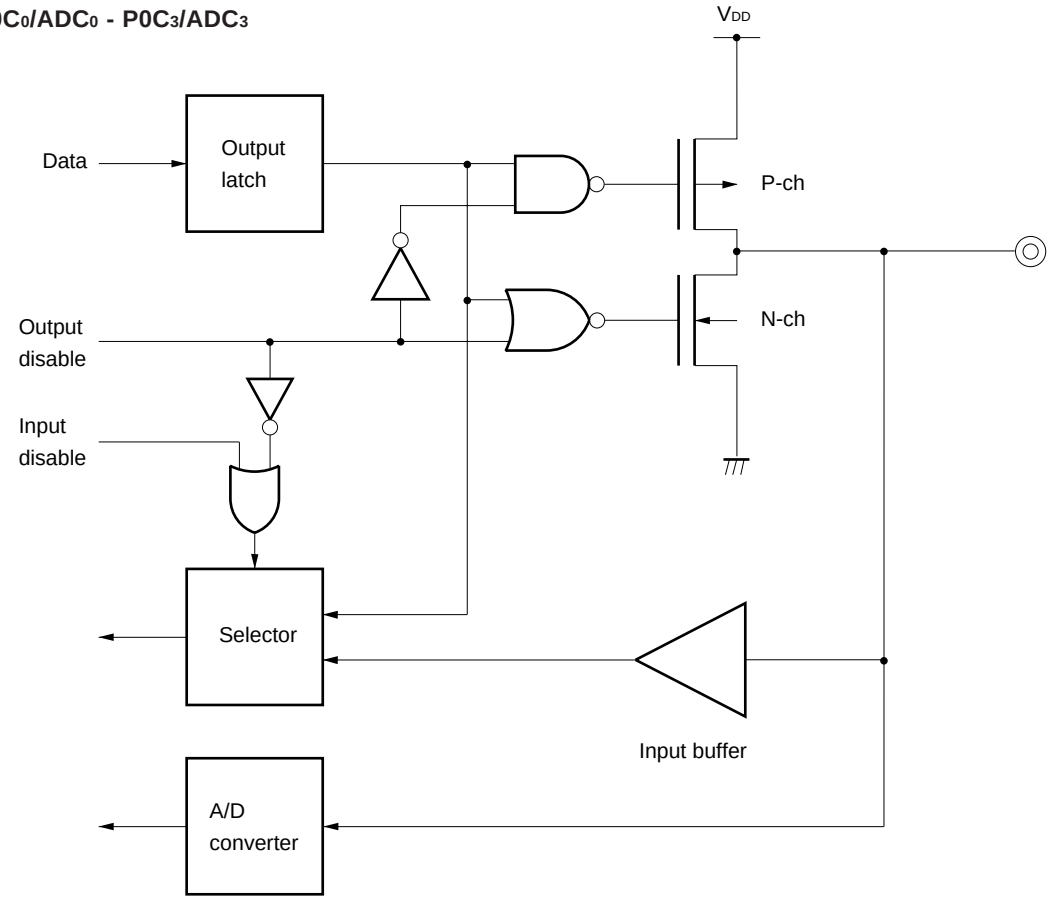
2.2 PIN INPUT/OUTPUT CIRCUIT

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

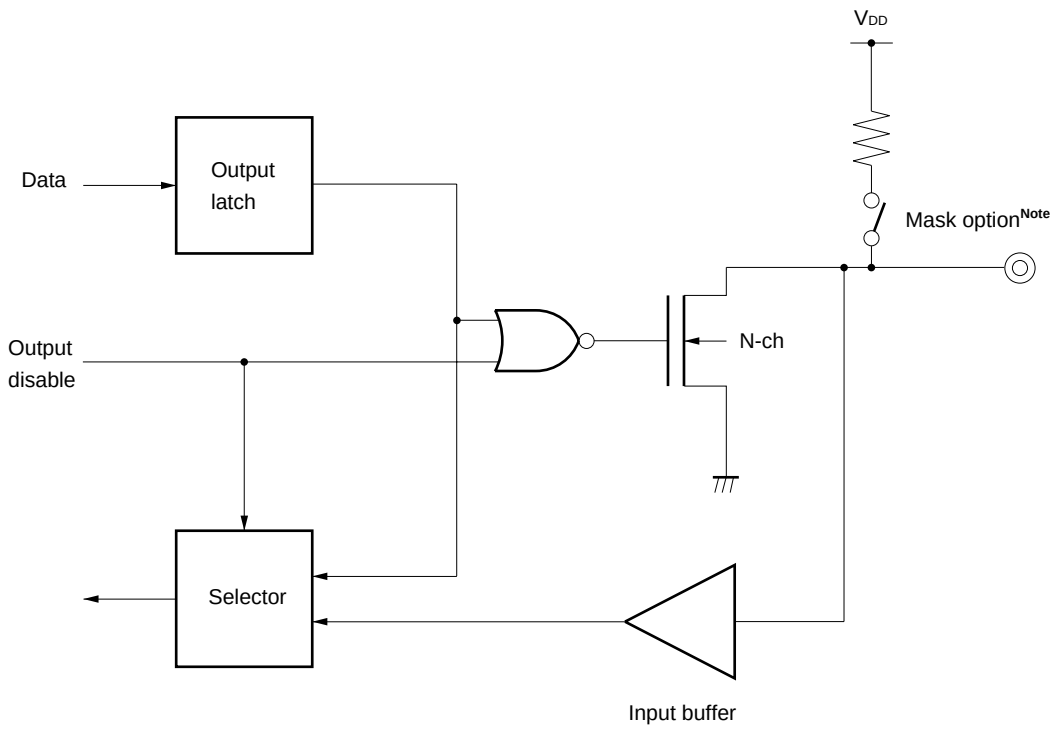
(1) $P0A_0$-$P0A_3$, $P0B_0$-$P0B_3$



(2) $P0C_0/ADC_0$ - $P0C_3/ADC_3$

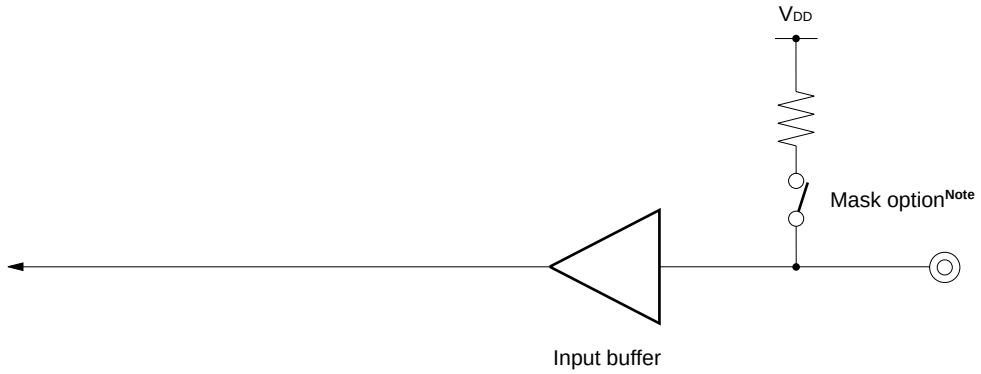


(3) $P0D_0$-$P0D_3$, $P1A_0$-$P1A_3$



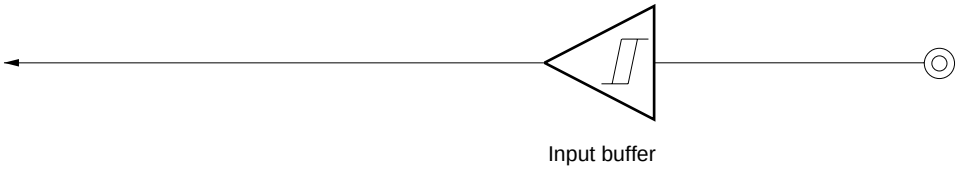
Note The μPD17P136A and 17P137A do not have a pull-up resistor as mask option.

(4) $P1B_0$

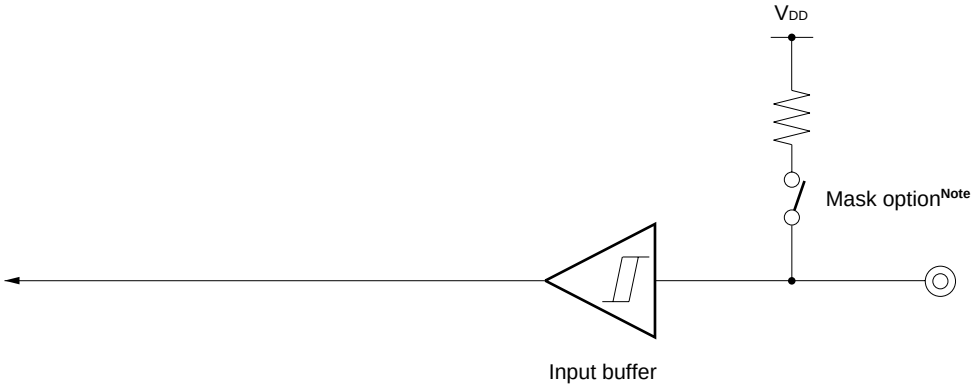


Note The μPD17P136A and 17P137A do not have a pull-up resistor as mask option.

(5) INT



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



Note The μPD17P136A and 17P137A do not have a pull-up resistor as mask option.

2.3 PROCESSING OF UNUSED PINS

The unused pins should be handled as follows:

★ **Table 2-1. Processing of Unused Pins**

Pin Name			Recommended Processing	
			Internal	External
Port	Input mode	P0A, P0B	Connect pull-up resistor by software	Open
		P0C	—	Connect each pin to V_{DD} or GND via resistor[Note 1]
		P0D, P1A	Pull-up resistor not connected by mask option	Directly connect to GND
			Pull-up resistor connected by mask option	Open
		$P1B_0$[Note2]	Pull-up resistor not connected by mask option	Directly connect to GND
	Output mode	P0A, P0B, P0C (CMOS port)	—	Open
		P0D, P1A (N-ch open-drain ports)	Outputs low level without pull-up resistor connected by mask option	
			Outputs high level without pull-up resistor connected by mask option	
External interrupt (INT)			Pull-up resistor not connected by mask option	Directly connect to V_{DD} or GND
			Pull-up resistor connected by mask option	Open
$\overline{\text{RESET}}$[Note3] (when only internal power-ON/power-down reset function is used)			Pull-up resistor not connected by mask option	Directly connect to V_{DD}
			Pull-up resistor connected by mask option	
V_{ADC}			—	Directly connect to V_{DD}

Notes 1. When connecting an external pull-up resistor (to V_{DD} via resistor) or pull-down resistor (to GND via resistor), make sure that the driving voltage and current consumption of the port are not exceeded. When connecting a pull-up or pull-down resistor with a high resistance to a port pin, make sure that noise is not superimposed on the pin. Generally, the resistance of the pull-up or pull-down resistor is about several kΩ, though it varies depending on the application circuit.

2. Because the $P1B_0$ pin is multiplexed with a test mode setting function, do not connect a pull-up resistor to this pin using the mask option. Directly connect it to GND.

3. In an application circuit where high reliability is required, be sure to input the $\overline{\text{RESET}}$ signal from an external source. Because the $\overline{\text{RESET}}$ pin is multiplexed with a mode setting function, directly connect it to V_{DD} if not use.

Caution It is recommended that the I/O mode, pull-up of resistors by software, and output levels of pins be fixed by repeatedly setting in each loop of the program.

Remark The μPD17P136A and 17P137A do not have a pull-up resistor as mask option.

2.4 NOTES ON USING $\overline{\text{RESET}}$ PIN AND P1B$_0$ PIN

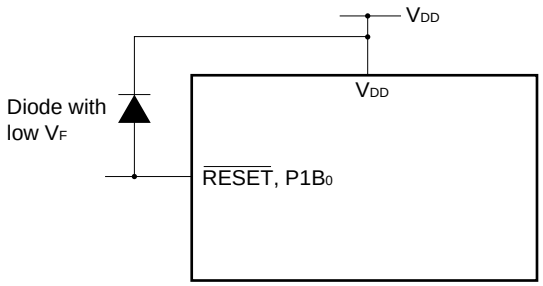
The $\overline{\text{RESET}}$ and P1B$_0$ pins have a function for setting a test mode in which the internal operations of the μPD17134A subseries are tested (for IC test), in addition to the functions described in **2.1 PIN FUNCTIONS**.

When a voltage exceeding V_{DD} is applied to either of these pins, the test mode is set. This means that, even during the normal operation, the test mode is set if a noise exceeding V_{DD} is applied. As a result, the operation may not be performed normally.

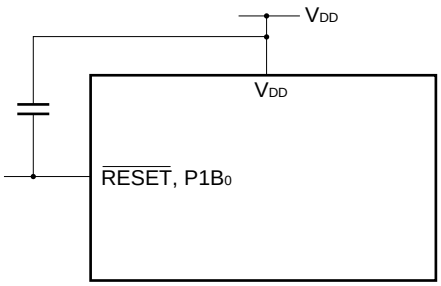
This is especially true if the wiring length of the $\overline{\text{RESET}}$ or P1B$_0$ pin is too long in which case a noise may be superimposed on the wiring.

Therefore, perform wiring so that noise may not be superimposed, by keeping the wiring length as short as possible. If noise is inevitable, take noise preventive measures by using an external component as illustrated below.

- **Connect a diode with low V_F between V_{DD} and $\overline{\text{RESET}}$/P1B$_0$**



- **Connect a capacitor between V_{DD} and $\overline{\text{RESET}}$/P1B$_0$**



[MEMO]

CHAPTER 3 PROGRAM COUNTER (PC)

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

3.1 PROGRAM COUNTER CONFIGURATION

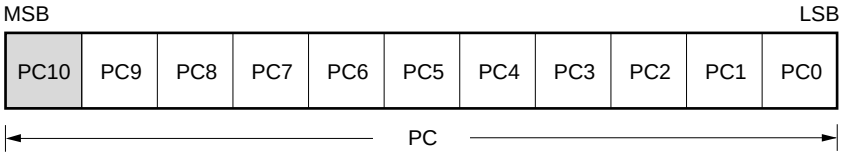
Figure 3-1 shows the configuration of the program counter.

The program counters of the μPD17134A and μPD17135A are 10-bit binary counters.

The program counters of the μPD17136A, μPD17137A, μPD17P136A, and μPD17P137A are 11-bit binary counters.

This program counter is incremented whenever an instruction is executed.

Figure 3-1. Program Counter



Remark The shaded part is effective only in the case of μPD17136A/17137A/17P136A/17P137A.

3.2 PROGRAM COUNTER OPERATION

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

3.2.1 to 3.2.7 explain program counter operation during execution of each instruction.

Figure 3-2. Value of the Program Counter after Instruction Execution

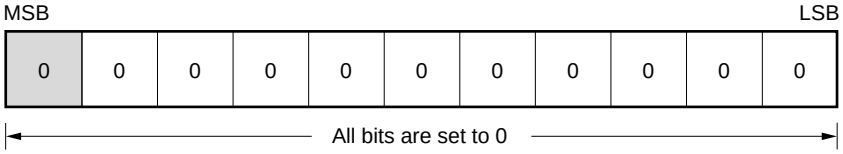
Instruction \ Program counter bit	Program counter value										
	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
At reset	0	0	0	0	0	0	0	0	0	0	0
BR addr CALL addr	Value set by the addr										
BR @AR CALL @AR (MOVT DBF, @AR)	Value in the address register (AR)										
RET RETSK RETI	Value in the address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

Remark The shaded part is effective only in the case of μPD17136A/17137A/17P136A/17P137A.

3.2.1 At Reset

By setting the $\overline{\text{RESET}}$ pin to low, the program counter is set to 0000H.

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



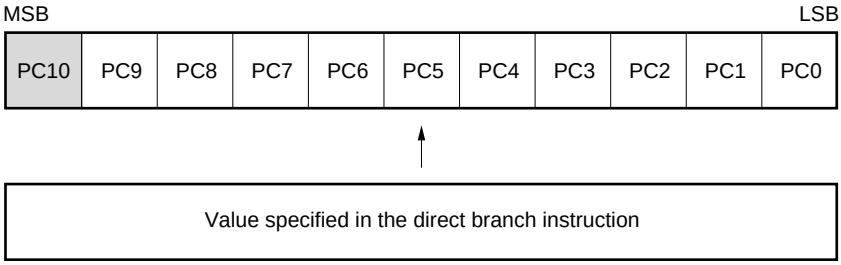
Remark The shaded part is effective only in the case of μPD17136A/17137A/17P136A/17P137A.

★ 3.2.2 During Execution of the Branch Instruction (BR)

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

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

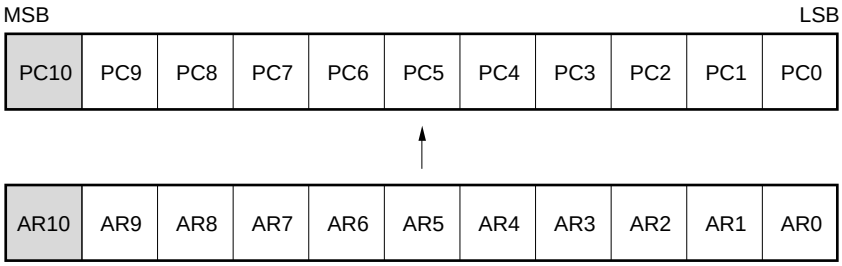
Figure 3-4. Value in the Program Counter during Execution of a BR addr Instruction



Remark The shaded part is effective only in the case of μPD17136A/17137A/17P136A/17P137A.

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

Figure 3-5. Value in the Program Counter during Execution of a BR @AR Instruction



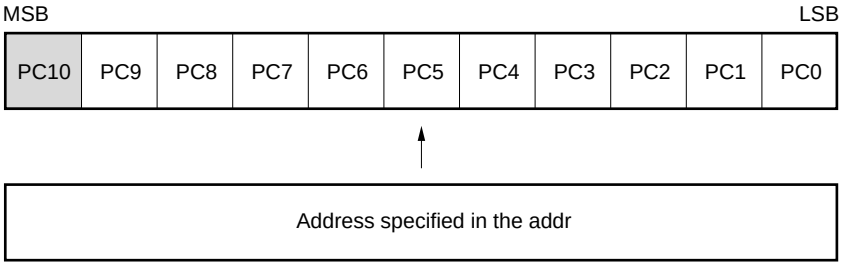
Remark The shaded part is effective only in the case of μPD17136A/17137A/17P136A/17P137A.

★ **3.2.3 During Execution of Subroutine Calls (CALL)**

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

A CALL addr causes the value in the program counter to be saved in the stack and then the address specified in the operand to be placed in the program counter. CALL addr can specify 000H-03FFH in the μPD17134A and 17135A, and 0000H-07FFH in the μPD17136A, 17137A, 17P136A, and 17P137A.

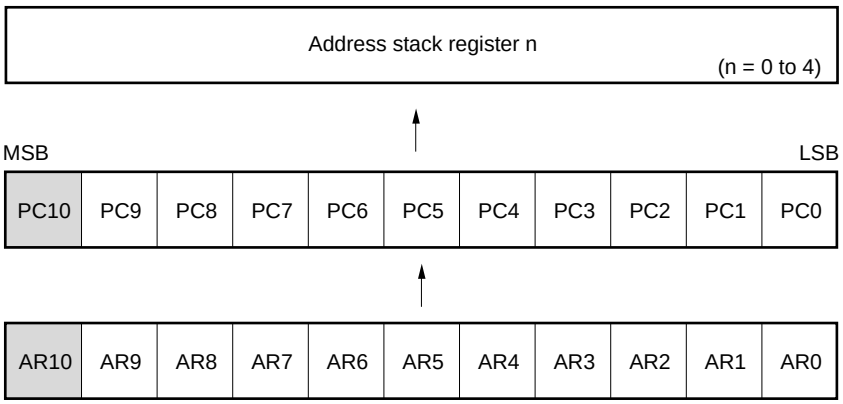
Figure 3-6. Value in the Program Counter during Execution of a CALL addr



Remark The shaded part is effective only in the case of μPD17136A/17137A/17P136A/17P137A.

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

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

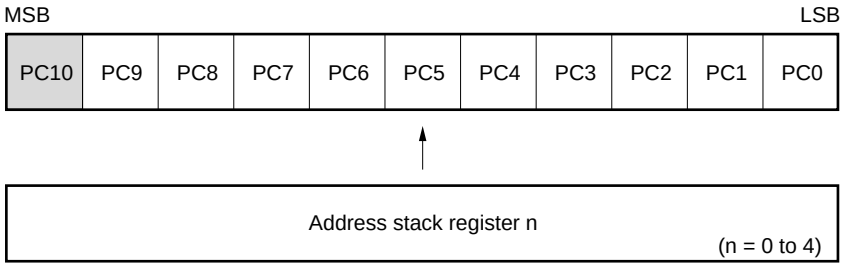


Remark The shaded part is effective only in the case of μPD17136A/17137A/17P136A/17P137A.

3.2.4 During Execution of Return Instructions (RET, RETSK, RETI)

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

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



Remark The shaded part is effective only in the case of μPD17136A/17137A/17P136A/17P137A.

3.2.5 During Table Reference (MOVT)

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

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

3.2.6 During Execution of Skip Instructions (SKE, SKGE, SKLT, SKNE, SKT, SKF)

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

3.2.7 When an Interrupt Is Received

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

[MEMO]

★

CHAPTER 4 PROGRAM MEMORY (ROM)

The program organization of the μPD17134A subseries is shown in Table 4-1.

Table 4-1. Program Memory Configuration

Product name	Program memory capacity	Program memory address
μPD17134A	2K bytes (1024 × 16 bits)	0000H-03FFH
μPD17135A		
μPD17136A	4K bytes (2048 × 16 bits)	0000H-07FFH
μPD17137A		
μPD17P136A		
μPD17P137A		

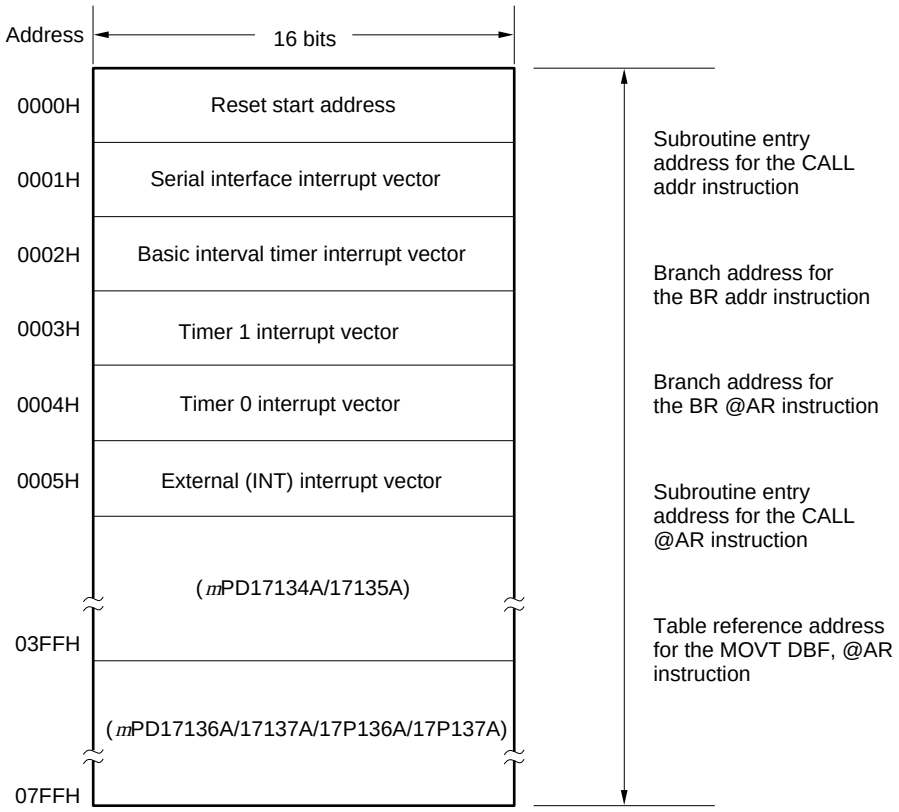
Program memory stores the program and the constant data table. The first area of the program memory is assigned to reset start and interrupt vector addresses.

The program memory address is specified by the program counter.

4.1 PROGRAM MEMORY CONFIGURATION

Figure 4-1 shows the program memory map. Branch instructions, subroutine calls, and table references can specify any address in program memory.

Figure 4-1. Program Memory Map for the μPD17134A Subseries



4.2 PROGRAM MEMORY USAGE

Program memory has the following two main functions:

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

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

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

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

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

4.2.1 Flow of the Program

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

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

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

Items (1) to (5) explain branching of the program using the interrupt function and instructions.

(1) Vector address

Table 4-2 shows the address to which the program is branched (vector address) when a reset or interrupt occurs.

Table 4-2. Vector Address for the μPD17134A Subseries

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

(2) Direct branch

A direct branch (BR addr) instruction branches a value of operand (addr) as an address. (In the case of the μPD17134A and μPD17135A, the most significant bit must be 0. If an address is specified outside of this range, an error will occur in the assembler.) A BR addr instruction can be used to branch to any address in program memory.

(3) Indirect branch

When executing an indirect branch (BR @AR), the program branches to the address specified by the value stored in the address register (AR). A BR @AR can be used to branch to any address in program memory. Also see **7.2 ADDRESS REGISTER (AR)**.

(4) Subroutine

To branch execution to a subroutine, the subroutine call (CALL) instruction is used.

The CALL instruction can be used in two ways: as a direct subroutine call instruction (CALL addr) that causes execution to branch using the value of the operand (addr) as an address, and as an indirect subroutine call instruction (CALL @AR) that causes execution to branch using the contents of an address register as an address.

To return from a subroutine, the RET or RETSK instruction is used. By executing the RET or RETSK instruction, execution is returned to the program memory address next to the one at which the CALL instruction was executed.

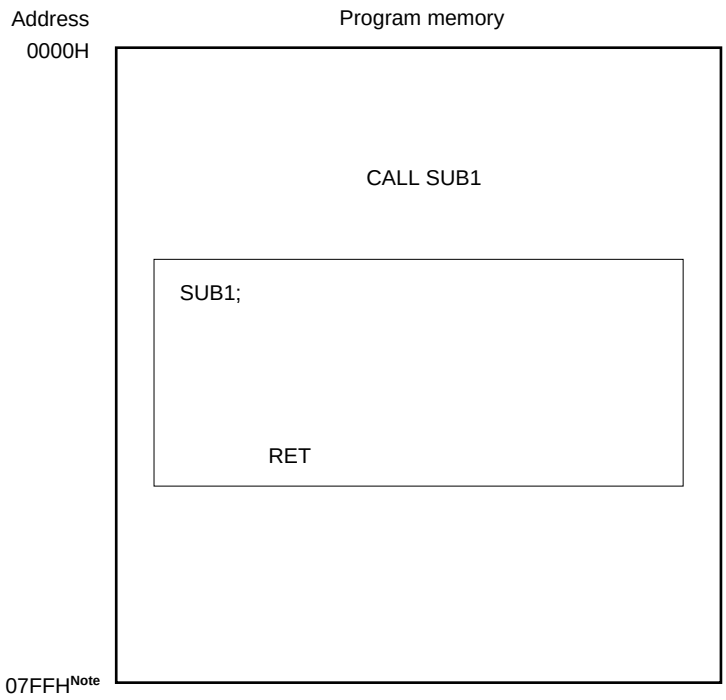
When the RETSK instruction is used, the first instruction after execution has returned from the subroutine is executed as a NOP instruction.

<1> Direct subroutine call

When using a direct subroutine call (CALL addr), the 11-bit instruction operand is used to specify a program memory address of the branched subroutine. (In the case of the μPD17134A and μPD17135A, the most significant bit must be 0. If an address is specified outside of this range, an error will occur in the assembler.)

Example

Figure 4-2. CALL addr Instruction



Note The program memory of the μPD17134A and μPD17135A is address 0000H to 03FFH.

<2> Indirect subroutine call

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

Also see **7.2 ADDRESS REGISTER (AR)**.

4.2.2 Table Reference

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

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

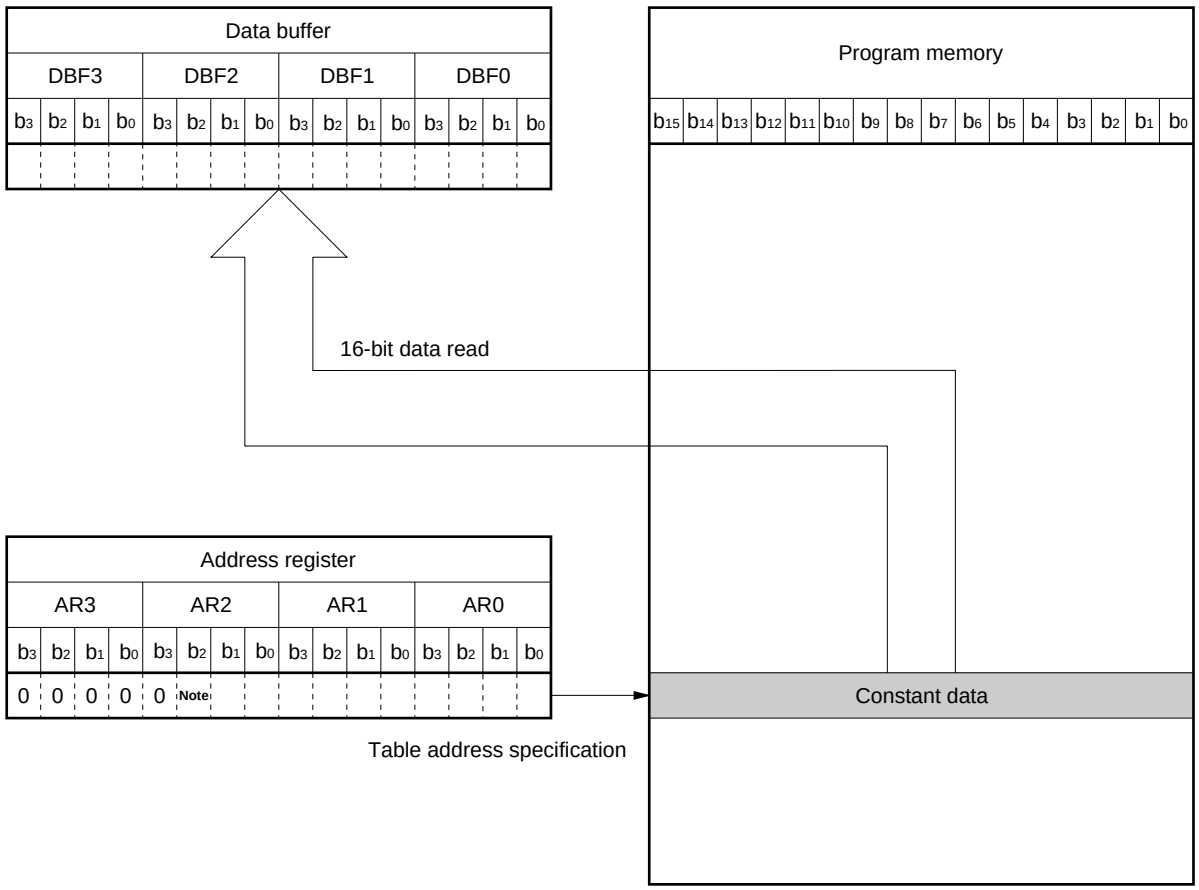
Since each location in program memory contains 16 bits of information, the MOVT instruction causes

16 bits of data to be stored in the data buffer. The address register can be used to table reference any location in program memory.

Caution Note that one level of the address stack is temporarily used when performing table reference. Be sure not to exceed the stack level that can be used. Also see 7.2 ADDRESS REGISTER (AR) and CHAPTER 10 DATA BUFFER (DBF).

Remark Two instruction cycles are required to execute the table reference instruction, but this is an exception.

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



Note This bit is fixed to 0 in the case of the μPD17134A and μPD17135A.

(1) Constant data table

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

Example 1. Program to read data in a constant data table.

```
OFFSET      MEM     0.00H                   ;  Area to store the offset address.
ROMREF:
            BANK0
                                            ;  Stores the start address of the  constant data
                                            ;  table in the AR register.
            MOV         AR3, #.DL.TABLE SHR 12 AND 0FH
            MOV         AR2, #.DL.TABLE SHR 8 AND 0FH
            MOV         AR1, #.DL.TABLE SHR 4 AND 0FH
            MOV         AR0, #.DL.TABLE AND 0FH

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

            ADD         AR0, OFFSET         ;  Adds the offset address.
            ADDC        AR1, #0
            ADDC        AR2, #0
            ADDC        AR3, #0
            MOVT        DBF, @AR            ;  Reads the constant data.

TABLE:
            DW      0001H                   ; When OFFSET = 0H
            DW      0002H
            DW      0004H
            DW      0008H
            DW      0010H
            DW      0020H
            DW      0040H
            DW      0080H
            DW      0100H
            DW      0200H
            DW      0400H
            DW      0800H
            DW      1000H
            DW      2000H
            DW      4000H
            DW      8000H                   ; When OFFSET = 0FH

            END
```

(2) Branch address table

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

Example 2. Program to branch to the address of the branch address table.

```
OFFSET      MEM     0.00H                   ; Area to store the offset address.
ROMREF:
            BANK0                           ; Stores the start address of the constant data
                                            ; table in the AR register.
            MOV         AR3, #.DL.TABLE SHR 12 AND 0FH
            MOV         AR2, #.DL.TABLE SHR 8 AND 0FH
            MOV         AR1, #.DL.TABLE SHR 4 AND 0FH
            MOV         AR0, #.DL.TABLE AND 0FH

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

            ADD         AR0, OFFSET         ; Adds the offset address.
            ADDC        AR1, #0
            MOVT        DBF, @AR            ; Reads the branch address
            PUT         AR, DBF             ; AR ← Branch address
            BR          @AR

TABLE:
            DW      0001H                   ; When OFFSET = 0H
            DW      0002H
            DW      0004H
            DW      0008H
            DW      0010H
            DW      0020H
            DW      0040H
            DW      0080H
            DW      0100H
            DW      0200H                   ; When OFFSET = 9H

            END
```

[MEMO]

CHAPTER 5 DATA MEMORY (RAM)

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

5.1 DATA MEMORY CONFIGURATION

Figure 5-1 shows the configuration of data memory.

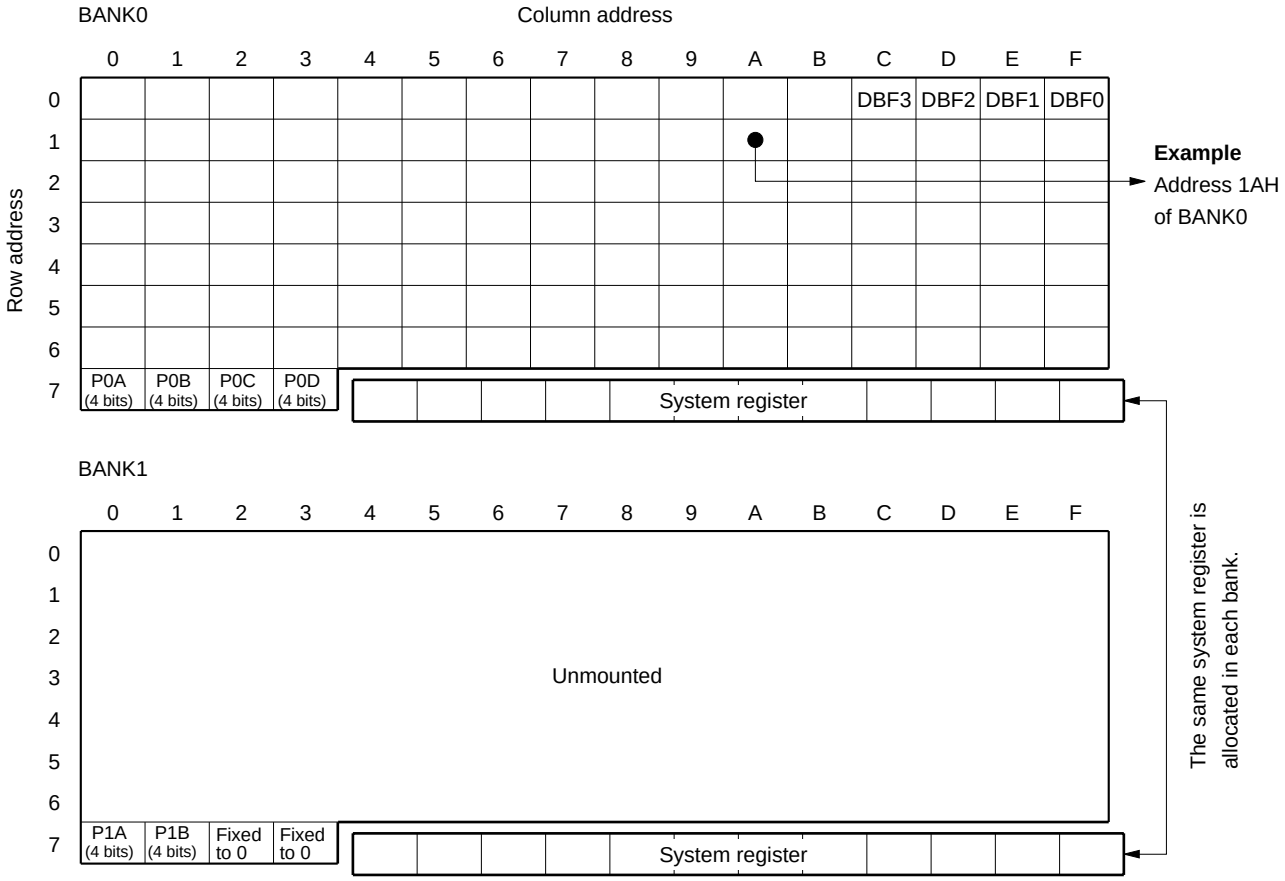
Data memory is divided into two areas called banks: BANK0 and BANK1.

An address is allocated to the data memory for each bank. An address consists of 4 bits of memory called “a nibble”.

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

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

★ **Figure 5-1. Data Memory Configuration**



Caution No hardware is assigned to addresses 00H through 6FH in BANK1. Do not use this area. If the contents of this area are read, the value is undefined. An instruction to write data to this area is invalid.

5.1.1 System Register (SYSREG)

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

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

For details, refer to **CHAPTER 7 SYSTEM REGISTER (SYSREG)**.

Figure 5-2. System Register Configuration

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

5.1.2 Data Buffer (DBF)

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

Figure 5-3 shows the configuration of the data buffer.

Figure 5-3. Data Buffer Configuration

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

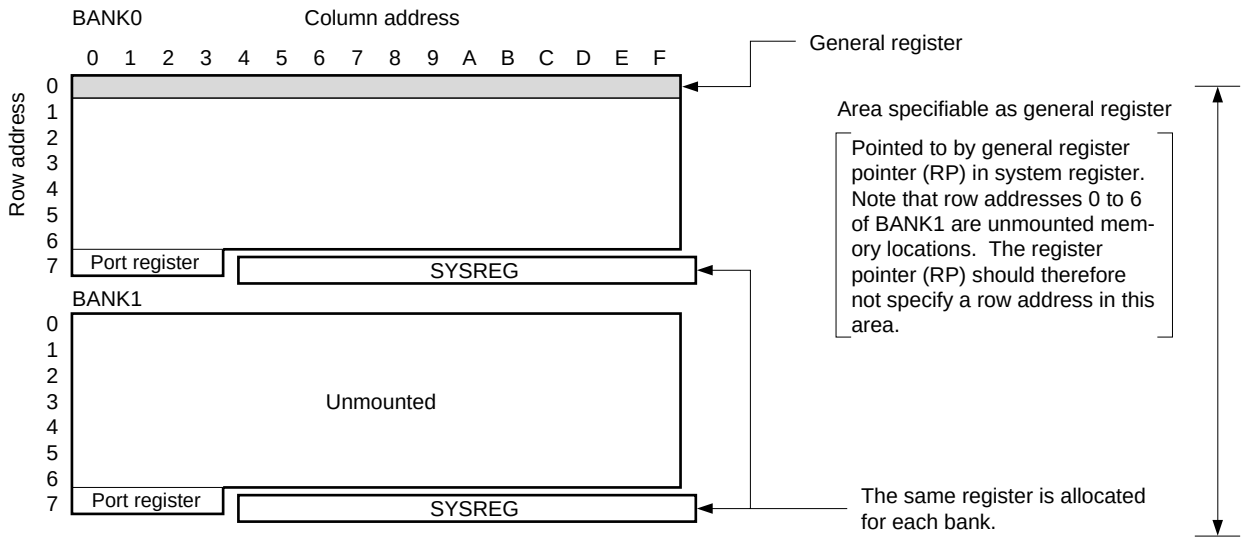
5.1.3 General Register (GR)

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

This arbitrary row address in an arbitrary bank is specified by the register pointer (RP) in the system register (SYSREG).

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

Figure 5-4. General Register (GR) Configuration



5.1.4 Port Registers

A port register consists of eight nibbles allocated at addresses 70H to 73H in each bank of the data memory.

As shown in Figure 5-5, the high-order 3 bits of address 71H of BANK1 and all of addresses 72H and 73H of BANK1 are always set to 0.

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

Figure 5-5. Port Register Configuration

Port register																	
Address		70H				71H				72H				73H			
Symbol	BANK0	P0A				P0B				P0C				P0D			
		P0A3	P0A2	P0A1	P0A0	P0B3	P0B2	P0B1	P0B0	P0C3	P0C2	P0C1	P0C0	P0D3	P0D2	P0D1	P0D0
	BANK1	P1A				P1B											
		P1A3	P1A2	P1A1	P1A0	Fixed to “0”			P1B0	Fixed to “0”				Fixed to “0”			

5.1.5 General Data Memory

General data memory is all the data memory not used by the port and system registers (SYSREG). In other words, general data memory consists of 112 nibbles in BANK0.

5.1.6 Unmounted Data Memory

There is no hardware mounted at addresses 00H to 6FH of BANK1. Any attempt to read this area will yield undefined value. Writing data to this area is invalid and should therefore not be attempted.

★

CHAPTER 6 STACK

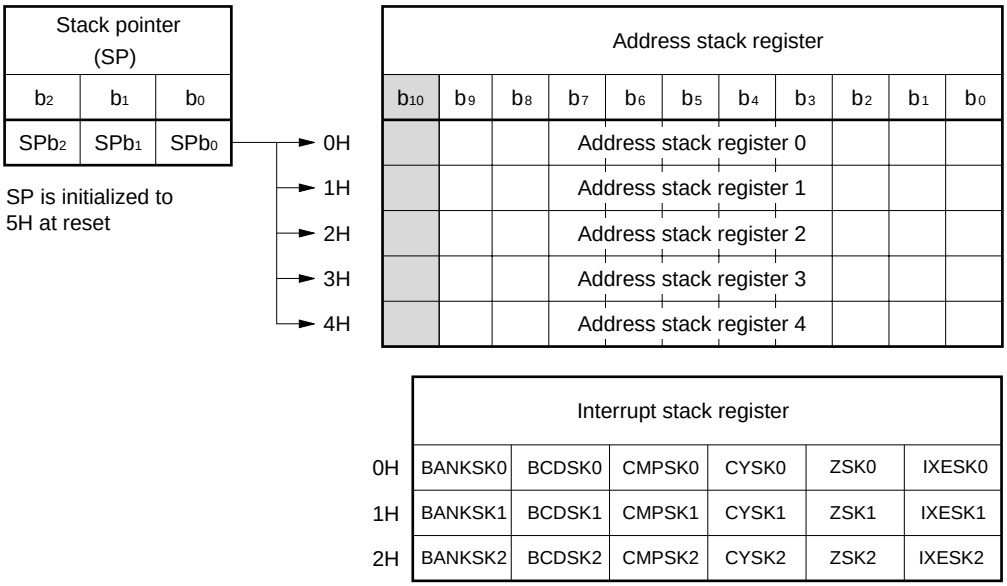
The stack is a register used to save information such as the program return address and the contents of the system register during execution of subroutine calls or interrupts.

6.1 STACK CONFIGURATION

Figure 6-1 shows the stack configuration.

The stack consists of the following parts: one 3-bit binary counter stack pointer, five 10-bit (μPD17134A, 17135A)/ 11-bit (μPD17136A, 17137A, 17P136A, 17P137A) address stack registers, and three 6-bit interrupt stack registers.

Figure 6-1. Stack Configuration



Remark The shaded part is effective only in the case of μPD17136A/17137A/17P136A/17P137A.

6.2 FUNCTIONS OF THE STACK

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

6.3 ADDRESS STACK REGISTERS (ASRs)

Five 11-bit address stack registers (ASRs) are provided as shown in Figure 6-1. The functions of these registers are as follows:

- Store a return address when the CALL addr or CALL @AR instruction is executed, when the first instruction cycle of the “MOVT DBF, @AR” instruction is executed, or when an interrupt is accepted.
- Store the contents of an address register (AR) when the PUSH AR instruction is executed. The ASR to which the data is to be stored is specified by decrementing the value of the stack pointer (SP) by one when the instruction is executed.
- Restore the contents of the ASR (return address) specified by the stack pointer to the program counter and increment the value of the stack pointer by one when the RET or RETSK instruction is executed, when the second instruction cycle of the “MOVT DBF, @AR” instruction is executed, or when the RETI instruction is executed.
- Transfer the value of the ASR specified by the stack pointer to an address register and decrement the value of the stack pointer by one when the POP AR instruction is executed.

Caution **If the stack pointer underflows as a result of executing the CALL addr or CALL @AR instruction or servicing an interrupt, it is assumed that a hang-up occurs. Consequently, the internal reset signal is generated, the hardware is initialized, and the program is started from address 0000H.**

Remark The size of the ASR differs depending on the model. The μPD17134A and 17135A have five 10-bit ASRs, while the μPD17136A, 17137A, 17P136A, and 17P137A have five 11-bit ASRs.

6.4 INTERRUPT STACK REGISTERS (INTSKs)

Three 5-bit interrupt stack registers (INTSKs) are provided as shown in Figure 6-1. The functions of these registers are as follows:

- Five flags (BCD, CMP, CY, Z, and IXE) in the program status word (PSWORD) in the system register (SYSREG) to be explained shortly are saved to the INTSK when an interrupt occurs. After the flags have been saved, all the bits of the BANK and PSWORD are cleared to 0.
- The contents of INTSK are restored to the PSWORD when the RETI instruction is executed.
- INTSK saves data each time an interrupt has been accepted.

Caution **If interrupts are accepted exceeding 3 levels, the first data is lost.**

6.5 STACK POINTER (SP) AND INTERRUPT STACK REGISTERS

The stack pointer is a 3-bit binary counter that specifies the addresses of the five address stack registers as shown in Figure 6-1, and is assigned to address 01H of the register file. The value of the stack pointer is initialized to 5H at reset.

- The value of SP is decremented by one when the CALL addr or CALL @AR instruction is executed, when the first instruction cycle of the “MOVT DBF, @AR” instruction is executed, or when an interrupt is accepted.
- The value of SP is incremented by one when the RET or RETSK instruction is executed, when the second instruction cycle of the “MOVT DBF, @AR” instruction is executed, when the POP AR instruction is executed, or when the RETI instruction is executed.

When an interrupt is accepted, the counter of the interrupt stack registers is also decremented by one in addition to the SP. The value of the counter of the interrupt stack registers is incremented by one only when the RETI instruction is executed.

Table 6-1. Operation of Stack Pointer

Instruction	Value of stack pointer (SP)	Counter of interrupt stack registers
CALL addr CALL @AR MOVT, DBF @AR (1st instruction cycle) PUSH AR	–1	Not affected
RET RETSK MOVT DBF, @AR (2nd instruction cycle) POP AR	+1	
Accepting interrupt	–1	–1
RETI	+1	+1

Remark Two instruction cycles are required to execute the “MOVT DBF, @AR” instruction, but this is an exception.

Because the stack pointer (SP) is a 3-bit binary counter, it can take a value 0H to 7H. If the value of the stack pointer is 6 or more, however, an internal reset signal is generated (to prevent a hang-up). This is because only five address stack registers are available.

Because the stack pointer is located on the register file, its value can be directly read by manipulating the register file with the POKE instruction. The value of the stack pointer is also changed at this time, but the values of the address stack registers are not affected. Of course, the stack pointer can also be read by using the PEEK instruction.

The value of the stack pointer is 5H at reset.

6.6 STACK OPERATION

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

6.6.1 On Execution of Instructions CALL, RET, RETSK

Table 6-2 shows operation of the stack pointer (SP), address stack register, and the program counter (PC) during execution of CALL, RET, and RETSK.

Table 6-2. Operation of the Instructions CALL, RET, and RETSK

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

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

6.6.2 Table Reference (MOVT DBF, @AR Instruction)

Table 6-3 shows the operation during table reference.

Table 6-3. Stack Operation during Table Reference

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

Caution When the "MOVT DBF, @AR" instruction is executed, one level of the address stack is temporarily used. Exercise care not to exceed the usable stack level.

Remark Two instruction cycles are required to execute the "MOVT DBF, @AR" instruction. This is an exception.

6.6.3 Operation on Execution of Interrupt Receipt and RETI Instruction

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

Table 6-4. Operation during Interrupt Receipt and RETI Instruction

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

6.7 STACK NESTING LEVELS AND THE PUSH AND POP INSTRUCTIONS

During execution of operations such as subroutine calls and returns, the stack pointer (SP) simply functions as a 3-bit counter which is incremented and decremented by one. When the value in the stack pointer is 0H and a CALL or MOVT instruction is executed or an interrupt is received, the stack pointer is decremented to 7H. The μPD17134A subseries treat this condition as a fault and generates an internal reset signal.

In order to avoid this condition, when the address stack register is being used frequently, the PUSH and POP instructions are used to save the address stack register.

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

Table 6-5. Stack Operation during the PUSH and POP Instructions

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

[MEMO]

CHAPTER 7 SYSTEM REGISTER (SYSREG)

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

7.1 SYSTEM REGISTER CONFIGURATION

Figure 7-1 shows the allocation address of the system register in data memory. As shown in Figure 7-1, the system register is allocated in addresses 74H to 7FH of data memory, independently of the banks. This means that each bank has the same system register at addresses 74H to 7FH.

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

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

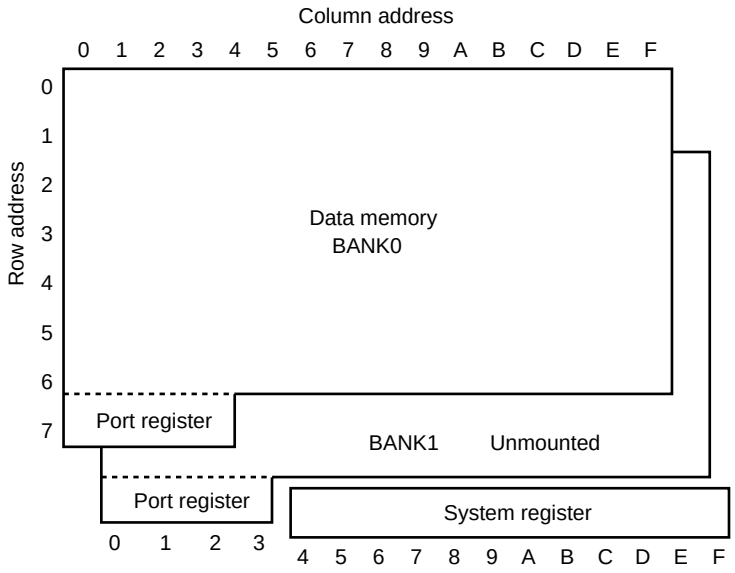


Figure 7-2 shows the configuration of the system register. As shown in Figure 7-2, the system register consists of the following seven registers.

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

Figure 7-2. System Register Configuration

Address	74H	75H	76H	77H	78H	79H	7AH	7BH	7CH	7DH	7EH	7FH
Name	Address register (AR)				Window register (WR)	Bank register (BANK)	Index register (IX) Data memory row address pointer (MP)			General register pointer (RP)		Program status word (PSWORD)
Symbol	AR3	AR2	AR1	AR0	WR	BANK	IXH MPH	IXM MPL	IXL	RPH	RPL	PSW
Bit	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0
Data	0 0 0 0	0 Note	(AR)			0 0 0 (BANK)	M P E	0 0 0 (MP)	(IX)	0 0 0 (RP)		B C D / C M P / C Y / Z / I X E
Initial value when reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	Undefined	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Note This bit is fixed to 0 in the case of the μPD17134A and μPD17135A.

7.2 ADDRESS REGISTER (AR)

7.2.1 Address Register Configuration

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

As shown in Figure 7-3, the address register consists of the 16 bits in address 74H to 77H (AR3 to AR0) of the system register. However, since the high-order 5 or 6 bits are always set to 0, the address register is actually 11 or 10 bits. When the system is reset, all 16 bits of the address register are reset to 0.

Figure 7-3. Address Register Configuration

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

Note This bit is fixed to 0 in the case of the μPD17134A and μPD17135A.

7.2.2 Address Register Functions

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

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

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

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

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

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

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

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

Also see **CHAPTER 6 STACK**.

(3) Indirect branch instruction (BR @AR)

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

(4) Indirect subroutine call (CALL @AR)

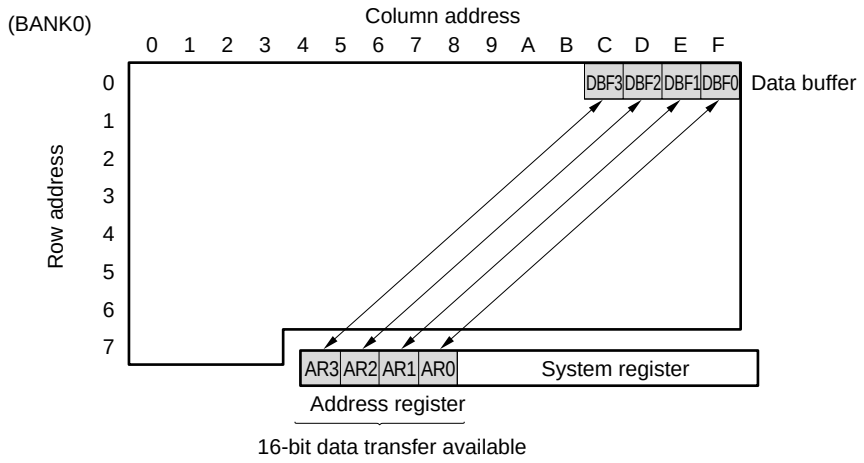
When the CALL @AR instruction is executed, the subroutine located at the address in program memory specified by the value in the address register is called.

★ **(5) Address register used as a peripheral hardware register**

The address register can be manipulated 4 bits at a time by using data memory manipulation instructions. The address register can also be used as a peripheral hardware register for transferring 16-bit data to the data buffer. In other words, by using the PUT AR, DBF and GET DBF AR instructions, the address register can be used to transfer 16-bit data to the data buffer.

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

Figure 7-4. Address Register Used as a Peripheral Circuit



7.3 WINDOW REGISTER (WR)

7.3.1 Window Register Configuration

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

As shown in Figure 7-5, the window register (WR) consists of four bits allocated at address 78H of the system register. The contents of the window register is undefined after reset. However, when $\overline{\text{RESET}}$ is used to release the system from HALT or STOP mode, the previous state is maintained.

Figure 7-5. Window Register Configuration

Address	78H			
Name	Window register			
Symbol	WR			
Bit	b_3	b_2	b_1	b_0
Data	←→			
Initial value when reset	Undefined			

7.3.2 Window Register Functions

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

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

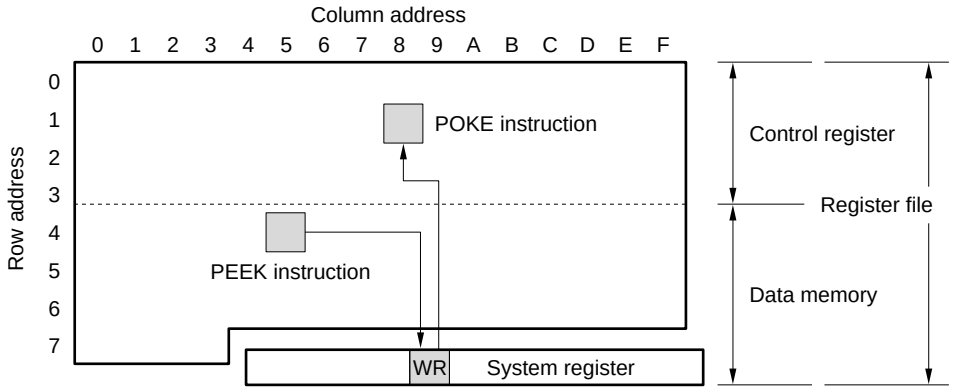
(1) PEEK WR, rf

As shown in Figure 7-6, the “PEEK WR, rf” instruction is used to transfer the contents of the register file specified by rf to the window register.

(2) POKE rf, WR

As shown in Figure 7-6, the “POKE rf, WR” instruction is used to transfer the contents of the window register to the file specified by rf.

Figure 7-6. Example of Window Register Operation



7.4 BANK REGISTER (BANK)

7.4.1 Bank Register Configuration

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

The bank register consists of four bits at address 79H (BANK) of the system register. However, since the three high-order bits are always set to 0, only the least significant bit is actually used.

All bits are set to 0 at reset.

Figure 7-7. Bank Register Configuration

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

7.4.2 Functions of Bank Register

The bank register is used to switch between the banks in data memory. Table 7-1 shows how the banks in data memory are specified by the value in the bank register.

Table 7-1. Specifying the Bank in Data Memory

Bank register				Bank in data memory
b_3	b_2	b_1	b_0	
0	0	0	0	BANK0
0	0	0	1	BANK1

Data memory is effectively divided into two banks by the bank register. When a data memory manipulation instruction is executed, the data memory in the bank specified by the bank register is manipulated.

Therefore, if the current bank is BANK0, in order to manipulate data memory in BANK1 (port registers), the bank register must be used to switch the current bank to BANK1.

The system register can be manipulated regardless of the state of the bank register.

For example, whether the instruction MOV 78H, #0 is executed for BANK0 or BANK1, the effect is the same; 0 is written to address 78H of the system register.

In addition, BANK becomes 0 after saved to the interrupt stack register.

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

7.5.1 Index Register (IX)

IX is used for address modification of the data memory. The difference between IX and MP is that IX modifies an address specified by a bank and operand m.

IX is allocated to a total of 12 bits of system register addresses 7AH (IXH), 7BH (IXM), and 7CH (IXL), as shown in Figure 7-8. Actually, however, only 11 bits, the low-order 3 bits of IXH, IXM, and IXL, function as IX. An index register enable flag (IXE) which enables address modification by IX is assigned to the least significant bit of PSW.

When IXE = 1, the address of the data memory specified by operand m is not m, but the result of ORing between m and IXM through IXL. The bank specified at this time is also indicated by ORing BANK and IXH.

Remark IXH of the μPD17134A subseries is fixed to "0", and the bank is not modified even when IXE = 1 (to prevent a bank other than 0 from being used).

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

MP is used for address modification of the data memory. The difference between IX and MP is that MP modifies the row address of an address indirectly specified by bank and operand @r.

MPH and IXH and MPL and IXM are assigned to the same address (addresses 7AH and 7BH of the system register) as shown in Figure 7-8. Actually, however, the low-order 3 bits of MPH and MPL, or a total of 7 bits, function as MP. A memory pointer enable flag (MPE) which enables address modification by MP is assigned to the most significant bit of MPH.

When MPE = 1, the bank and row address of the data memory indirectly specified by operand @r are not BANK and m_R, but the address specified by MP (the column address is specified by the contents of r independently of MPE). At this time, the low-order 3 bits of MPH and the most significant bit of MPL indicate BANK, and the low-order 3 bits of MPL indicate a row address.

Remark The low-order 3 bits of MPH and most significant bit of MPL of the μPD17134A subseries are fixed to "0", and bank 0 is always specified even when MPE = 1 (to prevent a bank other than 0 from being used).

Figure 7-8. Index Register Configuration

Address	7AH				7BH				7CH				7FH			
Name	Index register (IX) / Memory pointer (MP)												Low-order 4 bits of program status word (PSWORD)			
Symbolic name	IXH / MPH				IXM / MPL				IXL				PSW			
Bit	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Flag name	MPE															IXE
Data		← (IX) →														
		← (MP) →														
		0	0	0	0											
Initial value when reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7-9. Modification of Data Memory Address by Index Register and Memory Pointer

IXE	MPE	Data memory address specified by m											Indirect transfer address specified by @r										
		Bank				Row address			Column address				Bank				Row address			Column address			
		b_3	b_2	b_1	b_0	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_2	b_1	b_0	b_3	b_2	b_1	b_0
0	0	BANK				m							BANK				m_R			(r)			
0	1	BANK				m							MPH				MPL			(r)			
1	0	BANK				m							BANK				m_R			(r)			
		Logical OR											Logical OR										
		IXH				IXM			IXL				IXH				IXM						
1	1	Setting prohibited																					

BANK : Bank register
IX : Index register
IXE : Index enable flag
IXH : Bits 10 through 8 of index register
IXM : Bits 7 through 4 of index register
IXL : Bits 3 through 0 of index register
m : Data memory indicated by m_R and m_C
m_R : Data memory row address
m_C : Data memory column address

MP : Memory pointer
MPE : Memory pointer enable flag
MPH : High-order 3 bits of memory pointer
MPL : Low-order 4 bits of memory pointer
r : General register column address
RP : General register pointer
(×) : Contents addressed by ×
×: Direct address such as r

Table 7-2. Instructions Subject to Address Modification

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

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

As shown in Table 7-9, data memory addresses are not affected by the index register and the data memory row address pointer.

(1) Data memory manipulation instructions

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

```
R003    MEM    0.03H
M061    MEM    0.61H
        ADD    R003, M061     ; Addition in memories (0.03H) ← (0.03H) + (0.61H)
```

As shown in Figure 7-10, when the above instructions are executed, the data in general register address R003 and data memory address M061 are added together and the result is stored in general register address R003.

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

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

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

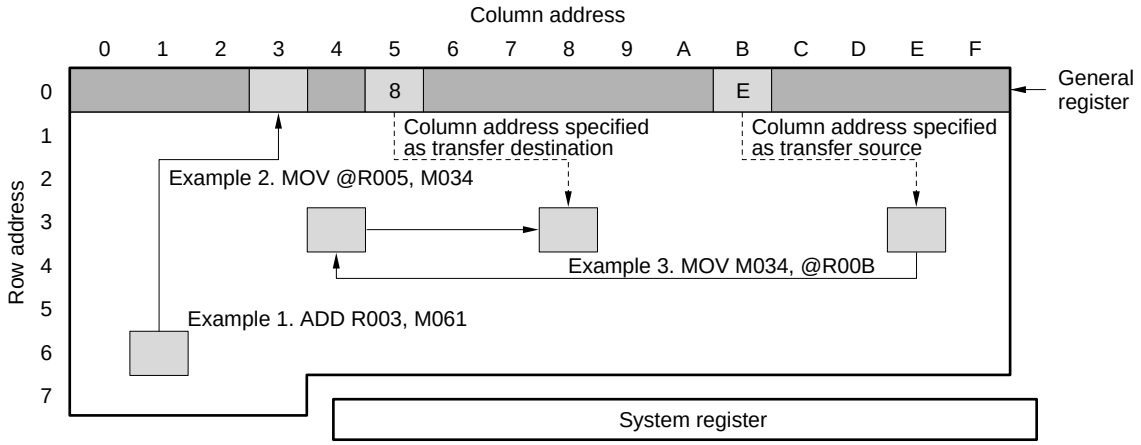
As shown in Figure 7-10, when the above instructions are executed, the data stored in data memory address M034 is transferred to data memory location 38H.
The "MOV @r, m" instruction transfers the contents of the data memory specified by m to a data memory address with the row address same as m and column address specified by @r.
In the above example, therefore, data at M034 is transferred to 38H whose row address is the same as that of M034 (= 3) and column address is specified by the contents of R005 (= 8).

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

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

As shown in Figure 7-10, when the above instructions are executed, the contents of data memory stored at address 3EH is transferred to data memory location M034.
The "MOV m, @r" instruction transfers the contents of the data memory of the address which the column address is specified by @r to a data memory address specified by m.
In the above example, therefore, data at 3EH is transferred to M034 whose row address is the same as that of M034 (= 3) and column address is specified by the contents of R00B (= 0EH).

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



Addresses in Example 1

ADD R003, M061

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

Addresses in Example 2

MOV @R005, M034

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

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

As shown in Figure 7-9, the indirect data transfer bank and row address specified by @r become the data memory row address pointer value only when general register indirect data transfer instructions (MOV @r, m and MOV m, @r) are used.

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

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

As shown in Figure 7-11, when the above instructions are executed, the contents of data memory address M034 is transferred to data memory location 68H.

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

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

Therefore the address in the above example is 68H.

This example is different from Example 2 in 7.5.3 when MPE = 0 for the following reasons: In this example, the data memory row address pointer is used to point to the indirect address bank and row address specified by @r. (In Example 2 in 7.5.3, the indirect address bank and row address are the same as m.)

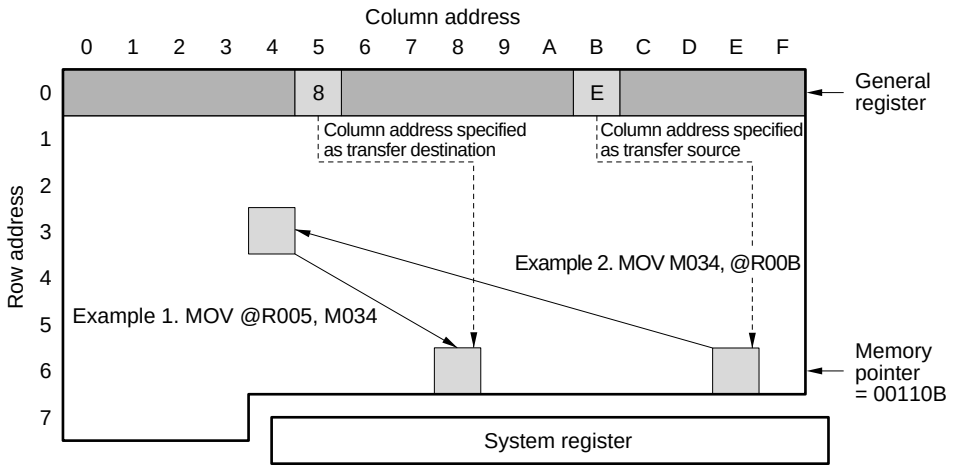
By setting MPE = 1, diagonal indirect data transfer can be performed using the general register.

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

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

As shown in Figure 7-11, when the above instructions are executed, the data stored in address 6EH is transferred to data memory location M034.

Figure 7-11. Operation Example When IXE = 0 and MPE = 1



Addresses in Example 1

MOV @R005, M034

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

Addresses in Example 2

MOV M034, @R00B

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

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

As shown in Figure 7-9, when a data memory manipulation instruction is executed, any bank or address in data memory specified by m can be modified using the index register.

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

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

An example is shown below.

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

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

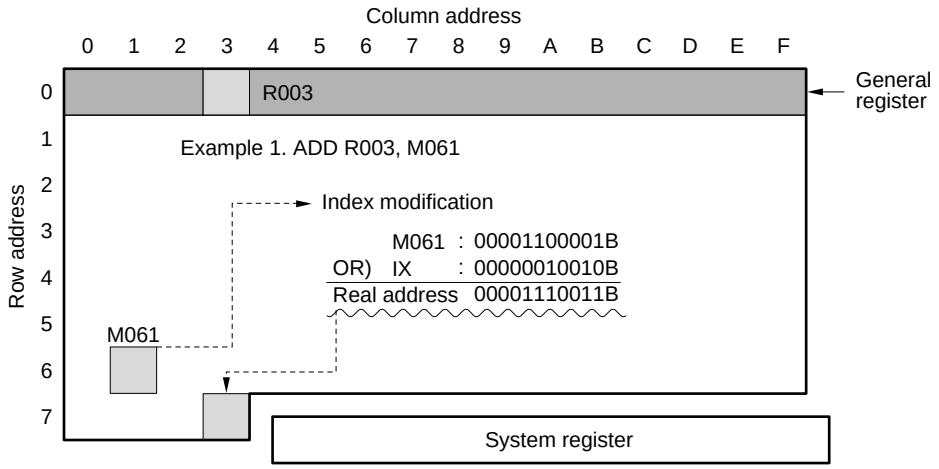
As shown in Figure 7-12, when the instructions of example 1 are executed, the value in data memory address 73H (real address) and the value in general register address R003 (address 03H) are added together and the result is stored in general register address R003.

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

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

As compared to when IXE = 0 (Examples in 7.5.3), in this example the data memory address being directly specified by m is modified by performing an OR operation on m and the index register.

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



Addresses in Example 1

ADD R003, M061

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

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

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

As shown in Figure 7-13, when the above instructions are executed, the contents of data memory address 35H is transferred to data memory location 38H.

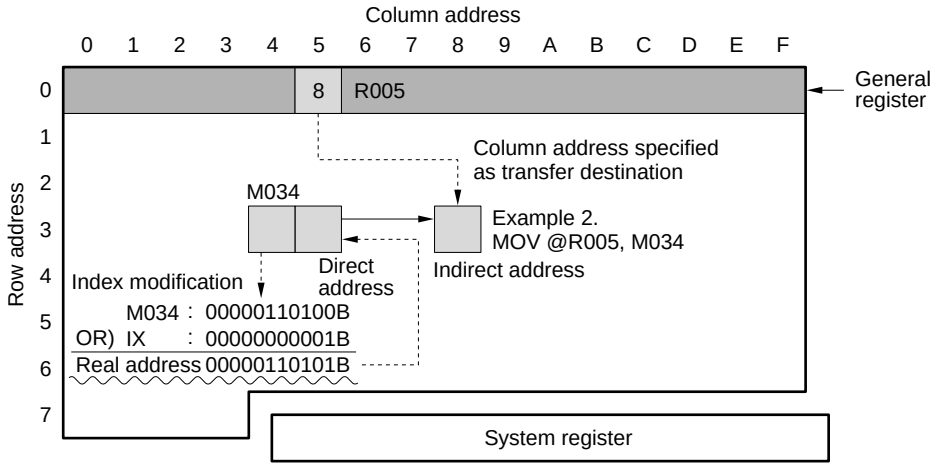
When the MOV @r, m instruction is executed when IXE = 1, the data memory address specified by m (direct address) is modified using the contents of the index register. The bank and row address of the indirect address specified by @r are also modified using the index register.

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

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

This example is different from Example 3 in 7.5.3 when IXE = 0 for the following reasons: In this example, the bank, row address and column address of the direct address specified by m are modified using the index register. The general register is transferred to the address specified by the column address of the modified data memory address and the same row address. (In Example 3 in 7.5.3, the direct address is not modified.)

Figure 7-13. Operation Example When IXE = 1 and MPE = 0



Example 3. Clearing data memory of 00H-0FH to 0

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

4. Processing an array

As shown in Figure 7-14, when an operation

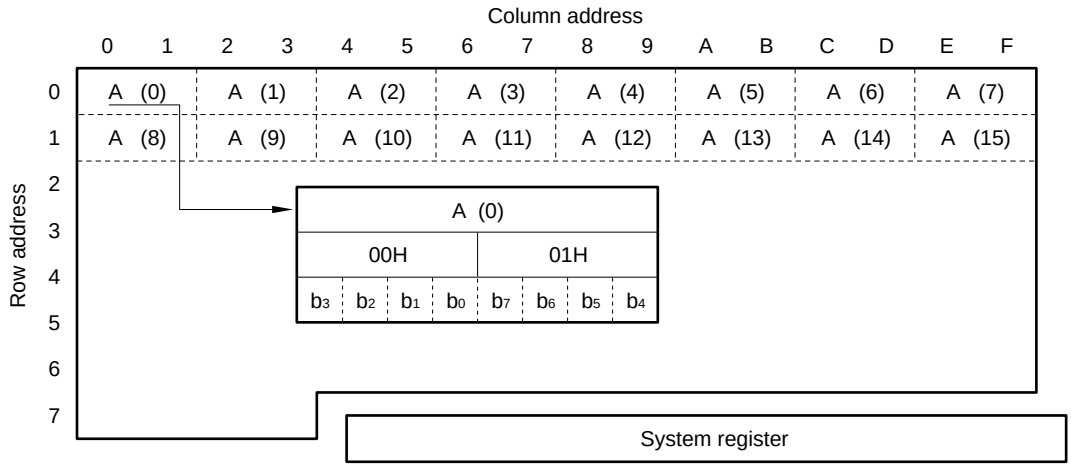
$A(N) \leftarrow A(N) + 4 \quad (0 \leq N \leq 15)$

is executed to element A (N) of a one-dimensional array with each element 8 bits long, the following instructions are executed.

```
M000  MEM   0.00H
M001  MEM   0.01H
      MOV   IXH, #0
      MOV   IXM, #N SHR 3            ; Sets offset of row address
      MOV   IXL, #N SHL 1 AND 0FH    ; Sets offset of column address
      OR    PSW, #.DF.IXE AND 0FH    ; IXE ← 1
      ADD   M000, #4
      ADDC  M001, #0                 ; A(N) ← A(N) + 4
```

In the above example, the value of N shifted 1 bit to the left (i.e., the value of N multiplied by 2) is set to the index register because one element is 8 bits long.

Figure 7-14. Operation Example When IXE = 1 and MPE = 0 (Array Processing)



7.6 GENERAL REGISTER POINTER (RP)

7.6.1 General Register Pointer Configuration

Figure 7-15 shows the configuration of the general register pointer.

Figure 7-15. General Register Pointer Configuration

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

As shown in Figure 7-15, the general register pointer consists of seven bits; four bits in system register address 7DH (RPH) and the high-order 3 bits of system register address 7EH (RPL). However, since the high-order 3 bits of address 7DH are always set to 0, the register effectively consists of four bits; the least significant bit of address 7DH and the high-order 3 bits of address 7EH.

All register bits are cleared to 0 at reset.

★ 7.6.2 Functions of the General Register Pointer

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

The general register consists of 16 nibbles in any single row of data memory. As shown in Figure 7-16, the general register pointer is used to indicate which row address is being used as the general register.

Since the general register pointer effectively consists of four bits, the data memory row addresses in which the general register can be placed are address 0H to 7H of BANK0 and BANK1. In other words, any row in data memory can be specified as the general register.

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

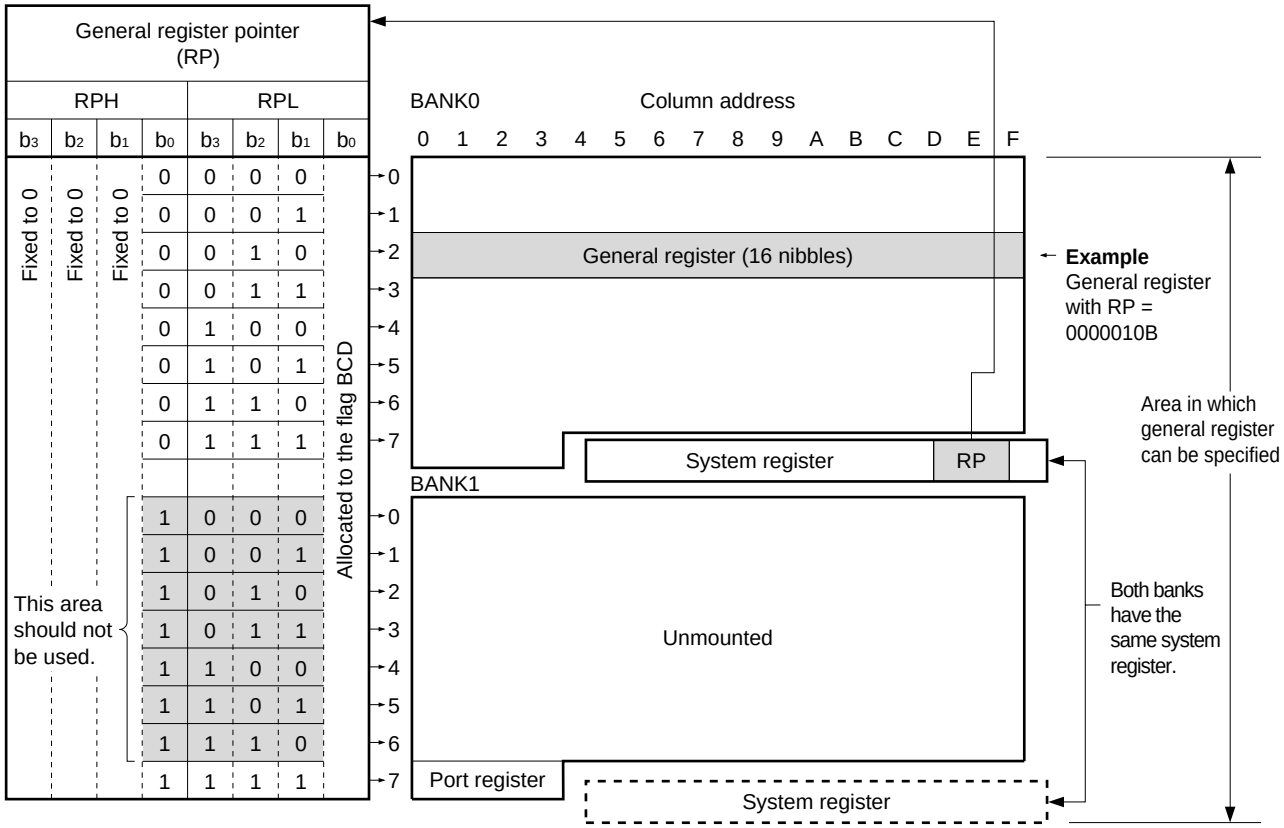
Note that row addressed 0H to 6H of BANK1 are unmounted memory locations and should therefore not be specified as locations for the general register.

For example, when instructions such as

ADD r,m and LD r,m

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

Figure 7-16. General Register Configuration



7.7 PROGRAM STATUS WORD (PSWORD)

7.7.1 Program Status Word Configuration

Figure 7-17 shows the configuration of the program status word.

Figure 7-17. Program Status Word Configuration

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

As shown in Figure 7-17, the program status word consists of five bits; the least significant bit of system register address 7EH (RPL) and all four bits of system register address 7FH (PSW).

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

★ All register bits are cleared to 0 at reset and after the contents of the interrupt stack register have been saved.

7.7.2 Functions of the Program Status Word

The flags of the program status word are used for setting conditions for arithmetic operations and data transfer instructions and for reflecting the status of operation results. Figure 7-18 shows an outline of the functions of the program status word.

Figure 7-18. Outline of Functions of the Program Status Word

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

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

7.7.3 Index Enable Flag (IXE)

The IXE flag is used to specify index modification on the data memory address when a data memory manipulation instruction is executed.

When the IXE flag is set to 1, an OR operation is performed on the data memory address and the index register (IX), and executes an instruction to the data memory with the result of the OR operation as the real address.

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

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

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

Table 7-3 shows how the CMP flag affects the setting and resetting of the Z flag.

Table 7-3. Zero Flag (Z) and Compare Flag (CMP)

Conditions	When CMP flag	When CMP flag is 1
When the result of an arithmetic operation is a value 0	Z ← 1	Z flag remains unchanged
When the result of an arithmetic operation is other than 0	Z ← 0	Z ← 0

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

Example of comparing 12-bit data

```
; Are 12-bit data stored in M001, M002, and M003 equal to 456H?
        CMP456:
                SET2    CMP, Z
                SUB     M001, #4    ; Data stored to M001, M002, and M003 are not lost
                SUB     M002, #5
                SUB     M003, #6
                ; CLR1  CMP         ; CMP is automatically cleared by bit judgement instruction
                SKT1    Z
                BR      DIFFER      ; ≠ 456 H
                BR      AGREE       ; = 456 H
```

7.7.5 Carry Flag (CY)

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

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

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

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

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

7.7.6 Binary-Coded Decimal Flag (BCD)

The BCD flag is used for BCD operations.

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

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

7.7.7 Notes Concerning Use of Arithmetic Operations

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

When an arithmetic operation is performed on the program status word, the result is stored in the program status word.

Below is an example.

Example

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

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

7.8 NOTES CONCERNING USE OF THE SYSTEM REGISTER

7.8.1 Reserved Words for the System Register

Because the system register is allocated in data memory, it can be used in any of the data memory manipulation instructions. As shown in Example 1 (using a 17K Series Assembler AS17K), because a data memory address can not be directly coded in an instruction operand, it needs to be defined as a symbol beforehand.

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

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

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

For information concerning reserved words, see **CHAPTER 20 ASSEMBLER RESERVED WORDS**.

```
Example 1.        MOV     34H, #0101B    ; Using a data memory address like 34H or 76H will cause an
                  MOV     76H, #1010B    ; error in  the assembler.
            M037  MEM 0.37H              ; Addresses in general data memory need to be defined as
                  MOV     M037, #0101B   ; symbols using the MEM directive.

        2.        MOV     AR1, #1010B    ; By using the reserved word AR1 (address 6H), there is no need
                                         ; to define the address as a symbol.
                                         ; Reserved word AR1 is defined in a device file with the directive
                                         ; “AR1 MEM 0.76H”.
```

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

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

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

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

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

Example 3.

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

Expanded macro

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

4.

```
SET1    BCD             ; Embedded macro
```

Expanded macro

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

```
CLR2 Z, CY              ; Identical address flag
```

Expanded macro

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

```
CLR2 Z, BCD             ; Different address flag
```

Expanded macro

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

7.8.2 Handling of System Register Addresses Fixed at 0

In dealing with system register area fixed at 0 (see **Figure 7-2**), there are a few points for which caution should be taken with regard to device, emulator and assembler operation.

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

(1) Concerning device operation

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

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

An error will be generated if a write instruction attempts to write 1 to an address fixed at 0.

Below is an example of the type of instructions that will cause the in-circuit emulator to generate an error.

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

        2.  MOV    IXL, #1111B     ;
            MOV    IXM, #1111B     ;
            MOV    IXH, #0001B     ; Attempts to write 1 to bit 0 (an address fixed at 0).
            ADD    IXL, #1         ;
            ADDC   IXM, #0         ;
            ADDC   IXH, #0         ; Attempts to write 1 to bit 0 (an address fixed at 0) as a result of
                                     operation.
```

However, when all valid bits are set to 1 as shown in Example 2, executing the instructions INC AR or INC IX will not cause an error to be generated by the in-circuit emulator. This is because when all valid bits of the address register and index register are set to 1, executing the INC instruction causes all bits to be set to 0. The only time the in-circuit emulator will not generate an error when an attempt is made to write the value 1 to an address fixed at 0 is when the address being written to is in the address register.

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

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

MOV BANK, #0100B

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

The following is the reason why an error is not generated in the assembler: the assembler does not know what data memory address is the object of the data memory manipulation instruction being executed.

The assembler generates an error only when the value n in the embedded macro BANKn is a value greater than 2:

This is because the assembler judges that embedded macros other than BANK0 and 1 cannot be used in the μPD17134A subseries.

[MEMO]

CHAPTER 8 GENERAL REGISTER (GR)

The general register (GR) is allocated in data memory. It can therefore be used directly for arithmetic operations and transferring data.

8.1 GENERAL REGISTER CONFIGURATION

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

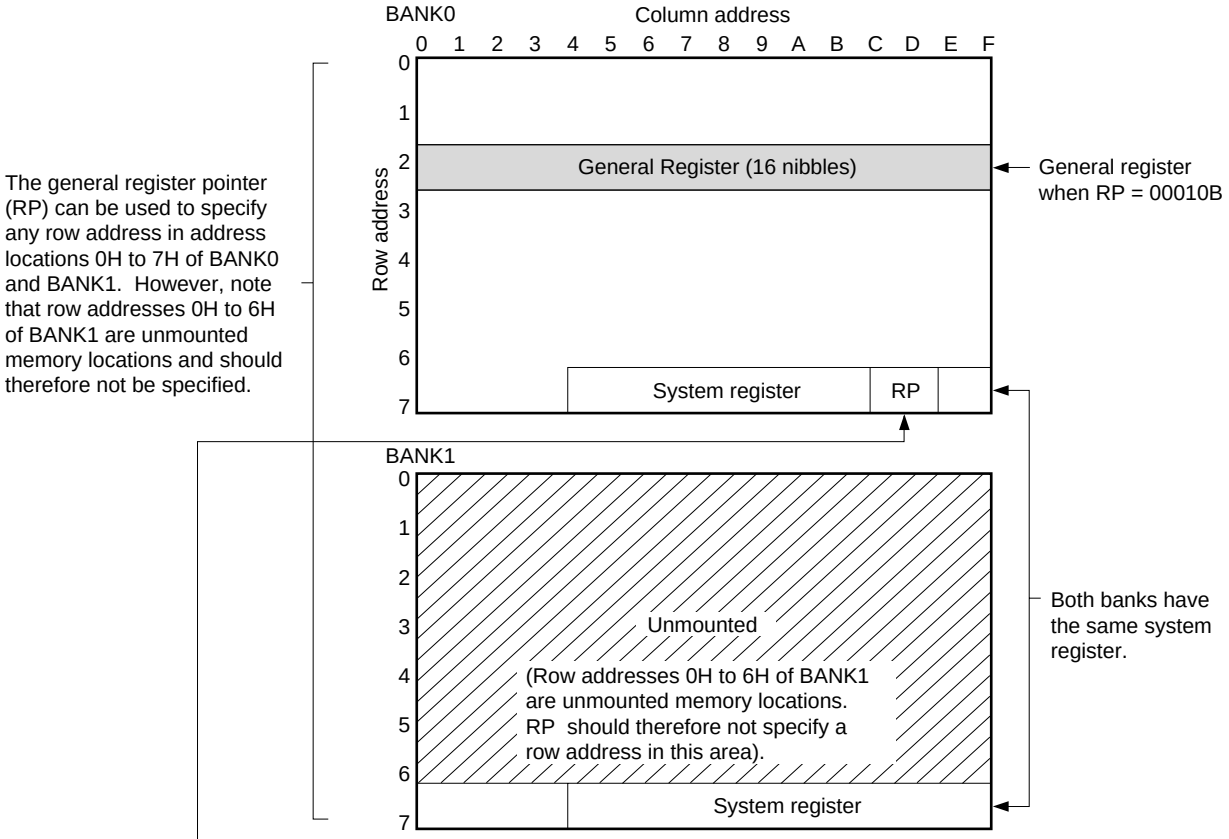
As shown in Figure 8-1, 16 nibbles in a single row address in data memory (16 × 4 bits) are used as the general register.

The register pointer (RP) in the system register is used to indicate which row address is to be used as the general register. Since the general register pointer effectively has four valid bits, the data memory row addresses in which the general register can be allocated are addresses 0H to 7H of BANK0 and BANK1. However, note that row addresses 0H to 6H of BANK1 are unmounted area and should therefore not be specified as locations for the general register.

8.2 FUNCTIONS OF THE GENERAL REGISTER

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

Figure 8-1. General Register Configuration



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

CHAPTER 9 REGISTER FILE (RF)

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

9.1 REGISTER FILE CONFIGURATION

9.1.1 Configuration of the Register File

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

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

In the same way as with data memory, the register file is divided into addresses in 4-bit units. It has a total of 128 nibbles specified in row addresses from 0H to 7H and column addresses from 0H to 0FH.

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

9.1.2 Relationship between the Register File and Data Memory

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

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

This means that the same memory exists in register file addresses 40H to 7FH and in data memory bank addresses 40H to 7FH.

Assuming that the current bank is BANK0, register file addresses 40H to 7FH are equivalent to addresses 40H to 7FH of BANK0 in data memory. When the current bank is BANK1, register file addresses 40H to 7FH are equivalent to address 40H to 7FH of BANK1 in data memory.

Figure 9-1. Register File Configuration

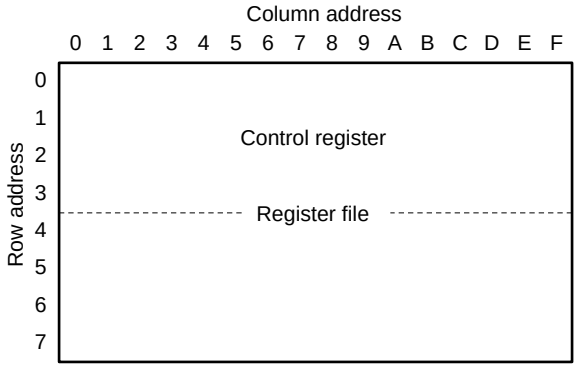
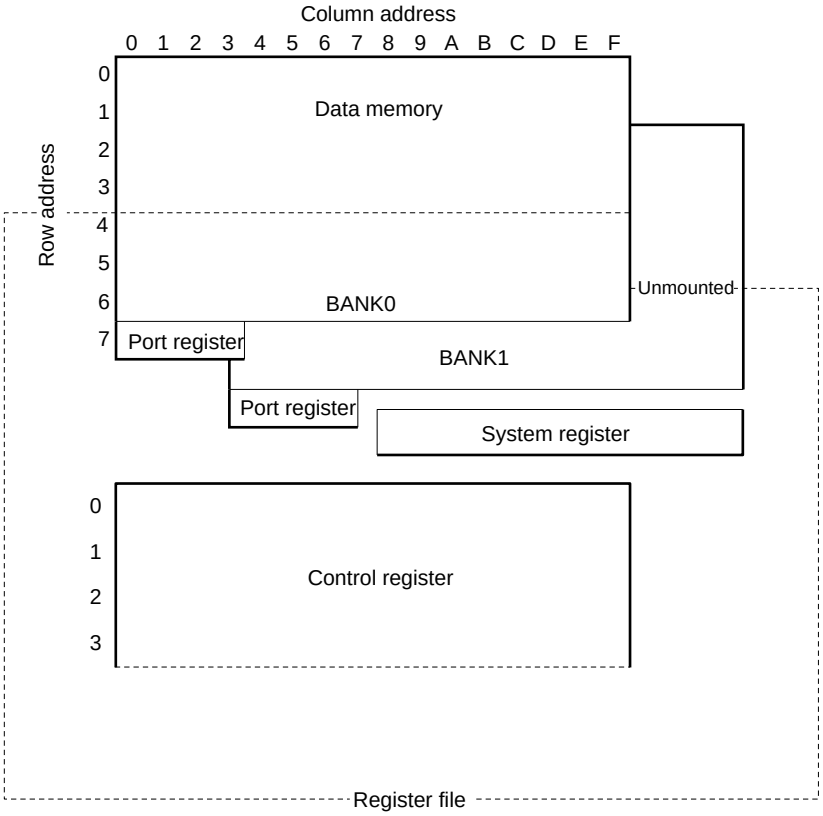


Figure 9-2. Relationship Between the Register File and Data Memory



9.2 FUNCTIONS OF THE REGISTER FILE

9.2.1 Functions of the Register File

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

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

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

9.2.2 Functions of Control Register

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

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

- Stack pointer (SP)
- Power-down reset
- 8-bit timer counter (TM0, TM1)
- AC zero cross detector (ZCROSS)
- A/D converter
- Basic interval timer (BTM)
- Ports
- Interrupt functions
- Serial interface (SIO)

9.2.3 Register File Manipulation Instructions

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

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

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

Below is an example using the PEEK and POKE instructions.

```
Example     RF02    MEM0.82H    ; Symbol definition
            RF1F    MEM0.9FH    ; Register file addresses 00H to 3FH must be defined with
            RF53    MEM0.53H    ; symbols as BANK0 addresses 80H to BFH.
            RF6D    MEM0.6DH    ; See 9.4 NOTES CONCERNING USE OF THE REGISTER FILE
            RF70    MEM1.70H    ; for details.
            RF71    MEM1.71H    ;
            BANK0
        ①   PEEK    WR, RF02    ;
        ②   POKE    RF1F, WR    ;
        ③   PEEK    WR, RF53    ;
        ④   POKE    RF6D, WR    ;
            BANK1                ;
        ⑤   PEEK    WR, RF02    ;
        ⑥   POKE    RF1F, WR    ;
        ⑦   PEEK    WR, RF70    ;
★       ⑧   POKE    RF72, WR    ;
```

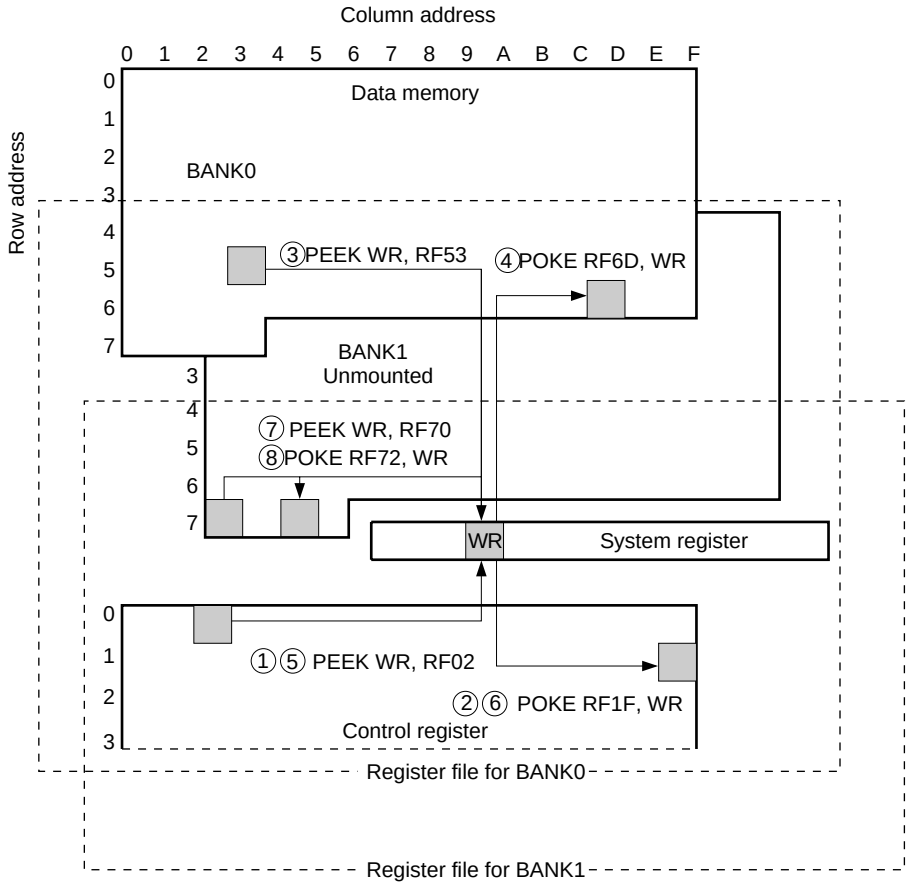
Figure 9-3 shows an example of register file operation.

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

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

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

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



9.3 CONTROL REGISTER

Figure 9-4 shows the configuration of the control register.

As shown in Figure 9-4, the control register consists of 64 nibbles (64 × 4 bits) allocated in register file addresses 00H to 3FH.

However, only 26 nibbles are actually used. The remaining 38 nibbles are allocated for registers which have not yet been implemented. Data should not be read from or written to this area.

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

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

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

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

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

Attempting to read data in the unused register address area (38 nibbles) will yield unpredictable values. In addition, attempting to write to this area has no effect.

9.4 NOTES CONCERNING USE OF THE REGISTER FILE

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

It is necessary to take note of the following notes concerning device operation and use of the 17K Series assembler (AS17K) and in-circuit emulator (IE-17K or IE-17K-ET) with regard to the read-only (R) and unused registers in the control register (register file addresses 00H to 3FH).

(1) Device operation

Writing to a read-only register has no effect.

Attempting to read data from an address in the unused data area will yield an undefined value. Attempting to write to an address in the unused data area has no effect.

(2) During use of the assembler (AS17K)

An error will be generated if an attempt is made to write to a read-only register.

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

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

Attempting to write to a read-only register has no effect. No error is generated.

Attempting to read data from an address in the unused data area will yield an undefined value. Attempting to write to an address in the unused data area has no effect. No errors are generated.

9.4.2 Register File Symbol Definitions and Reserved Words

Attempting to use a numerical value in a 17K Series assembler (AS17K) to specify a register file address in the rf operand of the “PEEK WR, rf” or “POKE rf, WR” instructions will cause an error to be generated.

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

Example 1. Case which causes an error to be generated

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

Case in which no error is generated

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

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

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

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

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

Example 2.

```
RF71    MEM          1.71H    ; Register file overlapping with data memory
RF02    MEM          0.82H    ; Control register

        BANK0
        PEEK    WR, RF71      ; RF71 becomes address 71H in BANK0.
        PEEK    WR, RF02      ; RF02 becomes address 02H in the control register.

        BANK1
        PEEK    WR, RF71      ; RF71 becomes address 71H in BANK1.
        PEEK    WR, RF02      ; RF02 becomes address 02H in the control register.
```

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

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

By using these embedded macro instructions, the contents of the register file can be manipulated in 1-bit unit.

Due to the fact that most of control register consists of 1-bit flags, the assembler (AS17K) has reserved words for use with these flags.

However, note that there is no reserved word for the stack pointer for its use as a flag. The only reserved word used for the stack pointer is the reserved word “SP”, for its use as data memory. For this reason, none of the above flag manipulation instructions can be used with the stack pointer.

Figure 9-4. Control Register Configuration (1/2)

Column address / Row address	Item	0				1				2				3				4				5				6				7			
0 (8)	Symbol					0	SP			SIOTS	SIOHIZ	SIOCK1	SIOCK0	WDTRES	0	0	WDTEN																
	When reset					0	1	0	1	0	0	0	0	0	0	0	0																
	Read/ Write					R/W				R/W				R/W																			
1 (9)	Symbol	0	0	0	PDRESEN	TM0EN	TM0RES	TM0CK1	TM0CK0	TM1EN	TM1RES	TM1CK1	TM1CK0	BTMISEL	BTMRES	BTMCK1	BTMCK0																
	When reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0																
	Read/ Write	R/W				R/W				R/W				R/W																			
2 (A)	Symbol	0	0	0	ADCSTRT	ADCSOFT	0	ADCCMP	ADCEND	ADCCH3	ADCCH2	ADCCH1	ADCCH0																				
	When reset	0	0	0	0	0	0	0	0	0	0	0	0																				
	Read/ Write	R/W				R/W		R		R/W																							
3 (B)	Symbol																																
	When reset																																
	Read/ Write																																

Remark The address in parentheses apply when the AS17K assembler is used.
The names of all the flags in the control registers are assembler reserved words saved in the device file. Using these reserved words is useful in programming. (See **CHAPTER 20 ASSEMBLER RESERVED WORDS**.)

Figure 9-4. Control Register Configuration (2/2)

8	9	A	B				C				D				E				F			
			TM0OSEL	0	0	SIOEN	0	0	P0BGPU	P0AGPU									0	0	0	INT
			0	0	0	0	0	0	0	0									0	0	0	Note
			R/W				R/W												R			
			P0C3IDI	P0C2IDI	P0C1IDI	P0C0IDI	P0CBIO3	P0CBIO2	P0CBIO1	P0CBIO0	0	0	0	ZCROSS					0	0	IEGMD1	IEGMD0
			0	0	0	0	0	0	0	0	0	0	0	0					0	0	0	0
			R/W				R/W				R/W								R/W			
			P0DBIO3	P0DBIO2	P0DBIO1	P0DBIO0	0	P1AGIO	P0BGIO	P0AGIO					0	0	0	IPSIO	IPBTM	IPTM1	IPTM0	IP
			0	0	0	0	0	0	0	0					0	0	0	0	0	0	0	0
			R/W				R/W								R/W				R/W			
			0	0	0	IRQSIO	0	0	0	IRQBTM	0	0	0	IRQTM1	0	0	0	IRQTM0	0	0	0	IRQ
			0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
			R/W				R/W				R/W				R/W				R/W			

Note The value of the INT flag changes every moment according to the status of the INT pin.

CHAPTER 10 DATA BUFFER (DBF)

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

The data buffer acts as a data storage area for the CPU peripheral circuit (address register, serial interface, timer 0, timer1, basic internal timer, and A/D converter) through use of the GET and PUT instructions. It also acts as data storage used for receiving and transferring data. By using the MOVT, DBF, and @AR instructions, fixed data in program memory can be read into the data buffer.

10.1 DATA BUFFER CONFIGURATION

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

As shown in Figure 10-1, the data buffer is allocated in address locations 0CH to 0FH in BANK0 and consists of a total of 16 bits (4 × 4 bits).

Figure 10-1. Allocation of the Data Buffer

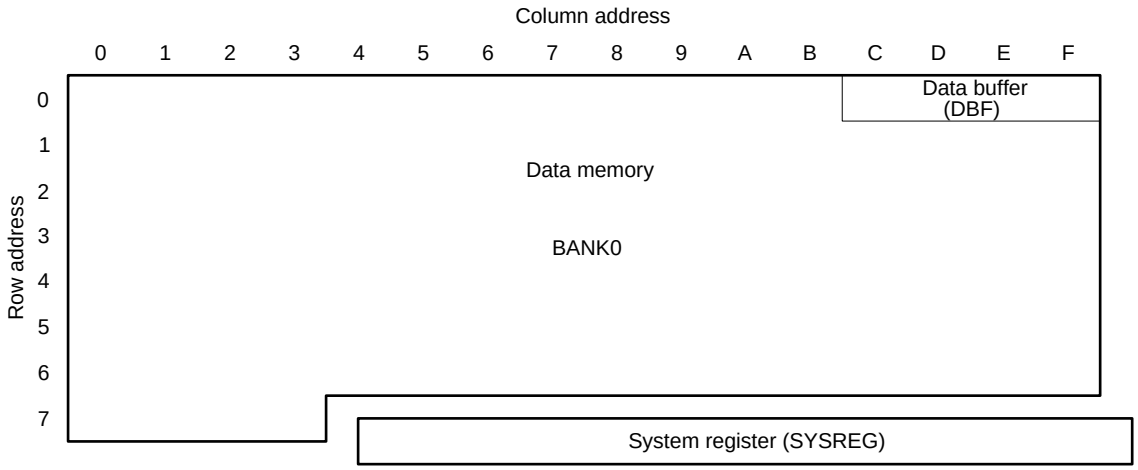


Figure 10-2 shows the configuration of the data buffer. As shown in Figure 10-2, the data buffer is made up of 16 bits with its LSB in bit 0 of address 0FH and its MSB in bit 3 of address 0CH.

Figure 10-2. Data Buffer Configuration

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

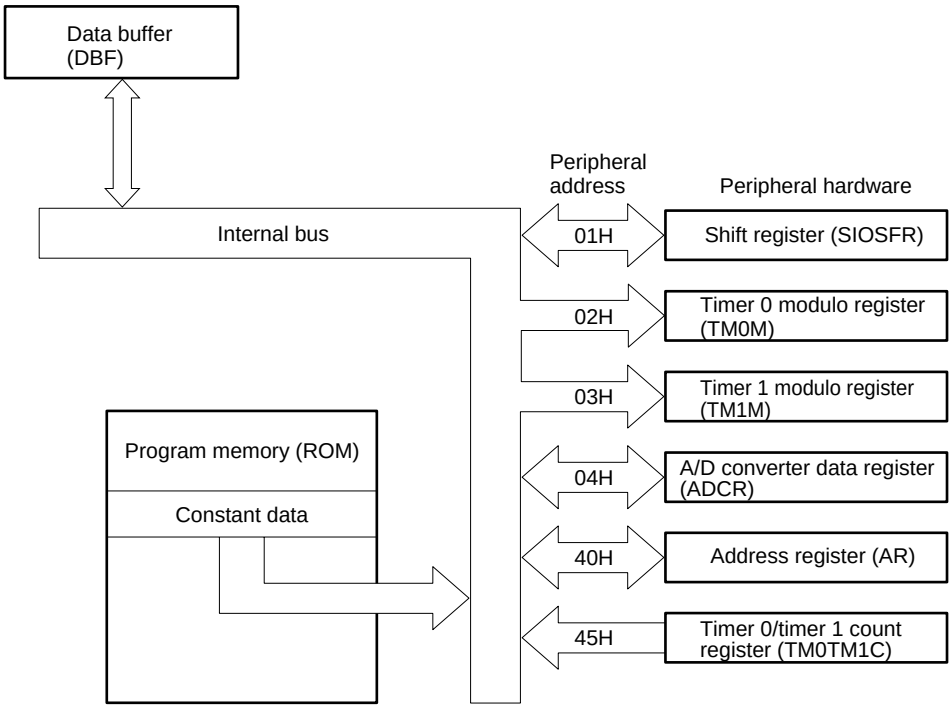
Because the data buffer is allocated in data memory, it can be used in any of the data memory manipulation instructions.

10.2 FUNCTIONS OF THE DATA BUFFER

The data buffer has two separate functions.

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

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



10.2.1 Data Buffer and Peripheral Hardware

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

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

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

PUT p, DBF: Write the data in the data buffer to the peripheral hardware address specified by p.

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

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

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

Table 10-1. Peripheral Hardware

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

Peripheral address	Name	Peripheral hardware	Direction of data		Actual bit length
			PUT	GET	
01H	SIOSFR	Serial interface	○	○	8 bits
02H	TM0M	Timer 0	○	×	8 bits
03H	TM1M	Timer 1	○	×	8 bits
04H	ADCR	A/D converter	○	○	8 bits

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

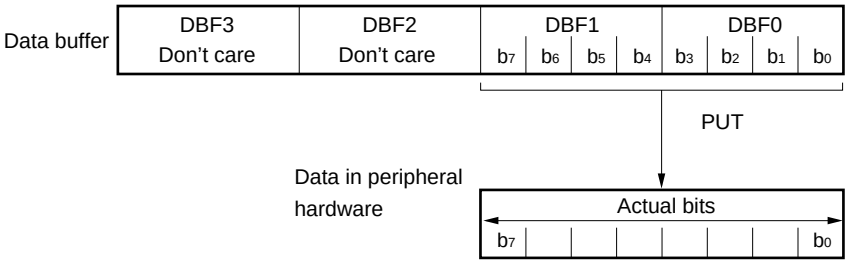
Peripheral address	Name	Peripheral hardware	Direction of data		Actual bit length
			PUT	GET	
40H	AR	Address register	○	○	10/11 bits[Note]
45H	TM0TM1C	Timer 0/timer 1 count register	×	○	16 bits

Note 10 bits for the μPD17134A and 17135A, and 11 bits for the μPD17136A and 17137A.

10.2.2 Data Transfer with Peripheral Hardware

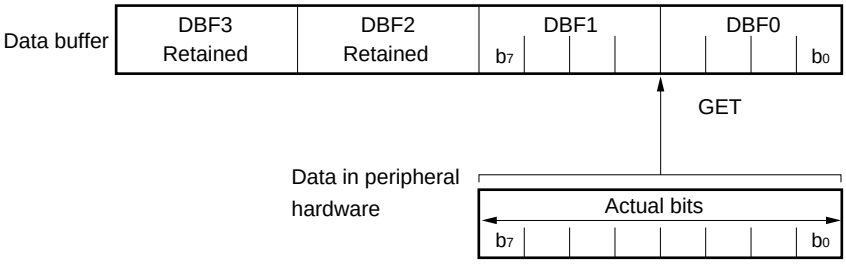
Data can be transferred between the data buffer and peripheral hardware in 8- or 16-bit units. Instruction execution time for a single PUT or GET instruction is the same regardless of whether 8 or 16 bits are being transferred.

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



When only 8 bits of data are being written from the data buffer, the high-order 8 bits (DBF3, DBF2) are "don't care" (any value can be written).

2. GET instruction (when the actual bits in peripheral hardware are the 8 bits from 0 to 7)



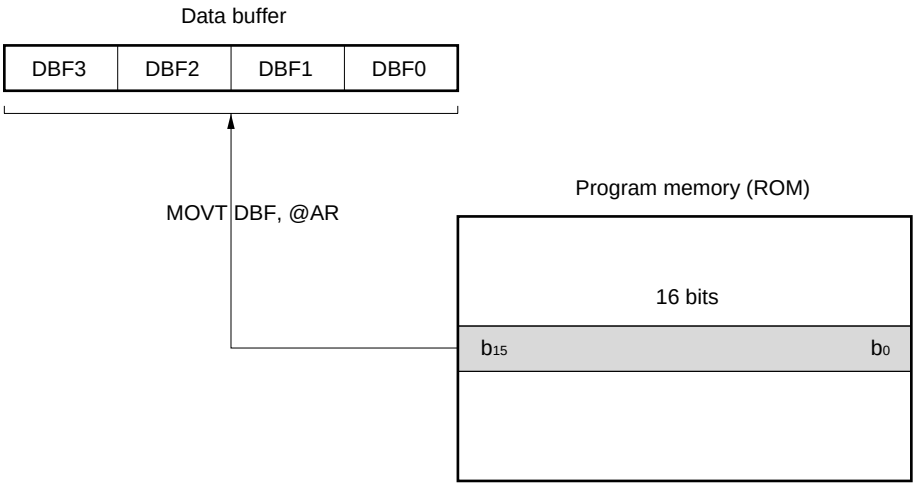
When 8 bits of data are being read into the data buffer, the values in the high-order 8 bits (DBF3, DBF2) remain unchanged.

10.2.3 Table Reference

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

The MOVT instruction is explained below.

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



[MEMO]

CHAPTER 11 ARITHMETIC AND LOGIC UNIT (ALU)

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

11.1 ALU BLOCK CONFIGURATION

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

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

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

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

11.2 FUNCTIONS OF THE ALU BLOCK

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

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

11.2.1 Functions of the ALU

The arithmetic operations consist of addition and subtraction. Arithmetic operations can be performed on the contents of the general register and data memory or on immediate data and the contents of data memory. Operations in binary are performed on 4 bits of data and operations in decimal are performed on one place (BCD operation).

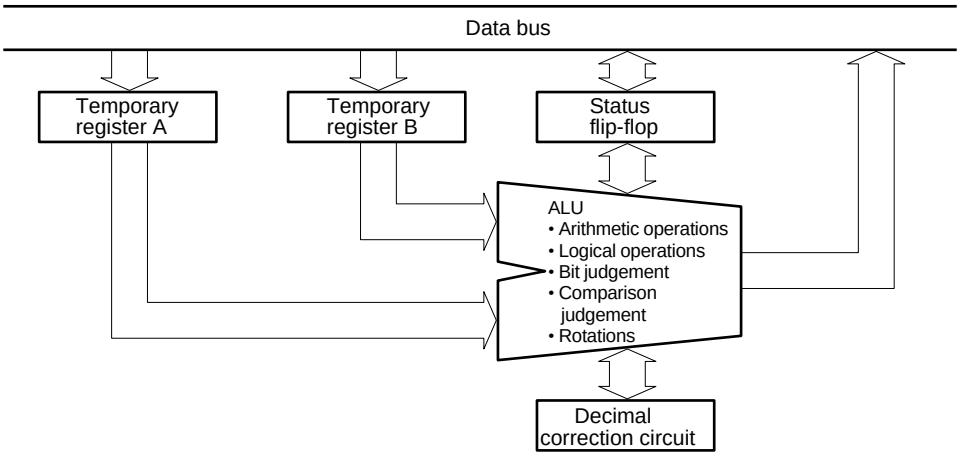
Logical operations include ANDing, ORing, and XORing. Their operands can be general register contents and data memory contents, or data memory contents and immediate data.

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

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

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

Figure 11-1. ALU Configuration



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

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

[MEMO]

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

ALU function		Instruction	Operation	Explanation
Arithmetic operations	Addition	ADD r, m	(r) ← (r) + (m)	Adds contents of general register and data memory. Result is stored in general register.
		ADD m, #n4	(m) ← (m) + n4	Adds immediate data to contents of data memory. Result is stored in data memory.
		ADDC r, m	(r) ← (r) + (m) + CY	Adds contents of general register, data memory and carry flag. Result is stored in general register.
		ADDC m, #n4	(m) ← (m) + n4 + CY	Adds immediate data, contents of data memory and carry flag. Result is stored in data memory.
	Subtraction	SUB r, m	(r) ← (r) – (m)	Subtracts contents of data memory from contents of general register. Result is stored in general register.
		SUB m, #n4	(m) ← (m) – n4	Subtracts immediate data from data memory. Result is stored in data memory.
		SUBC r, m	(r) ← (r) – (m) – CY	Subtracts contents of data memory and carry flag from contents of general register. Result is stored in general register.
		SUBC m, #n4	(m) ← (m) – n4 – CY	Subtracts immediate data and carry flag from data memory. Result is stored in data memory.
Logical operations	Logical OR	OR r, m	(r) ← (r) ∨ (m)	OR operation is performed on contents of general register and data memory. Result is stored in general register.
		OR m, #n4	(m) ← (m) ∨ n4	OR operation is performed on immediate data and contents of data memory. Result is stored in data memory.
	Logical AND	AND r, m	(r) ← (r) ∧ (m)	AND operation is performed on contents of general register and data memory. Result is stored in general register.
		AND m, #n4	(m) ← (m) ∧ n4	AND operation is performed on immediate data and contents of data memory. Result is stored in data memory.
	Logical XOR	XOR r, m	(r) ← (r) ∀ (m)	XOR operation is performed on contents of general register and data memory. Result is stored in general register.
		XOR m, #n4	(m) ← (m) ∀ n4	XOR operation is performed on immediate data and contents of data memory. Result is stored in data memory.
Bit Judgement	True	SKT m, #n	CMP ← 0, if (m) ∧ n=n, then skip	Skips next instruction if all bits in data memory specified by n are TRUE (1). Result is not stored.
	False	SKF m, #n	CMP ← 0, if (m) ∧ n=0, then skip	Skips next instruction if all bits in data memory specified by n are FALSE (0). Result is not stored.
Comparison judgement	Equal	SKE m, #n4	(m) – n4, skip if zero	Skips next instruction if immediate data equals contents of data memory. Result is not stored.
	Not equal	SKNE m, #n4	(m) – n4, skip if not zero	Skips next instruction if immediate data is not equal to contents of data memory. Result is not stored.
	≥	SKGE m, #n4	(m) – n4, skip if not borrow	Skips next instruction if contents of data memory is greater than or equal to immediate data. Result is not stored.
	<	SKLT m, #n4	(m) – n4, skip if borrow	Skips next instruction if contents of data memory is less than immediate data. Result is not stored.
Rotation	Rotate to the right	RORC r	$\rightarrow CY \rightarrow (r)_{b3} \rightarrow (r)_{b2} \rightarrow (r)_{b1} \rightarrow (r)_{b0}$	Rotate contents of the general register along with the CY flag to the right. Result is stored in general register.

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

ALU Function	Difference in operation because of program status word (PSWORD)					
Arithmetic operation	Value of BCD flag	Value of CMP flag	Operation	CY flag	Z flag	Modification when IXE = 1
	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	Dont'care (retained)	Don't care (retained)	Not affected	Don't care (retained)	Don't care (retained)	Executed
Bit judgement	Dont'care (retained)	Reset	Not affected	Don't care (retained)	Don't care (retained)	Executed
Comparison	Dont'care (retained)	Don't care (retained)	Not affected	Don't care (retained)	Don't care (retained)	Executed
Rotation	Dont'care (retained)	Don't care (retained)	Not affected	Value of b_0 of general register	Don't care (retained)	Executed

11.2.2 Functions of Temporary Registers A and B

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

11.2.3 Functions of the Status Flip-flop

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

(1) Z flag

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

(i) When CMP = 0

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

(ii) When CMP = 1

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

(2) CY flag

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

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

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

Only affected by arithmetic operations and rotations.

(3) CMP flag

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

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

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

(4) BCD flag

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

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

These flags can also be set through direct manipulation of the values in the program status word. At this time, the corresponding flag in the status flip-flop is also manipulated.

11.2.4 Operations in 4-Bit Binary

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

11.2.5 Operations in BCD

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

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

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

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

11.2.6 Operations in the ALU Block

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

The first data operand is 4-bit data used to specify the contents of an address in the general register or data memory. The second data operand is 4-bit data used to either specify the contents of an address in data memory or to be used as an immediate value. For example, in the instruction

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

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

ADD m, #n4

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

RORC r

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

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

Caution should be taken with regard to the following points:

(1) Arithmetic operations are affected by the CMP and BCD flags in the program status word.
(2) Logical operations are not affected by the CMP or BCD flag in the program status word. Logical operations do not affect the Z or CY flags.
(3) Bit judgement causes the CMP flag in the program status word to be reset.
(4) When an arithmetic operation, logical operation, bit judgement, comparison judgement, or rotation is being executed and the IXE flag in the program status word is set (1), address modification is performed using the index register.

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

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

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

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

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

Table 11-3. Types of Arithmetic Operations

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

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

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

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

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

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

Addition and subtraction are performed in 4-bit binary.

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

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

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

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

BCD operations are performed.

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

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

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

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

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

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

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

0, 0000B to 1, 0011B
CY CY

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

1, 0110B to 1, 1111B
CY CY

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

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

BCD operations are performed.

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

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

Example

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

By executing the instructions in steps numbered (1), (2), and (3), the 12 bits in memory locations M1, M2, and M3 and the immediate data (321) can be compared in decimal.

11.3.5 Notes Concerning Use of Arithmetic Operations

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

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

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

11.4 LOGICAL OPERATIONS

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

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

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

Table 11-4. Logical Operations

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

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

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

11.5 BIT JUDGEMENTS

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

The TRUE (1) and FALSE (0) bit judgements use SKT and SKF instruction, respectively

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

Bit judgements are not affected by the BCD flag in the program status word (PSWORD) and bit judgements do not cause either the CY or Z flag in the program status word (PSWORD) to be set. However, when an SKT or SKF instruction is executed, the CMP flag is reset (0). When the index enable flag (IXE) is set (1), index modification is performed using the index register. For information concerning index modification using the index register, see **CHAPTER 7 SYSTEM REGISTER (SYSREG)**.

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

Table 11-6. Bit Judgement Instructions

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

11.5.1 TRUE (1) Bit Judgement

The TRUE (1) bit judgement instruction (SKT m, #n) is used to determine whether or not the bits specified by n in the 4 bits of data memory m are TRUE (1). When all bits specified by n are TRUE (1), this instruction causes the next instruction to be skipped.

Example

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

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

11.5.2 FALSE (0) Bit Judgement

The FALSE (0) bit judgement instruction (SKF m, #n) is used to determine whether or not the bits specified by n in the 4 bits of data memory m are FALSE (0). When all bits specified by n are FALSE (0), this instruction causes the next instruction to be skipped.

Example

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

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

11.6 COMPARISON JUDGEMENTS

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

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

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

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

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

Table 11-7. Comparison Judgement Instructions

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

11.6.1 “Equal to” Judgement

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

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

Example

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

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

11.6.2 "Not Equal to" Judgement

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

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

Example

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

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

11.6.3 "Greater Than or Equal to" Judgement

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

Example

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

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

11.6.4 "Less Than" Judgement

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

Example

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

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

11.7 ROTATIONS

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

The RORC instruction is used for rotation to the right.

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

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

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

11.7.1 and 11.7.2 explain rotation.

11.7.1 Rotation to the Right

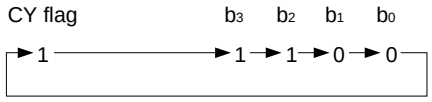
The instruction used for rotation to the right (RORC r) rotates the contents of the general register in the direction of its least significant bit.

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

Example 1.

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

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



Basically, when rotation to the right is performed, the following operation is executed:

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

2.

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

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

11.7.2 Rotation to the Left

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

Example

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

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

CHAPTER 12 PORTS

12.1 PORT 0A ($P0A_0$, $P0A_1$, $P0A_2$, $P0A_3$)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

If P0CnIDI is 0 (n = 0 to 3), the P0Cn/ADCn pin functions as a port. If P0CnIDI is 1 (n = 0 to 3), the P0Cn/ADCn pin functions as the analog input pin of the A/D converter.

ADCCH0 and ADCCH1 (bits 1 and 0 at address 22H) in the register file are used to select the input pin for A/D conversion.

To use P0C pins as A/D converter input pins, set P0CBIOn = 0 so that they are set as input ports. (See **13.3 A/D CONVERTER**.)

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

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

(n = 0 to 3)

P0CnIDI RF: 1BH	P0CBIOn RF: 1CH	Function	BANK0 72H	
			Write	Read
0	0	Input port	Possible P0C latch	Pin status
	1	Port output	Possible P0C latch	P0C latch contents
1	0	A/D converter analog input[Note1]	Possible P0C latch	P0C latch contents
	1	Output port and A/D converter analog input[Note2]	Possible P0C latch	P0C latch contents

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

2. Functions as an output port. At this time, the analog input voltage changes affected by the port output. When using this pin as an analog input pin, be sure to set P0CBIOn to 0.

12.4 PORT 0D ($P0D_0/\overline{SCK}$, $P0D_1$/SO, $P0D_2$/SI, $P0D_3/\overline{TM0OUT}$)

Port 0D is a 4-bit input/output port with an output latch. It is mapped into address 73H of BANK0 in data memory. The output format is N-ch open-drain output. The mask option can be used to specify that a pin contain a pull-up resistor in 1-bit unit.

Input or output can be specified in 1-bit unit. Input/output is specified with P0DBIO0 to P0DBIO3 (address 2BH) in the register file.

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

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

Port 0D can also be used for serial interface input/output or timer 0 output. SIOEN (0BH bit 0) in the register file is used to switch ports ($P0D_0$ to $P0D_2$) to serial interface input/output ($\overline{SCK}$, SO, SI) and vice versa. TM0OSEL (bit 3 at address 0BH) in the register file is used to switch a port ($P0D_3$) to timer 0 output ($\overline{TM0OUT}$) and vice versa. If TM0OSEL = 1 is selected, 1 is output at timer 0 reset. This output is inverted every time a timer 0 count value matches the modulo register contents.

Table 12-4. Register File Contents and Pin Functions

(n = 0 to 3)

Register file value			Pin function			
TM0OSEL RF: 0BH Bit 3	SIOEN RF: 0BH Bit 0	P0DBIOn RF: 2BH Bit n	$P0D_0/\overline{SCK}$	$P0D_1$/SO	$P0D_2$/SI	$P0D_3/\overline{TM0OUT}$
0	0	0	Input port			
		1	Output port			
	1	0	$\overline{SCK}$	SO	SI	Input port
		1				Output port
1	0	0	Input port			$\overline{TM0OUT}$
		1	Output port			
	1	0	$\overline{SCK}$	SO	SI	
		1				

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

Port mode		Contents read from the port register (0.73H)
Input port		Pin status
Output port		Output latch contents
$\overline{SCK}$	An internal clock is selected as a shift clock.	Output latch contents
	An external clock is selected as a shift clock.	Pin status
SI		Pin status
SO		Not defined
$\overline{TM0OUT}$		Output latch contents

Caution Using the serial interface causes the output latch for the P0D1/SO pin to be affected by the contents of the SIOSFR (shift register). So, reset the output latch before using the pin as output port.

12.5 PORT 1A ($P1A_0$, $P1A_1$, $P1A_2$, $P1A_3$)

Port 1A is a 4-bit input/output port with an output latch. It is mapped into address 70H of BANK1 in data memory. The output format is N-ch open-drain output. The mask option can be used to specify that a pin contain a pull-up resistor in 1-bit unit.

Input or output can be specified in 4-bit units. Input/output is specified by P1AGIO (bit 2 at address 2CH) in the register file.

When P1AGIO is 0, each pin of port 1A is used as input port. If a read instruction is executed for the port register, pin statuses are read. When P1AGIO is 1, each pin of port 1A is used as output port and the contents written in the output latch are output to pins. If a read instruction is executed when pins are output ports, the contents of the output latch, rather than pin statuses, are fetched.

At reset, P1AGIO is set to 0 and all P1A pins become input ports. The contents of the port output latch are 0.

Table 12-6. Writing into and Reading from the Port Register (1.70H)

(n = 0 to 3)

P1AGIOn RF: 2CH, bit 2	Pin input/output	BANK1 70H	
		Write	Read
0	Input	Possible	P1A pin status
1	Output	Write to the P1A latch	P1A latch contents

12.6 PORT 1B ($P1B_0$)

Port 1B is a 1-bit input-dedicated port. It is mapped into address 71H of BANK1 in data memory. The mask option can be used to specify that pull-up resistors be contained in $P1B_0$ pins.

Port 1B is the input-dedicated port. At reading, only the least significant bit is valid and a value is read into it. At writing, no value changes. Value 0 is always read into the high-order 3 bits of the port register.

12.7 PORT CONTROL REGISTER

12.7.1 Input/Output Switching by Group I/O

Ports which switch input/output in 4-bit unit are called group I/O. Port 0A, port 0B, and port 1A are used as group I/O. The register shown in the figure below is used for input/output switching.

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

RF: 2CH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	P1AGIO	P0BGIO	P0AGIO
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

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

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

P1AGIO	Function
0	Sets port 1A to input mode.
1	Sets port 1A to output mode.

12.7.2 Input/Output Switching by Bit I/O

Ports which switch input/output in 1-bit unit are called bit I/O. Port 0C and port 0D are used as bit I/O. The register shown in the figure below is used for input/output switching.

Figure 12-2. Port Control Register of Bit I/O (1/2)

RF: 1CH

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

Read = R, write = W

P0CBIO0	Function
0	Sets $P0C_0$ to input mode.
1	Sets $P0C_0$ to output mode.

P0CBIO1	Function
0	Sets $P0C_1$ to input mode.
1	Sets $P0C_1$ to output mode.

P0CBIO2	Function
0	Sets $P0C_2$ to input mode.
1	Sets $P0C_2$ to output mode

P0CBIO3	Function
0	Sets $P0C_3$ to input mode.
1	Sets $P0C_3$ to output mode.

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

RF: 2BH

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

Read = R, write = W

P0DBIO0	Function
0	Sets $P0D_0$ to input mode.
1	Sets $P0D_0$ to output mode.

P0DBIO1	Function
0	Sets $P0D_1$ to input mode.
1	Sets $P0D_1$ to output mode.

P0DBIO2	Function
0	Sets $P0D_2$ to input mode.
1	Sets $P0D_2$ to output mode.

P0DBIO3	Function
0	Sets $P0D_3$ to input mode.
1	Sets $P0D_3$ to output mode.

12.7.3 Specifying Pull-Up Resistor Incorporation Using Software

Pull-up resistor incorporation can be specified in 4-bit units using P0AGPU and P0BGPU (address 0CH) in the register file.

Figure 12-3. Specifying Pull-Up Resistor Incorporation Using Software

RF: 0CH

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

Read = R, write = W

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

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

[MEMO]

13.1 8-BIT TIMERS/COUNTERS (TM0 AND TM1)

The μPD17134A subseries has two channels of 8-bit timers/counters: timer 0 (TM0) and timer 1 (TM1).

These two timers can be used in combination as a 16-bit timer by using the count up signal of timer 0 as the count pulse for timer 1.

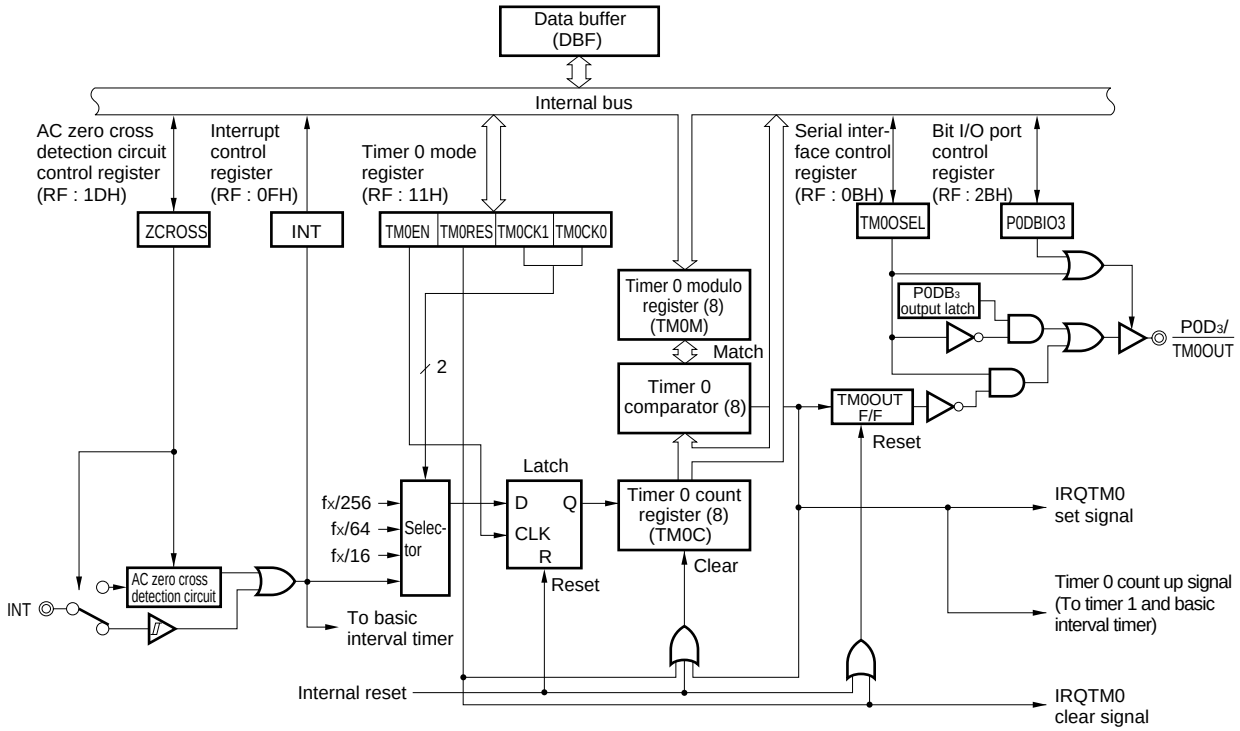
These timers are controlled by manipulating the hardware with the PUT/GET instruction and registers in the register file with the PEEK/POKE instruction.

13.1.1 8-Bit Timers/Counters Configuration

Figure 13-1 shows the configuration of the 8-bit timers/counters. An 8-bit timer/counter consists of an 8-bit count register, 8-bit modulo register, a comparator that compares the value of the count register and the value of the modulo register, and a selector that selects a count pulse.

Cautions 1. The modulo register is a write-only register.
2. The count register is a read-only register.

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



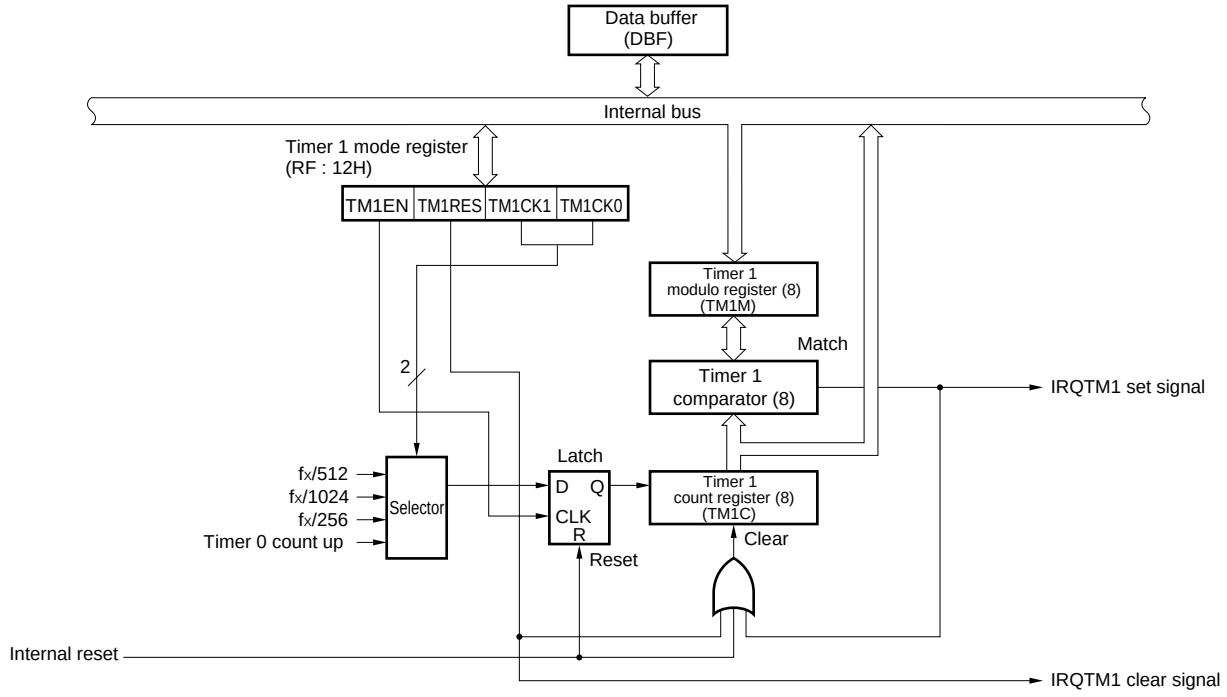


Figure 13-2. Timer 0 Mode Register

RF : 11H

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

Read = R, Write = W

TM0CK1	TM0CK0	Selects count pulse of timer 0
0	0	$f_X/256$
0	1	$f_X/64$
1	0	$f_X/16$
1	1	External clock from INT pin

TM0RES	Resets timer 0
0	Does not affect timer 0
1	Resets timer 0 count register and IRQTM0

Remark TM0RES is automatically cleared to 0 after it has been set to 1.
This bit is always 0 when it is read.

TM0EN	Starts timer 0
0	Stops counting by timer 0
1	Starts counting by timer 0

Remark TM0EN can be used as a status flag that detects the counting status of timer 0
(1: counting in progress, 0: counting is stopped).

Figure 13-3. Timer 1 Mode Register

RF : 12H

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

TM1CK1	TM1CK0	Selects count pulse of timer 1
0	0	$f_X/512$
0	1	$f_X/1024$
1	0	$f_X/256$
1	1	Count up signal from timer 0

TM1RES	Resets timer 1
0	Does not affect timer 1
1	Resets timer 1 count register and IRQTM 1

Remark TM1RES is automatically cleared to 0 after it has been set to 1. This bit is always 0 when it is read.

TM1EN	Starts timer 1
0	Stops counting by timer 1
1	Starts counting by timer 1

Remark TM1EN can be used as a status flag that detects the counting status of timer 0 (1: counting in progress, 0: counting is stopped).

13.1.2 Operation of 8-Bit Timers/Counters

(1) Count register

The count register of timers 0 and 1 is an 8-bit up counter whose initial value is 00H, and is incremented each time a count pulse has been input.

The count register is initialized to 00H in the following cases.

(1) When this product is reset (refer to **CHAPTER 17 RESET**).

(2) When the contents of the 8-bit modulo register and the value of the count register coincide, and the comparator generates a coincidence signal.

(3) In the case of timer 0, when “1” is written to TM0RES of the register file.
In the case of timer 1, when “1” is written to TM1RES of the register file.

(2) Modulo register

The modulo register of timers 0 and 1 determines the count value of the count register and its initial value is set to FFH.

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

(3) Comparator

The comparator of timers 0 and 1 outputs a coincidence signal when the value of the count register and the value of the modulo register coincide. If the value of the modulo register is the initial value FFH, for example, the comparator outputs the coincidence signal when the count register counts 256.

The coincidence signal output from the comparator clears the contents of the count register to 0, and automatically sets interrupt request flags (IRQTM0 and IRQTM1) to “1”. If the EI instruction (that enables accepting interrupts) is executed, and if the interrupt enable flags (IPTM0 and IPTM1) are set at this time, interrupts are accepted. When an interrupt has been accepted, the interrupt request flag (IRQTM0 or IRQTM1) is cleared to “0”, and program execution branches to a specified interrupt routine.

13.1.3 Selecting Count Pulse

The count pulse for timer 0 is selected by TM0CK0 and TM0CK1.

As the count pulse, a pulse resulting from dividing the system clock (f_X) by 256, 64, or 16, or an external count pulse input from the INT pin can be selected.

At reset, f_X/256 is selected as a count pulse because TM0CK0 = 0 and TM0CK1 = 0.

The count pulse for timer 1 is selected by TM1CK0 and TM1CK1.

As the count pulse, a pulse resulting from dividing f_X by 1024, 512, or 256, or the count up signal from timer 0 can be selected.

Timer 1 is also used to generate oscillation stabilization time on power application or at reset. Therefore, the initial values are TM1CK0 = 0 and TM1CK1 = 0, and f_X/512 is selected as the count pulse.

Because TM1EN = 1 as the initial condition, the μPD17134A subseries starts program execution from address 0000H after it has been reset at f_X = 8 MHz and after about 16.4 ms (about 65.5 ms at 2 MHz) (refer to **CHAPTER 17 RESET**).

13.1.4 Setting Count Value to Modulo Register

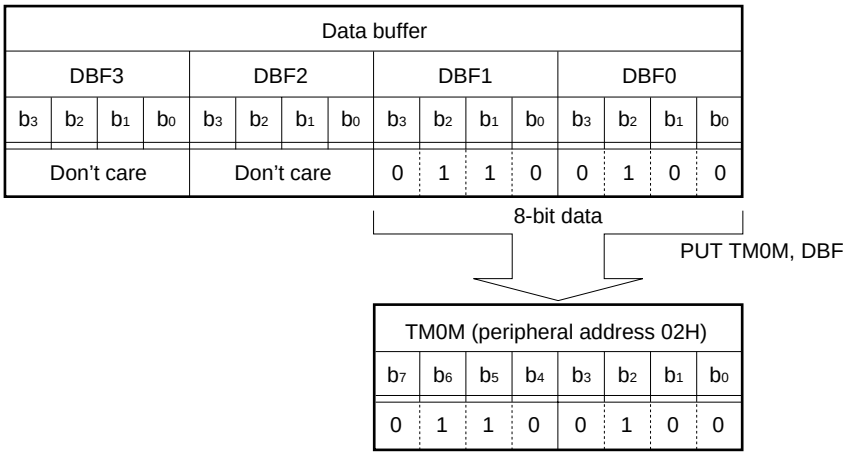
A value is set to the modulo register by using the PUT instruction via DBF (data buffer). The peripheral address of the modulo register of timer 0 is assigned to 02H, and that of timer 1 is assigned to 03H.

To transfer a value by using the PUT instruction, the data of the low-order 8 bits of DBF (DBF1 and DBF0) are transferred to the modulo register. Figure 13-4 shows an example of the modulo register of timer 0.

Figure 13-4. Setting Count Value to Modulo Register

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

```
CONTDATL  DAT   4H              ; Assigns CONTDATL to 4H by using symbol definition instruction
CONTDATH  DAT   6H              ; Assigns CONTDATH to 6H by using symbol definition instruction
          MOV   DBF0, #CONTDATL ;
          MOV   DBF1, #CONTDATH ;
          PUT   TM0M, DBF       ; Transfers data by using reserved word “TM0M”
```



Caution The range of the value to that can be set to the modulo register is 01H to FFH. If 00H is set, the normal count operation is not performed.

The modulo register is a write-only register. Therefore, the set value of the modulo register cannot be read. Even if the “PUT TM0M, DBF” or “PUT TM1M, DBF” instruction is executed while the 8-bit timer/counter is operating, the count is operating is not stopped.

13.1.5 Reading Value of Count Register

The values of the count registers of timers 0 and 1 are read simultaneously by using the GET instruction via DBF (data buffer).

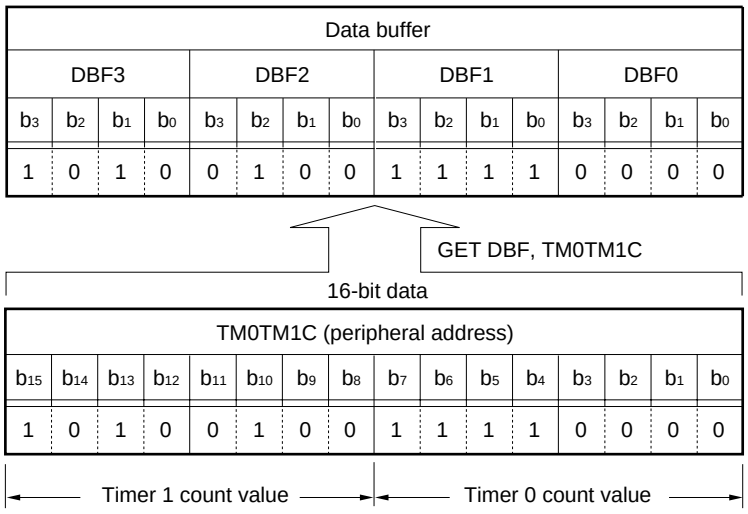
The values of the count registers of timers 0 and 1 are assigned to peripheral address 45H. The high-order 8 bits of this address are assigned to the count value of timer 1, and the low-order 8 bits are assigned to the count value of timer 0.

The values of the count registers can be read to DBF by using the GET instruction. While the GET instruction is being executed, the count registers stop counting and hold the current count value. If a count pulse is input to the timer while the timer is operating and the GET instruction is being executed, the count value is held, the value of the count register is incremented by one after the GET instruction has been executed, and the timer continues counting.

Therefore, the timer does not count erroneously even if the GET instruction is executed while the timer is operating, unless two or more count pulses are input to the timer in one instruction cycle.

Figure 13-5. Reading Count Value of Count Register

Example of using GET DBF, TM0TM1C; reserved word DBF and TM0TM1C when count value of timer 0 is F0H and count value of timer 1 is A4H



13.1.6 Setting of Interval Time

The time interval at which the comparator outputs the coincidence signal is determined by the value set to the modulo register. The set value N of the modulo register is calculated from interval time T [sec] as follows:

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

$$N = T \times f_{CP} - 1 \text{ or } N = \frac{T}{TCP} - 1 \text{ (where, N = 1 to 255)}$$

f_{CP} : Frequency of count pulse [Hz]
T_{CP} : Cycle of count pulse [sec] ($1/f_{CP}$ = resolution)

- **Example of calculating count value from interval time and program**

  - **Example of setting 7 ms to timer 1 as interval time (system clock: f_X = 8 MHz)**
    Suppose one wanted to set the interval timer to 7 ms. It is impossible to set an interval time of exactly 7 ms from an 8-MHz system clock. To set an interval time closest to 7 ms, therefore, calculate the count value by selecting a count pulse (f_X/256, resolution: 32 μs) at which the resolution is maximum.

    Example of calculation T = 7 ms, Resolution = 32 μs

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

$$= \frac{7 \times 10^{-3}}{32 \times 10^{-6}} - 1$$

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

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

Program example

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

INITFLG TM1EN, TM1RES, TM1CK1, NOT TM1CK0
                        ; Sets TM1EN and TM1RES, sets count pulse of timer 1 to “fx/256”, and starts
                        ; counting, by using embedded macroinstruction “INITFLG”
```

13.1.7 Error of Interval Time

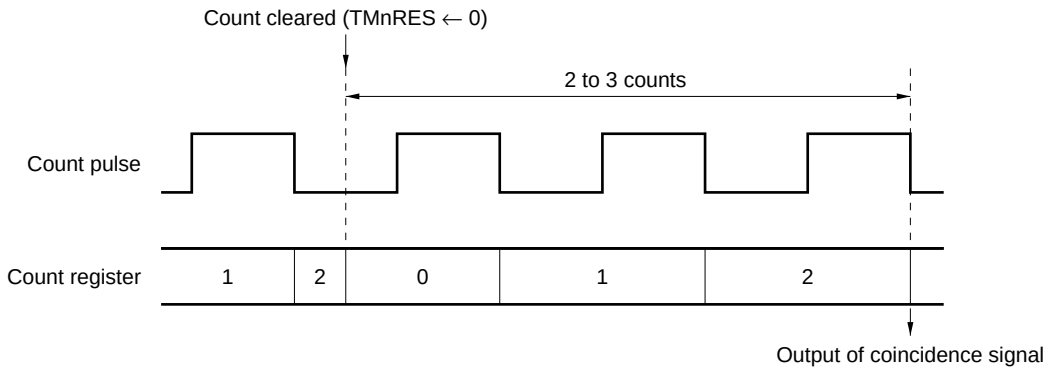
The interval time may include an error of up to –1.5 count, especially if the value set value of the modulo register is low.

(1) Error when count register is cleared to 0 during counting (maximum error: –1 count)

The count register of the 8-bit timer/counter is cleared to 0 when the TMnRES flag is set to 1. However, the divider circuit that generates a count pulse from the system clock is not reset.

Therefore, an error of 1 cycle of the count pulse may be generated at the first count if the TMnRES flag is set to 1 and the count value is cleared to 0 during counting. An example of counting where 2 is set to the modulo register is shown below.

Figure 13-6. Error When Count Register Is Cleared to 0 During Counting



In this example, the coincidence signal must be output each time the count value has reached 3. However, the coincidence signal is output when the count value reaches 2 for the first time after the count has been cleared.

The above error also occurs when TMnRES ← 1 at the same time as TMnEN = 1 ← 0.

(2) Error when counting is started from count stop status (maximum error: –1.5 count)

The count register of the 8-bit timer is cleared to 0 by setting the TMnRES flag to 1. However, the divider circuit that generates a count pulse from the system clock is not reset.

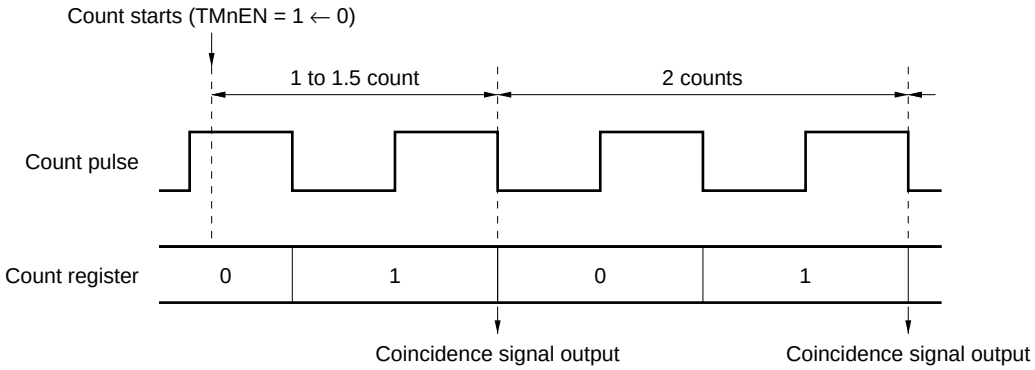
If the TMnEN flag is set to 1 and counting is started from the count stop status, the timing of the first counting differs as follows depending on whether the count pulse starts with a low level or a high level.

If count pulse starts with high level: First count at the next rising
If count pulse starts with low level: First count on starting of counting

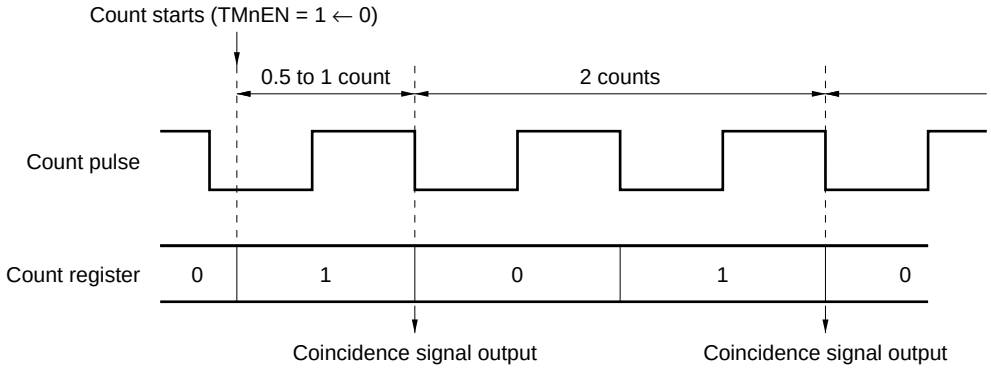
Therefore, an error of –0.5 to 1.5 count occurs until the coincidence signal is output for the first time after counting has been started. An example of counting where the modulo register is set to 1 is shown below.

Figure 13-7. Error When Counting Is Started from Count Stop Status

(a) If counting is started when count pulse is high (error: –0.5 to –1 count)



(b) If counting is started when count pulse is low (error: –1 to –1.5 count)



In this example, the coincidence signal must be output each time the count value has reached 2. However, the first coincidence signal is output when the count value is 1.5 at maximum or 0.5 at minimum (error: –0.5 to –1.5 count).

The above error also occurs during oscillation stabilization wait time because the timer is also used to generate the oscillation stabilization wait time.

13.1.8 Timer 0 Output

The $POD_3/\overline{TM0OUT}$ pin functions as timer 0 output pin by setting the TM0OSEL flag to “1”. At this time, the value of P0DBIO3 is irrelevant.

Timer 0 has an internal flip-flop for outputting a coincidence signal. The output of this flip-flop is inverted each time the comparator has output the coincidence signal. If the TM0OSEL flag is set to “1”, the content of this flip-flop is output to the $P0D_3/\overline{TM0OUT}$ pin.

The $P0D_3/\overline{TM0OUT}$ pin is an N-ch open-drain output pin and can be connected to a pull-up resistor by mask option. If the pull-up resistor is not connected, the $P0D_3/\overline{TM0OUT}$ pin goes into a high-impedance state as the initial status.

The internal timer 0 output flip-flop starts operating as soon as TM0EN has been set to 1. To make sure that timer 0 output always starts from the initial status, set TM0RES to 1 and reset the flip-flop before starting counting.

Figure 13-8. Timer 0 Output Setting Register

RF : 0BH

	Bit 3	Bit 2	Bit 1	Bit 0
	TM0OSEL	0	0	SIOEN
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

SIOEN	Function
0	$P0D_0/\overline{SCK}$, $P0D_1$/SO, and $P0D_2$/SI pins function as port pins.
1	$P0D_0/\overline{SCK}$, $P0D_1$/SO, and $P0D_2$/SI pins function as serial interface pins.

Caution This bit is not directly related to output setting of timer 0.

TM0OSEL	Function
0	$P0D_3/\overline{TM0OUT}$ pin functions as port pin.
1	$P0D_3/\overline{TM0OUT}$ pin outputs coincidence signal of timer 0.

13.2 BASIC INTERVAL TIMER (BTM)

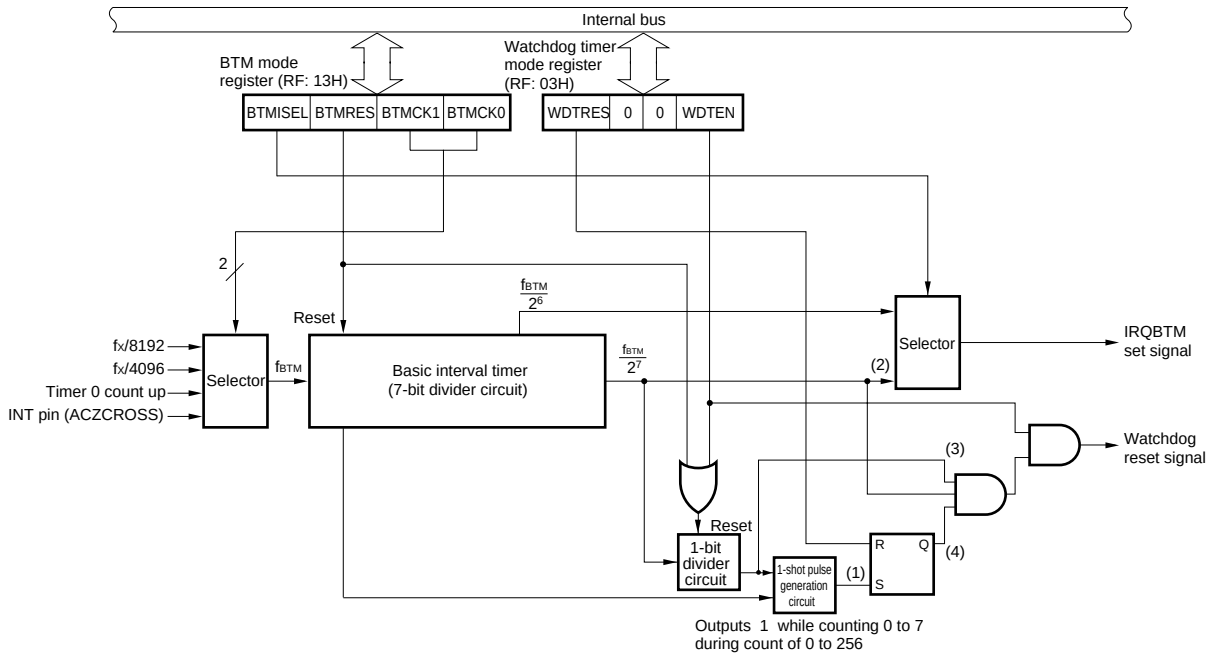
The μPD17134A subseries has a 7-bit basic interval timer.
This basic interval timer has the following functions.

(1) Generates reference time.
(2) Selects and counts wait time when standby mode is released.
(3) Serves as watchdog timer that detects program hang-up.

13.2.1 Basic Interval Timer Configuration

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

Figure 13-9. Basic Interval Timer Configuration



Remark (1) through (4) in the figure corresponding to the signals in the timing chart in Figure 13-12.

13.2.2 Registers Controlling Basic Interval Timer

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

Figure 13-10. BTM Mode Register

RF : 13H

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

Read = R, write = W

BTMCK1	BTMCK0	Selects count pulse of BTM
0	0	f_X/8192 (execution time of 512 instructions)
0	1	f_X/4096 (execution time of 256 instructions)
1	0	Count up of timer 0
1	1	INT pin (information on INT pin that has gone through AC zero cross detection circuit when ZCROSS = 1)

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

Remark BTMRES is automatically cleared to 0 after it has been set to 1.
This bit is always “0” when it is read.

BTMISEL	Selects interval time
0	Sets count pulse divided by 128 as interval time.
1	Sets count pulse divided by 32 as interval time.

Figure 13-11. Watchdog Timer Mode Register

RF : 03H

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

Read = R, write = W

WDTEN	Enables watchdog timer function
0	Watchdog timer stops.
1	Watchdog timer starts operating.

Remarks 1. WDTEN cannot be cleared to 0 by program.
2. WDTEN is automatically cleared to 0 after it has been set to 1. This bit is always 0 when it is read.

WDTRES	Resets watchdog timer
0	Does not affect watchdog timer.
1	Sets flip-flop that holds overflow carry of BTM used by watchdog timer.

Remark WDTRES is automatically cleared to 0 after it has been set to 1. This bit is always 0 when it is read.

13.2.3 Operation of Basic Interval Timer

The basic interval timer is a 7-bit binary counter that always counts up by using a count pulse specified by the BTM mode register. Counting operation cannot be stopped.

The interval time of the basic interval timer can be changed by using the BTMISEL bit of the BTM mode register. When BTMISEL = 0, the interval time is the count pulse divided by 128 ($128/f_{BTM}$); when BTMISEL = 1, the interval time is the count pulse divided by 32 ($32/f_{BTM}$).

The contents of the counter are not cleared to 0 even if the interval time is changed.

13.2.4 Watchdog Timer Function

The basic interval timer can also be used as a watchdog timer to detect a program hang-up.

(1) Function of watchdog timer

The watchdog timer is a counter that generates a reset signal at fixed intervals. By inhibiting the generation of this reset signal each time through program, the system can be reset (and started from address 0000H) if it has overrun due to an external noise (i.e., if the watchdog timer is not reset within the time set by program). This function can prevent the system from overrunning even if the program is caused to jump to an unexpected routine by an external noise and enter an infinite loop, because a reset signal is generated at fixed intervals.

(2) Operation of the watchdog timer

When “1” is set to WDTEN, the 1-bit divider is enabled to operate, and consequently, the basic interval timer operates as an 8-bit watchdog timer.

Once the watchdog timer has been started, it cannot be stopped until the device is reset and WDTEN is cleared to 0.

Generation of the reset signal by the watchdog timer can be inhibited in the following two ways:

(i) Repeat setting WDTRES in program.

(ii) Repeat setting BTMRES in program.

In the case of (i), WDTRES must be set while the count value of the watchdog timer is between 8 and 191 (immediately before it reaches 192). Therefore, “SET1 WDTRES” must be executed at least once at a timing shorter than the cycle in which the count value of the watchdog timer reaches 184.

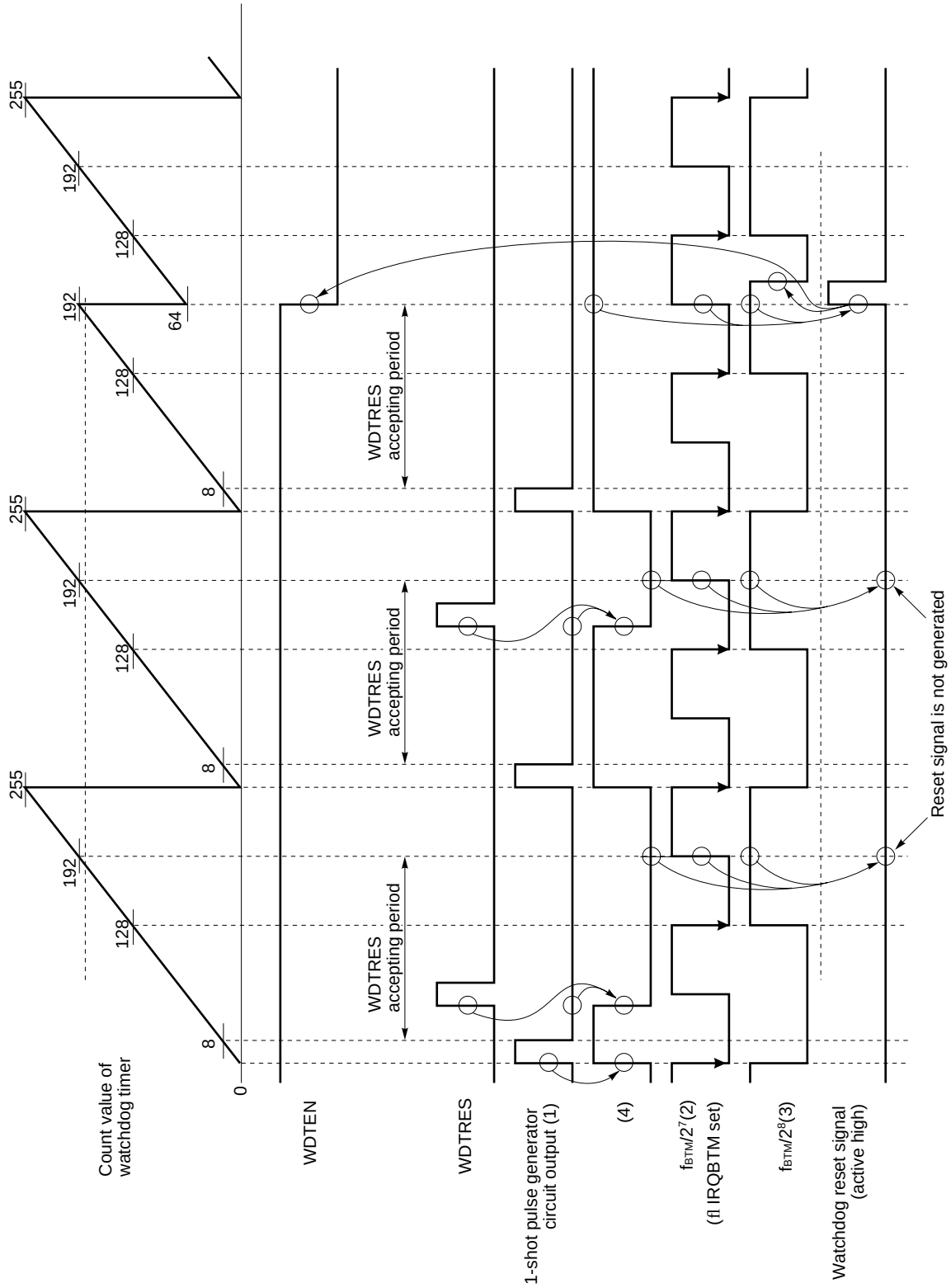
In the case of (ii), BTMRES must be set until the count value of the basic interval timer (BTM) reaches 128. Therefore, “SET BTMRES” must be executed at least once at a timing shorter than the cycle in which the count value of BTM reaches 128. In this case, however, interrupt processing cannot be performed by BTM.

Caution BTM is not reset even if WDTEN is set. Therefore, be sure to set BTMRES and reset BTM before setting WDTEN first.

Example

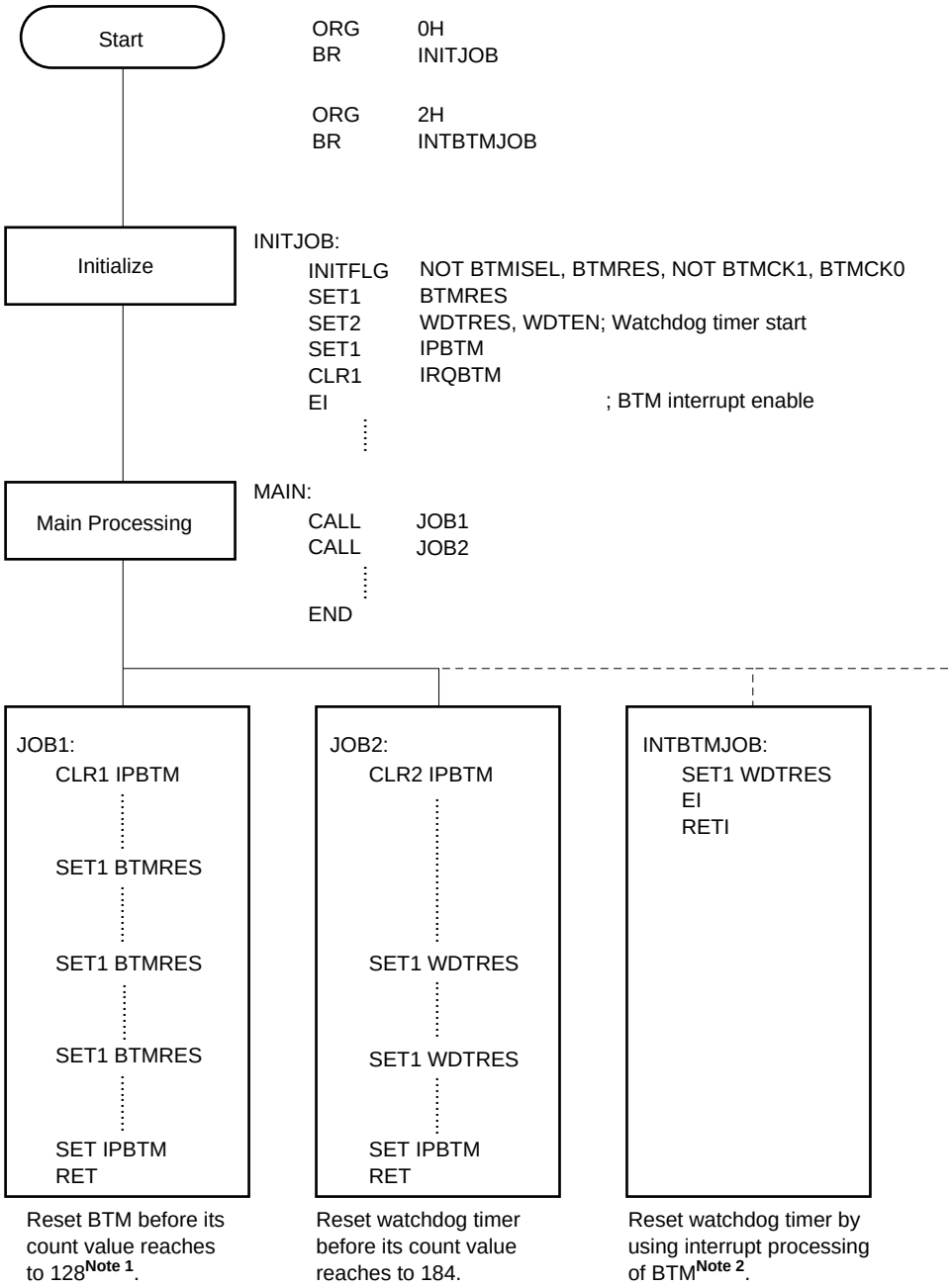
```
        ⋮
SET1    BTMRES
SET2    WDTEN, WDTRES
        ⋮
```

Figure 13-12. Timing Chart of Watchdog Timer (with WDTRES Flag Used)



(3) Program example of watchdog timer

Program Example



Notes 1. Interrupt processing by BTM cannot be performed in the method to reset counter before BTM overflows.

2. Although the method of resetting the watchdog timer by using the BTM interrupt processing is easier to program than the other two methods, its program hang-up detection rate is lower than that of the other two.

13.3 A/D CONVERTER

μPD17134A subseries contains an 8-bit resolution A/D converter with 4-channel analog input ($P0C_0/ADC_0$ - $P0C_3/ADC_3$).

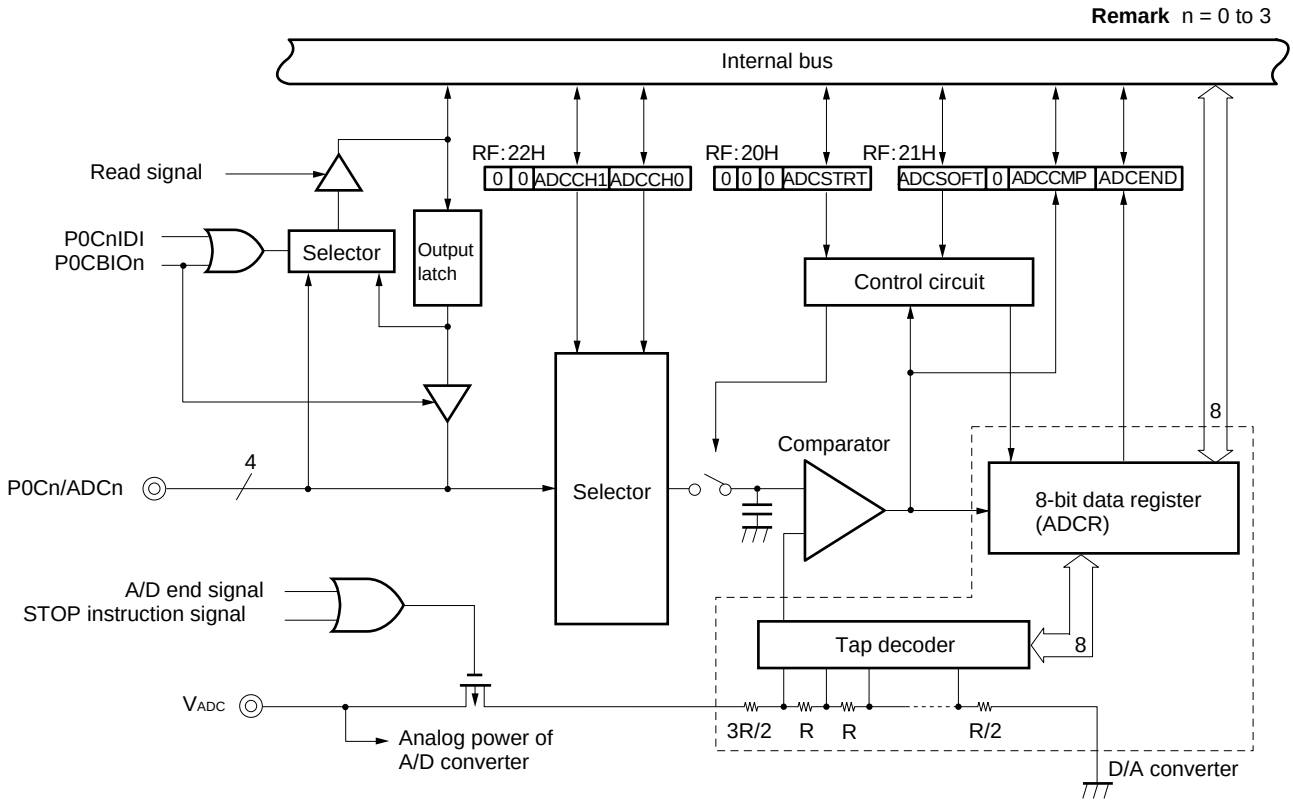
The A/D converter uses the successive approximation method. The following two operation modes are available:

(1) Successive mode: 8-bit A/D conversion occurs starting at high-order bits.
(2) Single mode: Comparison occurs with an arbitrary voltage value set in the 8-bit data register.

13.3.1 A/D Converter Configuration

Figure 13-13 shows the A/D converter configuration.

Figure 13-13. Block Diagram of the A/D Converter



Cautions 1. The 8-bit data register (ADCR) is cleared to 00H when the STOP instruction has been executed.

2. If the HALT instruction is executed during A/D conversion, a current keeps flowing between V_{ADC} and GND.

13.3.2 Functions of A/D Converter

(1) ADC_0 – ADC_3

These pins are used to input 4-channel analog voltage to the A/D converter. The A/D converter contains a sample hold circuit. Analog input voltage is internally retained during A/D conversion.

(2) V_{ADC}

This pin is used to input the power supply and the reference voltage for the A/D converter.
A signal input to ADC_0 to ADC_3 is converted to a digital signal based on voltage applied across V_{ADC} and GND.
To reduce the current consumption of the microcontroller, the A/D converter has a function for automatically stopping the current which flows into the V_{ADC} pin when the converter is not operating. Current flows into the V_{ADC} pin in the following cases.

<1> Successive mode (ADCSOFT=0)
From when the ADCSTRT flag is set (1) until the ADCEND flag is set (1).

<2> Single mode (ADCSOFT=1)
From when the ADCSTRT flag is set (1) or from when a value of the 8-bit data register is written until the result of comparison by the comparator is written in the ADCCMP flag.

Caution If the HALT Instruction is executed while the A/D conversion is in progress, the A/D converter stops conversion. Note that, in this case, the HALT mode is set with current flowing to the V_{ADC} pin. When the HALT mode has been released, the A/D conversion is resumed. At this time, however, the value of ADCR is undefined, and the correct conversion result cannot be obtained.

Remark A/D conversion is stopped if the STOP instruction is executed while the conversion is in progress. In this case, the A/D converter is initialized, and the current to the V_{ADC} pin is also cut. The A/D converter remains stopped even if the STOP mode has been released.

(3) 8-bit data register (ADCR)

In the successive mode, this 8-bit data register stores A/D conversion results for successive approximation. It is read by the GET instruction. In the single mode, the data in this register is converted to analog voltage by the internal D/A converter and the comparator compares this voltage with an analog signal input from the ADCn pin. A value can be written in this register by using the PUT instruction.

(4) Comparator

The comparator compares an analog input voltage from a pin with voltage output from the D/A converter. Value 1 is output if analog input voltage from the pin is high. Value 0 is output if this voltage is low. The comparison result is stored in the 8-bit data register (ADCR) in the successive mode. It is stored in the ADCCMP flag in the single mode.

(5) A/D converter control register

Figure 13-14 shows the A/D converter control register.

Figure 13-14. A/D Converter Control Register (1/2)

RF: 21H

	Bit 3	Bit 2	Bit 1	Bit 0
	ADCSOFT	0	ADCCMP	ADCEND
Read/write	R/W		R	
Initial value when reset	0	0	0	0

Read = R, write = W

ADCEND	End of A/D conversion
0	Initial status or during A/D conversion.
1	Indicates the end of A/D conversion in successive mode. Cleared to 0 by setting (1) or resetting ADCSTRT.

ADCCMP	Compare result (valid only in the single mode)
0	Analog input voltage is lower than output voltage of the internal D/A converter.
1	Analog input voltage is higher than output voltage of the internal D/A converter.

Remarks 1. In the single mode, the flag content is valid for the third and subsequent instructions after ADCSTRT is set (1) or data is set in ADCR until ADCSTRT or ADCR is set again.

2. In the successive mode, a value changes according to an A/D conversion value. However, the bit for this value cannot be identified.

3. ADCCMP is automatically cleared to 0 when “PUT ADCR, DBF” instruction is executed.

ADCSOFT	A/D operation mode selection flag
0	Successive mode
1	Single mode

Figure 13-14. A/D Converter Control Register (2/2)

RF: 20 H

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

Read = R, write = W

ADCSTRT	Start of A/D operation
0	Initial status or during A/D conversion.
1	Cleared to 0 automatically after A/D conversion (successive or single mode[Note]) starts.

Note With the μPD17134A subseries, ADCR is reset to 0 if the ADCSTRT flag is set, regardless of the A/D conversion mode. In the single mode, start conversion by writing a value to ADCR.

RF: 22H

	Bit 3	Bit 2	Bit 1	Bit 0
	ADCCH3	ADCCH2	ADCCH1	ADCCH0
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

ADCCH1	ADCCH0	Analog input channel selection
0	0	ADC_0 is selected.
0	1	ADC_1 is selected.
1	0	ADC_2 is selected.
1	1	ADC_3 is selected.

Fixed to 0. (Dummy flag)

13.3.3 Setting Values in the 8-bit Data Register (ADCR)

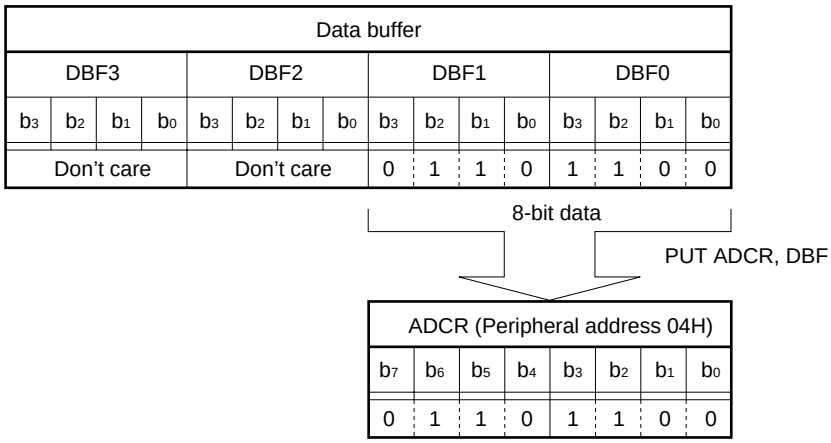
A value is set in the 8-bit data register via the data buffer (DBF) using the PUT instruction in the same way as for comparison voltage setting in the single mode.

The peripheral address for the 8-bit data register (ADCR) of the A/D converter is assigned to 04H. If a value is sent to ADCR by the PUT instruction, only the low-order 8 bits (DBF1, DBF0) of DBF are valid. DBF3 and DBF2 values do not affect ADCR.

Figure 13-15. Setting a Value in the 8-Bit Data Register (ADCR)

Example of setting 6CH in ADCR

```
CONTDATL  DAT   0CH          ; CONTDATL is assigned to 0CH by using a symbol definition instruction.
CONTDATH  DAT   06H          ; CONTDATH is assigned to 06H by using a symbol definition instruction.
          MOV   DBF0, #CONTDATL;
          MOV   DBF1, #CONTDATH;
          PUT   ADCR, DBF    ; Data is transferred using reserved words ADCR and DBF.
```



13.3.4 Reading Values from the 8-bit Data Register (ADCR)

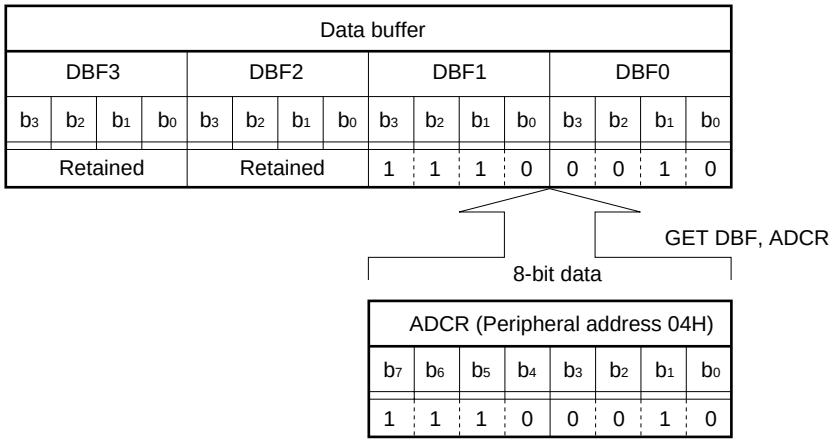
A value is read from the 8-bit data register (ADCR) via the data buffer (DBF) using the GET instruction.

The 8-bit data register (ADCR) of the A/D converter has peripheral address 04H and only its low-order 8 bits (DBF1, DBF0) are valid. Execution of the GET instruction does not affect the high-order 8 bits of DBF.

Figure 13-16. Reading Values from the 8-bit Data Register (ADCR)

The result from 8-bit A/D conversion is E2H.

```
GET  DBF, ADCR      ; Example of using reserved words DBF and ADCR
```

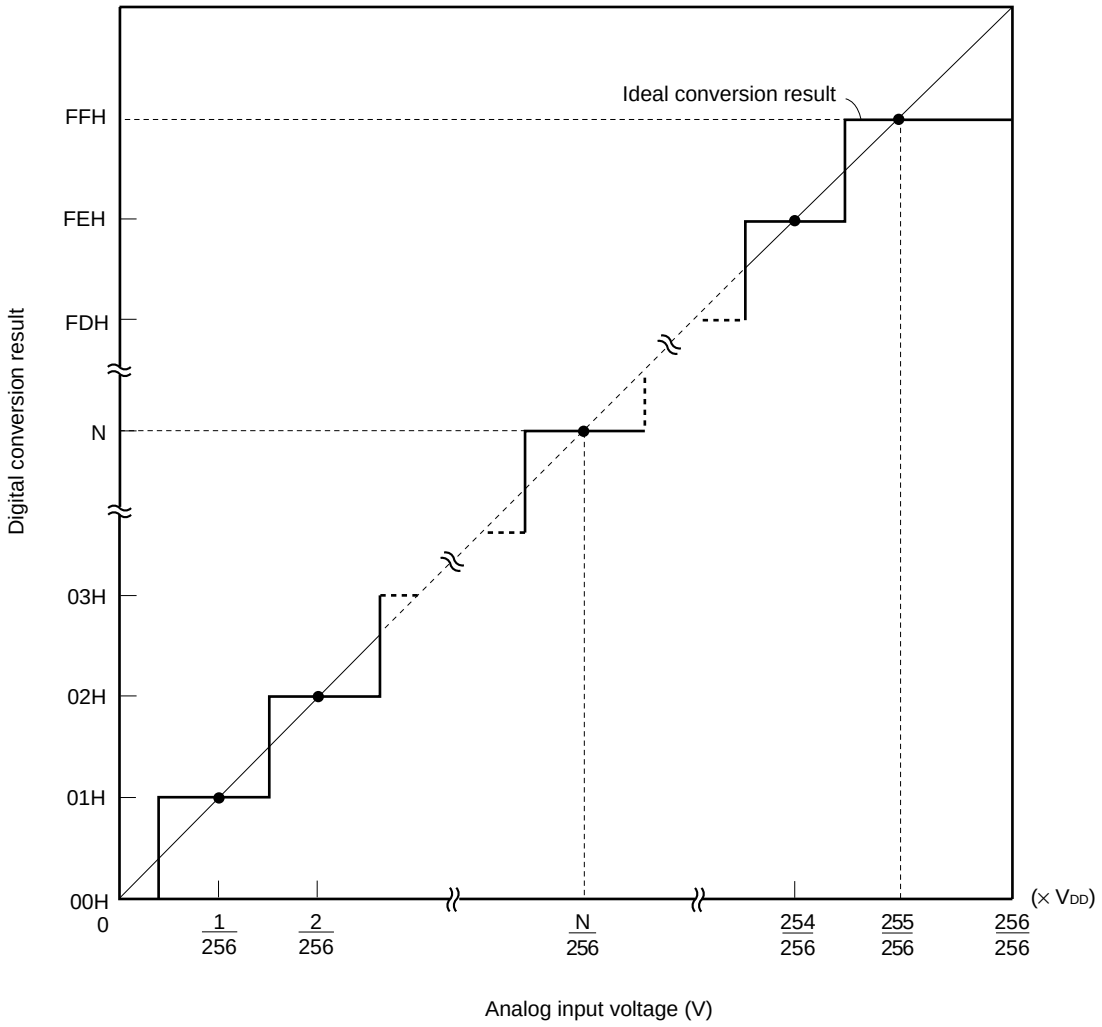


13.3.5 A/D Converter Operation

The A/D converter operates in two modes: successive mode and single mode. The mode can be switched by setting the ADCSOFT flag.

ADCSOFT	Operation mode of A/D converter
0	Successive mode (A/D conversion)
1	Single mode (Compare operation)

Figure 13-17. Relationship between the Analog Input Voltage and Digital Conversion Result



(1) Successive mode

(a) Outline of successive mode

In the successive mode, the A/D converter performs conversion in 8-bit units by means of successive approximation, and the result of the conversion is automatically stored to an 8-bit data register (ADCR). An analog input voltage and the voltage output by the internal D/A converter are compared by the internal comparator, and data for conversion is sequentially obtained from 8 bits of data, starting from the most significant bit. A time of 25 instructions is required to complete converting the 8 bits of data. The completion of the 8-bit A/D conversion is indicated by setting of the ADCEND flag to 1.

(b) Operation in successive mode

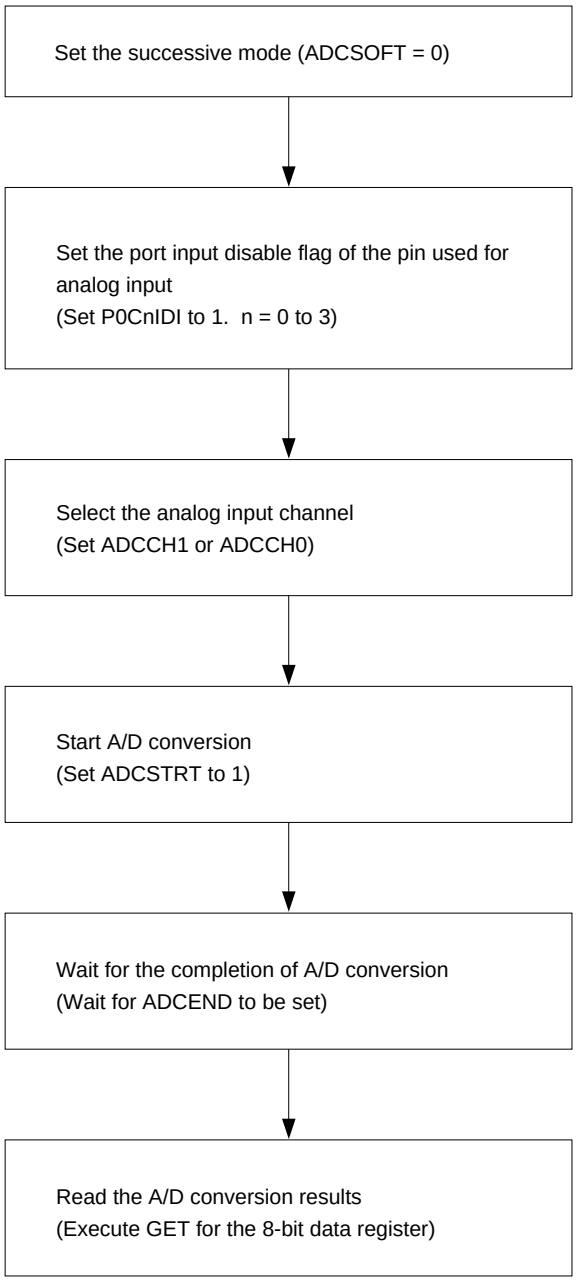
When ADCSOFT = 0, the A/D converter is set in the successive mode.

By setting P0CnIDI to 1 before starting A/D conversion, use of a pin used as an analog input pin of the A/D converter as a port pin is prohibited. This is to prevent an increase in the through current of the input buffer of the port if the voltage of the pin specified as an analog input pin reaches the intermediate level. After that, an analog input signal is selected by ADCCH1 and ADCCH0. A/D conversion is started by setting the ADCSTRT flag to 1. The ADCSTRT flag is cleared to 0 immediately after A/D conversion has been started.

While A/D conversion is in progress, the internal hardware performs successive approximation, starting from the most significant bit of the 8 bits of data. The conversion result is stored to an 8-bit data register on a bit-by-bit basis. Converting 1 bit of data requires a time of three instructions. If a resolution of 8 bits is not required, therefore, the time required can be calculated from the number of instructions executed, and the data being converted can be extracted before the ADCEND flag is set.

The completion of the A/D conversion is indicated by setting of the ADCEND flag which takes place as soon as data has been stored to the least significant bit of the 8-bit data register.

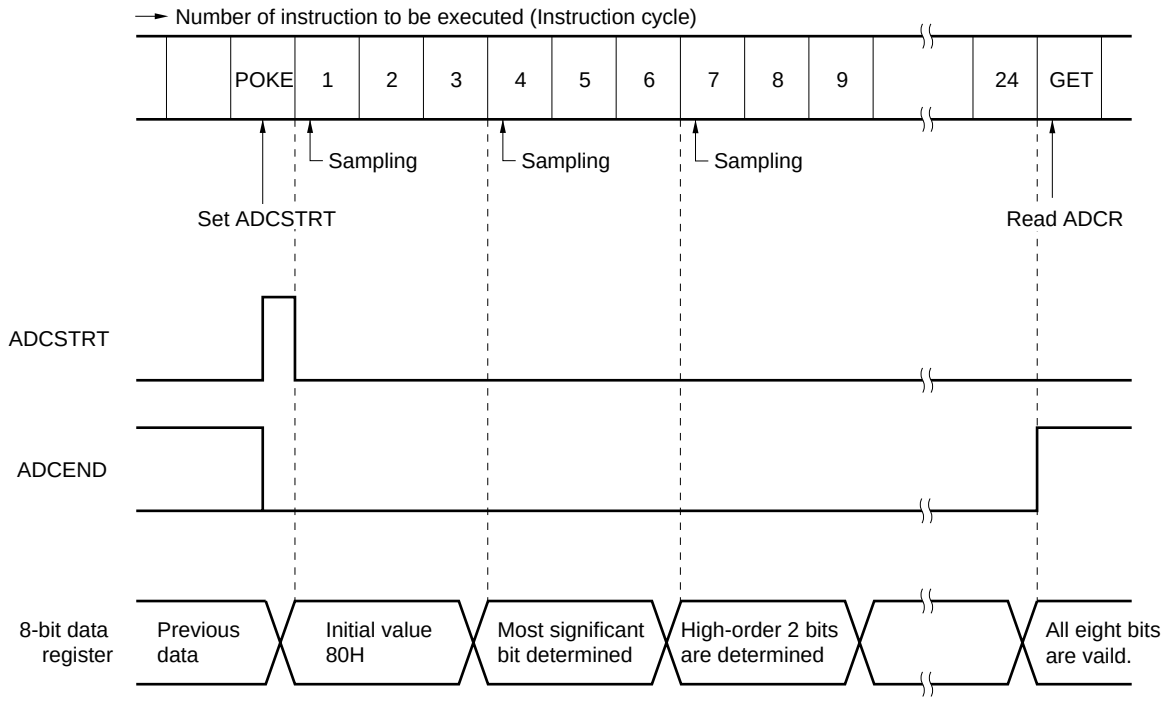
Figure 13-18. Using the Successive Mode for the A/D Converter



(c) Successive mode A/D conversion timing

Figure 13-16 shows the A/D conversion timing in the successive mode.

Figure 13-19. A/D Conversion Timing in the Continuous Mode



Caution Sampling is executed eight times while A/D conversion is performed once. Therefore, if the analog input voltage changes substantially during A/D conversion, conversion is not performed accurately. To obtain the accurate conversion result, it is necessary to keep changes in the analog input voltage as small as possible during A/D conversion.

One sampling time = 14/f_x (1.75 μs, 8 MHz[Note])
Sampling repeat cycle = 48/f_x (6 μs, 8 MHz[Note])

Note The guaranteed oscillation range of the μPD17134A, 17136A, and 17P136A is 400 kHz to 2.4 MHz.

Table 13-1. Data Conversion Time for the A/D Converter

Number of instructions executed after ADCSTRT is set to 1[Note]	Bits for which A/D conversion is completed (valid bits when ADCR is read)
4 instructions	Most significant bit
7 instructions	High-order 2 bits
10 instructions	High-order 3 bits
13 instructions	High-order 4 bits
16 instructions	High-order 5 bits
19 instructions	High-order 6 bits
22 instructions	High-order 7 bits
25 instructions	All 8 bits

Note Includes GET instruction to read data from ADCR.

(2) Single Mode

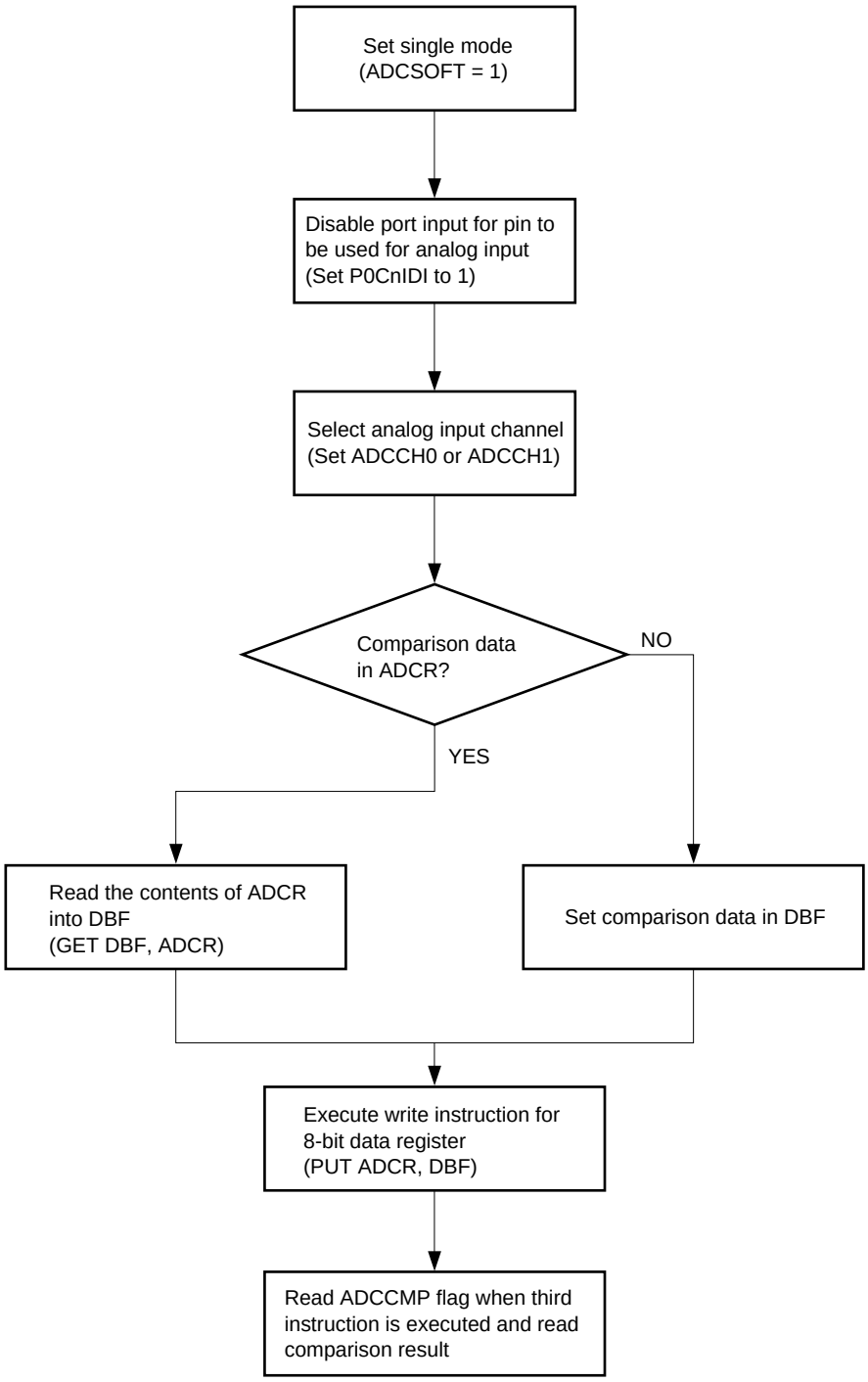
(a) Overview of single mode

In the single mode, data in the 8-bit data register (ADCR) is compared with voltage subjected to D/A conversion and with an analog input signal from a pin.
The comparison result appears in the ADCCMP flag.

(b) Explanation of single mode operation

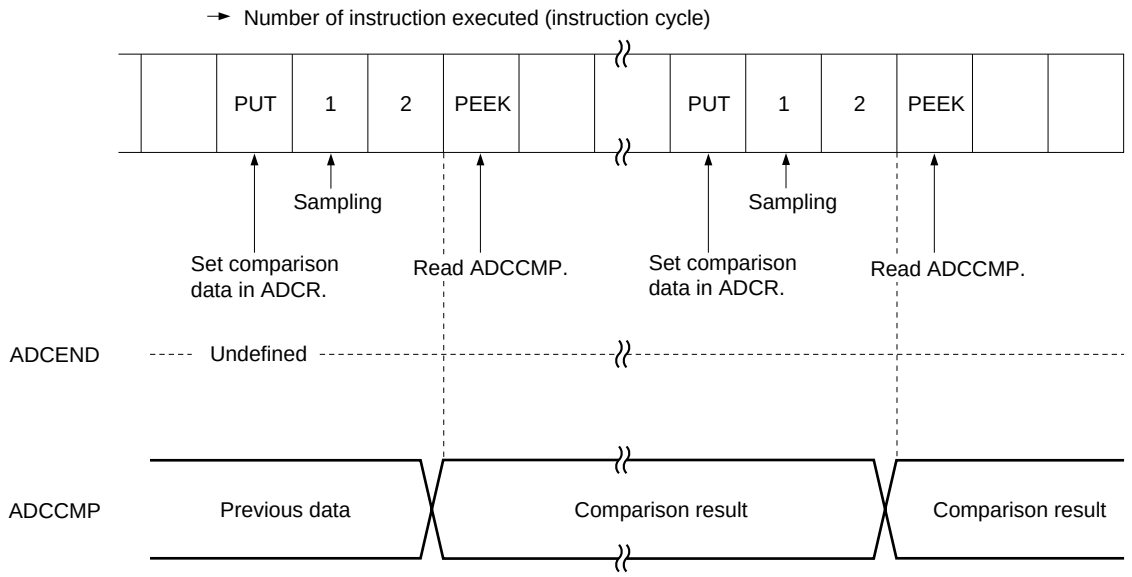
If ADCSOFT is 1, the A/D converter function enters the single mode.
Before single mode operation starts, port input is disabled for the pin to be used for analog input by setting P0CnIDI to 1. (This is done for the same reason as in the successive mode.)
To start single mode operation, execute a write instruction (PUT ADCR, DBF) for the 8-bit data register (ADCR) when ADCSOFT is 1.
The comparison result in single mode appears in ADCCMP at the execution of the third instruction after a PUT instruction is executed to write to the 8-bit data register (ADCR). At this time, the ADCEND flag becomes undefined.

Figure 13-20. Using the Single Mode for the A/D Converter



(c) Single mode operation (comparison) timing

Figure 13-21. Single Mode Operation (Comparison) Timing



In the single mode, comparison is started when compare data is set to ADCR (by executing the PUT instruction), and the result of conversion can be read by using the PEEK instruction after execution of the third instruction.
The ADCCMP flag is cleared to 0 when an instruction that writes ADCR is executed.

Caution Before setting a value to ADCR, be sure to set ADCSOFT to 1. A value cannot be set to ADCR while ADCSOFT is 0 (the "PUT ADCR, DBF" instruction is invalidated).

One sampling time = 14/f_X (1.75 μs, f_X = 8 MHz)

13.4 SERIAL INTERFACE (SIO)

The serial interface consists of an shift register (SIOSFR, 8 bits), serial mode register, and serial clock counter. It is used for serial data input/output.

13.4.1 Functions of the Serial Interface

This serial interface provides three signal lines: serial clock input pin ($\overline{SCK}$), serial data output pin (SO), and serial data input pin (SI). It allows 8 bits to be sent or received in synchronization with clocks. It can be connected to peripheral input/output devices using any method with a mode compatible to that used by the μPD7500 series or 75X series.

(1) Serial clock

Three types of internal clocks and one type of external clock are able to be selected. If an internal clock is selected as a shift clock, it is automatically output to the $P0D_0/\overline{SCK}$ pin.

Table 13-2. Serial Clock List

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

f_X: System Clock oscillation frequency

(2) Transfer operation

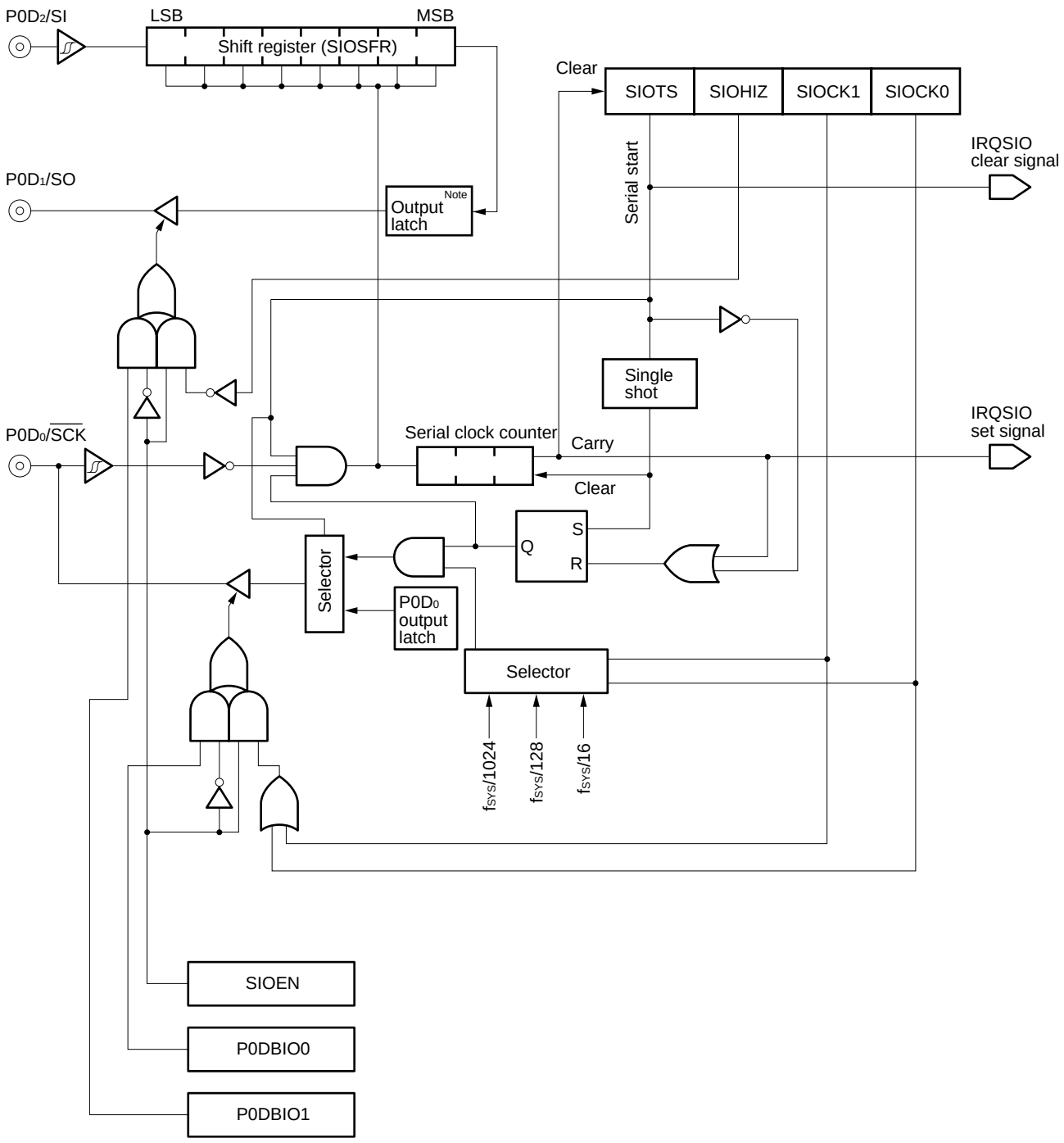
Each pin of port 0D ($P0D_0/\overline{SCK}$, $P0D_1$/SO, and $P0D_2$/SI) functions as a serial interface pin when SIOEN is set to 1. If SIOTS is set to 1 at this time, the operation is started in synchronization with the falling of the external or internal clock. If SIOTS is set, IRQSIO is automatically cleared.

Transfer is started from the most significant bit of the shift register in synchronization with the falling of the serial clock, and the information on the SI pin is stored to the shift register, starting from the least significant bit, in synchronization with the rising of the serial clock.

When transfer of 8-bit data has been completed, SIOTS is automatically cleared, and IRQSIO is set.

Remark When executing serial transfer, transfer is started only from the most significant bit of the shift register. It cannot be started from the least significant bit. The status of the SI pin is loaded to the shift register in synchronization with the rising of the serial clock.

Figure 13-22. Block Diagram of the Serial Interface



Note The output latch of the shift register is shared with P0D1. If an output instruction is executed to P0D1, therefore, the status of the output latch of the shift register is accordingly changed.

13.4.2 3-wire Serial Interface Operation Modes

Two modes can be used for the serial interface. If the serial interface function is selected, the $P0D_2$/SI pin always takes in data in synchronization with the serial clock.

- 8-bit transmission reception mode (simultaneous transmission and reception)
- 8-bit reception mode (SO pin: in the high-impedance state)

Table 13-3. Operating Mode of the Serial Interface

SIOEN	SIOHIZ	$P0D_2$/SI pin	$P0D_1$/SO pin	Operating mode of the serial interface
1	0	SI	SO	8-bit transmission/reception mode
1	1	SI	$P0D_1$ (input)	8-bit reception mode
0	×	$P0D_2$ (I/O)	$P0D_1$ (I/O)	General-purpose port mode

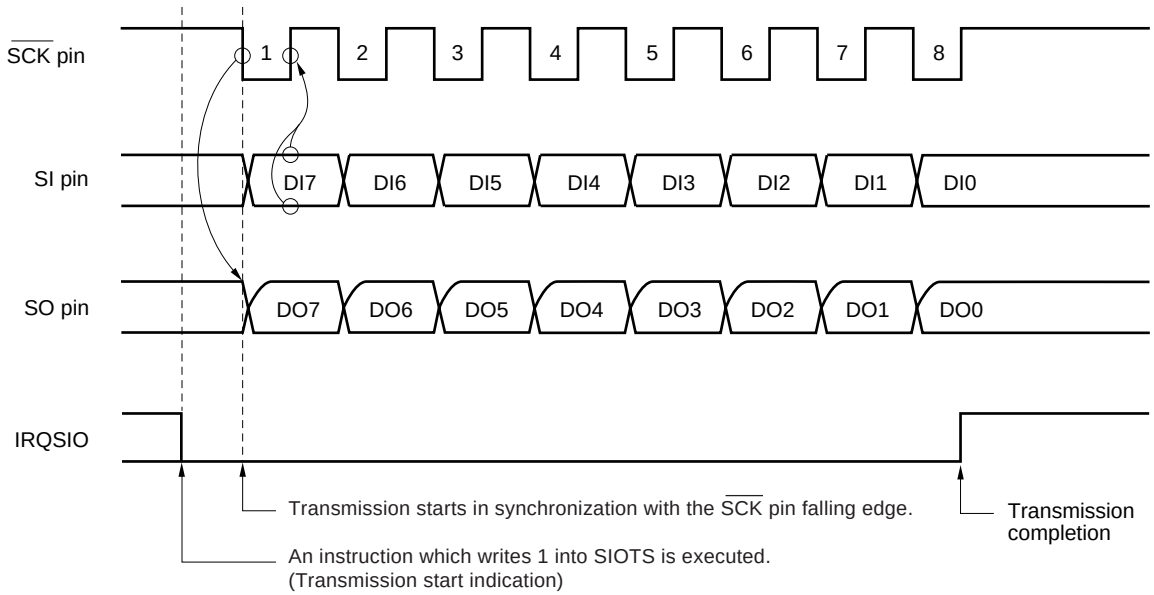
×: Don't care

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

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

Every time the serial clock counter (3-bit counter) counts eight serial clocks, the interrupt request flag (IRQSIO←1) is set to 1.

Figure 13-23. Timing of 8-Bit Transmission and Reception Mode (Simultaneous Transmission and Reception)

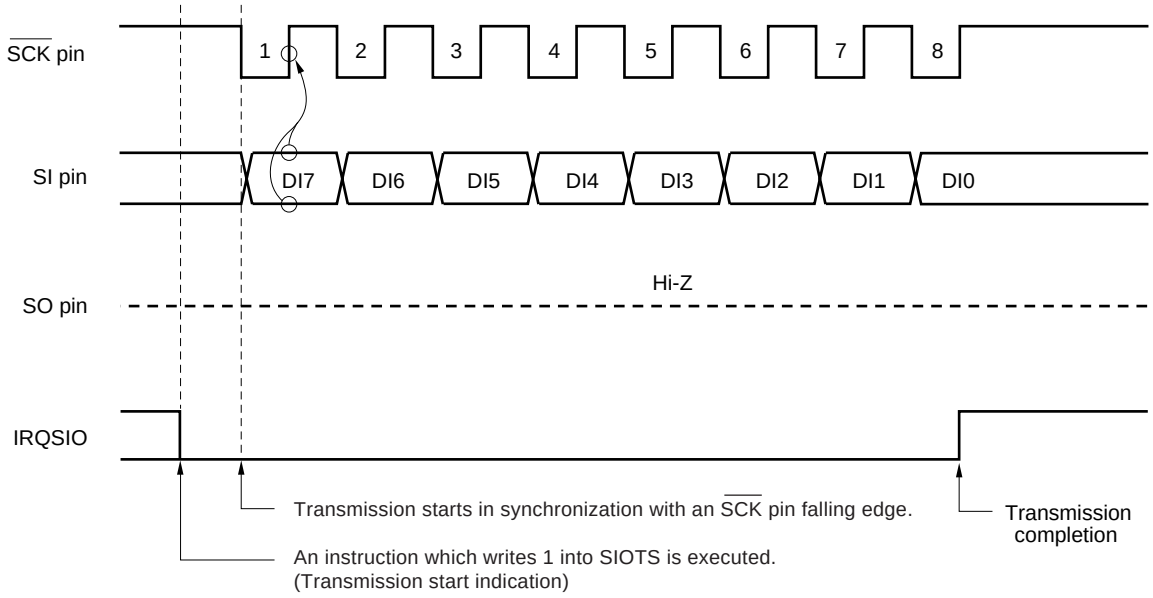


Remark DI: Serial data input
DO: Serial data output

(2) Clock synchronization 8-bit transmission and reception mode (SO pin output high impedance)

The $P0D_1$/SO pin goes into a high-impedance state when SIOHIZ = 1. If supply of the serial clock is started by writing “1” to SIOTS at this time, only the reception function of the serial interface is enabled.
Because the $P0D_1$/SO pin goes into a high-impedance state, it can be used as an input port pin ($P0D_1$).

Figure 13-24. Timing of the Clock Synchronization 8-Bit Reception Mode



Remark DI: Serial data input

(3) Operation stop mode

If the value in SIOTS (RF: address 02H, bit 3) is 0, the serial interface enters operation stop mode. In this mode, no serial transfer occurs.
In this mode, the shift register does not perform shifting and can be used as an ordinary 8-bit register.

Figure 13-25. Serial Interface Control Register (1/2)

RF: 02H

	Bit 3	Bit 2	Bit 1	Bit 0
	SIOTS	SIOHIZ	SIOCK1	SIOCK0
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

SIOCK1	SIOCK0	Selection of the serial clock
0	0	External clock ($\overline{SCK}$ pin)
0	1	$f_X/16$
1	0	$f_X/128$
1	1	$f_X/1024$

SIOHIZ	Function selection of the $P0D_1$/SO pin
0	Serial data output (SO pin)
1	Input port output high impedance ($P0D_1$ pin)

SIOTS	Start and stop of serial transmission (at writing)
0	Forced termination of the shift register (Disables intermediate restart).
1	Start of serial transfer operation • At internal clock selection Starts operation specifying the internal division signal of the system clock as a serial clock. • At external clock selection Starts operation in synchronization with an $\overline{SCK}$ pin falling edge.

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

Figure 13-25. Serial Interface Control Register (2/2)

RF: 0BH

	Bit 3	Bit 2	Bit 1	Bit 0
	TM0OSEL	0	0	SIOEN
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

SIOEN	SIO operation enable
0	The pins $P0D_0/\overline{SCK}$, $P0D_1/SO$, $P0D_2/SI$ function as ports.
1	The pins $P0D_0/\overline{SCK}$, $P0D_1/SO$, $P0D_2/SI$ function as the serial interface.

Remark See also **CHAPTER 12**.

TM0OSEL	Selecting function of the $P0D_3/\overline{TM0OUT}$ pin
0	The $P0D_3/\overline{TM0OUT}$ pin is used as a port.
1	The $P0D_3/\overline{TM0OUT}$ pin is used for timer 0 output.

Caution This is not related to the serial interface directly.

13.4.3 Setting Values in the Shift Register

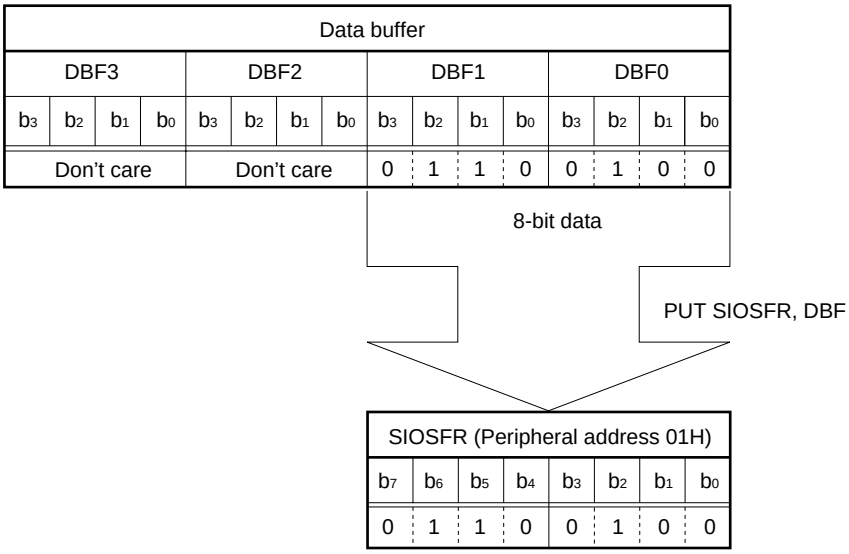
Values are set in the shift register via the data buffer (DBF) using the PUT instruction.

The peripheral address of the shift register is 01H. When sending a value to the shift register using the PUT instruction, only the low-order 8 bits (DBF1, DBF0) of DBF are valid. The DBF3 and DBF2 values do not affect the shift register.

Figure 13-26. Setting a Value in the Shift Register

Example of setting value 64H in the shift register

```
SIODATL  DAT  4H              ; SIODATL is assigned to 4H using symbol definition.
SIODATH  DAT  6H              ; SIODATH is assigned to 6H using symbol definition.
         MOV  DBF0, #SIODATL  ;
         MOV  DBF1, #SIODATH  ;
         PUT  SIOSFR, DBF     ; Value is transmitted using reserved word SIOSFR.
```

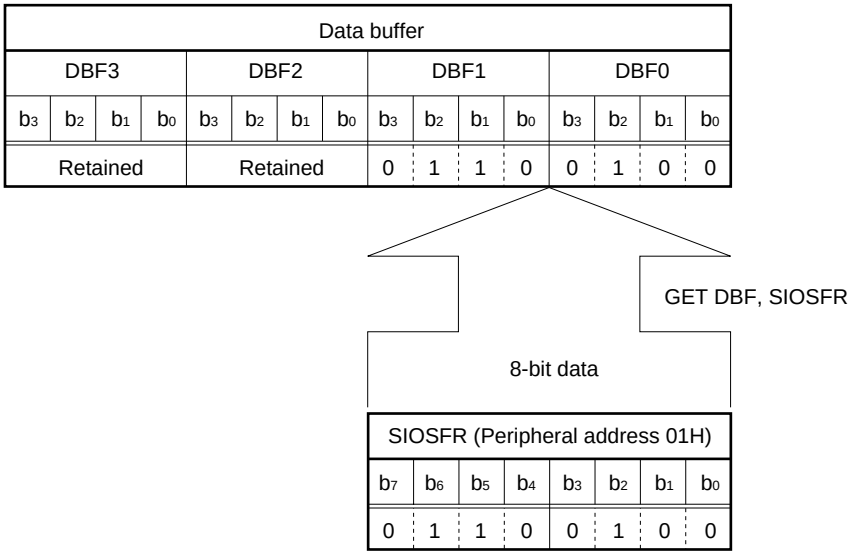


13.4.4 Reading Values from the Shift Register

A value is read from the shift register via the data buffer (DBF) using the GET instruction. The shift register has peripheral address 01H and only the low-order 8 bits (DBF1, DBF0) are valid. Executing the GET instruction does not affect the high-order 8 bits of DBF.

Figure 13-27. Reading a Value from the Shift Register

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



★

CHAPTER 14 INTERRUPT FUNCTIONS

The μPD17134A subseries has four internal interrupt functions and one external interrupt function. It can be used in various applications.

The interrupt control circuit of the μPD17134A subseries has the features listed below. This circuit enables very high-speed interrupt processing.

(a) Used to determine whether an interrupt can be accepted with the interrupt mask enable flag (INTE) and interrupt enable flag (IP×××).
(b) The interrupt request flag (IRQ×××) can be tested or cleared. (Interrupt generation can be checked by software.)
(c) Multiple interrupts are possible (up to three levels).
(d) Standby mode (STOP, HALT) can be released by an interrupt request. (Release source can be selected by the interrupt enable flag.)

Caution In interrupt processing, the bank register and the BCD, CMP, CY, Z, and IXE flags are saved in the stack automatically by the hardware for up to three levels of multiple interrupts. The DBF and WR are not saved by the hardware when peripheral hardware such as the timers or A/D converter is accessed in interrupt processing. It is recommended that the DBF and WR be saved in RAM by the software at the beginning of interrupt processing. Saved data can be loaded back into the DBF and WR immediately before the end of interrupt processing.

14.1 INTERRUPT SOURCE TYPES AND VECTOR ADDRESSES

For every interrupt in the μPD17134A subseries, when the interrupt is accepted, a branch occurs to the vector address associated with the interrupt source. This method is called the vectored interrupt method. Table 14-1 lists the interrupt source types and vector addresses.

If two or more interrupts occur simultaneously, or if two or more pending interrupts are enabled at the same time, processing is performed according to the priorities shown in Table 14-1.

Table 14-1. Interrupt Source Types

Interrupt source	Priority	Vector address	IRQ flag	IP flag	IEG flag	Internal/ external	Remarks
INT pin (RF: 0FH, bit 0)	1	0005H	IRQ RF: 3FH, bit 0	IP RF: 2FH, bit 0	IEGMD0,1 RF: 1FH	External	Rising edge or falling edge can be selected.
Timer 0	2	0004H	IRQTM0 RF: 3EH, bit 0	IPTM0 RF: 2FH, bit 1	–	Internal	
Timer 1	3	0003H	IRQTM1 RF: 3DH, bit 0	IPTM1 RF: 2FH, bit 2	–	Internal	
Basic interval timer	4	0002H	IRQBTM RF: 3CH, bit 0	IPBTM RF: 2FH, bit 3	–	Internal	
Serial interface	5	0001H	IRQSIO RF: 3BH, bit 0	IPSIO RF: 2EH, bit 0	–	Internal	

14.2 HARDWARE COMPONENTS OF THE INTERRUPT CONTROL CIRCUIT

The flags of the interrupt control circuit are explained below.

(1) Interrupt Request Flag and the Interrupt Enable Flag

The interrupt request flag (IRQ×××) is set to 1 when an interrupt request occurs. When interrupt processing is executed, the flag is automatically cleared to 0.

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

(2) EI/DI instruction

The EI/DI instruction is used to determine whether an accepted interrupt is to be executed.

If the EI instruction is executed, the interrupt enable flag (INTE) for enabling interrupt reception is set. If the interrupt is accepted, INTE is cleared to 0. Since the INTE flag is not registered in the register file, flag status cannot be checked by instructions.

The DI instruction clears the INTE flag to 0 and disables all interrupts.

At reset the INTE flag is cleared to 0 and all interrupts are disabled.

Table 14-2. Interrupt Request Flag and Interrupt Enable Flag

Interrupt request flag	Signal for setting the interrupt request flag	Interrupt enable flag
IRQ	Set by edge detection of an INT pin input signal. A detection edge is selected by IEGMD0 or IEGMD1.	IP
IRQTM0	Set by a match signal from timer 0.	IPTM0
IRQTM1	Set by a match signal from timer 1.	IPTM1
IRQBTM	Set by an overflow (reference time interval signal) from the basic interval timer.	IPBTM
IRQSIO	Set by a serial data transmission end signal from the serial interface.	IPSIO

Figure 14-1. Interrupt Control Register (1/6)

RF: 0FH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	0	INT
Read/write	R			
Initial value when reset	0	0	0	**Note**

Read = R, write = W

INT	Status of the INT pin
0	Sets logical status to 0 during PEEK instruction execution.
1	Sets logical status to 1 during PEEK instruction execution.

Note Values are not latched and so change momentarily according to pin logic. Once the IRQ flag is set, however, it remains set until an interrupt is accepted. The POKE instruction to address 0FH is invalid.

RF: 1FH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	IEGMD1	IEGMD0
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

IEGMD1	IEGMD0	Selection of the interrupt detection edge of the INT pin
0	0	Interrupt at the rising edge
0	1	Interrupt at the falling edge
1	0	Interrupt at both edges
1	1	

Figure 14-1. Interrupt Control Register (2/6)

RF: 3FH

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

Read = R, write = W

When read

IRQ	INT pin interrupt request
0	No interrupt request has been issued from the INT pin or an INT pin interrupt is being processed.
1	An interrupt request from the INT pin occurs or an INT pin interrupt is being held.

When write

IRQ	INT pin interrupt request
0	An interrupt request from the INT pin is forcibly released.
1	An interrupt request from the INT pin is forced to occur.

RF: 3EH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	0	IRQTM0
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

When read

IRQTM0	TM0 interrupt request
0	No interrupt request has been issued from timer 0 or a timer 0 interrupt is being processed.
1	The contents of the timer 0 count register matches that of the timer 0 modulo register and an interrupt request occurs. Or a timer 0 interrupt request is being held.

When write

IRQTM0	TM0 interrupt request
0	An interrupt request from timer 0 is forcibly released.
1	An interrupt request from timer 0 is forced to occur.

Remark If TM0RES is set to 1, IRQTM0 is cleared to 0.

Figure 14-1. Interrupt Control Register (3/6)

RF: 3DH

	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	0	IRQTM1
Read/write	R/W			
Initial value when reset	0	0	0	1

Read = R, write = W

When read

IRQTM1	TM1 interrupt request
0	No interrupt request has been issued from timer 1 or a timer 1 interrupt is being processed.
1	The contents of the timer 1 count register matches that of the timer 1 modulo register and an interrupt request occurs. Or a timer 1 interrupt request is being held.

When write

IRQTM1	TM1 interrupt request
0	An interrupt request from timer 1 is forcibly released.
1	An interrupt request from timer 1 is forced to occur.

Remark If TM1RES is set to 1, IRQTM1 is cleared to 0. IRQTM1 is cleared to 0 also immediately after the execution of the STOP instruction.

RF: 3CH

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

Read = R, write = W

When read

IRQBTM	BTM interrupt request
0	No interrupt request has been issued from the basic interval timer or a basic interval timer interrupt is being processed.
1	The basic interval timer overflows and an interrupt request occurs. Or a basic interval timer interrupt request is being held.

When write

IRQBTM	BTM interrupt request
0	An interrupt request from the basic interval timer is forcibly released.
1	An interrupt request from the basic interval timer is forced to occur.

Remark If BTMRES is set to 1, IRQBTM is cleared to 0.

Figure 14-1. Interrupt Control Register (4/6)

RF: 3BH

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

Read = R, write = W

When read

IRQSIO	SIO interrupt request
0	No interrupt request has been issued from the serial interface or a serial interface interrupt is being processed.
1	Serial interface transmission is completed and an interrupt request occurs. Or, a serial interface

When write

IRQSIO	SIO interrupt request
0	An interrupt request from the serial interface is forcibly released.
1	An interrupt request from the serial interface is forced to occur.

Figure 14-1. Interrupt Control Register (5/6)

RF: 2FH

	Bit 3	Bit 2	Bit 1	Bit 0
	IPBTM	IPTM1	IPTM0	IP
Read/write	R/W			
Initial value when reset	0	0	0	0

Read = R, write = W

IP	INT pin interrupt enable
0	Disables an interrupt from the INT pin. Holds an interrupt even if the IRQ flag is set to 1.
1	Enables an interrupt from the INT pin. Executes the EI instruction. If the IRQ flag is set to 1, executes interrupt processing.

IPTM0	TM0 interrupt enable
0	Disables an interrupt from timer 0. Holds an interrupt even if the IRQTM0 flag is set to 1.
1	Enables an interrupt from timer 0. Executes the EI instruction. If the IRQTM0 flag is set to 1, executes interrupt processing.

IPTM1	TM1 interrupt enable
0	Disables an interrupt from timer 1. Holds an interrupt even if the IRQTM1 flag is set to 1.
1	Enables an interrupt from timer 1. Executes the EI instruction. If the IRQTM1 flag is set to 1, executes interrupt processing.

IPBTM	BTM interrupt enable
0	Disables an interrupt from the basic interval timer. Holds an interrupt even if the IRQBTM flag is set to 1.
1	Enables an interrupt from the basic interval timer. Executes the EI instruction. If the IRQBTM flag is set to 1, executes interrupt processing.

Figure 14-1. Interrupt Control Register (6/6)

RF: 2EH

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

Read = R, write = W

IPSIO	SIO interrupt enable
0	Disables an interrupt from the serial interface. Holds an interrupt even if the IRQSIO flag is set to 1.
1	Enables an interrupt from the serial interface. Executes the EI instruction. If the IRQSIO flag is set to 1, executes interrupt processing.

14.3 INTERRUPT SEQUENCE

14.3.1 Receiving an Interrupt

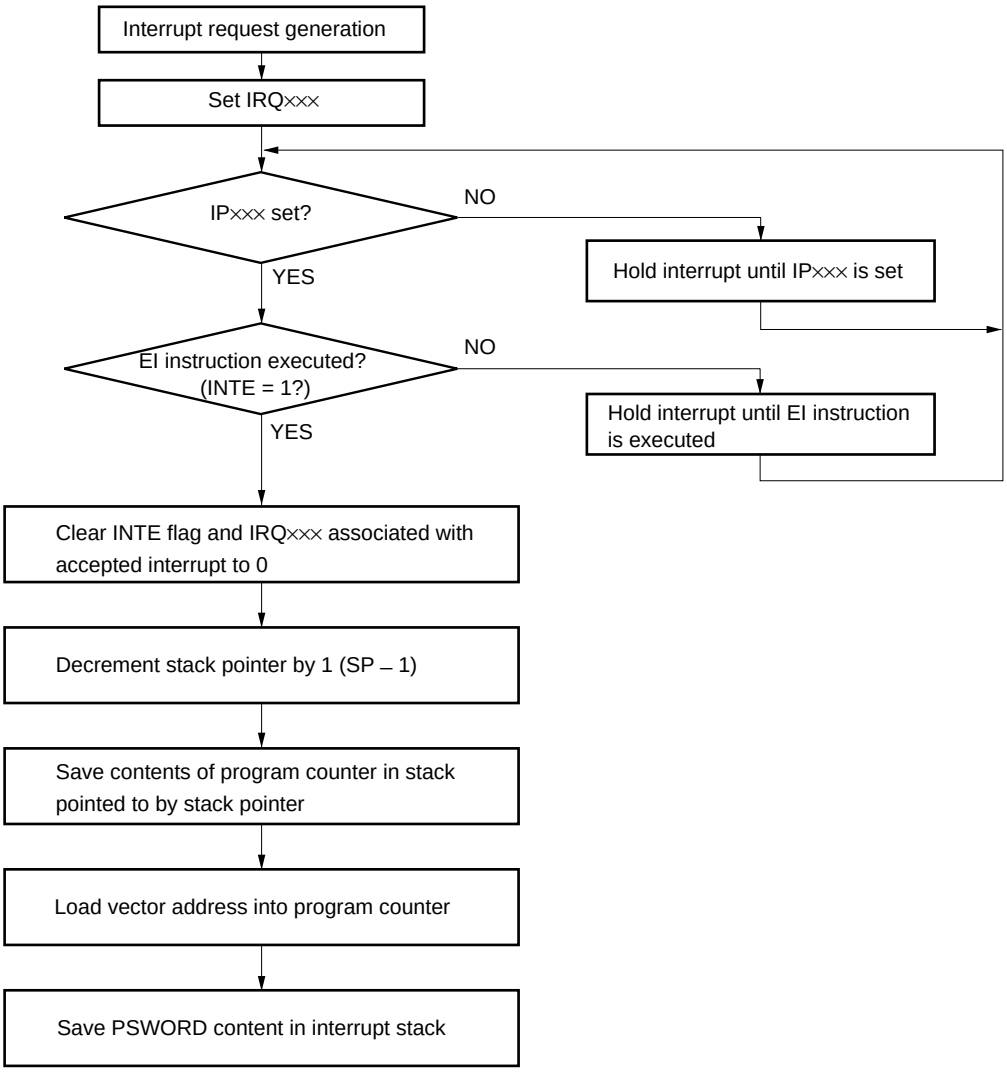
When an interrupt is accepted, interrupt processing starts after the instruction cycle of the instruction being executed is completed. The program flow is transferred to a vector address. However, if an interrupt occurs during MOVT or EI instruction, or if an instruction that satisfies the skip condition is executed, the interrupt processing is started two instruction cycles later.

When interrupt processing starts, one level of the address stack register is consumed to store the program return address, and one level of the interrupt stack register is consumed to save BANK and PSWORD in the system register.

If two or more interrupts occur or are enabled, interrupt processing is executed in descending order of priority. A lower-priority interrupt is held until a higher-priority interrupt is processed.

See priorities shown in Table 14-1.

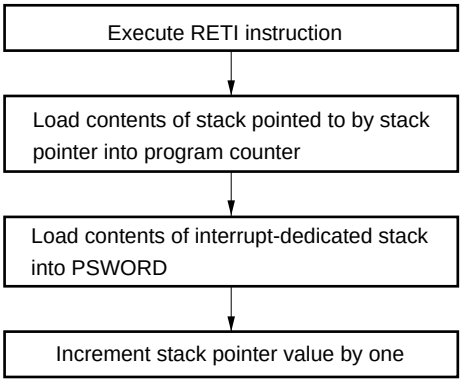
Figure 14-2. Interrupt Processing Procedure



14.3.2 Return from the Interrupt Routine

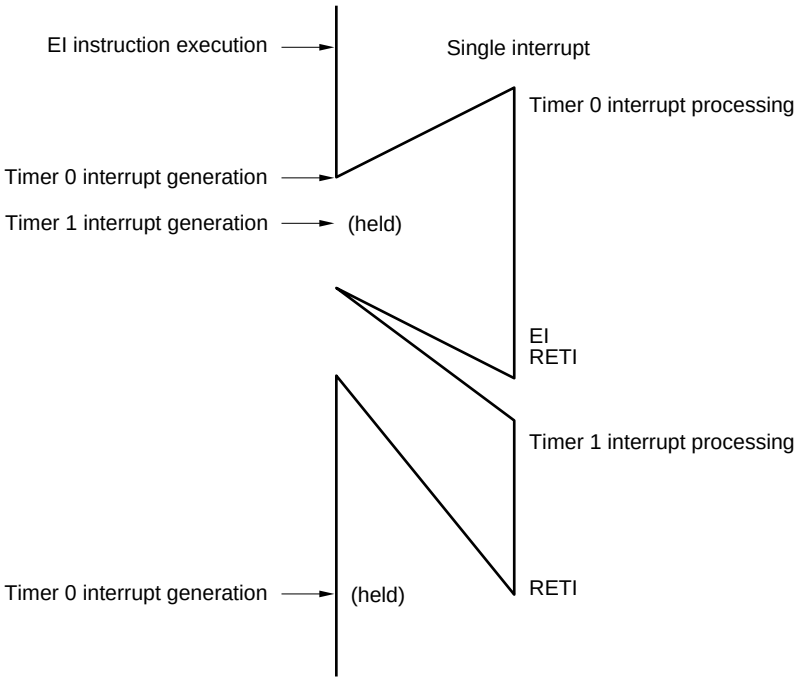
Execute the RETI instruction to return from the interrupt processing routine. During the RETI instruction cycle, processing in the figure below occurs.

Figure 14-3. Return from Interrupt Processing



Caution The INTE flag is not set for the RETI instruction.
Interrupt processing is completed. To handle a pending interrupt successively, execute the EI instruction immediately before the RETI instruction and set the INTE flag to 1.
To execute the RETI instruction following the EI instruction, no interrupt is accepted between EI instruction execution and RETI instruction execution. This is because the EI instruction sets the INTE flag to 1 after the execution of the subsequent instruction is completed.

Example



14.3.3 Interrupt Accepting Timing

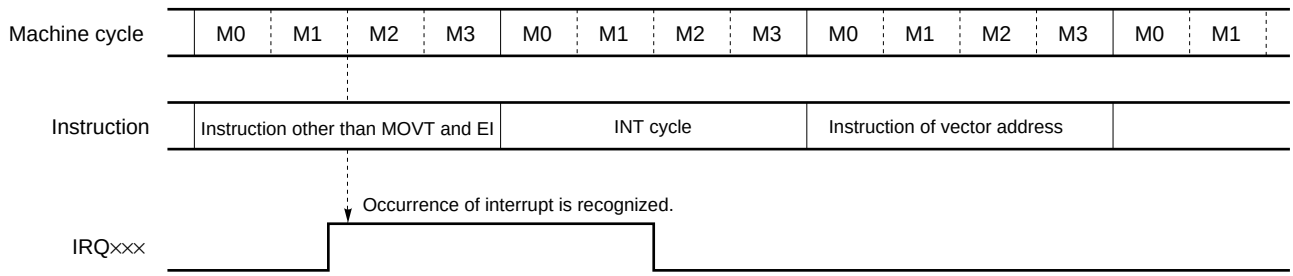
Figure 14-4 shows a timing chart that illustrates how interrupts are accepted.

The μPD17134A subseries xecutes one instruction in 16 clocks or in 1 instruction cycle. One instruction cycle consists of four states, M0 to M3, with each state made up of 4 clocks.

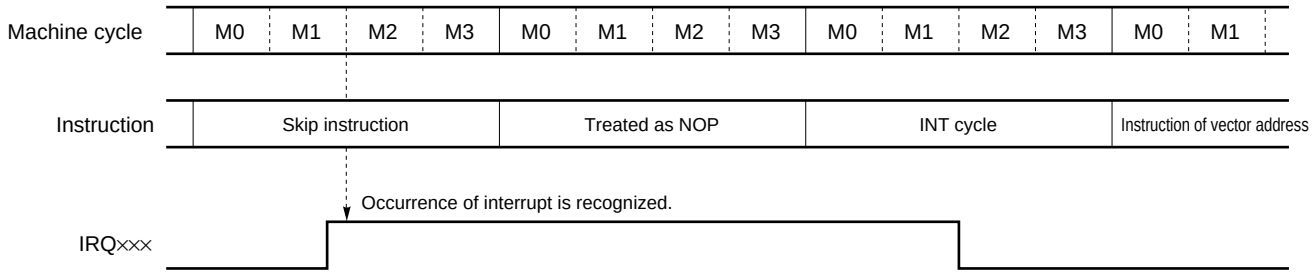
An interrupt occurs asynchronously in respect to the program operation. The program recognizes the occurrence of the interrupt at the leading edge of state M2.

Figure 14-4. Interrupt Accepting Timing (When INTE = 1, IP××× = 1) (1/3)

(1) If interrupt occurs before M2 of instruction other than MOVT and EI



(2) If skip condition of skip instruction is satisfied in (1)



(3) If interrupt occurs after M2 of instruction other than MOVT and EI

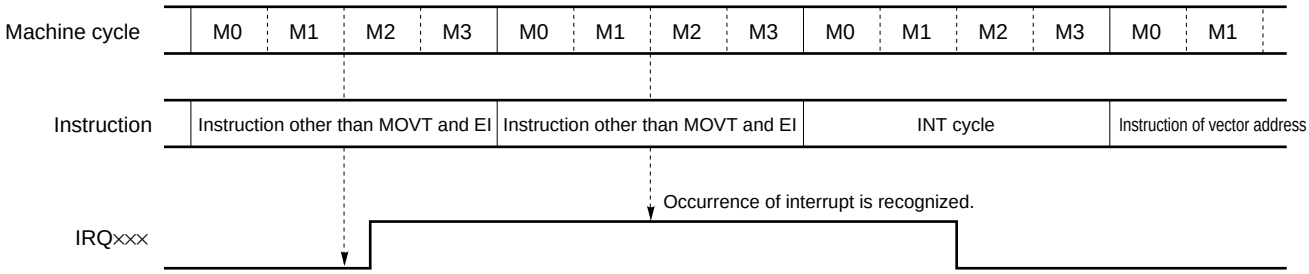
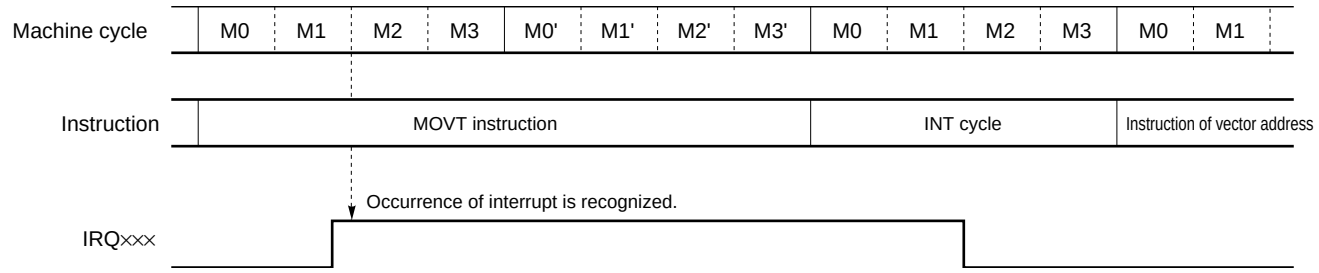
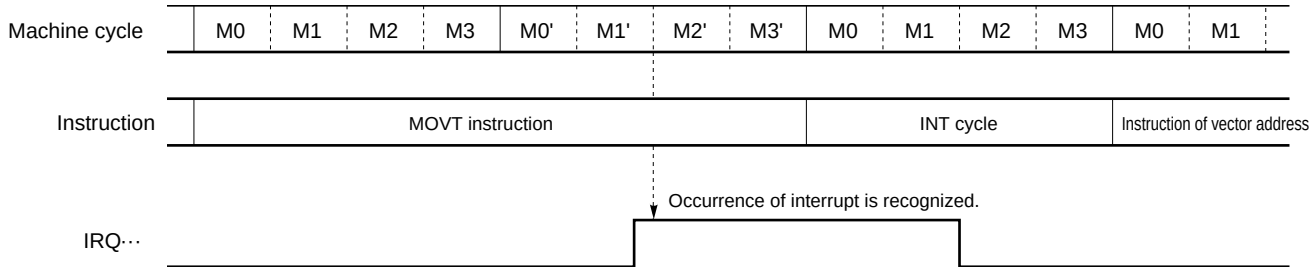


Figure 14-4. Interrupt Accepting Timing (When INTE = 1, IP×××= 1) (2/3)

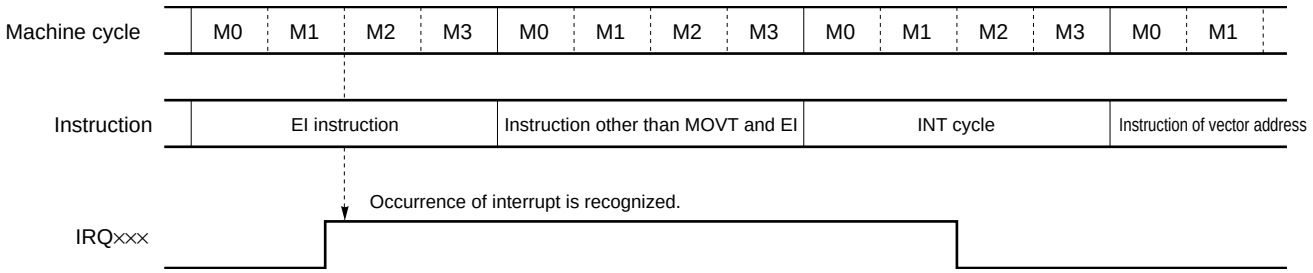
(4) If interrupt occurs before M2 of MOVT instruction



(5) If interrupt occurs before M2' of MOVT instruction



(6) If interrupt occurs before M2 of EI instruction



(7) If interrupt occurs after M2 of EI instruction

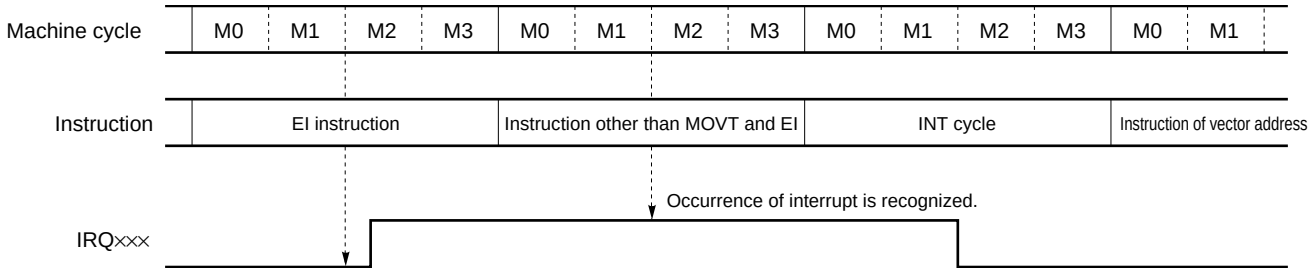
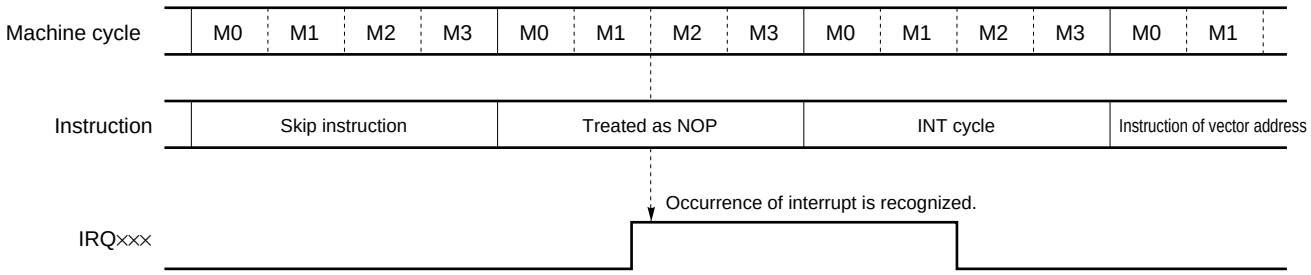


Figure 14-4. Interrupt Accepting Timing (When INTE = 1, IP×××= 1) (3/3)

(8) If interrupt occurs during skip of skip instruction (treated as NOP)



Remarks 1. The INT cycle is for preparation of an interrupt. In this cycle, the contents of PC and PSWORD are saved, and IRQ××× is cleared.

2. The MOVT instruction exceptionally requires 2 instruction cycles.

3. The EI instruction is designed so that multiplexed interrupt does not occur when program execution returns from interrupt processing.

14.4 MULTI-INTERRUPT

Multi-interrupt is a method that executes interrupt processing of other interrupt source B and C during the interrupt processing for an interrupt source A as shown in Figure 14-5.

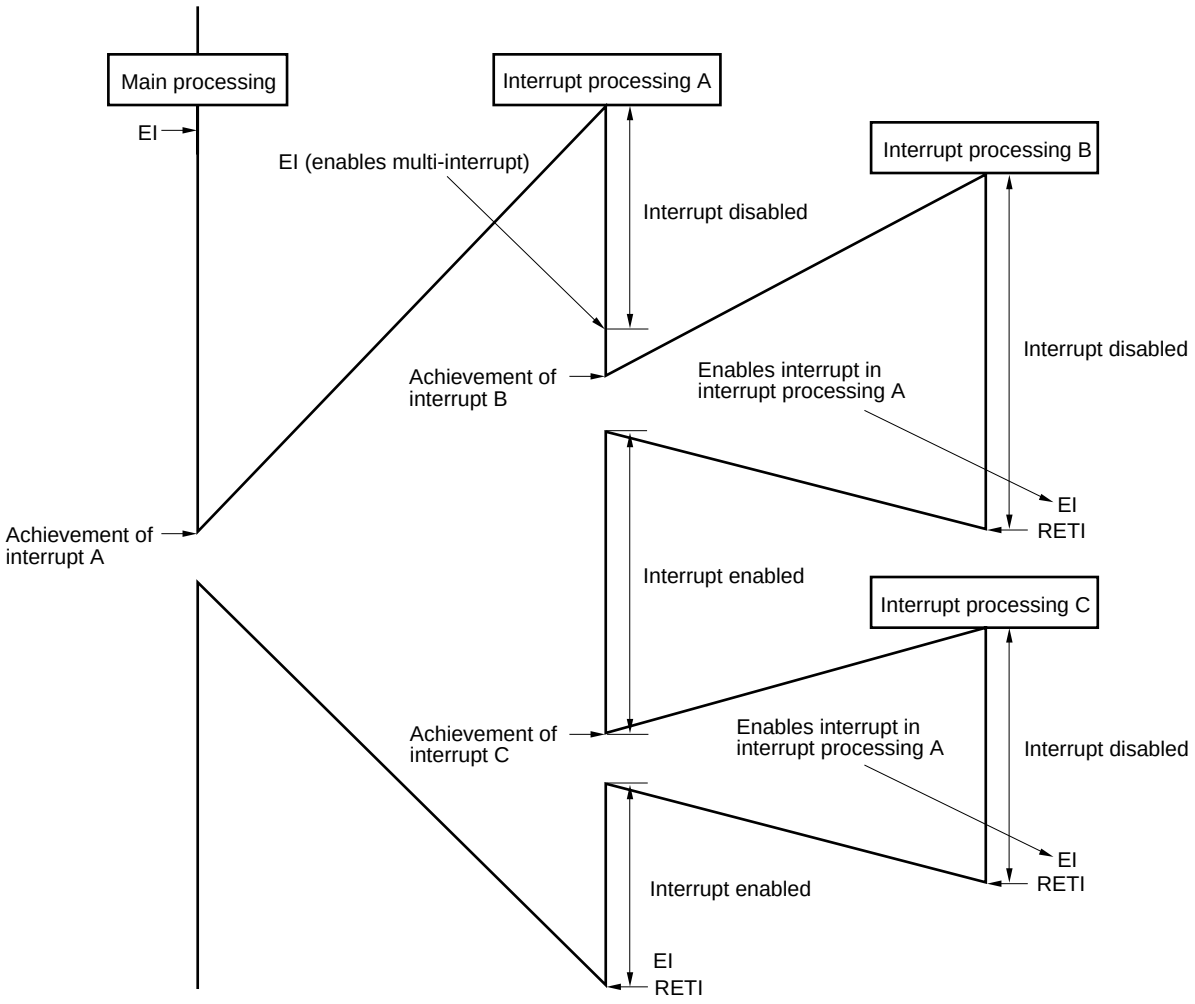
Nesting level at this time is also called interrupt level.

Pay attention to the following points when using multi-interrupt.

(1) Priority of interrupt source

(2) Limit of interrupt levels by interrupt stack (maximum 3 levels for the μPD17134A subseries)

Figure 14-5. Example of Multi-interrupt



As shown in Figure 14-5, INTE flag is cleared automatically and becomes interrupt disable state when interrupt has been achieved. Therefore, when executing multi-interrupt processing, execute EI instruction during interrupt processing.

Caution Maximum number of interrupt levels is 3. When achieving interrupt, interrupt stack register and address stack register are consumed by one level. Address stack register is consumed by MOVT instruction and PUSH instruction other than CALL instruction. Pay attention to the nesting level of address stack.

14.5 PROGRAM EXAMPLE OF INTERRUPT

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

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

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

  In the following program, after generating external interrupt, the signal from INT pin becomes effective after confirming that there is no change in the level of INT pin two times in every 100 μs.

  Example

```
WAITCNT   MEM      0.00H          ; Counter of wait processing
CHKRAM    MEM      0.01H
KEYON     FLG      0.01H.3        ; If key turns ON (even just once), KEYON = 1
CHK100U   FLG      0.01H.0        ; CHK100U = 1 only when passing 100 μs during WAIT loop
          ORG      0H
          BR       JOB_INIT
          ORG      5H
          BR       INT_JOB
          :
JOB_INIT:
          MOV      WAITCNT, #0    ; Clears RAM and the flag on RAM
          MOV      CHKRAM, #0     ;
          INITFLG  NOT IEGMD1, IEGMD0
                                  ; Rising edge is effective for the interrupt from INT pin
          CLR1     IRQ
          SET1     IP
          EI
          :
MAIN:
          CALL     55JOB
          CALL     55JOB
          :
          BR       MAIN
```

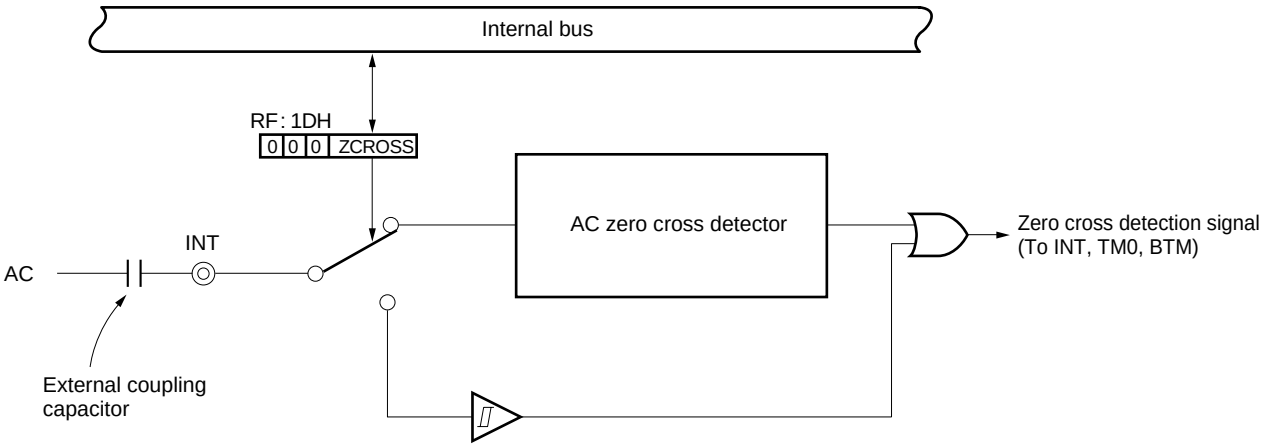
```
INT_JOB:
        NOP                     ; Loop which executes waiting for 100 μs at 8 MHz
        NOP                     ; 2 μs (1 instruction) × 5 instructions × 10 times
                                ; (count value at WAIT)
        ADD     WAITCNT, #01    ;
        SKE     WAITCNT, #0A    ;
        BR      INT_JOB         ;
        SKF1    INT             ; Check the level of INT pin
        BR      KEY_NO          ; If INT pin is high level, interrupt is invalid, and returns
                                ; to main processing
        SKF1    CHK100U         ; First wait?  (CHK100U = 0?)
        BR      WAIT_END        ; If it is the first time, wait again after setting CHK100U.
                                ; In the case of the second time, finish wait processing
        SET1    CHK100U         ;
        BR      INT_JOB
WAIT_END:
        SET1    KEY_ON          ; Judges that there is key input
KEY_NO:
        CLR1    CHK100U         ; CHK100U ← 0
        EI
        RETI
```

[MEMO]

CHAPTER 15 AC ZERO CROSS DETECTION

The INT pin is the interrupt signal input pin and timer count clock input pin. It also used as an AC zero cross detector input pin. This pin can be selected by writing 1 in ZCROSS (RF: 1DH bit 0).

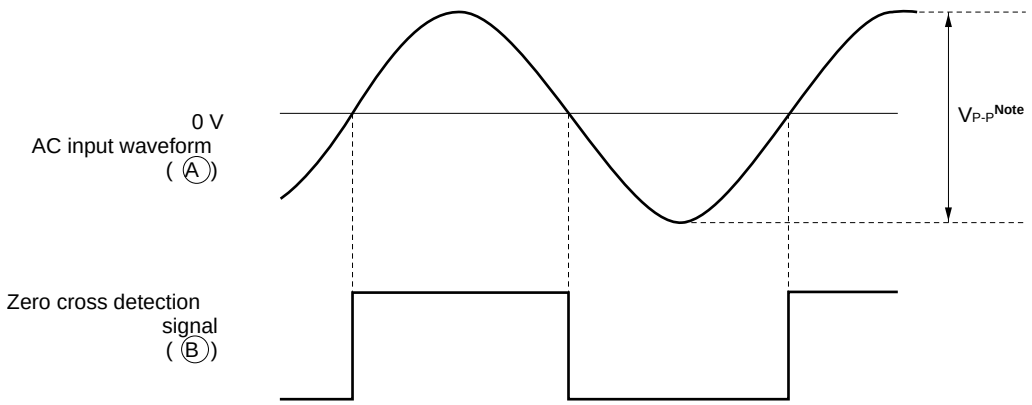
Figure 15-1. Block Diagram for the AC Zero Cross Detector



Caution When the AC zero cross detection circuit is used, the current consumption slightly increases (to 15 μA TYP.) even in the standby mode. To prevent an increase in the current consumption, clear ZCROSS to 0, and fix the input voltage of the INT pin to the high or low level.

The zero cross detector consists of a high gain amplifier which uses the self-bias method. It biases the input to the switching point and causes digital displacement in response to slight displacement of INT pin input. It detects changes of an AC signal from minus to plus and vice versa. This signal is input through the external coupling capacitor. The signal changes 0 to 1 and vice versa at each displacement point.

Figure 15-2. Zero Cross Detection Signal



Note The range of the input voltage when the INT pin is used as the input pin of the AC zero cross circuit is 1.0 V_{P-P} to 3.0 V_{P-P}.
Because the AC zero cross circuit does not have a function to reject noise, input a signal from which noise has been eliminated in advance to this circuit.

A pulse generated in the zero cross detector can be used as a timer 0 count clock and basic interval timer count clock in the same way as when the pulse does not go through the zero cross detector. The pulse is sent to the interrupt control circuit. Interrupt processing starts if an INT pin interrupt is enabled. To accept an interrupt, set IEGMD0 (RF: 1FH bit 0) and IEGMD1 (RF: 1FH bit 1) to select a signal rising edge, falling edge, or both rising and falling edges.

★

CHAPTER 16 STANDBY FUNCTION

16.1 OVERVIEW OF THE STANDBY FUNCTION

The μPD17134A subseries has a standby function to reduce the current consumption. The standby function can be used in two modes which can be selected as the application requires: STOP and HALT modes.

In the STOP mode, the system clock is stopped. Therefore, the current consumption of the CPU in this mode is only the leakage current. This mode is effective for holding the contents of the data memory without the CPU operating.

In the HALT mode, oscillation of the system clock continues, but the CPU is stopped because supply of the clock to the CPU is stopped. The current consumption in this mode is greater than in the STOP mode. However, operation can be resumed immediately after the HALT mode has been released because the system clock is oscillating. In both the STOP and HALT Modes, the contents of the data memory and registers, and the status of the output latch of the output port immediately before the standby mode is set are retained (except STOP 0000B). Therefore, set the port status to reduce the overall current consumption of the system before setting a standby mode.

Table 16-1. Status in Standby Mode

		STOP mode	HALT mode
Setting instruction		STOP instruction	HALT instruction
System clock oscillation circuit		Oscillation stops	Oscillation continues
Operating status	CPU	• Operation stops	
	RAM	• Retains previous status	
	Port	• Retains previous status[Note]	
	TM0	• Can operate only when INT input is selected as count pulse • Stops if system clock is selected (count value is retained)	• Can operate
	TM1	• Operation stops (count value is reset to “0”) (count up is also disabled)	• Can operate
	BTM	• Operation stops (count value is retained)	• Can operate
	SIO	• Can operate only when external clock is selected as serial clock[Note]	• Can operate
	A/D	• Operation stops[Note] (ADCR ← 00H)	• Can operate
	INT	• Can operate	• Can operate

Note When STOP 0000B is executed, these pins are set in the input port mode including when the multiplexed function of the pin is used.

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

2. The standby mode is not set if both the interrupt request flag and interrupt enable flag are set and the interrupt is specified as the condition to release the standby mode.

16.2 HALT MODE

16.2.1 Setting HALT Mode

The HALT mode is set when the HALT instruction is executed.

Operand $b_3b_2b_1b_0$ of the HALT instruction specifies the condition under which the HALT mode is released.

Table 16-2. HALT Mode Release Condition

Format: HALT $b_3b_2b_1b_0$

Bit	HALT mode release condition[Note 1]
b_3	Enables release by IRQ×××× when 1[Notes 2, 4]
b_2	Fixed to “0”
b_1	Enables forced release by IRQTM1 when 1[Notes 3, 4]
b_2	Fixed to “0”

Notes 1. HALT 0000B enables only reset ($\overline{\text{RESET}}$ input, power-ON/power-down reset).

2. IP××× must be 1.

3. The HALT mode is released regardless of the status of IPTM1.

4. Even if the HALT instruction is executed while IRQ××× = 1, the HALT instruction is ignored (treated as a NOP instruction), and the HALT mode is not set.

16.2.2 Start Address after HALT Mode Is Released

The address from which program execution is started after the HALT mode has been released differs depending on the release condition and interrupt enable condition.

Table 16-3. Start Address after HALT Mode Is Released

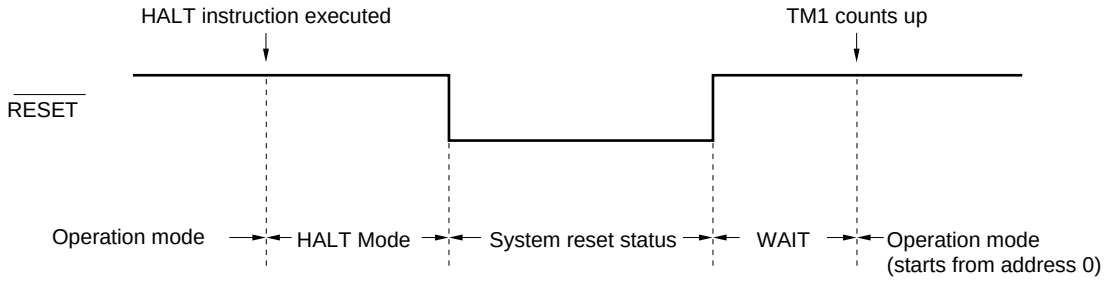
Release condition	Start address after HALT mode is released
Reset[Note 1]	Address 0
IRQ×××[Note 2]	Address next to HALT instruction in DI status Interrupt vector in EI status (if two or more IRQ××× are set, interrupt vector with highest priority)

Notes 1. $\overline{\text{RESET}}$ input and power-ON/power-down reset are valid.

2. IP××× must be 1 except when the HALT mode is forcibly released by IRQTM1.

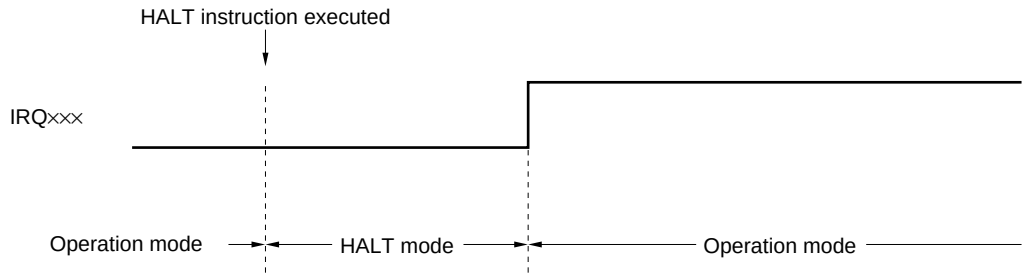
Figure 16-1. Releasing HALT Mode

(a) By $\overline{\text{RESET}}$ input

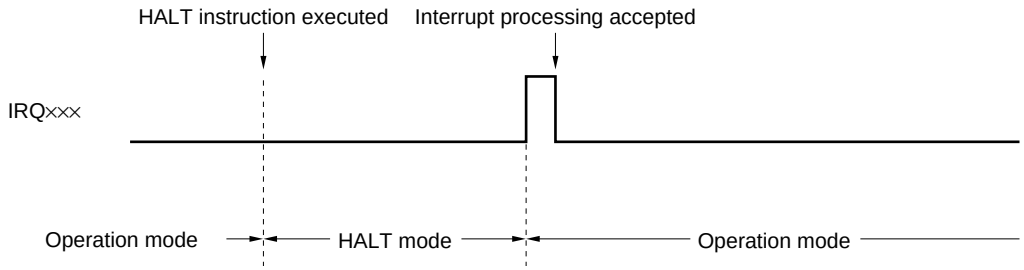


WAIT: Wait time until TM1 counts 256 clocks divided by 512
$256 \times 512/f_{CC} + \alpha$ (approx. 65 ms + α, f_{CC} = 2 MHz)
α: Oscillation growth time (differs depending on resonator)

(b) By IRQ××× (in DI status)



(c) By IRQ××× (in EI status)



16.2.3 HALT Mode Setting Conditions

(1) Forced releasing by IRQTM1

	Setting conditions
Release by external clock	● Timer 0 and timer 1 are used as 16-bit timer (TM0CK1 = 1, TM0CK0 = 1, TM1CK1 = 1, TM1CK0 = 1) ● Timer 0 and timer 1 are enabled to operate (TM0EN = 1, TM1EN = 1) ● Interrupt flag of timer 1 is cleared (IRQTM1 = 0)
Release by internal clock	● Timer 1 is enabled to operate ● Interrupt request flag of timer 1 is cleared (IRQTM1 = 0)

(2) Release by interrupt request flag (IRQ×××)

● Peripheral hardware used to release HALT mode is enabled to operate in advance.

Timer 0	Operation enabled (TM0EN = 1)
Timer 1	Operation enabled (TM1EN = 1)
Timer 0 + timer 1	Timer 1 selects count up signal from timer 0 as count pulse (TM1CK1 = 1, TM1CK0 = 1). Timer 0 and timer 1 are enabled to operate (TM0EN = 1, TM1EN = 1)
Basic interval timer	Always enabled to operate
Serial interface	Serial interface circuit is enabled to operate (SIOTS = 1, SIOEN = 1)
INT pin	Edge selected

- Clear the interrupt request flag (IRQ×××) of the peripheral hardware used to release the HALT mode to 0.
- Set the interrupt enable flag (IP×××) of the peripheral hardware used to release the HALT mode to 1.

Caution Be sure to include a NOP instruction immediately before the HALT instruction.
By doing so, a time of one instruction is created between the IRQ××× manipulation instruction and HALT instruction. Consequently, clearing IRQ××× is correctly reflected on the HALT instruction in the case, for example, of the CLR1 IRQ××× instruction (refer to Example 1 below). Unless a NOP instruction is described immediately before the HALT instruction, the CLR1 IRQ××× instruction is not correctly reflected on the HALT instruction, and the HALT mode is not set (Example 2).

Example 1. Correct program

```
        ⋮
(Setting of IRQxxx)
        ⋮

  CLR1    IRQxxx
  NOP             ; Describe NOP instruction immediately before HALT instruction.
                  ; (Clearing of IRQxxx is correctly reflected on HALT instruction.)
  HALT    1000B   ; Correctly execute HALT instruction (HALT mode is set).
        ⋮
        ⋮
        ⋮
```

2. Incorrect program

```
        ⋮
(Setting of IRQxxx)
        ⋮

  CLR1    IRQxxx  ; Clearing of IRQxxx is not reflected on HALT instruction.
                  ; (It is reflected on instruction next to HALT instruction.)
  HALT    1000B   ; HALT instruction is ignored (HALT mode is not set).
        ⋮
        ⋮
        ⋮
```

16.3 STOP MODE

16.3.1 Setting of STOP Mode

The STOP mode is set by executing the STOP instruction.

The operand $b_3b_2b_1b_0$ of the STOP instruction specifies the condition under which the STOP mode is to be released.

Table 16-4. STOP Mode Release Condition

Format: STOP $b_3b_2b_1b_0$

Bit	STOP mode release condition[Note 1]
b_3	Enables release of STOP mode by IRQ××× when 1[Note 2]
b_2	Fixed to “0”
b_1	Fixed to “0”
b_0	Fixed to “0”

Notes 1. STOP 0000B enables only reset ($\overline{\text{RESET}}$ input or power-ON/power-down reset). The internal circuitry of the microcontroller is initialized to the status immediately after reset when STOP 0000B is executed.

2. IP××× must be 1. The STOP mode cannot be released by IRQTM1.
Even if the STOP instruction is executed when IRQ××× = 1, the STOP instruction is ignored (treated as NOP), and the STOP mode is not set.

16.3.2 Start Address After STOP Mode Is Released

The address from which program execution is started after the STOP mode has been released differs depending on the release condition and interrupt enable condition.

Table 16-5. Start Address after STOP Mode Is Released

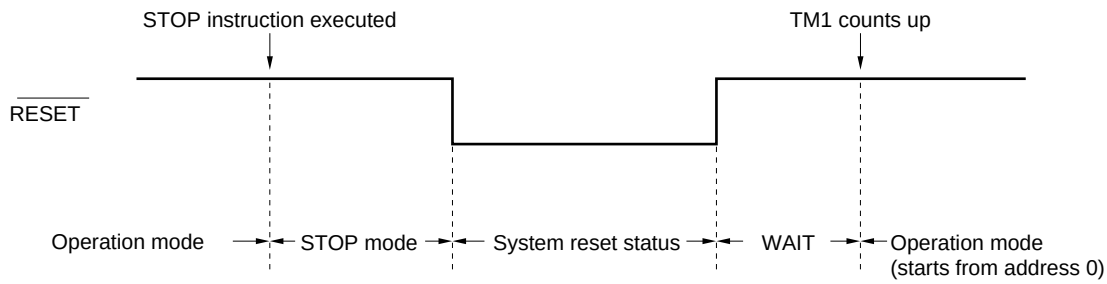
Release condition	Start address after STOP mode is released
Reset[Note 1]	Address 0
IRQ×××[Note 2]	Address next to that of STOP instruction in DI status
	Interrupt vector in EI status (If two or more IRQ××× are set, interrupt vector with highest priority)

Notes 1. Only $\overline{\text{RESET}}$ input and power-ON/power-down reset are valid.

2. IP××× must be 1. The STOP mode cannot be released by IRQTM1.

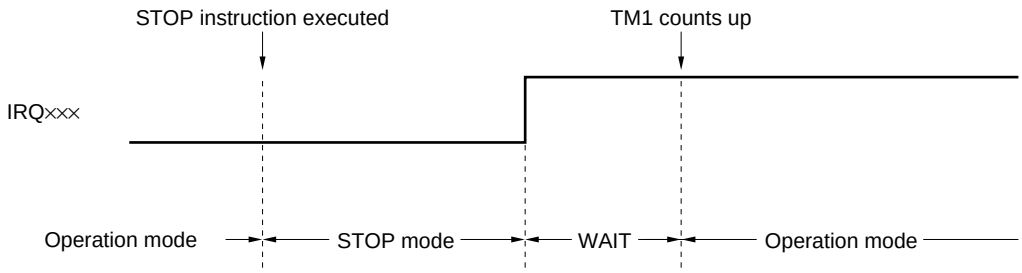
Figure 16-2. Releasing STOP Mode

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



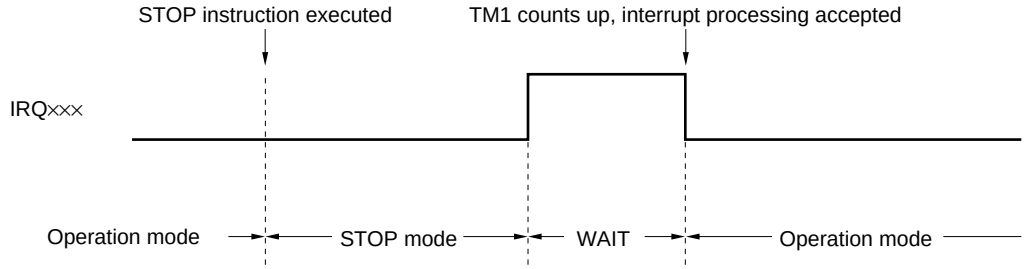
WAIT: Wait time until TM1 counts 256 clocks divided by 512
$256 \times 512/f_{CC} + \alpha$ (approx. 65 ms + α, f_{CC} = 2 MHz)
α: Oscillation growth time (differs depending on resonator)

(b) Releasing STOP mode by IRQ××× (in DI status)



WAIT: Wait time until TM1 counts (n + 1) clocks divided by m
$(n + 1) \times m/f_{CC} + \alpha$ (n and m are values immediately before STOP mode is set)
α: oscillation growth time (differs depending on resonator)

(c) Releasing STOP mode by IRQ××× (in EI status)



WAIT: Wait time until TM1 counts (n + 1) clocks divided by m $(n + 1) \times m/f_{CC} + \alpha$
(n and m are values immediately before STOP mode is set)
α: oscillation growth time (differs depending on resonator)

16.3.3 STOP Mode Setting Conditions

When STOP mode is to be released by IRQ×××

Releasing by IRQ	● Selects edge of signal to be input from INT pin (IEGMD1, IEGMD0). ● Sets modulo register value of timer 1 (that creates oscillation stabilization wait time). ● Clears interrupt request flag (IRQ) of INT pin to 0. ● Sets interrupt enable flag (IP) of INT pin to 1.
Releasing by IRQSIO	● Selects external clock input from $\overline{\text{SCK}}$ pin as source clock (SIOCK1 = 0, SIOCK0 = 0). ● Enables serial interface to operate (SIOTS = 1). ● Sets modulo register value of timer 1 (that sets oscillation stabilization time). ● Clears interrupt request flag of serial interface (IRQSIO) to 0. ● Sets interrupt enable flag of serial interface (IPSIO) to 1.
Releasing by IRQTM0	● Selects external clock input from INT pin as source clock of timer 0 (TM0CK1 = 1, TM0CK0 = 1). ● Sets modulo register value of timer 0. ● Sets modulo register value and source clock of timer 1 (that creates oscillation stabilization time). ● Enables timer 0 to operate (TM0EN = 1). ● Clears interrupt request flag (IRQTM0) to 0 ● Sets interrupt enable flag of timer 0 (IPTM0) to 1.

Caution **Be sure to include a NOP instruction before the STOP instruction. By doing so, a time of one instruction is created between the IRQ×××× manipulation instruction and STOP instruction. As a result, clearing IRQ××××, for example, is correctly reflected on the STOP instruction when the IRQ×××× instruction is executed (refer to Example 1 below). Unless a NOP instruction is described immediately before the STOP instruction, the CLR1 IRQ×××× instruction is not reflected on the STOP instruction, and the STOP mode is not set (Example 2).**

Example 1. Correct program

```
        ⋮
(Setting of IRQ××××)
        ⋮

  CLR1
  NOP       IRQ××××   ; Describe NOP instruction immediately before the STOP instruction.
                      ; (Clearing IRQ×××× is correctly reflected on the STOP instruction.)
  STOP      1000B     ; STOP instruction is correctly executed (STOP mode is set).
        ⋮
```

2. Incorrect program

```
        ⋮
(Setting of IRQ××××)
        ⋮

  CLR1      IRQ××××   ; Clearing IRQ×××× is not reflected on the STOP instruction.
                      ; (It is reflected on the instruction next to the STOP instruction.)
  STOP      1000B     ; The STOP instruction is ignored (STOP mode is not set).
        ⋮
```

[MEMO]

★ CHAPTER 17 RESET

The μPD17134A subseries is reset in the following four ways.

(1) By $\overline{\text{RESET}}$ input
(2) Power-ON/power-down reset that resets the microcontroller on power application or when supply voltage drops
(3) Watchdog timer that resets the microcontroller in case of a program hang-up
(4) Reset because of overflow/underflow of address stack

The power-ON/power-down reset function is effective when the supply voltage is 4.5 to 5.5 V.

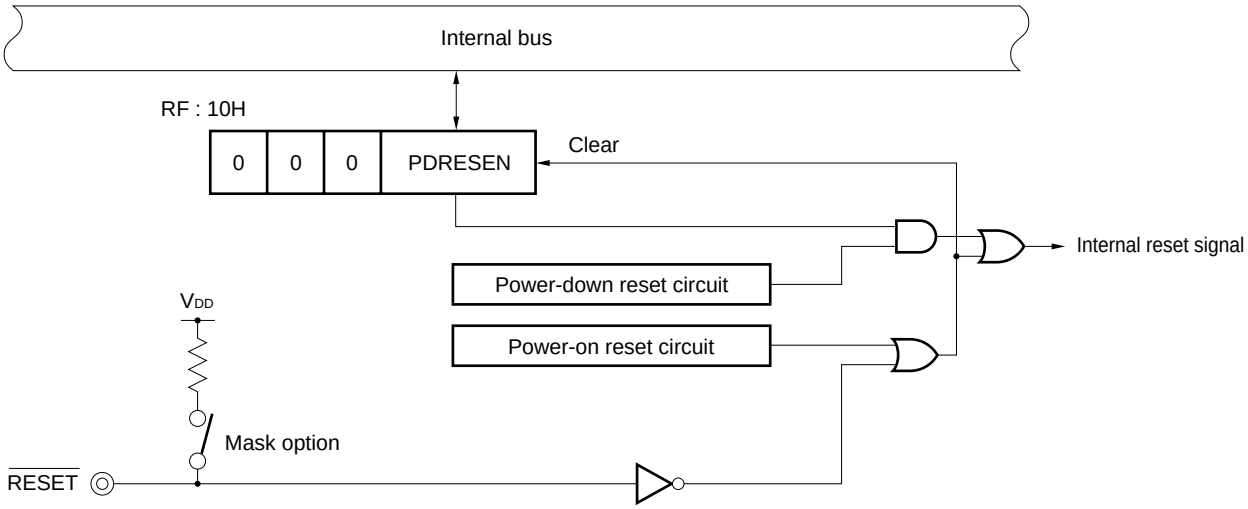
17.1 RESET FUNCTION

The reset function is used to initialize the device operation. How the device is initialized differs depending on the type of reset effected.

Table 17-1. Hardware Status at Reset

Hardware \ Reset method		• $\overline{\text{RESET}}$ input during operation • Internal power-ON/ power-DOWN reset during operation	• $\overline{\text{RESET}}$ input in standby mode • Internal power-ON/ power-DOWN reset in standby mode	• Overflow of watchdog timer • Overflow and underflow of stack
Program counter		0000H	0000H	0000H
Port	I/O mode	Input	Input	Input
	Output latch	0	0	Undefined
General-purpose data memory	Other than DBF	Undefined	Retains previous status	Undefined
	DBF	Undefined	Undefined	Undefined
System register	Other than WR	0	0	0
	WR	Undefined	Retains previous status	Undefined
Control register		SP = 5H, IRQTM1 = 1, TM1EN = 1, IRQBTM = 0, and INT = status of INT pin at that time. Others are 0. Refer to **CHAPTER 9 REGISTER FILE (RF)**.		SP = 5H, INT = status of INT pin at that time. Others retain previous status.
Timer 0 and timer 1	Count register	00H	00H	Timer 0: 00H, timer 1: undefined
	Modulo register	FFH	FFH	FFH
Counter of basic interval timer		Undefined	Undefined	Undefined (40H if watchdog timer overflows)
Shift register of serial interface (SIOSFR)		Undefined	Retains previous status	Undefined
Data register of A/D converter (ADCR)		00H	00H	00H

Figure 17-1. Reset Block Configuration



17.2 RESETTING

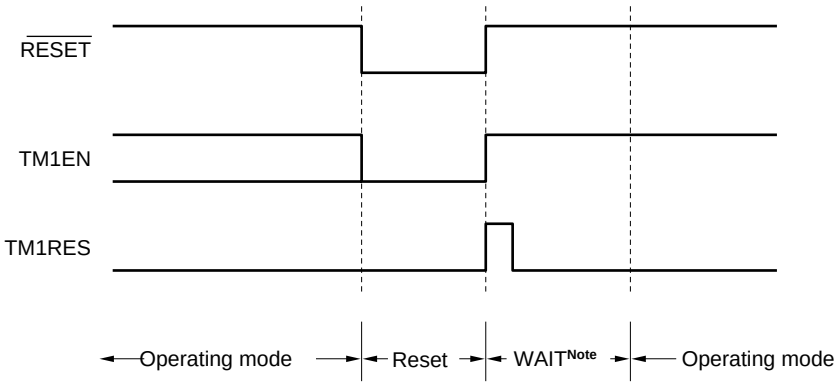
Operation when system reset is caused by the $\overline{\text{RESET}}$ pin is shown in the figure below.

If the $\overline{\text{RESET}}$ pin is set from low to high, system clock oscillation starts and an oscillation stabilization wait occurs with the timer 1. Program execution starts from address 0000H.

If power-on reset is used, the reset signals shown in Figure 17-2 are internally generated. Operation is the same as that when reset is caused externally by the $\overline{\text{RESET}}$ pin.

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

Figure 17-2. Reset Operation



Note This is oscillation stabilization wait time. Operating mode is set when timer 1 counts system clocks (f_{CC}) 512 × 256 counts approx. 65 ms at f_{CC} = 2 MHz).

17.3 POWER-ON/POWER-DOWN RESET FUNCTION

The μPD17134A subseries is provided with two reset functions to prevent malfunctions from occurring in the microcontroller. They are the power-on reset function and power-down reset function. The power-on reset function resets the microcontroller when it detects that power was turned on. The power-down reset function resets the microcontroller when it detects drops in the power voltage.

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

Caution When designing an application circuit that calls for high reliability, do not depend on the internal power-ON/power-DOWN reset function only. Make sure that an external $\overline{\text{RESET}}$ signal is input.

17.3.1 Conditions Required to Enable the Power-On Reset Function

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

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

(1) The power voltage must be within 4.5 to 5.5 V during normal operation, including the standby state.
(2) The frequency of the system clock oscillator must be 400 kHz to 4 MHz. **Note**
(3) The power-down reset function must be enabled during normal operation, including the standby state.
(4) The power voltage must rise from 0 V to the specified voltage.
(5) The time it takes for the power voltage to rise from 0 to 2.7 V must be shorter than the oscillation stabilization wait time (system clock f_{CC} = 512 × 256 counts, about 65 ms, at f_{CC} = 2 MHz) counted in timer 1.

Note Applies to the μPD17135A, 17137A, and 17P137A.
When the μPD17134A, 17136A, or 17P136A is used, f_{CC} = 400 kHz to 2 MHz.

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

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

17.3.2 Power-On Reset Function and Operation

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

The power-on reset circuit operates under a lower voltage than the other internal circuits. It initializes the microcontroller regardless whether the oscillation circuit is operating. When the reset is terminated, timer 1 counts the number of oscillation pulses sent from the oscillator until it reaches the specified value. Within this period, oscillation becomes stable and the power voltage applied to the microcontroller enters the range (V_{DD} = 2.7 to 5.5 V at 400 kHz to 4 MHz) in which the microcontroller is guaranteed to operate.

When this period elapses, the microcontroller enters normal operation mode. Figure 17-3 shows an example of the power-on reset operation.

Functions of the power-on reset

(1) This circuit always monitors the voltage applied to the V_{DD} pin.

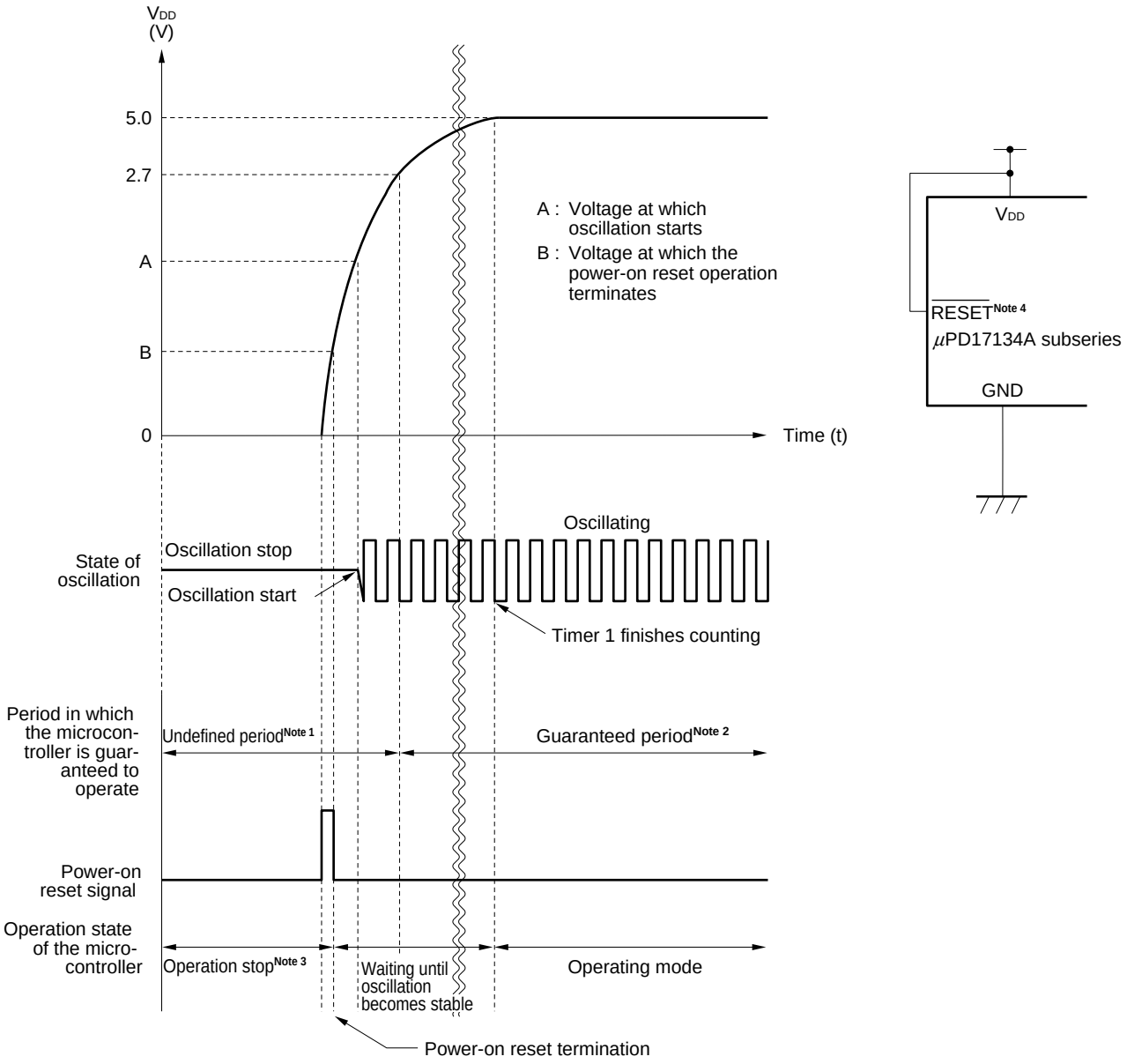
(2) This circuit resets the internal circuit of the microcomputer, regardless of whether the oscillator circuit operates or not, when the supply voltage rises, until the voltage reaches the power-ON reset clear voltage (V_{DD} = 1.5 V TYP.).[Note]

(3) This circuit stops oscillation during the reset operation.

(4) When reset is released, timer 1 counts oscillation pulses. The microcontroller waits until oscillation becomes stable and the power voltage becomes V_{DD} = 2.7 V or higher.

Note The internal circuit of the microcontroller is not reset until the supply voltage reaches the level at which the internal circuit can operate (i.e., internal reset signal can be accepted).

Figure 17-3. Example of the Power-On Reset Operation



Notes 1. During the operation-undefined period, not all of the operations specified for the μPD17134A subseries are guaranteed. The power-on reset operation is guaranteed in this period.

2. The operation-guaranteed period refers to the time in which all the operations specified for the μPD17134A subseries are guaranteed.

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

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

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

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

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

17.3.4 Power-Down Reset Function and Operation

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

When this function detects a power voltage drop, it issues the reset signal to the microcontroller. It then initializes the microcontroller. Stopping oscillation during reset prevents the power voltage in the microcontroller from fluctuating out of control. When the specified power voltage recovers and the power-down reset operation is terminated, the microcontroller waits the time required for stable oscillation using the timer. The microcontroller then enters normal operation (starts from address 0).

Figure 17-4 shows an example of the power-down operation. Figure 17-5 shows an example of reset operation during the period from power-down reset to power recovery.

Functions of the power-down reset

(1) This circuit always monitors the voltage applied to the V_{DD} pin.

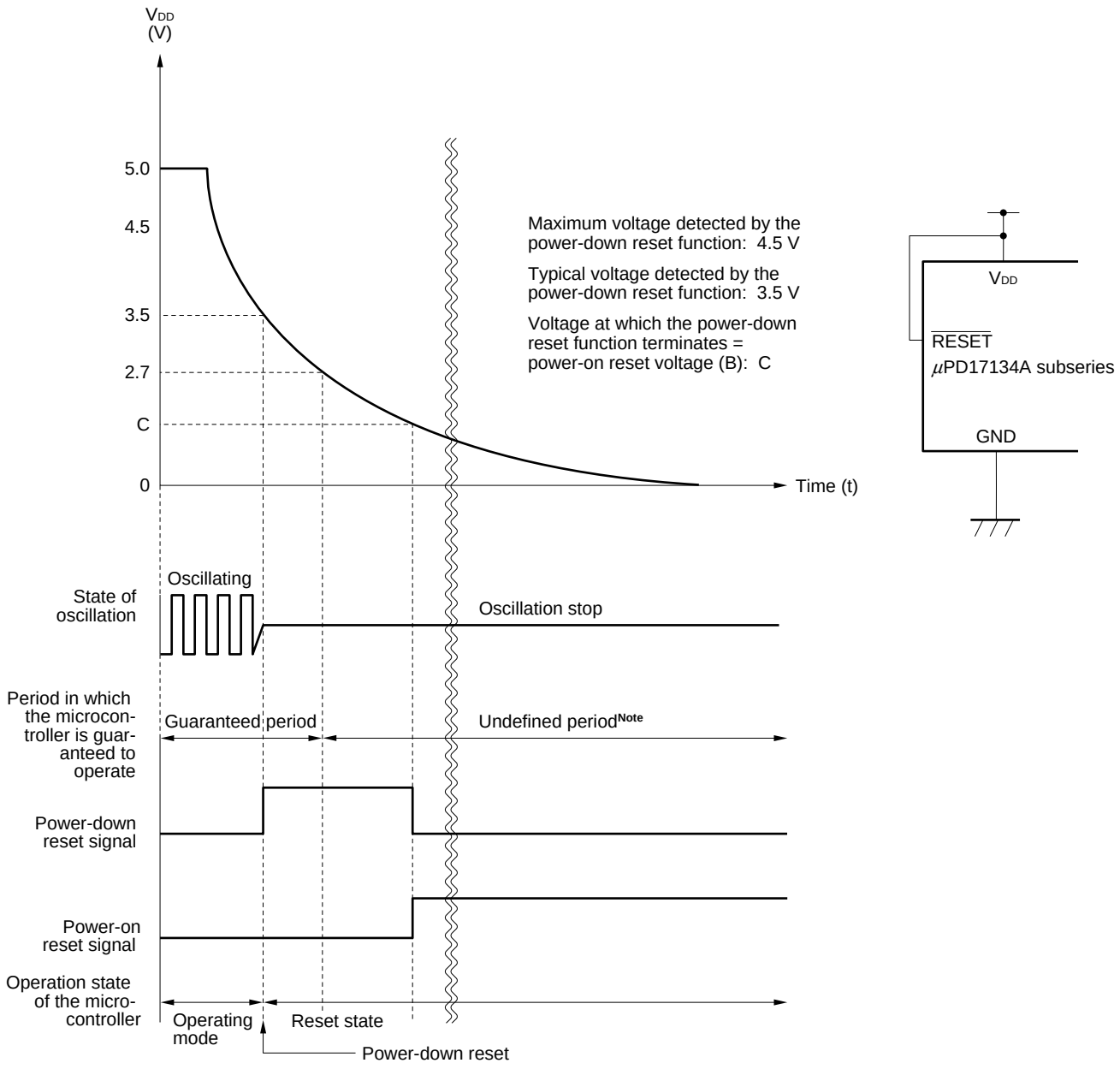
(2) When this circuit detects a power voltage drop, it issues a reset signal to the other parts of the microcontroller. It continues to send this reset signal until the power voltage recovers or all the functions in the microcontroller stop.

(3) This circuit stops oscillation during the reset operation to prevent software crashes.
When the power voltage recovers to the low-voltage detection level (3.5 V TYP., 4.5 V MAX.) before the power-down reset function stops, the microcontroller waits the time required for stable oscillation using timer 1, then enters normal operation mode.

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

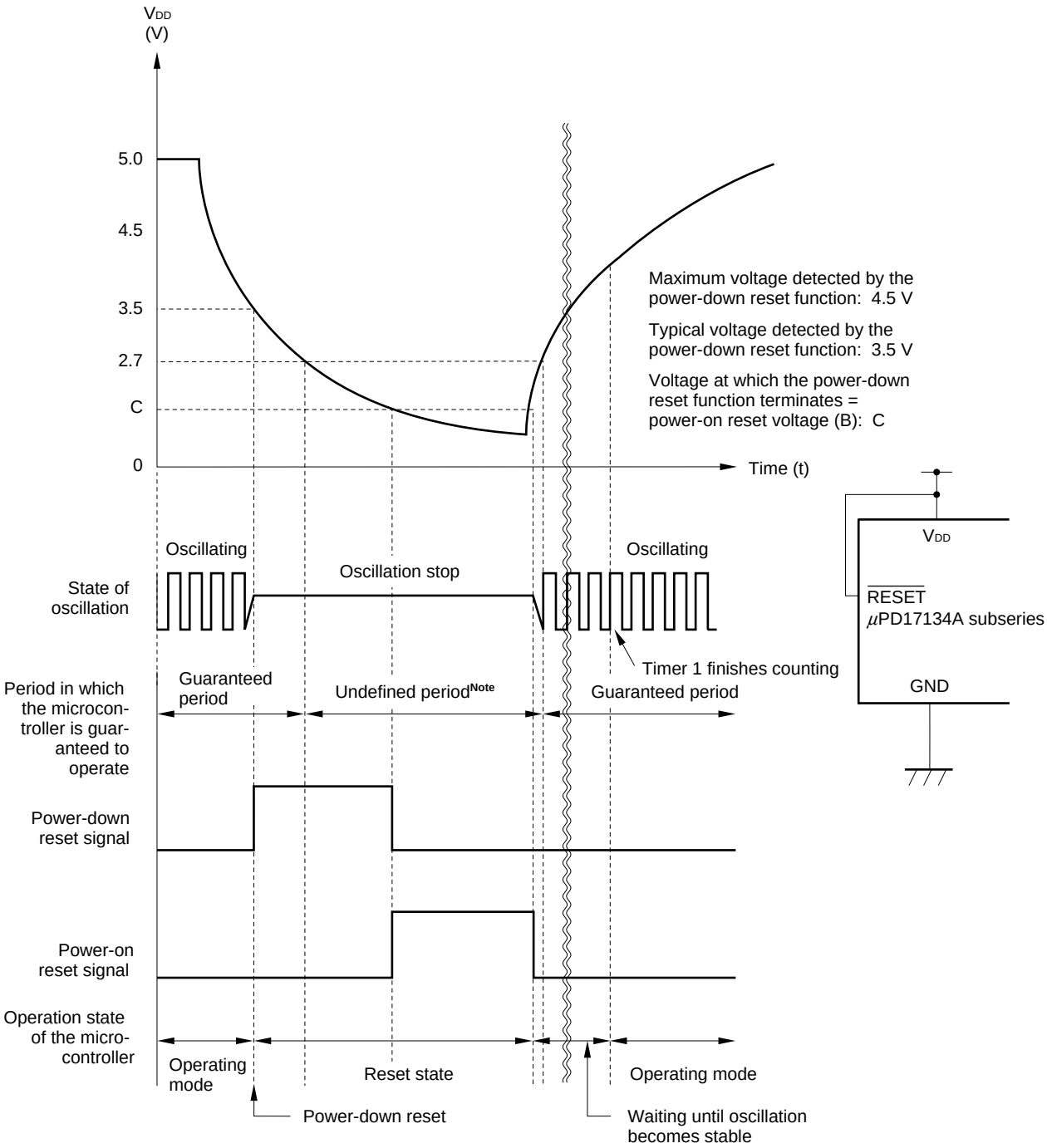
(5) After the power-down reset function stops and the power voltage recovers before it reaches 0 V, the microcontroller waits using timer 1 until oscillation becomes stable and the power voltage (V_{DD}) reaches 2.7 V. The microcontroller then enters normal operation mode.

Figure 17-4. Example of the Power-Down Reset Operation



Note In the operation-undefined period, not all the operations specified for the μPD17134A subseries are guaranteed. The power-down reset operation, which continues to issue a reset signal until all the functions in the microcontroller stop, is guaranteed in this period.

Figure 17-5. Example of Reset Operation during the Period from Power-Down Reset to Power Recovery



Note In the operation-undefined period, not all the operations specified for the μPD17134A subseries are guaranteed. The power-down reset operation, which continues to issue the reset signal until all the functions in the microcontroller stop, is guaranteed in this period.

[MEMO]

CHAPTER 18 ONE-TIME PROM WRITING/VERIFYING

The on-chip program memories of the μPD17P136A and 17P137A are is a 2048 × 16-bit one-time PROM.

Pins listed in Table 18-1 are used for one-time PROM writing/verifying. The address is updated by the clock signal input from the CLK pin.

Caution PIB$_0$/V$_{PP}$ pin is used as V$_{PP}$ pin in program writing/verifying mode. Therefore, there is a possibility of overrunning of the microcontroller when higher voltage than V$_{DD}$ + 0.3 V is applied to PIB$_0$/V$_{PP}$ pin in normal operation mode. Pay careful attention to pin protection.

Table 18-1. Pins Used for Writing/Verifying Program Memory

Pin	Function
V$_{PP}$	Applies program voltage. Apply +12.5 V to this pin.
V$_{DD}$	Power supply pin. Apply +6 V to this pin.
$\overline{\text{RESET}}$	System reset input pin. Used for initializing all states before setting program memory writing/verifying mode.
CLK	Clock input for updating address. Updates program memory address by inputting four pulses.
MD$_0$-MD$_3$	Select operation mode.
D$_0$-D$_7$	8-bit data I/O pins.

18.1 DIFFERENCES BETWEEN MASK ROM VERSION AND ONE-TIME PROM MODEL

The μPD17P136A and 17P137A are microcontrollers replacing the program memory of the on-chip mask ROM version μPD17136A and 17137A to one-time PROM. Table 18-2 shows the differences between mask ROM version and one-time PROM version.

Differences between each product are only its program memory, program size, address register size, and whether it can specify mask option or not. The CPU function and internal peripheral hardware of each product are the same. Therefore, the μPD17P136A can be used for evaluating program of the μPD17134A/17136A in system development. Also, the μPD17P137A can be used for evaluating the μPD17135A/17137A in the same way.

Table 18-2. Differences Between Mask ROM Version and One-Time PROM Version

Item	μPD17134A/17135A	μPD17136A/17137A	μPD17P136A/17P137A
ROM	Mask ROM		One-time PROM
	1024 × 16 bits (0000H to 03FFH)	2048 × 16 bits (0000H to 07FFH)	
Program counter	10 bits	11 bits	
Address register			
Address stack register			
P0D, P1A, and P1B pins and pull-up resistor of $\overline{\text{RESET}}$ pin	Mask option		Not available
V_{PP} pin and operating mode select pin	Not available		Provided
★ Quality grade	Standard Special [(A), (A1)]		Standard

★ **Caution The PROM model is highly compatible with the mask ROM model in terms of functions but its internal ROM circuit and electrical characteristics are partially different from those of the mask ROM model. To replace the PROM model with the mask ROM model, thoroughly evaluate the application by using a sample of the mask ROM model.**

18.2 OPERATION MODE WHEN PROGRAM MEMORY IS WRITTEN/VERIFIED

The μPD17P136A and 17P137A enter a program memory write/verify mode when they have been reset for a fixed time (V_{DD} = 5 V, $\overline{\text{RESET}}$ = 0 V) and then +6 V is applied to the V_{DD} pin and +12.5 V to the V_{PP} pin. In this mdoe, the operation modes shown in the table below can be selected depending on the setting of the MD_0 through MD_3 pins. Connect V_{ADC} directly to V_{DD}. Connect all the other pins to GND via pull-down resistor.

Table 18-3. Setting 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	Program memory address 0 clear
		L	H	H	H	Write mode
		L	L	H	H	Verify mode
		H	×	H	H	Program inhibit mode

Remark ×: don't care (L or H)

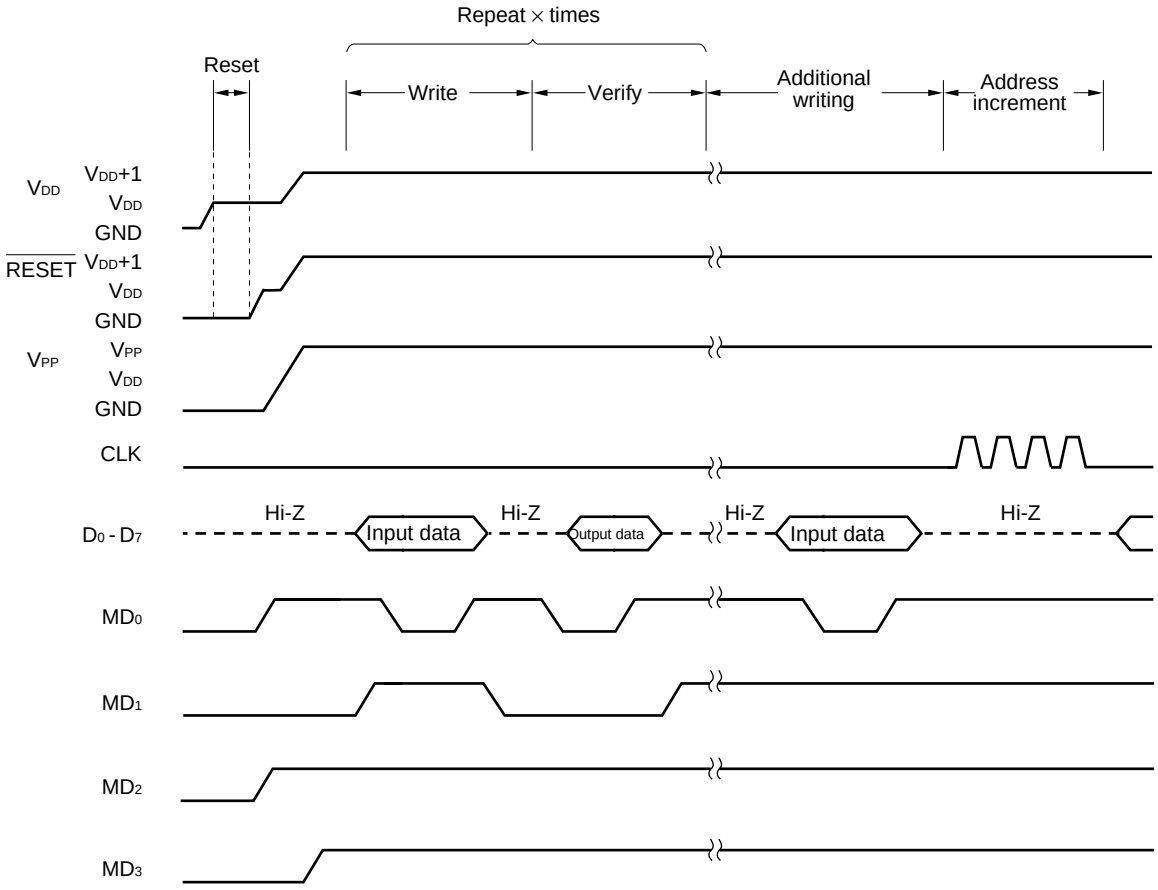
18.3 WRITING PROCEDURE OF PROGRAM MEMORY

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

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

Figure 18-1 shows the procedures of (2) through (12).

Figure 18-1. Procedure of Program Memory Writing

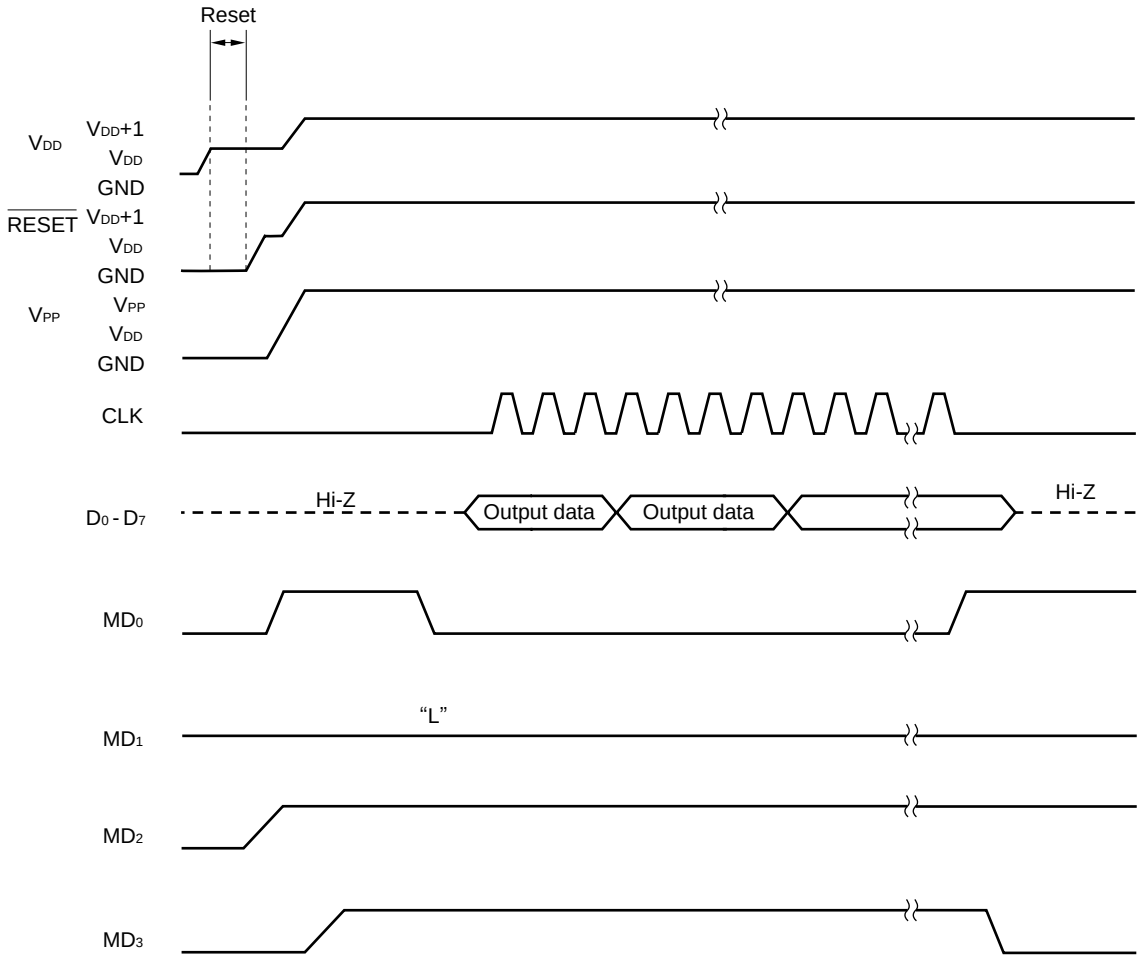


18.4 READING PROCEDURE OF PROGRAM MEMORY

(1) Connect V_{ADC} directly to V_{DD}, and all the other pins to GND via pull-down resistor. Make the CLK pin low.
(2) Apply 5 V to the V_{DD} pin. Make V_{PP} pin and $\overline{RESET}$ pin low.
(3) Wait for 10 μs. Then, apply 5 V to $\overline{RESET}$ pin.
(4) Set the program memory address 0 clear mode using mode selector pins.
(5) Apply 6 V to V_{DD} and $\overline{RESET}$ and 12.5 V to V_{PP}.
(6) Set mode selector pins to the program inhibit mode.
(7) Set the verify mode. When clock pulses are input to the CLK pin, data for each address can be sequentially output with four clocks as one cycle.
(8) Set the program inhibit mode.
(9) Set the program memory address 0 clear mode.
(10) Change the voltage of V_{DD} and V_{PP} pins to 5 V.
(11) Turn off the power.

Figure 18-2 shows the program reading procedure (2) through (9).

Figure 18-2. Procedure of Program Memory Reading



CHAPTER 19 INSTRUCTION SET

19.1 OVERVIEW OF THE INSTRUCTION SET

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	AR		
		INC	IX		
		MOVT	DBF, @AR		
		BR	@AR		
		CALL	@AR		
		RET			
		RETSK			
		EI			
		DI			
		RETI			
		PUSH	AR		
		POP	AR		
		GET	DBF, p		
		PUT	p, DBF		
		PEEK	WR, rf		
		POKE	rf, WR		
		RORC	r		
		STOP	s		
		HALT	h		
		NOP			
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	CALL	addr
1 1 0 1	D			MOV	m, #n4
1 1 1 0	E			SKT	m, #n
1 1 1 1	F			SKF	m, #n

19.2 LEGEND

AR	:	Address register
ASR	:	Address stack register indicated by stack pointer
addr	:	Program memory address (11 bits)
BANK	:	Bank register
CMP	:	Compare register
CY	:	Carry flag
DBF	:	Data buffer
h	:	Halt release condition
INTEF	:	Interrupt enable flag
INTR	:	Register saved automatically to interrupt stack
INTSK	:	Interrupt stack register
IX	:	Index register
MP	:	Data memory row address pointer
MPE	:	Memory pointer enable flag
m	:	Data memory address indicated by m_R and m_C
m_R	:	Data memory row address (high-order)
m_C	:	Data memory column address (low-order)
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
(X)	:	Contents addressed by X

19.3 LIST OF THE INSTRUCTION SET

Group	Mnemonic	Operand	Operation	Machine code			
				OP code	Operand		
Add	ADD	r, m	(r) ← (r) + (m)	00000	m_R	m_C	r
		m, #n4	(m) ← (m) + n4	10000	m_R	m_C	n4
	ADDC	r, m	(r) ← (r) + (m) + CY	00010	m_R	m_C	r
		m, #n4	(m) ← (m) + n4 + CY	10010	m_R	m_C	n4
	INC	AR	AR ← AR + 1	00111	000	1001	0000
		IX	IX ← IX + 1	00111	000	1000	0000
Subtract	SUB	r, m	(r) ← (r) – (m)	00001	m_R	m_C	r
		m, #n4	(m) ← (m) – n4	10001	m_R	m_C	n4
	SUBC	r, m	(r) ← (r) – (m) – CY	00011	m_R	m_C	r
		m, #n4	(m) ← (m) – n4 – CY	10011	m_R	m_C	n4
Logical Operation	OR	r, m	(r) ← (r) ∨ (m)	00110	m_R	m_C	r
		m, #n4	(m) ← (m) ∨ n4	10110	m_R	m_C	n4
	AND	r, m	(r) ← (r) ∧ (m)	00100	m_R	m_C	r
		m, #n4	(m) ← (m) ∧ n4	10100	m_R	m_C	n4
	XOR	r, m	(r) ← (r) ∀ (m)	00101	m_R	m_C	r
		m, #n4	(m) ← (m) ∀ n4	10101	m_R	m_C	n4
Judge	SKT	m, #n	CMP ← 0, if (m) ∧ n = n, then skip	11110	m_R	m_C	n
	SKF	m, #n	CMP ← 0, if (m) ∧ n = 0, then skip	11111	m_R	m_C	n
Compare	SKE	m, #n4	(m) –n4, skip if zero	01001	m_R	m_C	n4
	SKNE	m, #n4	(m) –n4, skip if not zero	01011	m_R	m_C	n4
	SKGE	m, #n4	(m) –n4, skip if not borrow	11001	m_R	m_C	n4
	SKLT	m, #n4	(m) –n4, skip if borrow	11011	m_R	m_C	n4
Rotate	RORC	r	→ CY → $(r)_{b3}$ → $(r)_{b2}$ → $(r)_{b1}$ → $(r)_{b0}$ →	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	DBF, @AR	SP ← SP –1, ASR ← PC, PC ← AR, DBF ← (PC), PC ← ASR, SP ← SP +1	00111	000	0001	0000

Group	Mnemonic	Operand	Operation	Machine code			
				OP code	Operand		
Transfer	PUSH	AR	SP ← SP –1, ASR ← AR	00111	000	1101	0000
	POP	AR	AR ← ASR, SP ← SP+1	00111	000	1100	0000
	PEEK	WR, rf	WR ← (rf)	00111	rf_R	0011	rf_C
	POKE	rf, WR	(rf) ← WR	00111	rf_R	0010	rf_C
	GET	DBF, p	DBF ← (p)	00111	p_H	1011	p_L
	PUT	p, DBF	(p) ← DBF	00111	p_H	1010	p_L
Branch	BR	addr	PC ← addr	01100	addr		
		@AR	PC ← AR	00111	000	0100	0000
Sub-routine	CALL	addr	SP ← SP –1, ASR ← PC, PC ← addr	11100	addr		
		@AR	SP ← SP –1, ASR ← PC, PC ← AR	00111	000	0101	0000
	RET		PC ← ASR, SP ← SP +1	00111	000	1110	0000
	RETSK		PC ← ASR, SP ← SP +1 and skip	00111	001	1110	0000
	RETI		PC ← ASR, INTR ← INTSK, SP ← SP +1	00111	100	1110	0000
Interrupt	EI		INTEF ← 1	00111	000	1111	0000
	DI		INTEF ← 0	00111	001	1111	0000
Others	STOP	s	STOP	00111	010	1111	s
	HALT	h	HALT	00111	011	1111	h
	NOP		No operation	00111	100	1111	0000

19.4 ASSEMBLER (AS17K) EMBEDDED 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	n = 0, 1

19.5 INSTRUCTIONS

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

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

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

Sets carry flag CY, if a carry occurs as a result of the addition. Resets the carry flag, if no carry occurs.
If the addition result is other than zero, zero flag Z is reset, regardless of compare flag CMP.
If the result is zero with the compare flag reset (CMP = 0), the zero flag Z is set.
If the result is zero with the compare flag set (CMP = 1), the zero flag Z is not changed.
Addition can be executed in binary 4 bits or BCD. The BCD flag for the PSWORD specifies what kind of addition is to be executed.

<3> Example 1

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

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

Example 2

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

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

Note

Register	RP							
	RPH				RPL			
Bit	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Data	0	0	0	Bank	Row Address			B C D

RP (general register pointer) is assigned in the system register, as shown above.
Therefore, to set bank 0 and row address 2 in a general register, 00H must be stored in RPH and 04H, in RPL.
In this case, the subsequent arithmetic operation is executed in binary 4-bit operation, because the BCD flag is reset.

Example 3

To add the address 0.6FH contents to the address 0.03H contents and store the result in address 0.3H. At this time, data memory address 0.6FH can be specified, by selecting data memory address 2FH, if IXE = 1, IXH = 0, IXM = 4, and IXL = 0, i.e., IX = 0.40H.

(0.03H) ← (0.03H) + (0.6FH)

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

```
MEM003      MEM     0.03H
MEM02F      MEM     0.2FH
            MOV     RPH, #00H           ; General register bank 0
            MOV     RPL, #00H           ; General register row address 0
            MOV     IXH, #00H           ; IX ← 00001000000B
            MOV     IXM, #04H           ;
            MOV     IXL, #00H           ;
            SET1    IXE                 ; IXE flag ← 1
            ADD     MEM003, MEM02F      ; IX                    00001000000B (0.40H)
                                        ; Bank operand OR )     00000101111B (0.2FH)
                                        ; Specified address     00001101111B (0.6FH)
```

Example 4

To add the address 0.3FH contents to the address 0.03H contents and store the result in address 0.03H. At this time, data memory address 2.3FH can be specified by specifying data memory address 2FH, if IXE = 1, IXH = 0, IXM = 1, and IXL = 0, i.e., IX = 0.10H.

(0.03H) ← (0.03H) + (0.3FH)

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

```
MEM003   MEM    0.03H
MEM02F   MEM    0.2FH
         MOV    BANK, #00H
         MOV    RPH, #00H          ; General register bank 0
         MOV    RPL, #00H          ; General register row address 0
         MOV    IXH, #00H          ; IX ← 00000010000B (0.10H) Note
         MOV    IXM, #01H
         MOV    IXL, #00H
         SET1   IXE                ; IXE flag ← 1
         ADD    MEM003, MEM02F     ; IX                    00000010000B (0.10H)
                                   ; Bank operand OR )     00000101111B (0.2FH)
                                   ; Specified address     00100111111B (0.3FH)
```

Note

Register	IX											
	IXH				IXM				IXL			
Bit	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
Data	M P E	0	Bank 0	0		Row Address				Column address		

IX (index register) is assigned in the system register, as shown above,
Therefore, to specify IX = 0.10H, 00H must be stored in IXH. 01H in IXM, and 00H in IXL.
In this case, MP (memory pointer) for general register indirect transfer is invalid, because the MPE flag (memory pointer enable) is reset.

<4> Caution

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

```
MEM013   MEM    0.13H
MEM02F   MEM    0.2FH
         MOV    RPH, #00H          ; General register bank 0
         MOV    RPL, #00H          ; General register row address 0
         ADD    MEM013, MEM02F
                Indicates the general register column address.
                The low-order 4 bits (in this case, 03H) are valid
```

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

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

<1> OP code

10	8 7	4 3	0
10000	m_R	m_C	n4

<2> Function

When CMP = 0, (m) ← (m) + n4

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

When CMP = 1, (m) + n4

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

Sets carry flag CY, if a carry occurs as a result of the addition; resets the carry flag if no carry occurs.
If the addition result is other than zero, zero flag Z is reset, regardless of compare flag CMP.
If the result is zero with the compare flag reset (CMP = 0), the zero flag Z is set.
If the result is zero with the compare flag set (CMP = 1), the zero flag Z is not changed.
Addition can be executed in binary 4 bits or BCD. The BCD flag for the PSWORD specifies which kind of addition is to be executed.

<3> Example 1

To add 5 to the address 0.2FH contents, 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 address 0.6FH contents and store the result in address 0.6FH. At this time, data memory address 0.6FH can be specified by selecting data memory address 2FH, if IXE = 1, IXH = 0, IXM = 4, and IXL = 0, i.e., IX = 0.40H.

```
            (0.6FH) ← (0.6FH) + 05H
                      └── Address obtained as result of ORing index register contents, 0.40H,
                          and data memory address 0.2FH
MEM02F      MEM     0.2FH
            MOV     BANK, #00H      ; Data memory bank 0
            MOV     IXH, #00H       ; IX ← 00001000000B (0.40H)
            MOV     IXM, #04H
            MOV     IXL, #00H
            SET1    IXE             ; IXE flag ← 1
            ADD     MEM02F, #05H    ; IX                    00001000000B (0.40H)
                                    ; Bank operand OR )     00000101111B (0.2FH)
                                    ; Specified address     00001101111B (0.6FH)
```

Example 3

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

★

```
            (0.2FH) ← (0.2FH) + 05H
                      └── Address obtained as result of ORing index register contents, 0.00H,
                          and data memory address 0.2FH
MEM02F      MEM     0.2FH
            MOV     BANK, #00H      ; Data memory bank 0
            MOV     IXH, #00H       ; IX ← 00000000000B
            MOV     IXM, #00H
            MOV     IXL, #00H
            SET1    IXE             ; IXE flag ← 1
            ADD     MEM02F, #05H    ; IX                    00000000000B (0.00H)
                                    ; Bank operand OR )     00000101111B (0.2FH)
                                    ; Specified address     00000101111B (0.2FH)
```

<4> Caution

When the CMP flag = 1, the addition result is not stored.
When the BCD flag = 1, the decimal addition result 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 data memory contents to the general register contents with carry flag CY, and stores the result in general register.

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

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

★ By using this ADDC instruction, one or more nibbles can be easily added.
Sets carry flag CY, if a carry occurs as a result of the addition; resets the carry flag if no carry occurs.
If the addition result is other than zero, zero flag Z is reset, regardless of compare flag CMP.
If the result is zero with the compare flag reset (CMP = 0), the zero flag Z is set.
If the result is zero with the compare flag set (CMP = 1), the zero flag Z is not changed.
Addition can be executed in binary 4 bits or BCD. The BCD flag for PSWORD specifies which kind of addition is to be executed.

<3> Example 1

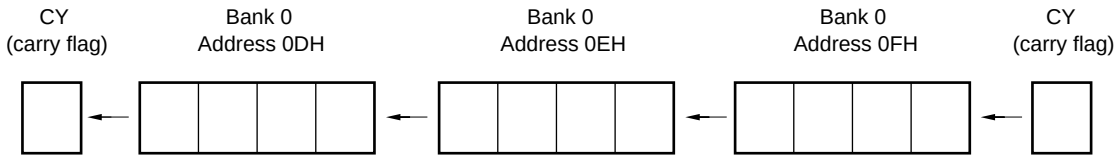
To add the 12-bit contents for addresses 0.0DH through 0.0FH to the 12-bit contents for addresses 0.2DH through 0.2FH, and store the result in the 12-bit contents for address 0.0DH to 0.0FH, when row address 0 (0.00H–0.0FH) of bank 0 is specified as a general register:

(0.0FH) ← (<u>0.0FH</u>) + (0.2FH)
(0.0EH) ← (<u>0.0EH</u>) + (0.2EH) + CY
(0.0DH) ← (<u>0.0DH</u>) + (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       ; Low-order nibble
         ADDC    MEM00E, MEM02E
         ADDC    MEM00D, MEM02D       ; High-order nibble
```

Example 2

To shift the 12-bit contents for addresses 0.2DH through 0.2FH 1 bit to the left, when row address 2 in bank 0 (0.20H–0.2FH) is specified as a general register:




```
MEM00D   MEM     0.0DH
MEM00E   MEM     0.0EH
MEM00F   MEM     0.0FH
MEM02D   MEM     0.2DH
MEM02E   MEM     0.2EH
MEM02F   MEM     0.2FH
         MOV     RPH, #00H          ; General register bank 0
         MOV     RPL, #04H          ; General register row address 2
         MOV     BANK, #00H         ; Data memory bank 0
         ADDC    MEM00F, MEM02F
         ADDC    MEM00E, MEM02E
         ADDC    MEM00D, MEM02D
```

Example 3

To add the address 0.0FH contents to the addresses 0.40H through 0.4FH contents, 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 for addresses 0.40H through 0.42H to the 12-bit contents for addresses 0.0DH through 0.0FH, and store the result in 12-bit contents for addresses 0.0DH through 0.0FH:

(0.0DH) ← (0.0DH) + (0.40H)
(0.0EH) ← (0.0EH) + (0.41H) + CY
(0.0FH) ← (0.0FH) + (0.42H) + CY

```
MEM000   MEM    0.00H
MEM001   MEM    0.01H
MEM002   MEM    0.02H
MEM00D   MEM    0.0DH
MEM00E   MEM    0.0EH
MEM00F   MEM    0.0FH
         MOV    BANK, #00H          ; Data memory bank 0
         MOV    RPH, #00H           ; General register bank 0
         MOV    RPL, #00H           ; General register row address 0
         MOV    IXH, #00H           ; IX 00001000000 (0.40H)
         MOV    IXM, #04H
         MOV    IXL, #00H
         SET1   IXE                 ; IXE flag ← 1
         ADD    MEM00D, MEM000      ; (0.0DH) ← (0.0DH) + (0.40H)  ; Low-order nibble
         ADDC   MEM00E, MEM001      ; (0.0EH) ← (0.0EH) + (0.41H)
         ADDC   MEM00F, MEM002      ; (0.0FH) ← (0.0FH) + (0.42H)  ; High-order nibble
```

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

<1> OP code

10 – 8	7 – 4	3 – 0
10010	m_R	m_C	n4

<2> Function

When CMP = 0, (m) ← (m) + n4 + CY

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

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

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

Sets carry flag CY, if a carry occurs as a result of the addition. Resets the carry flag, if no carry occurs.
If the addition result is other than zero, zero flag Z is reset, regardless of compare flag CMP.
If the result is zero with the compare flag reset (CMP = 0), the zero flag Z is set.
If the result is zero with the compare flag set (CMP = 1), the zero flag Z is not changed.
Addition can be executed in binary or BCD. The BCD flag for PSWORD specifies which kind of addition is to be executed.

<3> Example 1

To add 5 to the 12-bit contents for 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 for addresses 0.4DH through 0.4FH and store the result in addresses 0.4DH through 0.4FH:

```
            (0.4FH) ← (0.4FH) + 05H
            (0.4EH) ← (0.4EH) + CY
            (0.4DH) ← (0.4DH) + CY
MEM00D      MEM     0.0DH
MEM00E      MEM     0.0EH
MEM00F      MEM     0.0FH
            MOV     BANK, #00H          ; Data memory bank 0
            MOV     IXH, #00H           ; IX ← 00001000000B (0.40H)
            MOV     IXM, #04H
            MOV     IXL, #00H
            SET1    IXE                 ; IXE flag ← 1
            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 address register AR contents.

<3> Example 1

To add 1 to the 16-bit contents for 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 program can be rewritten as follows, with addition instructions:

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

Example 2

To transfer table data, 16 bits (1 address) at a time, to DBF (data buffer), using the table reference instruction (for details, refer to **10.2.3 Table Reference**):

```
; Address        Table data
★       ORG      10H
        DW       0F3FFH
        DW       0A123H
        DW       0FFF1H
        DW       0FFF5H
        DW       0FF11H
         ⋮
        MOV      AR3, #0H          ; Table data address
        MOV      AR2, #0H          ; 0010H in address register
        MOV      AR1, #1H          ;
        MOV      AR0, #0H
LOOP:
★       MOVT     DBF, @AR          ; Reads table data to DBF
        :
        :
        :                          ; Table data reference processing
        :
        INC      AR                ; Increments address register by 1
        BR       LOOP
```

<4> Caution

The numbers of bits, for address registers AR3 through AR0, differ, depending on the microcontroller model to be used.

- μPD17134A/17135A : 10 bits
- μPD17136A/17137A/17P136A/17P137A : 11 bits

(6) INC IX — Increment index register

<1> OP code

10	8 7	4 3	0
00111	000	1000	0000

<2> Function

IX ← IX + 1

Increments the index register IX contents.

<3> Example 1

To add 1 to the 12-bit contents for 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
```

This program can be rewritten as follows, with addition instructions:

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

Example 2

To clear all the contents for data memory addresses 0.00H through 0.73H, using the index register:

★

```
        MEM000  MEM0.00H
        MOV     IXH, #00H       ; Sets index register contents in 00H in bank 0
        MOV     IXM, #00H       ;
        MOV     IXL, #00H
RAM clear:
        SET1    IXE             ; IXE flag ← 1
        MOV     MEM000, #00H    ; Writes 0 to data memory indicated by index register
        CLR1    IXE             ; IXE flag ← 0
        INC     IX
        SET2    CMP, Z          ; CMP flag ← 1, Z flag ← 1
        SUB     IXL, #03H       ; Checks whether index register  contents
        SUBC    IXM, #07H       ; are 73H in bank 0
        SUBC    IXH, #00H       ;
        SKT1    Z               ; Loops until contents of index register becomes
        BR      RAM clear       ; 73H of bank 0
```

19.5.2 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 data memory contents from the general register contents, and stores the result in general register.

When CMP = 1, (r) – (m)

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

Sets carry flag CY, if a borrow occurs as a result of the subtraction. Resets the carry flag, if no borrow occurs.
If the subtraction result is other than zero, zero flag Z is reset, regardless of compare flag CMP.
If the result is zero with the compare flag reset (CMP = 0), the zero flag Z is set.
If the result is zero with the compare flag set (CMP = 1), the zero flag Z is not changed.
Subtraction can be executed in binary 4 bits or BCD. The BCD flag for PSWORD specifies which kind of subtraction is to be executed.

<3> Example 1

To subtract the address 0.2FH contents from the address 0.03H contents, store the result in address 0.03H, when row address 0 (0.00H–0.0FH) in bank 0 is specified as a general register (RPH = 0, RPL = 0):

```
          (0.03H) ← (0.03H) + (0.2FH)
MEM003    MEM     0.03H
MEM02F    MEM     0.2FH
          SUB     MEM003, MEM02F
```

Example 2

To subtract the address 0.2FH contents from the address 0.23H contents, when row address 2 (0.20H–0.2FH) in bank 0 is specified as the general register (RPH = 0, RPL = 4), and store the result in address 0.23H:

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

Example 3

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

```
            (0.03H) ← (0.03H) + (0.6FH)
MEM003      MEM     0.03H
MEM02F      MEM     0.2FH
            MOV     BANK, #00H          ; Data memory bank 0
            MOV     RPH, #00H           ; General register bank 0
            MOV     RPL, #00H           ; General register row address 0
            MOV     IXH, #00H           ; IX ← 00001000000B (0.40H)
            MOV     IXM, #04H           ;
            MOV     IXL, #00H           ;
            SET1    IXE                 ; IXE flag ← 1
            SUB     MEM003, MEM02F      ; IX 00001000000B (0.40H)
                                        ; Bank operand OR ) 00000101111B (0.2FH)
                                        ; Specified address   00001101111B (0.6FH)
```

Example 4

To subtract the address 0.3FH contents from the address 0.03H contents and store result in address 0.03H. At this time, data memory address 0.3FH can be specified by selecting data memory address 2FH, if IXE = 1, IXH = 0, IXM = 1, and IXL = 0, i.e., IX = 0.10H.

```
        (0.03H) ← (0.03H) + (0.3FH)
MEM003      MEM     0.03H
MEM02F      MEM     0.2FH
            MOV     BANK #00H           ; Data memory bank 0
            MOV     RPH, #00H           ; General register bank 0
            MOV     RPL, #00H           ; General register row address 0
            MOV     IXH, #00H           ; IX ← 00000010000B (0.10H)
            MOV     IXM, #01H           ;
            MOV     IXL, #00H           ;
            SET1    IXE                 ; IXE flag ← 1
            SUB     MEM003, MEM02F      ; IX 00000010000B (0.10H)
                                        ; Bank operand OR ) 00000101111B (0.2FH)
                                        ; Specified address   00000111111B (0.3FH)
```

<4> Caution

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

```
MEM013   MEM    0.13H
MEM02F   MEM    0.2FH
         MOV    RPH, #00H          ; General register bank 0
         MOV    RPL, #00H          ; General register row address 0
         SUB    MEM013, MEM02F
                Specify general register in 00H–0FH range
                (set register pointer row address other than 1).
```

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

(2) SUB m, #n4 **Subtract immediate data from data memory**

<1> OP code

10 – 8	7 – 4	3 – 0
10001	m_R	m_C	n4

<2> Function

When CMP = 0, (m) ← (m) – n4

Subtracts immediate data from the data memory contents, and stores the result in data memory.

When CMP = 1, (m) – n4

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

Sets carry flag CY, if a borrow occurs as a result of the subtraction. Resets the carry flag, if no borrow occurs.
If the subtraction result is other than zero, zero flag Z is reset, regardless of compare flag CMP.
If the result is zero with the compare flag reset (CMP = 0), the zero flag Z is set.
If the result is zero with the compare flag set (CMP = 1), the zero flag Z is not changed.
Subtraction can be executed in binary 4 bits or BCD. The BCD flag for PSWORD specifies which kind of subtraction is to be executed.

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

(0.6FH) ← (0.6FH) – 5

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

```
MEM02F   MEM    0.2FH
         MOV    BANK, #00H      ; Data memory bank 0
         MOV    IXH, #00H       ; IX ← 00001000000B (0.40H)
         MOV    IXM, #04H       ;
         MOV    IXL, #00H       ;
         SET1   IXE             ; IXE flag ← 1
         SUB    MEM02F, #05H    ; IX                  00001000000B (0.40H)
                                ; Bank operand OR )   00000101111B (0.2FH)
                                ; Specified address   00001101111B (0.6FH)
```

Example 3

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

(0.2FH) ← (0.2FH) – 5

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

```
MEM02F   MEM    0.2FH
         MOV    BANK0, #00H     ; Data memory bank 0
         MOV    IXH, #00H       ; IX ← 00000000000B (0.00H)
         MOV    IXM, #00H       ;
         MOV    IXL, #00H       ;
         SET1   IXE             ; IXE flag ← 1
         SUB    MEM02F, #05H    ; IX                  00000000000B (0.00H)
                                ; Bank operand OR )   00000101111B (0.2FH)
                                ; Specified address   00000101111B (0.2FH)
```

(3) SUBC r, m **Subtract data memory from general register with carry flag**

<1> OP code

10	8 7	4 3	0
00011	m_R	m_c	r

<2> Function

When CMP = 0, (r) ← (r) – (m) – CY

Subtracts the data memory contents from the general register contents with carry flag CY. Stores the result in general register. By using this SUBC instruction, 2 or more words can be easily subtracted.

When CMP = 1, (r) – (m) – CY

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

Sets carry flag CY, if a borrow occurs as a result of the subtraction. Resets the carry flag, if no borrow occurs.
If the subtraction result is other than zero, zero flag Z is reset, regardless of compare flag CMP.
If the result is zero with the compare flag reset (CMP = 0), the zero flag Z is set.
If the result is zero with the compare flag set (CMP = 1), the zero flag Z is not changed.
Subtraction can be executed in binary 4 bits or BCD. The BCD flag for PSWORD specifies which kind of subtraction is to be executed.

<3> Example 1

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

(0.0FH) ← (0.0FH) – (0.2FH)
(0.0EH) ← (0.0EH) – (0.2EH) – CY
(0.0DH) ← (<u>0.0DH</u>) + (0.2DH) – CY

```
MEM00D  MEM     0.0DH
MEM00E  MEM     0.0EH
MEM00F  MEM     0.0FH
MEM02D  MEM     0.2DH
MEM02E  MEM     0.2EH
MEM02F  MEM     0.2FH
        SUB     MEM00F,  MEM02F   ; Low-order nibble
        SUBC    MEM00E,  MEM02E
        SUBC    MEM00D,  MEM02D   ; High-order nibble
```

Example 2

To subtract the 12-bit contents for addresses 0.40H through 0.42H from the 12-bit contents for addresses 0.0DH through 0.0FH, and store the result in 12 bits for addresses 0.0DH through 0.0FH:

(0.0DH) ← (0.0DH) – (0.40H)
(0.0EH) ← (0.0EH) – (0.41H) – CY
(0.0FH) ← (0.0FH) + (0.42H) – CY

```
MEM000   MEM    0.00H
MEM001   MEM    0.01H
MEM002   MEM    0.02H
MEM00D   MEM    0.0DH
MEM00E   MEM    0.0EH
MEM00F   MEM    0.0FH
         MOV    BANK, #00H          ; Data memory bank 0
         MOV    RPH, #00H           ; General register bank 0
         MOV    RPL, #00H           ; General register row address 0
         MOV    IXH, #00H           ; IX ← 00001000000B (0.40H)
         MOV    IXM, #04H           ;
         MOV    IXL, #00H           ;
         SET1   IXE                 ; IXE flag ← 1
         SUB    MEMOOD, MEM000      ; (0.0DH) ← (0.0DH) – (0.40H)
         SUBC   MEM00E, MEM001      ; (0.0EH) ← (0.0EH) – (0.41H)
         SUBC   MEM00F, MEM002      ; (0.0FH) ← (0.0FH) – (0.42H)
```

(4) SUBC m, #n4 **Subtract immediate data from data memory with carry flag**

<1> OP code

10 – 8	7 – 4	3 – 0
10011	m_R	m_C	n4

<2> Function

When CMP = 0, (m) ← (m) – n4 – CY

Subtracts immediate data from the data memory contents with carry flag CY, and stores the result in data memory.

When CMP = 1, (m) – n4 – CY

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

Sets carry flag CY, if a borrow occurs as a result of the subtraction. Resets the carry flag, if no borrow occurs.
If the subtraction result is other than zero, zero flag Z is reset, regardless of compare flag CMP.
If the result is zero with the compare flag reset (CMP = 0), the zero flag Z is set.
If the result is zero with the compare flag set (CMP = 1), the zero flag Z is not changed.
Subtraction can be executed in binary or BCD. The BCD flag for PSWORD specifies which kind of subtraction is to be executed.

<3> Example 1

To subtract 5 from the 12-bit contents for addresses 0.0DH through 0.0FH and store the result in 12 bits for addresses 0.0DH through 0.0FH:

```
            (0.0FH) ← (0.0FH) – 05H
            (0.0EH) ← (0.0EH) – CY
            (0.0DH) ← (0.0DH) – CY
MEM00D      MEM       0.0DH
MEM00E      MEM       0.0EH
MEM00F      MEM       0.0FH
            SUB       MEM00F, #05H
            SUBC      MEM00E, #00H
            SUBC      MEM00D, #00H
```

Example 2

To subtract 5 from the 12-bit contents for addresses 0.4DH through 0.4FH and store the result in addresses 0.4DH through 0.4FH:

```
            (0.4FH) ← (0.4FH) – 05H
            (0.4EH) ← (0.4EH) – CY
            (0.4DH) ← (0.4DH) – CY
MEM00D      MEM       0.0DH
MEM00E      MEM       0.0EH
MEM00F      MEM       0.0FH
            MOV       BANK, #00H     ; Data memory bank 0
            MOV       IXH, #00H      ; IX ← 00001000000B (0.40H)
            MOV       IXM, #04H      ;
            MOV       IXL, #00H      ;
            SET1      IXE            ; IXE flag  ← 1
            SUB       MEM00F, #5     ; (0.4FH) ← (0.4FH) – 5
            SUBC      MEM00E, #0     ; (0.4EH) ← (0.4EH) – CY
            SUBC      MEM00D, #0     ; (0.4DH) ← (0.4DH) – CY
```

19.5.3 Logical Operation Instructions

(1) OR r, m — **OR between general register and data memory**

<1> OP code

10	8 7	4 3	0
00110	m_R	m_c	r

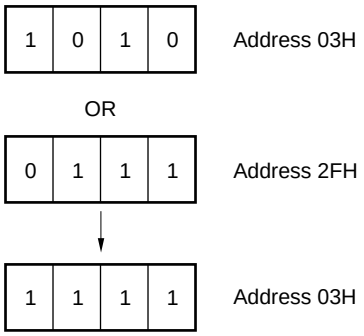
<2> Function

(r) ← (r) v (m)

ORs the general register contents with data memory. Stores the result in general register.

<3> Example 1

To OR the address 0.03H contents (1010B) and the address 0.2FH contents (0111B) and store the result (1111B) in address 0.03H:
(0.03H) ← (0.03H) v (0.2FH)




```
MEM003  MEM     0.03H
MEM02F  MEM     0.2FH
        MOV     MEM003, #1010B
        MOV     MEM02F, #0111B
        OR      MEM003, MEM02F
```

(2) OR m, #n4 — **OR between data memory and immediate data**

<1> OP code

10	8 7	4 3	0
10110	m_R	m_c	n4

<2> Function

(m) ← (m) v n4

ORs the data memory contents and immediate data. Stores the result in data memory.

<3> Example 1

To set bit 3 (MSB) for address 0.03H:
(0.03H) ← (0.03H) v 1000B

Address 0.03H

1	×	×	×

×: don't care

```
MEM003   MEM    0.03H
         OR     MEM003, #1000B
```

Example 2

To set all the bits for address 0.03H:

```
MEM003   MEM    0.03H
         OR     MEM003, #1111B
    or,
MEM003   MEM    0.03H
         MOV    MEM003, #0FH
```

(3) AND r, m **AND between general register and data memory**

<1> OP code

10	8 7	4 3	0
00100	m_R	m_c	r

<2> Function

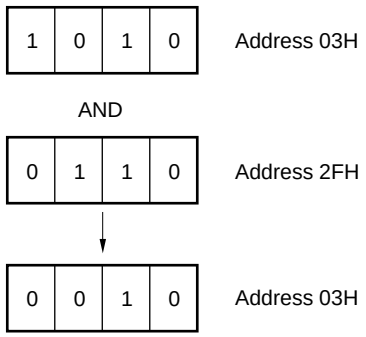
(r) ← (r) ∧ (m)

ANDs the general register contents with data memory and stores the result in general register.

<3> Example

To AND the address 0.03H (1010B) contents and the address 0.2FH (0110B) contents. To store the result (0010B) in address 0.03H:

(0.03H) ← (0.03H) ∧(0.2FH)




```
MEM003    MEM    0.03H
MEM02F    MEM    0.2FH
          MOV    MEM003, #1010B
          MOV    MEM02F, #0110B
          AND    MEM003, MEM02F
```

(4) AND m, #n4 **AND between data memory and immediate data**

<1> OP code

10	8 7	4 3	0
11110	m_R	m_c	n

<2> Function

(m) ← (m)∧ n4

ANDs the data memory contents and immediate data. Stores the result in data memory.

<3> Example 1

To reset bit 3 (MSB) for address 0.03H:

(0.03H) ← (0.03H) ∧ 0111B

Address 0.03H

0	×	×	×

×: don't care

```
MEM003    MEM    0.03H
          AND    MEM003, #0111B
```

Example 2

To reset all the bits for address 0.03H:

```
MEM003    MEM    0.03H
          AND    MEM003, #0000B
     or,
MEM003    MEM    0.03H
          MOV    MEM003, #00H
```

(5) XOR r, m **Exclusive OR between general register and data memory**

<1> OP code

10	8 7	4 3	0
00101	m_R	m_c	r

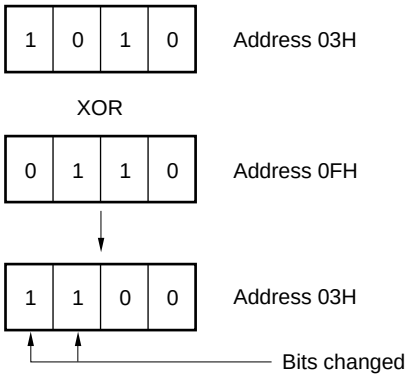
<2> Function

(r) ← (r) ∀ (m)

Exclusive-ORs the general register contents with data memory. Stores the result in general register.

<3> Example 1

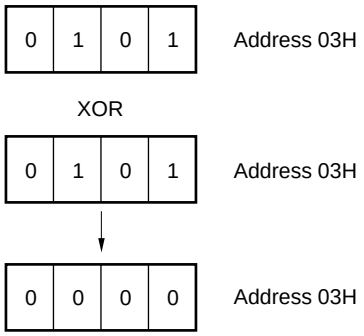
To compare the address 0.03H contents and the address 0.0FH contents. If different bits are found, set and store them in address 0.03H. If all the bits in address 0.03H are reset (i.e., the address 0.03H contents are the same as those for address 0.0FH), jumps to LBL1; otherwise, jumps to LBL2.
This example is for processing to compare the status of an alternate switch (address 0.03H contents) with the internal status (address 0.0FH contents) and to branch to another processing, if the switch status changes.




```
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

10	8 7	4 3	0
10101	m_R	m_c	n4

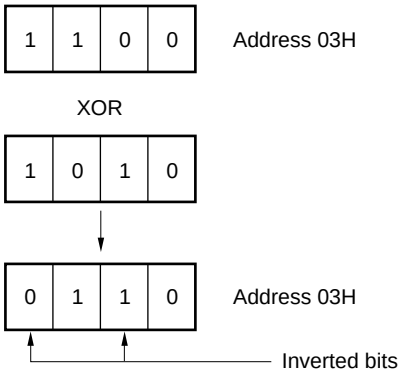
<2> Function

(m) ← (m) ∀ n4

Exclusive-ORs the data memory contents and immediate data. Stores the result in data memory.

<3> Example

To invert bits 1 and 3 in address 0.03H and store the result in address 03H:




```
MEM003    MEM    0.03H
          XOR    MEM003, #1010B
```

19.5.4 Judgment 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 data memory contents and immediate data is equal to n. (Executes as NOP instruction)

<3> Example 1

To jump to AAA, if bit 0 in address 03H is 1; if it is 0, jumps to BBB:

```
SKT   03H, #0001B
BR    BBB
BR    AAA
```

Example 2

To skip the next instruction, if both bits 0 and 1 in address 03H are 1.

```
SKT   03H, #0011B
```

	b_3	b_2	b_1	b_0	
Skip condition 03H	×	×	1	1	×: don't care

Example 3

The results of executing the following two instructions are the same:

```
SKT   13H, #1111B
SKE   13H, #0FH
```

(2) SKF m, #n **Skip next instruction if data memory bits are false**

<1> OP code

10	8 7	4 3	0
11111	m_R	m_C	n

★ **<2> Function**

CMP ← 0, if (m) ∧ n = 0, then skip

Skips the next one instruction, if the result of ANDing the data memory contents and immediate data is 0 (Executes as NOP instruction).

<3> Example 1

To store immediate data 00H to address 0FH in the data memory, if bit 2 in address 13H is 0; if it is 1, jumps to ABC:

```
MEM013    MEM    0.13H
MEM00F    MEM    0.0FH
          SKF    MEM013, #0100B
          BR     ABC
          MOV    MEM00F, #00H
```

Example 2

To skip the next instruction, if both bits 3 and 0 in address 29H are 0.

```
SKF       29H, #1001B
```

	b_3	b_2	b_1	b_0	
Skip condition 29H	0	×	×	0	×: don't care

Example 3

The results of executing the following two instructions are the same:

```
SKF       34H, #1111B
SKE       34H, #00H
```

19.5.5 Comparison Instructions

(1) SKE m, #n4 **Skip if data memory equal to immediate data**

<1> OP code

10	8	7	4	3	0
01001	m_R	m_c		n4	

<2> Function

(m) –n4, skip if zero

Skips the next one instruction, if the data memory contents are equal to the immediate data value (Executes as NOP instruction).

<3> Example

To transfer 0FH to address 24H, if the address 24H contents are 0; if not, jumps to OPE1:

```
MEM024    MEM    0.24H
          SKE    MEM024, #00H
          BR     OPE1
          MOV    MEM024, #0FH
OPE1      :
```

(2) SKNE m, #n4 **Skip if data memory not equal to immediate data**

<1> OP code

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 data memory contents are not equal to the immediate data value (Executes as NOP instruction).

<3> Example

To jump to XYZ, if the asddress 1FH contents are 1 and 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
1EH	0 0 1 1	1FH	0 0 0 1

```
MEM01E    MEM     0.1EH
MEM01F    MEM     0.1FH
          SKNE    MEM01F, #01H
          SKE     MEM01E, #03H
          BR      ABC
          BR      XYZ
```

The above program can be rewritten as follows, using compare and zero flags:

```
MEM01E    MEM     0.1EH
MEM01F    MEM     0.1FH
          SET2    CMP, Z          ; CMP flag ← 1, Z flag ← 1
          SUB     MEM01F, #01H
          SUBC    MEM01E, #03H
          SKT1    Z
          BR      ABC
          BR      XYZ
```

(3) SKGE m, #n4 **Skip if data memory greater than or equal to immediate data**

<1> OP code

10 – 8	7 – 4	3 – 0
11001	m_R	m_c	n4

★ **<2> Function**

(m) –n4, skip if not borrow

Skips the next one instruction, if the data memory contents are greater than or equal to the immediate data value (Executes as NOP instruction).

<3> Example

To execute RET, if 8-bit data stored in addresses 1FH (high-order) and 2FH (low-order) is greater than immediate data ‘17H’; if not, 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	m_R	m_c	n4

★ **<2> Function**

(m) –n4, skip if borrow

Skips the next one instruction, if the data memory contents are less than the immediate data value (Executes as NOP instruction).

<3> Example

To store 01H in address 0FH, if the address 10H contents are greater than immediate data ‘6’; if not, store 02H in address 0FH:

```
MEM00F    MEM     0.0FH
MEM010    MEM     0.10H
          MOV     MEM00F, #02H
          SKLT    MEM010, #06H
          MOV     MEM00F, #01H
```

19.5.6 Rotation Instructions

(1) RORC r **Rotate right general register with carry flag**

<1> OP code

			3 0
00111	000	0111	r

<2> Function

$$\rightarrow CY \rightarrow (r)_{b3} \rightarrow (r)_{b2} \rightarrow (r)_{b1} \rightarrow (r)_{b0} \rightarrow$$

Rotates the contents of general register indicated by r to right by 1 bit including carry flag.

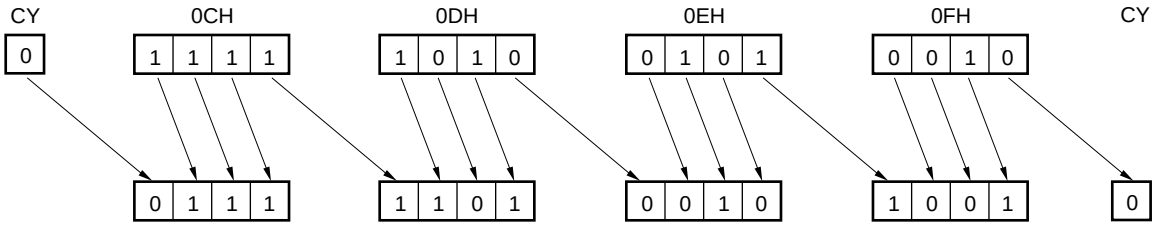
<3> Example 1

When row address 0 of bank 0 (0.00H – 0.0FH) is specified as general register (RPH = 0, RPL = 0), rotate the value of address 0.00H (1000B) to right by 1 bit to make it 0100B.

```
                (0.00H) ← (0.00H) ÷ 2
MEM000          MEM     0.00H
                MOV     RPH, #00H     ; General register bank 0
                MOV     RPL, #00H     ; General register row address 0
                CLR1    CY            ; CY flag ← 0
                RORC    MEM000
```

Example 2

When row address 0 of bank 0 (0.00H – 0.0FH) is specified as general register (RPH = 0, RPL = 0), rotate the data buffer DBF contents 0FA52H to right by 1 bit to make DBF contents 7D29H.




```
MEM00C          MEM     0.0CH
MEM00D          MEM     0.0DH
MEM00E          MEM     0.0EH
MEM00F          MEM     0.0FH
                MOV     RPH, #00H     ; General register bank 0
                MOV     RPL, #00H     ; General register row address 0
                CLR1    CY            ; CY flag ← 0
                RORC    MEM00C
                RORC    MEM00D
                RORC    MEM00E
                RORC    MEM00F
```

19.5.7 Transfer Instructions

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

<1> OP code

10	8 7	4 3	0
01000	m_R	m_C	r

<2> Function

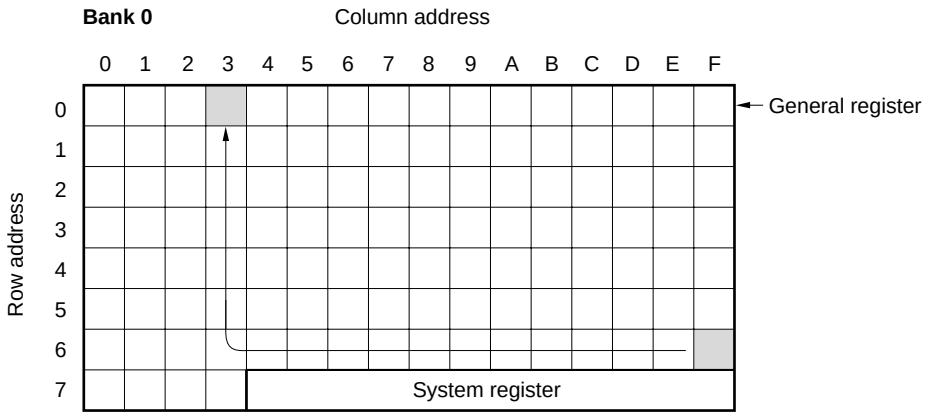
(r) ← (m)

Stores the data memory contents to general register.

<3> Example 1

To store the address 0.2FH contents to address 0.03H:

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



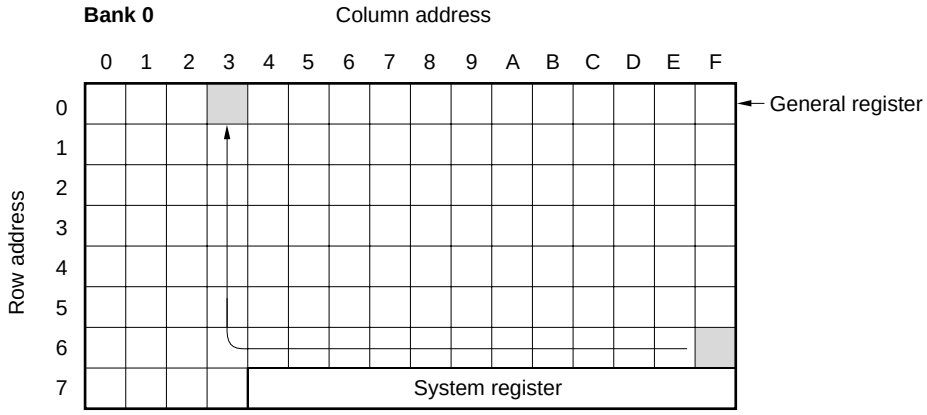
Example 2

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

IXH ← 00H
IXM ← 04H
IXL ← 00H
IXE flag ← 1
(0.03H) ← (0.6FH)

Address obtained as result of ORing index register contents, 040H, and data memory contents, 0.2FH

```
MEM003    MEM     0.03H
MEM02F    MEM     0.2FH
          MOV     IXH, #00H          ; IX ← 00001000000B (0.40H)
          MOV     IXM, #04H
          MOV     IXL, #00H
          SET1    IXE                ; IXE flag ← 1
          LD      MEM003, MEM02F
```



(2) ST m, r **Store general register to data memory**

<1> OP code

10	8 7	4 3	0
11000	m_R	m_c	r

<2> Function

(m) ← (r)

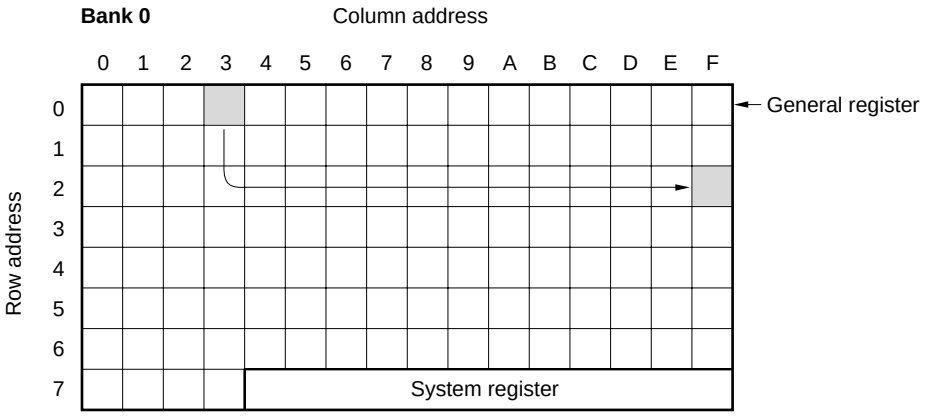
Stores the general register contents to data memory.

<3> Example 1

To store the address 0.03H contents to 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
```

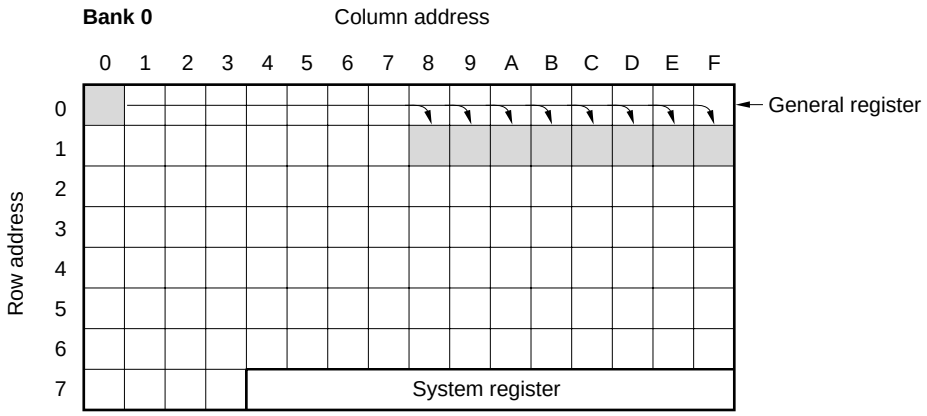


Example 2

To store the address 0.00H contents to addresses 0.18H through 0.1FH. The data memory addresses (18H – 1FH) are specified by the index register.

(0.18H) ← (0.00H)
(0.19H) ← (0.00H)
⋮
(0.1FH) ← (0.00H)

```
        MOV     IXH, #00H          ; IX ← 00000000000B (0.00H)
        MOV     IXM, #00H
        MOV     IXL, #00H          ; Specifies data memory address 0.00H
MEM018  MEM     0.18H
MEM000  MEM     0.00H
    LOOP1:
        SET1    IXE                ; IXE flag ← 1
        ST      MEM018, MEM000     ; (0.1XH) ← (0.00H)
        CLR1    IXE                ; IXE flag ← 0
        INC     IX                 ; IX ← IX + 1
        SKGE    IXL, #08H
        BR      LOOP1
```



(3) MOV @r, m **Move data memory to destination indirect**

<1> OP code

10	8	7	4	3	0
01010	m_R	m_c		r	

<2> Function

When MPE = 1
((MP), (r)) ← (m)

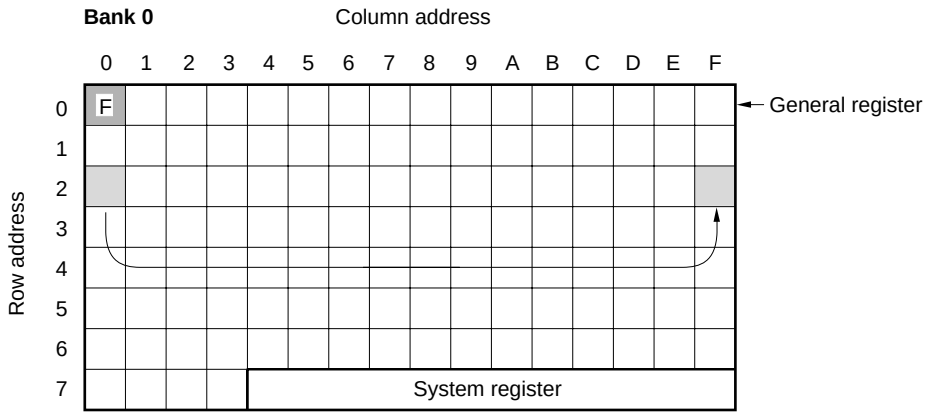
When MPE = 0
(BANK, m_R, (r)) ← (m)

Stores the data memory contents to the data memory addressed by the general register contents. When MPE = 0, transfer is performed in the same row address in the same bank.

<3> Example 1

To store the address 0.20H contents to address 0.2FH with the MPE flag cleared to 0. The transfer destination data memory address is at the same row address as the transfer source, and the column address is specified by the general register contents at address 0.00H.

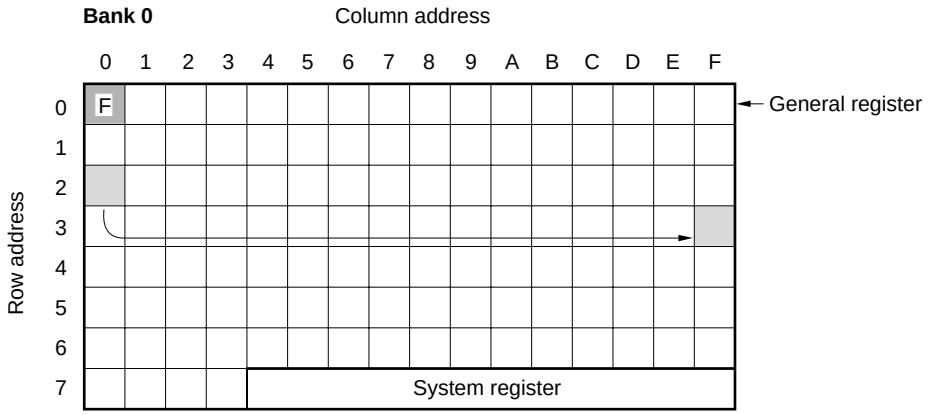
```
                (0.2FH) ← (0.20H)
MEM000          MEM     0.00H
MEM020          MEM     0.20H
                CLR1    MPE                 ; MPE flag ← 0
                MOV     MEM000, #0FH        ; Sets column address in general register
                MOV     @MEM000, MEM020     ; Store
```



Example 2

To store the address 0.20H contents to address 0.3FH, with the MPE flag set to 1. The row address for the transfer destination data memory address is specified by the memory pointer MP contents. The column address is specified by the general register contents at address 0.00H.

```
                (0.3FH) ← (0.20H)
MEM000          MEM     0.00H
MEM020          MEM     0.20H
                MOV     RPH, #00H           ; General register bank 0
                MOV     RPL, #00H           ; General register row address 0
                MOV     MEM000, #0FH        ; Sets column address in general register
                MOV     MPH, #00H           ; Sets row address in memory pointer
                MOV     MPL, #03H           ;
                SET1    MPE                 ; MPE flag ← 1
                MOV     @MEM000, MEM020     ; Store
```



(4) MOV m, @r **Move data memory to destination indirect**

<1> OP code

10 – 8	7 – 4	3 – 0
11010	m_R	m_C	r

<2> Function

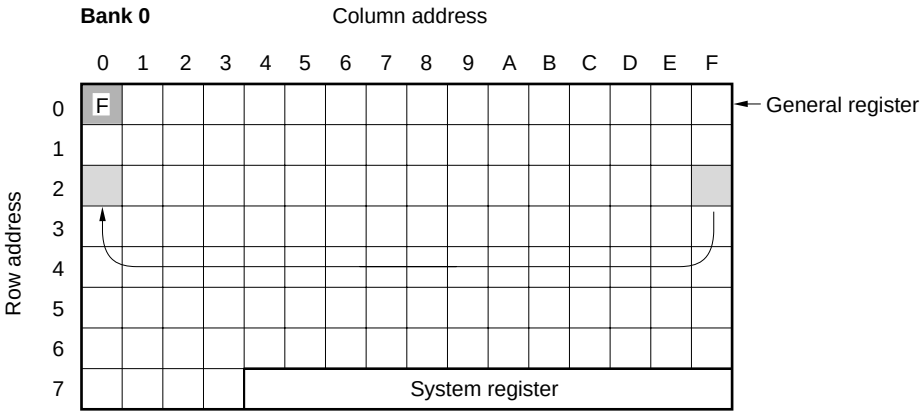
When MPE = 1
(m) ← (MP, (r))
When MPE = 0
(m) ← (BANK, m_R, (r))

Stores the data memory contents addressed by the general register contents to data memory. When MPE = 0, transfer is performed in the same row address in the same bank.

<3> Example 1

To store the address 0.2FH contents to address 0.20H, with the MPE flag cleared to 0. The transfer destination data memory address is at the same row address as the transfer source. The column address is specified by the general register contents at address 0.00H.

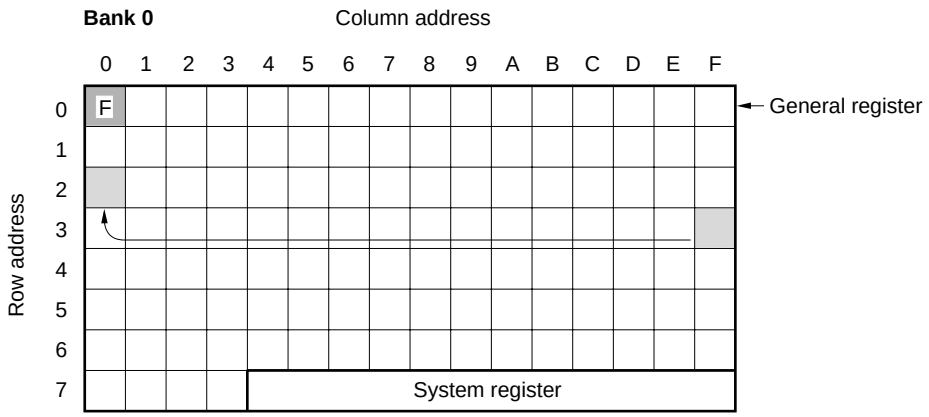
```
                (0.20H) ← (0.2FH)
MEM000          MEM     0.00H
MEM020          MEM     0.20H
                CLR1    MPE                 ; MPE flag ← 0
                MOV     MEM000, #0FH        ; Sets column address in general register
                MOV     MEM020, @MEM000     ; Store
```



Example 2

To store the address 0.3FH contents to address 0.20H, with the MPE flag set to 1. The row address for the transfer source data memory address is specified by the memory pointer MP contents. The column address is specified by the general register contents at address 0.00H.

```
                (0.20H) ← (0.3FH)
MEM000          MEM     0.00H
MEM020          MEM     0.20H
                MOV     MEM000, #0FH        ; Sets column address in general register
                MOV     MPH, #00H           ; Sets row address in memory pointer
                MOV     MPL, #03H           ;
                SET1    MPE                 ; MPE flag ← 1
                MOV     MEM020, @MEM000     ; Store
```



(5) MOV m, #n4 **Move immediate data to data memory**

<1> OP code

10 – 8	7 – 4	3 – 0
11101	m_R	m_c	n4

<2> Function

(m) ← n4

Stores immediate data to data memory.

<3> Example 1

To store immediate data 0AH to data memory address 0.50H:

```
                (0.5H) ← 0AH
MEM050          MEM     0.50H
                MOV     MEM050, #0AH
```

Example 2

To store immediate data 07H to address 0.32H, when data memory address 0.00H is specified with IXH = 0, IXM = 3, IXL = 2, and IXE flag = 1:

```
                (0.32H) ← 07H
MEM000          MEM     0.00H
                MOV     IXH, #00H           ; IX ← 00000110010B (0.32H)
                MOV     IXM, #03H
                MOV     IXL, #02H
                SET1    IXE                 ; IXE flag ← 1
                MOV     MEM000, #07H
```

(6) MOVT DBF, @AR — Move program memory data specified by AR to DBF

<1> OP code

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, to data buffer DBF.
Since this instruction temporarily uses one stack level, pay attention to nesting for subroutines and interrupts.

<3> Example

To transfer 16 bits of table data, specified by the values for address registers AR3, AR2, AR1, and AR0 in the system register, to data buffers DBF3, DBF2, DBF1, and DBF0:

```
; *
; ** Table data
; *
ORG     0010H
DW      0000000000000000B      ; (0000H)
DW      1010101111001101B      ; (0ABCDH)
                 :
; *
; ** Table reference program
; *
MOV     AR3, #00H              ; AR3 ← 00H  Sets 0011H in address register
MOV     AR2, #00H              ; AR2 ← 00H
MOV     AR1, #01H              ; AR1 ← 01H
MOV     AR0, #01H              ; AR0 ← 01H
MOVT    DBF, @AR               ; Transfers address 0011H data to DBF
```

In this case, the data are stored in DBF, as follows:

DBF3 = 0AH
DBF2 = 0BH
DBF1 = 0CH
DBF0 = 0DH

(7) PUSH AR — Push address register

<1> OP code

00111	000	1101	0000

<2> Function

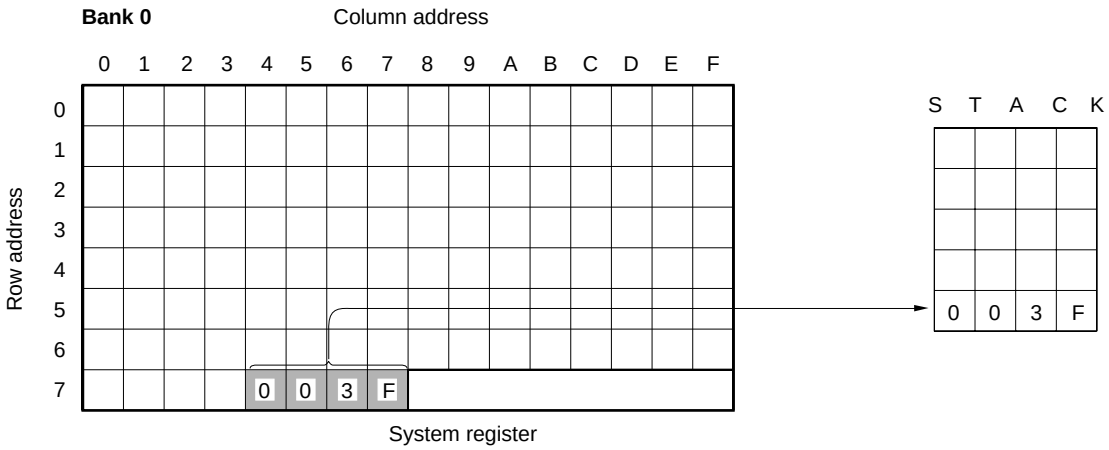
SP ← SP – 1,
ASR ← AR

Decrements stack pointer SP and stores the address register AR value to address stack register specified by stack pointer.

<3> Example 1

To set 003FH in 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 (next address of the data table) for a subroutine in the address register. Returns execution, if a data table exists after a subroutine:

```
        ⋮
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
```

If POP instruction is executed at this time, the contents of address register is “0011H” (the next address of CALL instruction).

(8) POP AR — Pop address register

<1> OP code

00111	000	1100	0000

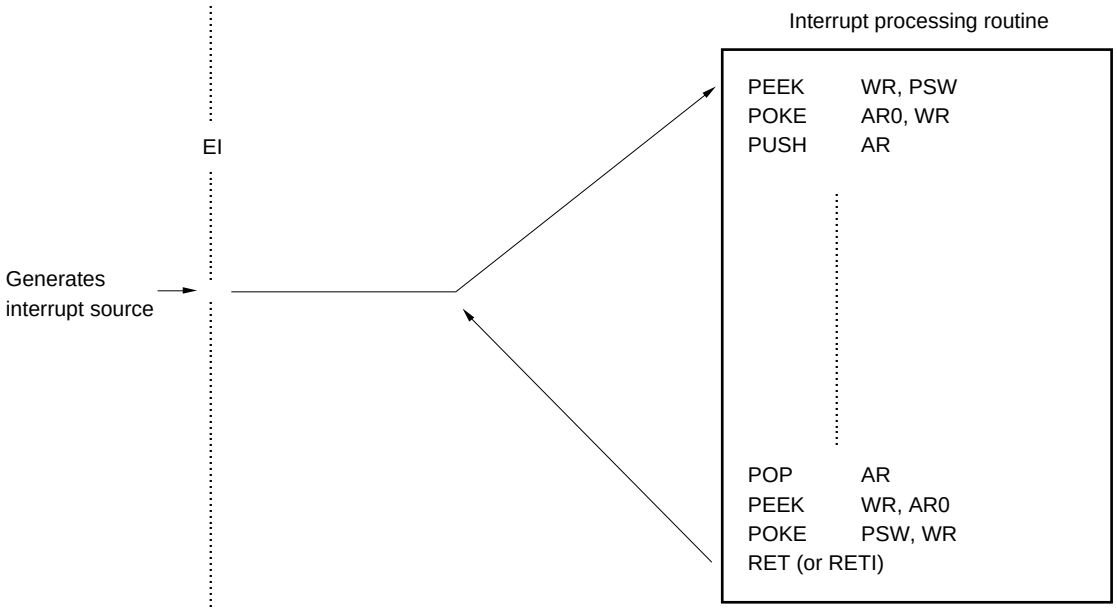
<2> Function

AR ← ASR,
SP ← SP + 1

Pops the contents of address stack register indicated by stack pointer to address register AR and then increments stack pointer SP.

<3> Example

If the PSW contents are changed, while an interrupt processing routine is being executed, the PSW contents are transferred to the address register through WR at the beginning of the interrupt processing and saved to address stack register by the PUSH instruction. Before the execution returns from the interrupt routine, the address register contents are restored through WR to PSW by the POP instruction.



(9) PEEK WR, rf — Peek register file to window register

<1> OP code

10 – 8	7 – 4	3 – 0
00111	rf_R	0011	rf_C

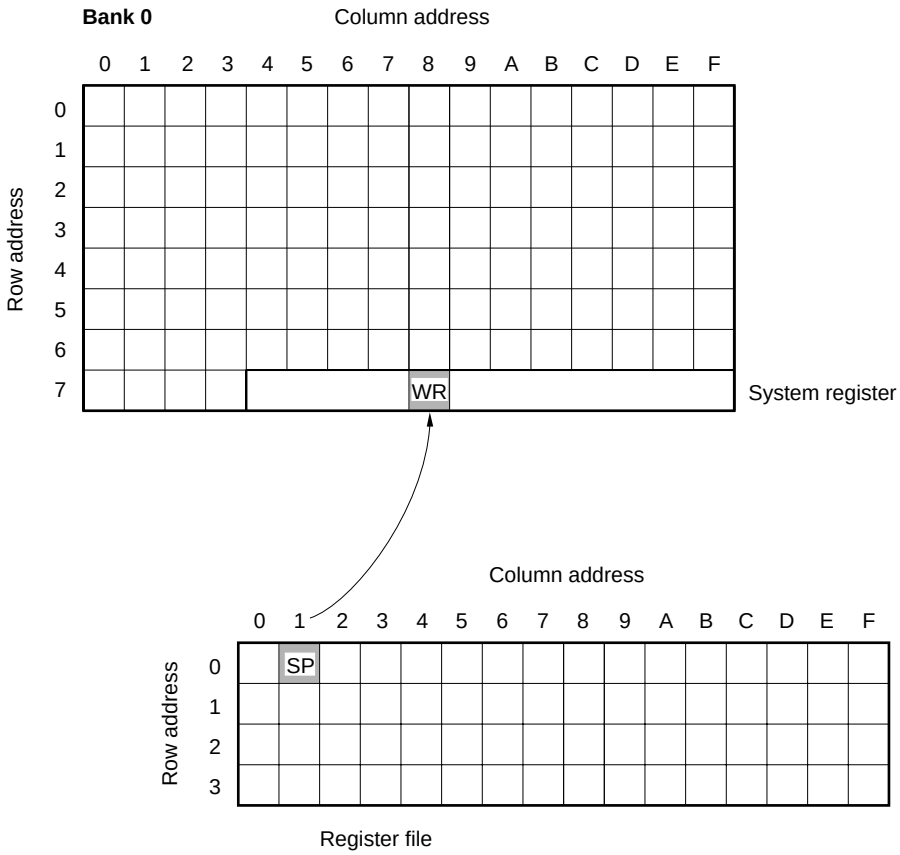
<2> Function

WR ← (rf)

Stores the register file contents to window register WR.

<3> Example 1

To store the stack pointer SP contents at address 01H in the register file to the window register:
PEEK WR, SP



(10) POKE rf, WR — Poke window register to register file

<1> OP code

10 – 8	7 – 4	3 – 0
00111	rf_R	0010	rf_C

<2> Function

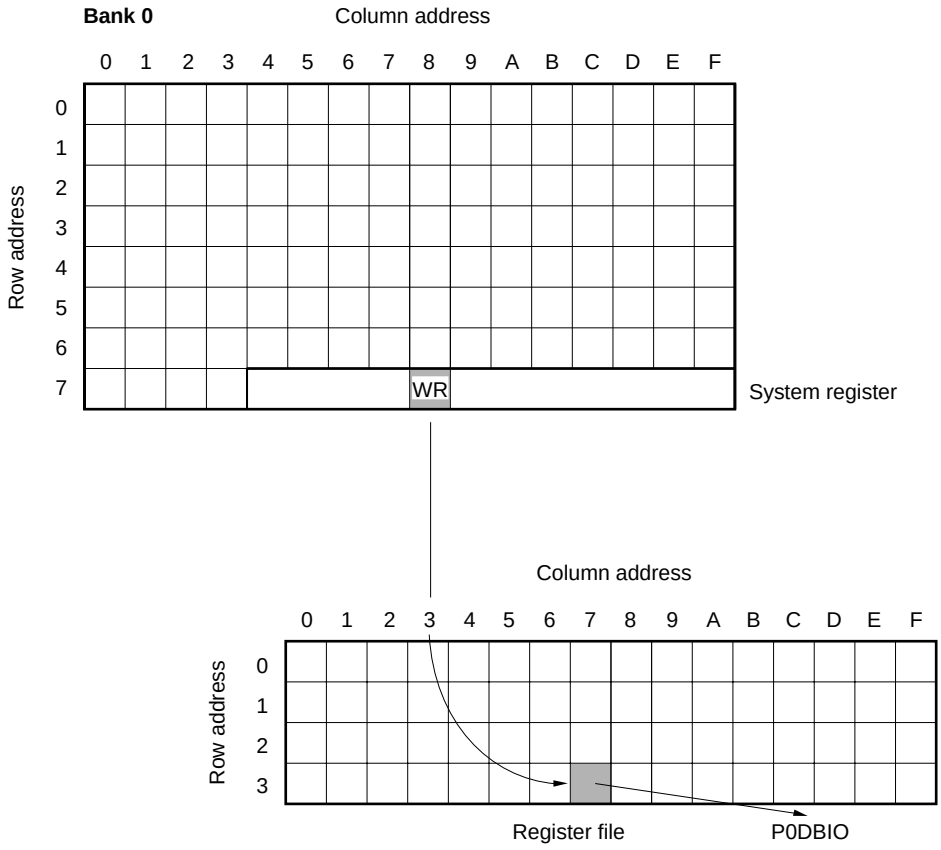
(rf) ← WR

Stores the window register WR contents to register file.

<3> Example

To store immediate data 0FH to P0DBIO for the register file through the window register:

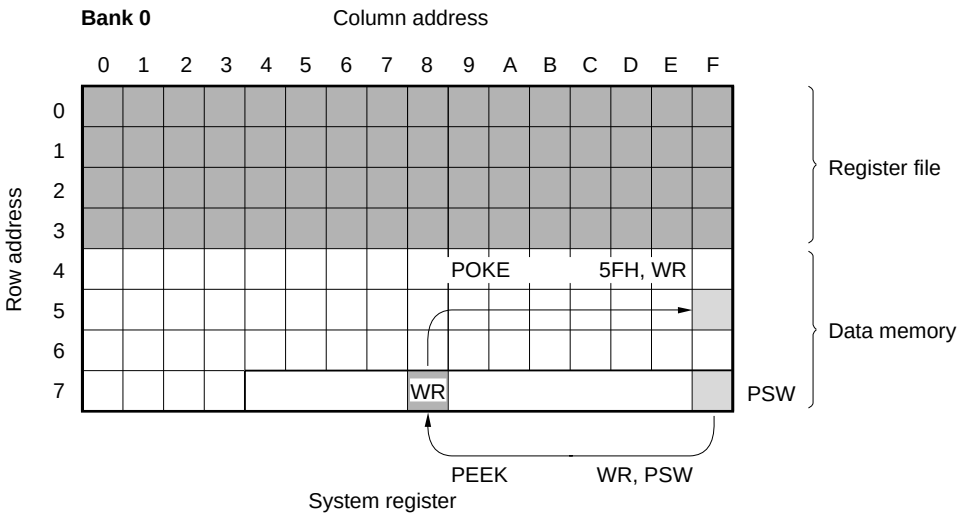
```
MOV     WR, #0FH
POKE    P0DBIO, WR      ; Sets all of P0D0, P0D1, P0D2, and P0D3 in output mode
```



★ **<4> Caution**

It seems that the same addresses 40H through 7FH of the data memory exist at addresses 40H through 7FH of the register file as for as the program is concerned.
The PEEK and POKE instructions can access addresses 40H through 7FH in each data memory bank, in addition to the register file. For example, these instructions can be used as follows:

```
MEM05F   MEM    0.5FH
         PEEK   WR, PSW       ; Stores PSW (7FH) contents in system register to WR
         POKE   MEM05F, WR    ; Stores WR contents to address 5FH in data memory
```



(11) GET DBF, p **Get peripheral data to data buffer**

<1> OP code

10 – 8	7 – 4	3 – 0
00111	p_H	1011	p_L

<2> Function

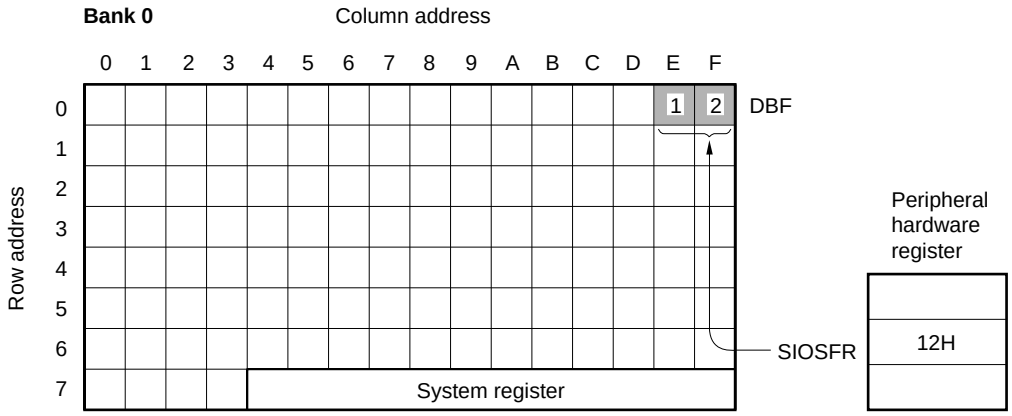
DBF ← (p)

Stores the peripheral register contents to data buffer DBF.
DBF is a 16-bit area of addresses 0H through 0FH of BANK0 of the data memory regardless of the value of the bank register.

<3> Example

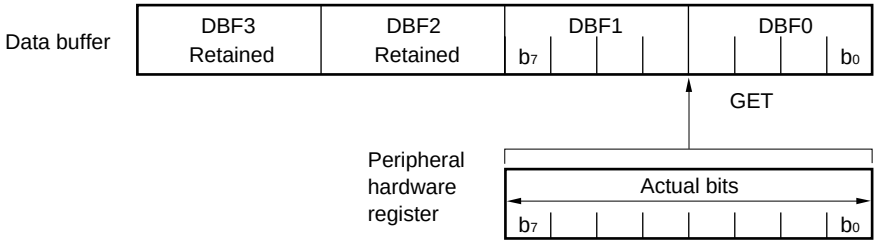
To store the 8-bit contents for shift register SIOSFR in the serial interface to data buffers DBF0 and DBF1:

```
GET      DBF, SIOSFR
```



★ **<4> Caution**

The data buffer is configured in 16 bits. However, the number of bits accessed differs depending on the peripheral hardware. For example, if the GET instruction is executed to a peripheral hardware register with a valid bit length of 8 bits, the contents of the peripheral hardware register are stored to the low-order 8 bits (DBF1, DBF0) of the data buffer DBF.



(12) PUT p, DBF — Put data buffer peripheral

<1> OP code

10	8	7	4	3	0
00111	p_H	1010	p_L		

★ **<2> Function**

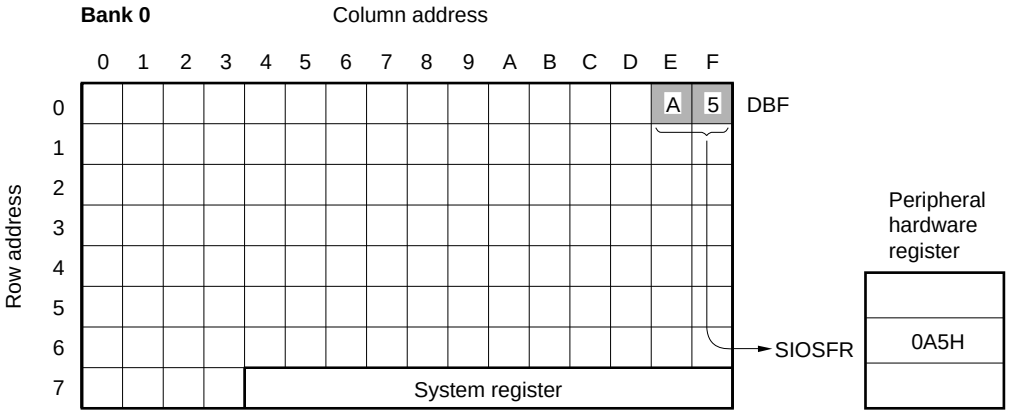
(p) ← DBF

Stores the data buffer DBF contents to peripheral hardware register.
DBF is a 16-bit area of addresses 0H through 0FH of BANK0 of the data memory regardless of the value of the bank register.

<3> Example

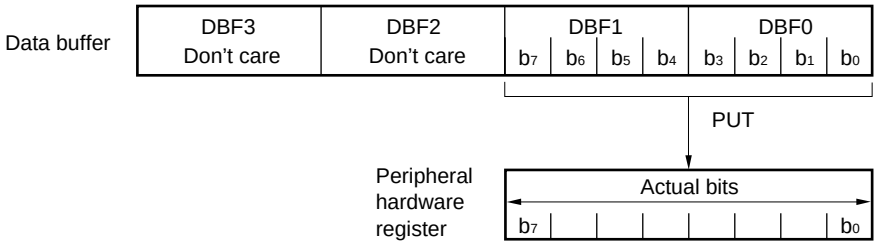
To set 0AH and 05H to data buffers DBF1 and DBF0, respectively, and transfer them to a peripheral register, shift register (SIOSFR) for serial interface:

```
MOV     BANK,     #00H        ; Data memory bank 0
MOV     DBF0,     #05H
MOV     DBF1,     #0AH
PUT     SIOSFR,   DBF
```



★ **<4> Caution**

The data buffer is configured in 16 bits. However, the number of bits accessed differs depending on the peripheral hardware. For example, if the GET instruction is executed to a peripheral hardware register with a valid bit length of 8 bits, the contents of the peripheral hardware register are stored to the low-order 8 bits (DBF1, DBF0) of the data buffer DBF.



19.5.8 Branch Instructions

(1) BR addr **Branch to the address**

<1> OP code

10	0
01100	addr

<2> Function

PC ← addr

Branches to an address specified by addr.

<3> Example

```
FLY    LAB     0FH      ; Defines FLY = 0FH
          :
          :
       BR      FLY      ; Jumps to address 0FH
          :
          :
       BR      LOOP1    ; Jumps to LOOP1
          :
          :
       BR      $ + 2    ; Jumps to an address 2 addresses lower than current address
          :
          :
       BF      $ – 3    ; Jumps to an address 3 addresses higher than current address
          :
          :
LOOP1:
```

(2) BR @AR **Branch to the address specified by address register**

<1> OP code

00111	000	0100	0000

<2> Function

PC ← AR

Branches to the program address, specified by address register AR.

<3> Example 1

To set 003FH in address register AR (AR0 – AR3) and jump to address 003FH by using the BR @AR instruction:

```
MOV   AR3,   #00H      ; AR3 ← 00H
MOV   AR2,   #00H      ; AR2 ← 00H
MOV   AR1,   #03H      ; AR1 ← 03H
MOV   AR0,   #0FH      ; AR0 ← 0FH
BR    @AR              ; Jumps to address 003FH
```

★ **Example 2**

To change the branch destination according to the data memory address 0.10H contents, as follows:

0.10H contents		Branch destination label
00H	→	AAA
01H	→	BBB
02H	→	CCC
03H	→	DDD
04H	→	EEE
05H	→	FFF
06H	→	GGG
07H	→	HHH
08H – 0FH	→	ZZZ

```
; *
; ** Jump table
; *
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  Sets AR to 001XH
        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
        BR      @AR              ; than 08H
```

<4> Caution

The number of bits, for address register AR3, AR2, AR1, and AR0, differs, depending on the microcontroller model to be used.

- μPD17134A/17135A : 10 bits
- μPD17136A/17137A/17P136A/17P137A : 11 bits

19.5.9 Subroutine Instructions

(1) CALL addr **Call subroutine**

<1> OP code

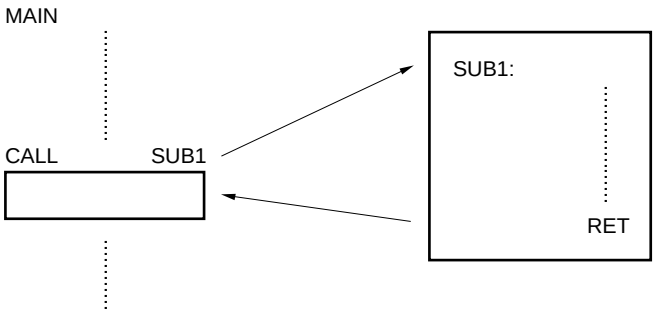
10	0
11100	addr

<2> Function

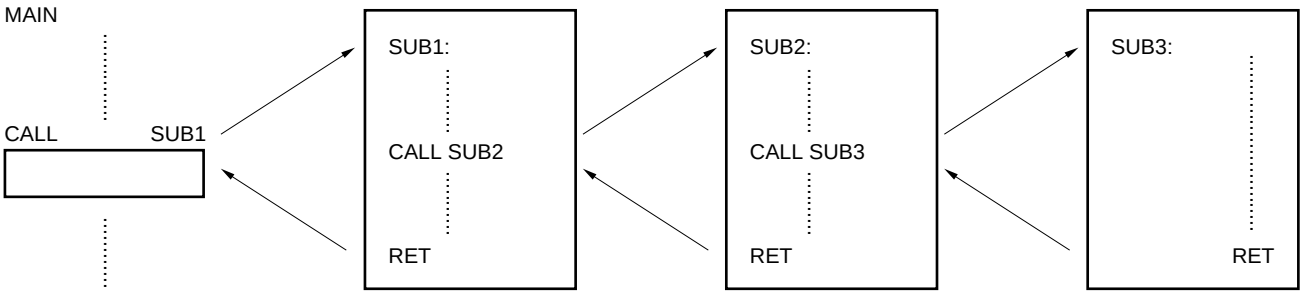
SP ← SP – 1, ASR ← PC,
PC ← addr

★ Increments and stores the program counter PC value to stack, and branches to a subroutine specified by addr.

<3> Example 1



Example 2



(2) CALL @AR — Call subroutine specified by address register

<1> OP code

00111	000	0101	0000

<2> Function

SP ← SP – 1,
ASR ← PC,
PC ← AR

Saves the program counter PC value to the stack, and branches the execution to a subroutine that starts from the address specified by address register AR.

<3> Example 1

To set 0020H in address register AR (AR0–AR3) and call the subroutine at address 0020H with the CALL @AR instruction:

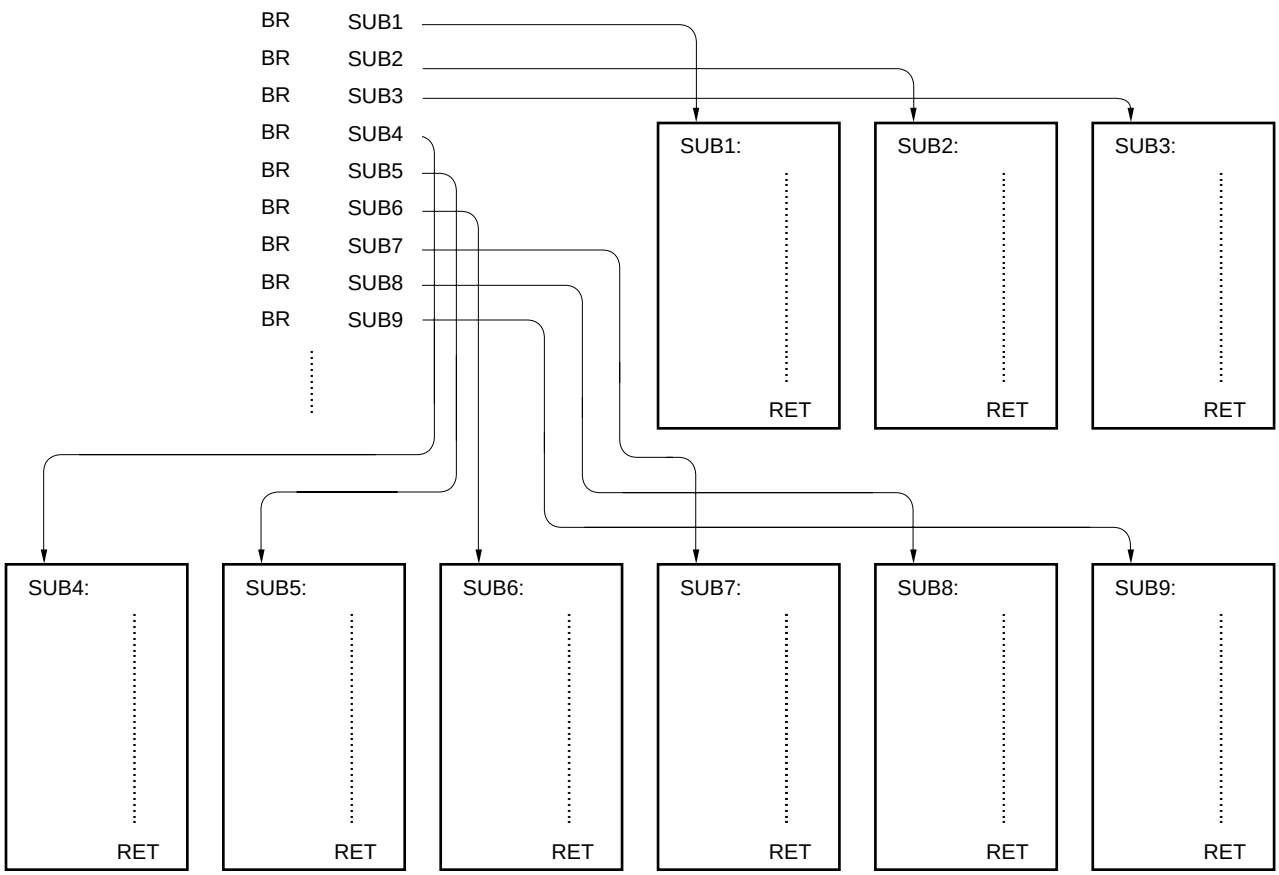
```
MOV     AR3,    #00H      ; AR3 ← 00H
MOV     AR2,    #00H      ; AR2 ← 00H
MOV     AR1,    #02H      ; AR1 ← 02H
MOV     AR0,    #00H      ; AR0 ← 00H
CALL    @AR               ; Calls subroutine at address 0020H
```

Example 2

To call the following subroutine by the data memory address 0.10H contents:

Contents of 0.10H		Subroutine
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 address register 001 · H
                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,   10H    ; AR0← 0.10H
                SKLT  AR0,   #08H   ; If AR0 is larger than 08H,
                MOV   AR0,   #08H   ; set AR0 to 08H
                CALL  @AR           ──────────► To jump table
                                    ◄──────────── Returns here when executing
                                                  RET instruction in each subroutine
```

<4> Caution

The number of bits, in address registers AR3, AR2, AR1, and AR0, differs, depending on the microcontroller model to be used.

- μPD17134A/17135A : 10 bits
- μPD17136A/17137A/17P136A/17P137A : 11 bits

(3) RET — Return to the main program from subroutine

<1> OP code

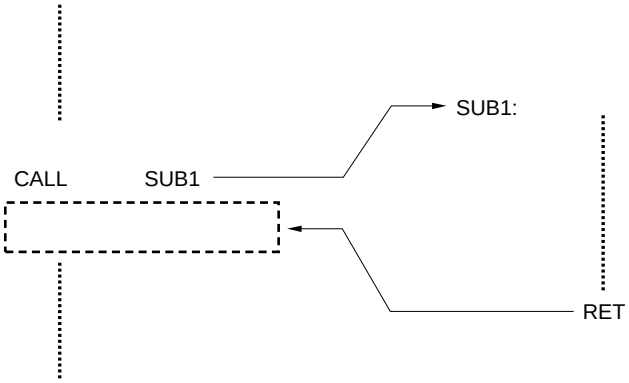
10	8 7	4 3	0
00111	000	1110	0000

<2> Function

PC ← ASR,
SP ← SP + 1,

Instruction to return to the main program from a subroutine.
Restores the return address, saved to the stack by the CALL instruction, to the program counter.

<3> Example



(4) RETSK — Return to the main program then skip next instruction

<1> OP code

00111	001	1110	0000

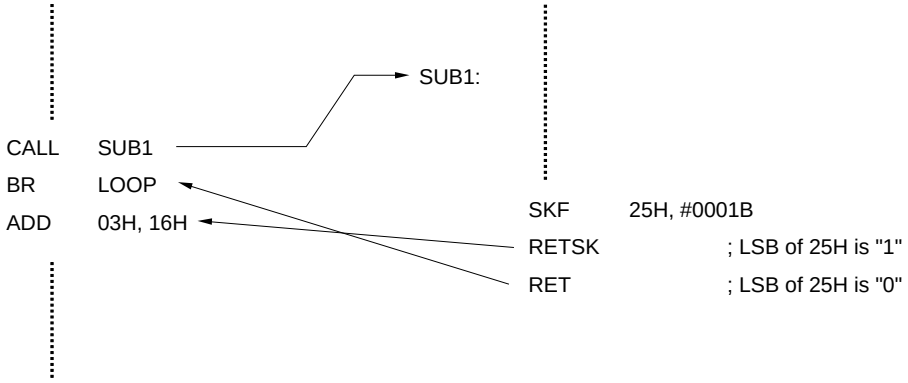
<2> Function

PC ← ASR, SP ← SP + 1 and skip

Instruction to return to the main program from a subroutine.
Skips the instruction next to the CALL instruction (i.e., treats that instructions as an NOP instruction).
Therefore, restores the return address, saved to the stack by the CALL instruction, to program counter PC and then increments the program counter.

<3> Example

To execute the RET instruction, if the LSB (least significant bit) content for address 25H in the data memory (RAM) is 0. The execution is returned to the instruction next to the CALL instruction. If the LSB is 1, execute the RETSK instruction. The execution is returned to the instruction following the one next to the CALL instruction (in this example, ADD 03H, 16H).



(5) RETI **Return to the main program from the interrupt service routine**

<1> OP code

00111	100	1110	0000

★ **<2> Function**

PC ← ASR, INTR ← INTSK, SP ← SP + 1

Instruction to return to the main program, from an interrupt service routine.
Restores the return address, saved to the stack by a vector interrupt, to the program counter.
Part of the system register (BANK, PSWORD) is also returned to the status before the occurrence of the vector interrupt.

19.5.10 Interrupt Instructions

(1) EI **Enable Interrupt**

<1> OP code

00111	000	1111	0000

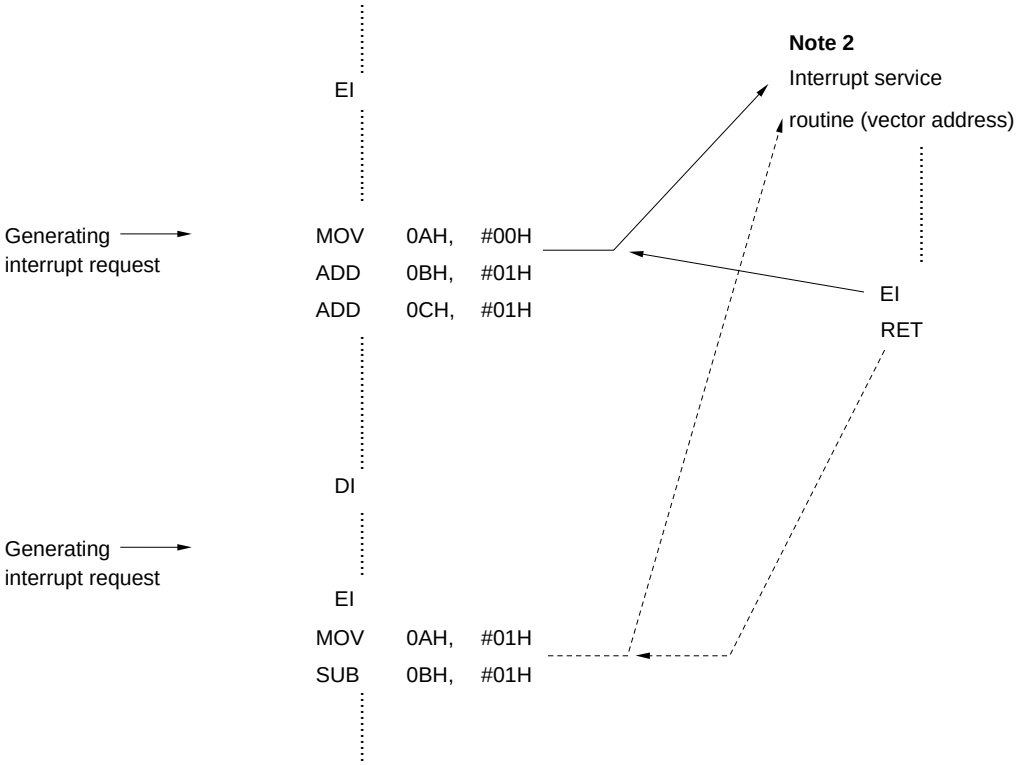
<2> Function

INTEF ← 1

Enables a vectored interrupt.
The interrupt is enabled after the instruction next to the EI instruction has been executed.

<3> Example 1

As shown in the following example, the interrupt is accepted after the instruction next to that, that has accepted the interrupt, has been completely executed (excluding an instruction that manipulates program counter). The flow then shifts to the vector address [Note1].

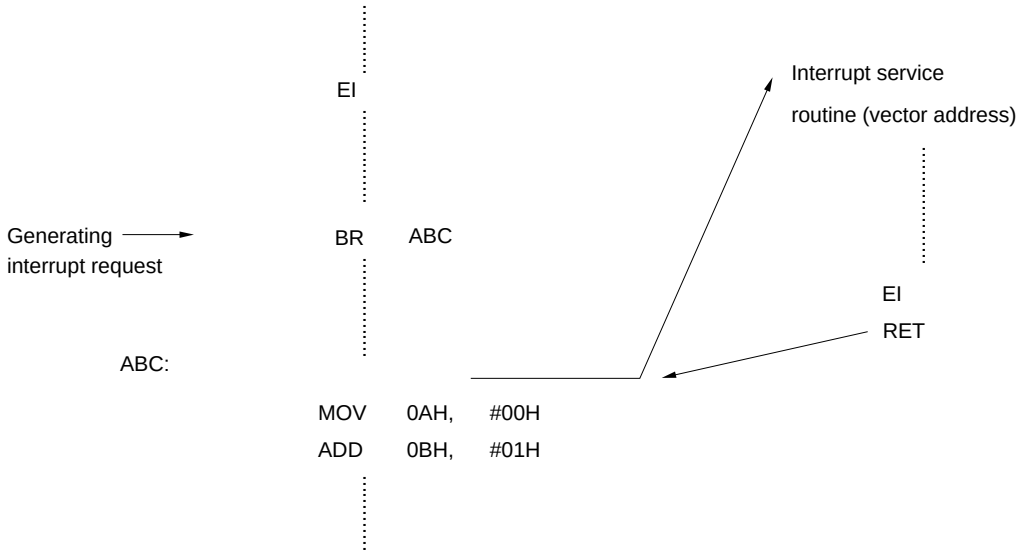


Notes 1. The vector address differs, depending on the interrupt to be accepted. Refer to **Table 14-1**.

2. The interrupt accepted in this example (an interrupt request is generated after the EI instruction has been executed and the execution flow shifts to an interrupt service routine) is the interrupt, whose interrupt enable flag (IP×××) is set. The program flow is not changed, without the interrupt enable flag set, even if an interrupt request is generated, after the EI instruction has been executed (therefore, the interrupt is not accepted). However, interrupt request flag (IRQ×××) is set, and the interrupt is accepted, as soon as the interrupt enable flag is set.

Example 2

An example of an interrupt, which occurs in response to an interrupt request being accepted counter PC is being executed:



(2) DI **Disable interrupt**

<1> OP code

00111	001	1111	0000

<2> Function

INTEF ← 0

Instruction to disable a vectored interrupt.

<3> Example

Refer to Example 1 in (1) EI.

19.5.11 Other Instructions

(1) STOP s **Stop CPU and release by condition s**

<1> OP code

			3 … 0
00111	010	1111	s

<2> Function

Stops the system clock and places the device in the STOP mode.
In the STOP mode, the power consumption for the device is minimized.
The condition, under which the STOP mode is to be released, is specified by operand (s).
For the stop releasing condition (s), refer to **16.3 STOP MODE**.

(2) HALT h **Halt CPU and release by condition h**

<1> OP code

			3 … 0
00111	011	1111	h

<2> Function

Places the device in the HALT mode.
In the HALT mode, the power consumption for the device is reduced.
The condition, under which the HALT mode is to be released, is specified by operand (h).
For HALT releasing condition (h), refer to **16.2 HALT MODE**.

(3) NOP **No operation**

<1> OP code

00111	100	1111	0000

<2> Function

Performs nothing and consumes one machine cycle.

[MEMO]

20.1 MASK OPTION DIRECTIVE

The μPD173134A, 17135A, 17136A, and 17137A have the following mask options.

- Internal pull-up resistor of $\overline{\text{RESET}}$ pin
- Internal pull-up resistor of $P0D_3$ through $P0D_0$ pins
- Internal pull-up resistor of $P1A_3$ through $P1A_0$ pins
- Internal pull-up resistor of $P1B_0$ pin

When developing a program, all the above mask options must be specified in the source program by using mask option directives.

20.1.1 Specifying Mask Option

The mask options are described in the assembler source program by using the following directives.

- OPTION and ENDOP directives
- Mask option definition directive

(1) OPTION and ENDOP directives

These directives specify the range in which the mask option is to be described (mask option definition block). Specify the mask option by describing the mask option directive in an area between the OPTION and ENDOP directives.

Description format

Symbol field	Mnemonic field	Operand field	Comment field
[label:]	OPTION		[;comment]
	⋮		
	ENDOP		

(2) Mask option definition directive

Table 20-1. Mask Option Definition Directive

Option	Definition directive and format	Operand	Definition
Internal pull-up resistor of $\overline{\text{RESET}}$ pin	OPTRES <operand>	OPEN	None
		PULLUP	Provided
Internal pull-up resistor of $P0D_3$ through $P0D_0$ pins	OPTP0D <operand 1>, ..., <operand 4>[Note 1]	OPEN	None
		PULLUP	Provided
Internal pull-up resistor of $P1A_3$ through $P1A_0$ pins	OPTP1A <operand 1>, ..., <operand 4>[Note 2]	OPEN	None
		PULLUP	Provided
Internal pull-up resistor of $P1B_0$ pin	OPTP1B <operand>	OPEN	Not used
		PULLUP	Used

Notes 1. <operand 1>, <operand 2>, <operand 3>, and <operand 4> specify the mask options of the $P0D_3$, $P0D_2$, $P0D_1$, and $P0D_0$ pins, respectively.

2. <operand 1>, <operand 2>, <operand 3>, and <operand 4> specify the mask options of the $P1A_3$, $P1A_2$, $P1A_1$, and $P1A_0$ pins, respectively.

(3) Example of describing mask option

```
; Example of describing mask option of μPD17134A subseries

  MASK_OPTION:

    OPTION                                  ; Start of mask option definition block
    OPTRES    PULLUP                        ; Internal pull-up resistor is connected to RESET pin.
    OPTP0D    PULLUP, PULLUP, OPEN, OPEN    ; Internal pull-up resistor is connected to P0D3 and P0D2 pins.
                                            ; P0D1 and P0D0 are open (externally pulled up).
    OPTP1A    PULLUP, OPEN, PULLUP, OPEN    ; P1A3 and P1A1 pins are connected to internal pull-up resistor.
                                            ; P1A2 and P1A0 pins are open (externally pulled up).
    OPTP1A    PULLUP                        ; P1B0 pin is connected to internal pull-up resistor.
    ENDOP                                   ; End of mask option definition block
```

20.2 RESERVED SYMBOLS

The reserved symbols defined in the μPD17134A, 17135A, 17136A, and 17137A device file (AS17134) are listed below.

System register (SYSREG)

Symbolic name	Attribute	Value	Read/ write	Description
AR3	MEM	0.74H	R	Bits 15 to 12 of the address register
AR2	MEM	0.75H	R/W	Bits 11 to 8 of the address register
AR1	MEM	0.76H	R/W	Bits 7 to 4 of the address register
AR0	MEM	0.77H	R/W	Bits 3 to 0 of the address register
WR	MEM	0.78H	R/W	Window register
BANK	MEM	0.79H	R/W	Bank register
IXH	MEM	0.7AH	R/W	Index register high
MPH	MEM	0.7AH	R/W	Data memory row address pointer high
MPE	FLG	0.7AH.3	R/W	Memory pointer enable flag
IXM	MEM	0.7BH	R/W	Index register middle
MPL	MEM	0.7BH	R/W	Data memory row address pointer low
IXL	MEM	0.7CH	R/W	Index register low
RPH	MEM	0.7DH	R/W	General register pointer high
RPL	MEM	0.7EH	R/W	General register pointer low
PSW	MEM	0.7FH	R/W	Program status word
BCD	FLG	0.7EH.0	R/W	BCD flag
CMP	FLG	0.7FH.3	R/W	Compare flag
CY	FLG	0.7FH.2	R/W	Carry flag
Z	FLG	0.7FH.1	R/W	Zero flag
IXE	FLG	0.7FH.0	R/W	Index enable flag

Data buffer (DBF)

Symbolic name	Attribute	Value	Read/ write	Description
DBF3	MEM	0.0CH	R/W	DBF bits 15 to 12
DBF2	MEM	0.0DH	R/W	DBF bits 11 to 8
DBF1	MEM	0.0EH	R/W	DBF bits 7 to 4
DBF0	MEM	0.0FH	R/W	DBF bits 3 to 0

Port register

Symbolic name	Attribute	Value	Read/ write	Description
P0A3	FLG	0.70H.3	R/W	Port 0A bit 3
P0A2	FLG	0.70H.2	R/W	Port 0A bit 2
P0A1	FLG	0.70H.1	R/W	Port 0A bit 1
P0A0	FLG	0.70H.0	R/W	Port 0A bit 0
P0B3	FLG	0.71H.3	R/W	Port 0B bit 3
P0B2	FLG	0.71H.2	R/W	Port 0B bit 2
P0B1	FLG	0.71H.1	R/W	Port 0B bit 1
P0B0	FLG	0.71H.0	R/W	Port 0B bit 0
P0C3	FLG	0.72H.3	R/W	Port 0C bit 3
P0C2	FLG	0.72H.2	R/W	Port 0C bit 2
P0C1	FLG	0.72H.1	R/W	Port 0C bit 1
P0C0	FLG	0.72H.0	R/W	Port 0C bit 0
P0D3	FLG	0.73H.3	R/W	Port 0D bit 3
P0D2	FLG	0.73H.2	R/W	Port 0D bit 2
P0D1	FLG	0.73H.1	R/W	Port 0D bit 1
P0D0	FLG	0.73H.0	R/W	Port 0D bit 0
P1A3	FLG	1.70H.3	R/W	Port 1A bit 3
P1A2	FLG	1.70H.2	R/W	Port 1A bit 2
P1A1	FLG	1.70H.1	R/W	Port 1A bit 1
P1A0	FLG	1.70H.0	R/W	Port 1A bit 0
P1B0	FLG	1.71H.0	R	Port 1B bit 0

Register file (control register)

Symbolic name	Attribute	Value	Read/ write	Description
SP	MEM	0.81H	R/W	Stack pointer
SIOTS	FLG	0.82H.3	R/W	SIO start flag
SIOHIZ	FLG	0.82H.2	R/W	SO pin state
SIOCK1	FLG	0.82H.1	R/W	SIO source clock selection flag bit 1
SIOCK0	FLG	0.82H.0	R/W	SIO source clock selection flag bit 0
WDTRES	FLG	0.83H.3	R/W	Watchdog timer reset flag
WDTEN	FLG	0.83H.0	R/W	Watchdog timer enable flag
TM0OSEL	FLG	0.8BH.3	R/W	Flag for switching timer 0 output and port
SIOEN	FLG	0.8BH.0	R/W	SIO enable flag
P0BGPU	FLG	0.8CH.1	R/W	P0B group pull-up selection flag (pull-up = 1)
P0AGPU	FLG	0.8CH.0	R/W	P0A group pull-up selection flag (pull-up = 1)
INT	FLG	0.8FH.0	R	INT pin status flag
PDRESEN	FLG	0.90H.0	R/W	Power-down reset enable flag
TM0EN	FLG	0.91H.3	R/W	Timer 0 enable flag
TM0RES	FLG	0.91H.2	R/W	Timer 0 reset flag
TM0CK1	FLG	0.91H.1	R/W	Timer 0 source clock selection flag bit 1
TM0CK0	FLG	0.91H.0	R/W	Timer 0 source clock selection flag bit 0
TM1EN	FLG	0.92H.3	R/W	Timer 1 enable flag
TM1RES	FLG	0.92H.2	R/W	Timer 1 reset flag
TM1CK1	FLG	0.92H.1	R/W	Timer 1 source clock selection flag bit 1
TM1CK0	FLG	0.92H.0	R/W	Timer 1 source clock selection flag bit 0
BTMISEL	FLG	0.93H.3	R/W	BTM interrupt request clock selection flag
BTMRES	FLG	0.93H.2	R/W	BTM reset flag
BTMCK1	FLG	0.93H.1	R/W	BTM source clock selection flag bit 1
BTMCK0	FLG	0.93H.0	R/W	BTM source clock selection flag bit 0
P0C3IDI	FLG	0.9BH.3	R/W	$P0C_3$ input port disable flag (ADC_3/$P0C_3$ selection)
P0C2IDI	FLG	0.9BH.2	R/W	$P0C_2$ input port disable flag (ADC_2/$P0C_2$ selection)
P0C1IDI	FLG	0.9BH.1	R/W	$P0C_1$ input port disable flag (ADC_1/$P0C_1$ selection)
P0C0IDI	FLG	0.9BH.0	R/W	$P0C_0$ input port disable flag (ADC_0/$P0C_0$ selection)
P0CBIO3	FLG	0.9CH.3	R/W	$P0C_3$ input/output selection flag (1 = output port)
P0CBIO2	FLG	0.9CH.2	R/W	$P0C_2$ input/output selection flag (1 = output port)
P0CBIO1	FLG	0.9CH.1	R/W	$P0C_1$ input/output selection flag (1 = output port)
P0CBIO0	FLG	0.9CH.0	R/W	$P0C_0$ input/output selection flag (1 = output port)
ZCROSS	FLG	0.9DH.0	R/W	Zerocross detector enable flag
IEGMD1	FLG	0.9FH.1	R/W	INT pin edge detection selection flag bit 1
IEGMD0	FLG	0.9FH.0	R/W	INT pin edge detection selection flag bit 0
ADCSTRT	FLG	0.0A0H.0	R/W	A/D converter start flag (always 0 when read)
ADCSOFT	FLG	0.0A1H.3	R/W	A/D converter software control flag (1 = single mode)
ADCCMP	FLG	0.0A1H.1	R/W	A/D converter comparison result flag (valid only in single mode)
ADCEND	FLG	0.0A1H.0	R/W	A/D converter conversion end flag

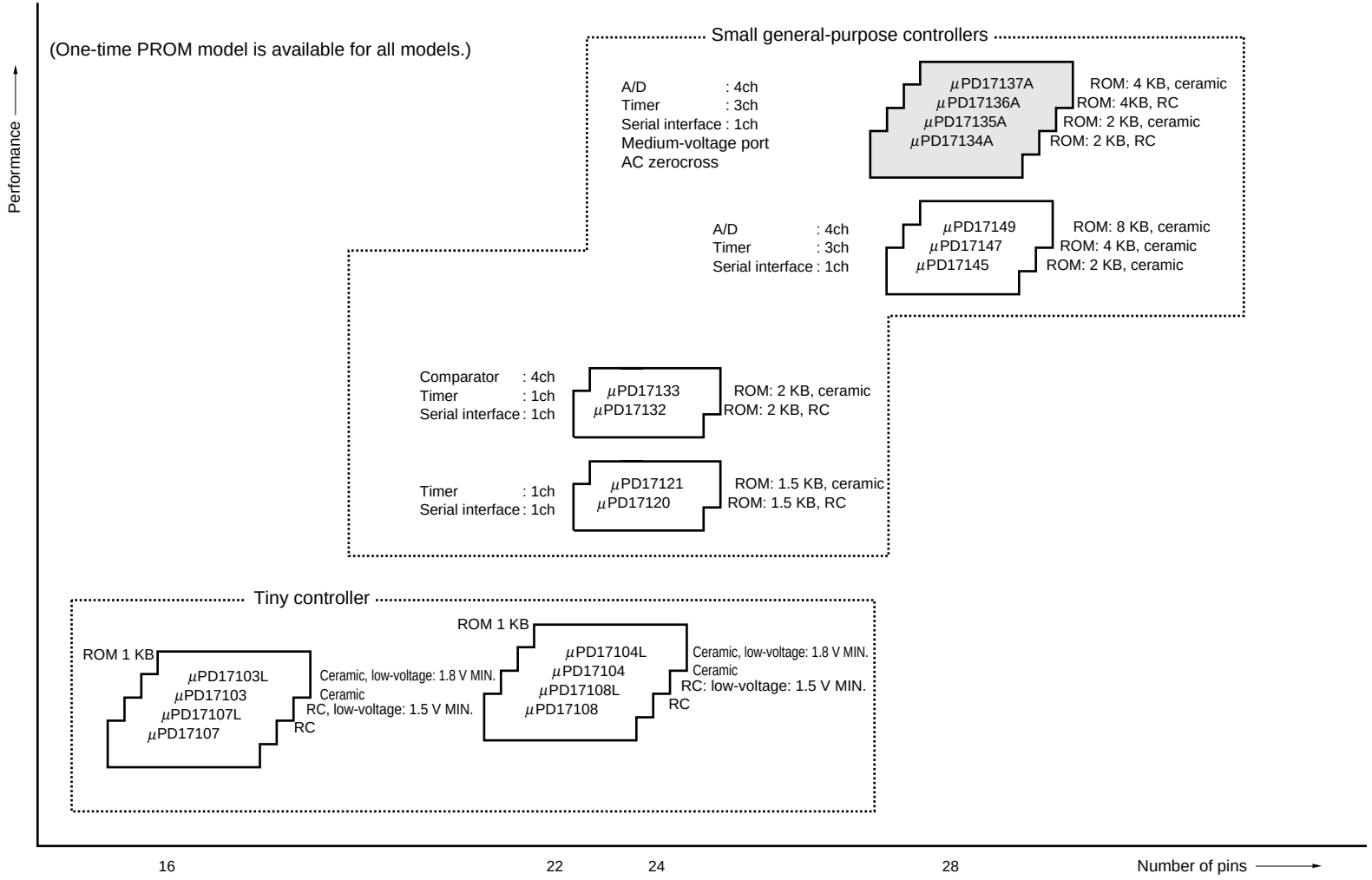
Symbolic name	Attribute	Value	Read/ write	Description
ADCCH3	FLG	0.0A2H.3	R/W	Dummy flag
ADCCH2	FLG	0.0A2H.2	R/W	Dummy flag
ADCCH1	FLG	0.0A2H.1	R/W	A/D converter channel selection flag bit 1
ADCCH0	FLG	0.0A2H.0	R/W	A/D converter channel selection flag bit 0
P0DBIO3	FLG	0.0ABH.3	R/W	$P0D_3$ input/output selection flag (1 = output port)
P0DBIO2	FLG	0.0ABH.2	R/W	$P0D_2$ input/output selection flag (1 = output port)
P0DBIO1	FLG	0.0ABH.1	R/W	$P0D_1$ input/output selection flag (1 = output port)
P0DBIO0	FLG	0.0ABH.0	R/W	$P0D_0$ input/output selection flag (1 = output port)
P1AGIO	FLG	0.0ACH.2	R/W	P1A group input/output selection flag (1 = all P1As are output ports.)
P0BGIO	FLG	0.0ACH.1	R/W	P0B group input/output selection flag (1 = all P0Bs are output ports.)
P0AGIO	FLG	0.0ACH.0	R/W	P0A group input/output selection flag (1 = all P0As are output ports.)
IPSIO	FLG	0.0AEH.0	R/W	SIO interrupt enable flag
IPBTM	FLG	0.0AFH.3	R/W	BTM interrupt enable flag
IPTM1	FLG	0.0AFH.2	R/W	TM1 interrupt enable flag
IPTM0	FLG	0.0AFH.1	R/W	TM0 interrupt enable flag
IP	FLG	0.0AFH.0	R/W	INT pin interrupt enable flag
IRQSIO	FLG	0.0BBH.0	R/W	SIO interrupt request flag
IRQBTM	FLG	0.0BCH.0	R/W	BTM interrupt request flag
IRQTM1	FLG	0.0BDH.0	R/W	TM1 interrupt request flag
IRQTM0	FLG	0.0BEH.0	R/W	TM0 interrupt request flag
IRQ	FLG	0.0BFH.0	R/W	INT pin interrupt request flag

Peripheral register

Symbolic name	Attribute	Value	Read/ write	Description
SIOSFR	DAT	01H	R/W	Peripheral address of the shift register
TM0M	DAT	02H	W	Peripheral address of the timer 0 modulo register
TM1M	DAT	03H	W	Peripheral address of the timer 1 modulo register
ADCR	DAT	04H	R/W	Peripheral address of A/D converter data register
TM0TM1C	DAT	45H	R	Peripheral address of timer 0 timer 1 count register
AR	DAT	40H	R/W	Peripheral address of the address register for GET, PUT, PUSH, CALL, BR, MOVT, and INC instructions

Others

Symbolic name	Attribute	Value	Description
DBF	DAT	0FH	Fixed operand value of PUT, GET, or MOVT instruction
IX	DAT	01H	Fixed operand value of INC instruction


Performance
(One-time PROM model is available for all models.)
Small general-purpose controllers
A/D : 4ch
Timer : 3ch
Serial interface : 1ch
Medium-voltage port
AC zerocross
μPD17137A
μPD17136A
μPD17135A
μPD17134A
ROM: 4 KB, ceramic
ROM: 4KB, RC
ROM: 2 KB, ceramic
ROM: 2 KB, RC
A/D : 4ch
Timer : 3ch
Serial interface : 1ch
μPD17149
μPD17147
μPD17145
ROM: 8 KB, ceramic
ROM: 4 KB, ceramic
ROM: 2 KB, ceramic
Comparator : 4ch
Timer : 1ch
Serial interface : 1ch
μPD17133
μPD17132
ROM: 2 KB, ceramic
ROM: 2 KB, RC
Timer : 1ch
Serial interface : 1ch
μPD17121
μPD17120
ROM: 1.5 KB, ceramic
ROM: 1.5 KB, RC
Tiny controller
ROM 1 KB
μPD17103L
μPD17103
μPD17107L
μPD17107
Ceramic, low-voltage: 1.8 V MIN.
Ceramic
RC, low-voltage: 1.5 V MIN.
RC
ROM 1 KB
μPD17104L
μPD17104
μPD17108L
μPD17108
Ceramic, low-voltage: 1.8 V MIN.
Ceramic
RC: low-voltage: 1.5 V MIN.
RC
16
22
24
28
Number of pins

[MEMO]

★

APPENDIX B COMPARISON OF FUNCTIONS BETWEEN μPD17135A, 17137A, AND μPD17145 SUBSERIES

(1/2)

		μPD17145	μPD17147	μPD17149	μPD17135A	μPD17137A
ROM		2K bytes	4K bytes	8K bytes	2K bytes	4K bytes
RAM		110 × 4 bits			112 × 4 bits	
Stack		Address stack × 5 levels Interrupt stack × 3 levels				
Instruction execution time (clock, supply voltage)		2 μs (f_X = 8 MHz, V_{DD} = 4.5 to 5.5 V) 4 μs (f_X = 4 MHz, V_{DD} = 3.6 to 5.5 V) 8 μs (f_X = 2 MHz, V_{DD} = 2.7 to 5.5 V)			2 μs (f_X = 8 MHz, V_{DD} = 4.5 to 5.5 V) 4 μs (f_X = 4 MHz, V_{DD} = 2.7 to 5.5 V)	
I/O	CMOS I/O	12 (P0A, P0B, P0C)				
	Input	2 ($P0F_0$, $P0F_1$)			1 ($P1B_0$)	
	Sense input	1 (INT) Can be pulled up by mask option			1 (INT)	
	N-ch open-drain I/O	8 (P0D, P0E, voltage: V_{DD}) P0D pull up: software P0E pull up: software			8 (P0D, P1A, voltage: 9 V) P0D pull up: mask option P1A pull up: mask option	
Internal pull-up resistor		100 kΩ TYP. (except P0D) 10 kΩ TYP. (P0D)			100 kΩ TYP.	
A/D converter (supply voltage)		8 bits × 4 channels (V_{DD} = 4.0 to 5.5 V)			8 bits × 4 channels (V_{DD} = 4.5 to 5.5 V)	
	Reference voltage pin	V_{REF} (V_{REF} = 2.5 to V_{DD})			None (V_{REF} = V_{ADC} = V_{DD})	
Timer	8 bits (TM0, TM1)	2 (timer output: $\overline{TM1OUT}$) TM0 clock : f_X/512 f_X/64 f_X/16 INT TM1 clock : f_X/8192 f_X/128 f_X/16 TM0 count up			2 (timer output: $\overline{TM0OUT}$) TM0 clock : f_X/256 f_X/64 f_X/16 INT TM1 clock : f_X/1024 f_X/512 f_X/256 TM0 count up	
	Basic interval timer (BTM)	1 (multiplexed with watchdog timer) Count pulse : f_X/16384 f_X/4096 f_X/512 f_X/16			1 (multiplexed with watchdog timer) Count pulse : f_X/8192 f_X/4096 TM0 count up INT	
Interrupt	External	1			1 (with AC zero cross detection function)	
	Internal	4 (TM0, TM1, BTM, SIO)				
SIO		1 (clocked 3-wire)				
	Output latch	Independent of $P0D_1$ latch			Multiplexed with $P0D_1$ latch	

(2/2)

	μPD17145	μPD17147	μPD17149	μPD17135A	μPD17137A
Standby function	HALT, STOP (with input pin $\overline{\text{RLS}}$ for releasing)			HALT, STOP	
Oscillation stabilization wait time	128 × 256 counts			512 × 256 counts	
POC function	Mask option			Internal	
Package	28-pin plastic SDIP (400 mil) 28-pin plastic SOP (375 mil)				
One-time PROM	μPD17P149			μPD17P137A	

Caution The μPD17145 subseries is not pin-compatible with the μPD17135A and 17137A. The μPD17145 subseries has no model equivalent to the μPD17134A and 17136A (RC oscillation type). For the electrical characteristics, refer to the Data Sheet of each model.

Remark fx: system clock oscillation frequency

APPENDIX C DEVELOPMENT TOOLS

The following support tools are available for developing programs for the μPD17134A subseries.

Hardware

Name	Outline
In-circuit emulator IE-17K IE-17K-ET[Note1] EMU-17K[Note2]	These are in-circuit emulators that can be commonly used with microcontrollers in 17K series. IE-17K and IE-17K-ET are connected to a host machine, NEC PC-9800 series or IBM PC/AT™, through RS-232-C. EMU-17K is mounted in expansion slot of NEC PC-9800 series that serves as host machine. When these in-circuit emulators are used in combination with the evaluation board (SE board) dedicated to each model of microcontroller, they operate as emulators corresponding to microcontroller. When these in-circuit emulators are used with man-machine interface software *SIMPLEHOST*™, a more sophisticated debugging environment can be created. EMU-17K also has a function that allows you to monitor data memory contents real-time.
SE board (SE-17134)	SE-17134 is an SE board for μPD17134A subseries series. It can be used alone for system evaluation or in combination with an in-circuit emulator for debugging.
Emulation probe (EP-17K28CT)	EP-17K28CT is an emulation probe for 17K series 28-pin shrink DIP (400 mil) and connects SE board and target system.
Emulation probe (EP-17K28GT)	EP-17K28GT is an emulation probe for 17K series 28-pin SOP (375 mil) and connects SE board and target system by being used with EV-9500GT-28[Note3].
Conversion adapter (EV-9500GT-28[Note3])	EV-9500GT-28 is an adapter for 28-pin SOP (375 mil) and is used to connect EP-17K28GT to target system.
PROM programmer AF-9703[Note4] AF-9704[Note4] AF-9705[Note4] AF-9706[Note4]	AF-9703, AF-9704, AF-9705, and AF-9706 are PROM programmers corresponding to μPD17P136A and 17P137A. When connected with program adapter AF-9808F, these programmers can be used to program μPD17P136A and 17P137A.
Programmer adapter (AF-9808F[Note4])	AF-9808F is an adapter for programming μPD17P136A and 17P137A, and is used in combination with AF-9703, AF-9704 or AF-9706.

Notes 1. Low-price model: external power supply type

2. This is a program of IC Corp. For details, consult IC.

3. Two EV-9500GT-28 are supplied as accessories with the EP-17K28GT. Five EV-9500GT-28's are optionally available as a set.

4. Manufactured by Ando Electric. For details, consult Ando Electric.

Software

Name	Description	Host machine	OS	Distribution media	Part number
17K series assembler (AS17K)	AS17K is an assembler applicable to the 17K series. In developing μPD17134A, 17135A, 17136A, and 17137A programs, AS17K is used in combination with a device file (AS17134).	PC-9800 series	MS-DOS™	5-inch, 2HD	μS5A10AS17K
				3.5-inch, 2HD	μS5A13AS17K
		IBM PC/AT	PC DOS™	5-inch, 2HC	μS7B10AS17K
Device file (AS17134)	AS17134 is a device file for the μPD17134A, 17135A, 17136A, and 17137A. It is used together with the assembler (AS17K) which is applicable to the 17K series.	PC-9800 series	MS-DOS	5-inch, 2HD	μS5A10AS17134
				3.5-inch, 2HD	μS5A13AS17134
		IBM PC/AT	PC DOS	5-inch, 2HC	μS7B10AS17134
Support software (*SIMPLEHOST*)	*SIMPLEHOST*, running on the Windows™, provides man-machine-interface in developing programs by using a personal computer and the in-circuit emulator.	PC-9800 series	MS-DOS / Windows	5-inch, 2HD	μS5A10IE17K
				3.5-inch, 2HD	μS5A13IE17K
		IBM PC/AT	PC DOS / Windows	5-inch, 2HC	μS7B10IE17K

Remark The supported OS versions are as follows:

OS	Version
MS-DOS	Ver. 3.30 to Ver. 5.00A[Note]
PC DOS	Ver. 3.1 to Ver. 5.0[Note]
Windows	Ver. 3.0 to Ver. 3.1

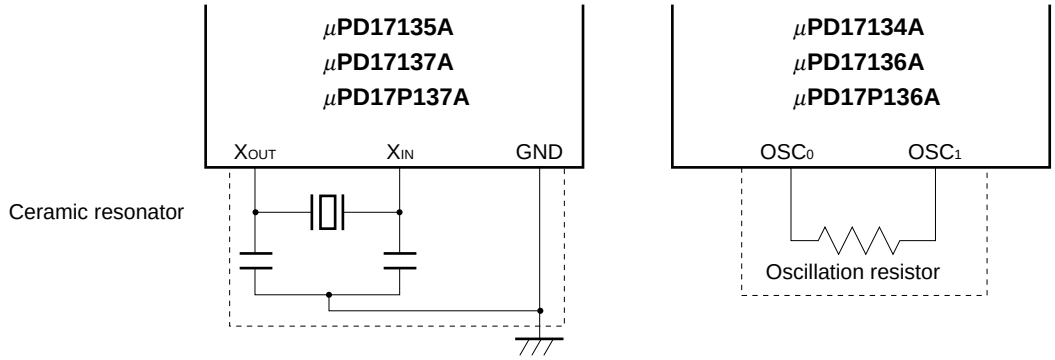
Note Although MS-DOS Ver. 5.00/5.00A and PC DOS Ver. 5.0 have a task swap function, this function cannot be used with this software.

★ APPENDIX D NOTES ON CONFIGURATION OF SYSTEM CLOCK OSCILLATION CIRCUIT

The system clock oscillation circuit oscillates by using a ceramic resonator connected across the X1 and X2 pins or an oscillation resistor connected across the OSC_1 and OSC_0 pins.

Figure D-2 shows the external circuits of the system clock oscillation circuit.

Figure D-1. External Circuit of System Clock Oscillation Circuit



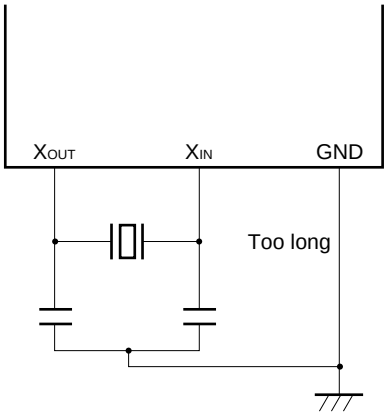
Caution Wire the system clock oscillation circuit so that the resistance component and inductance component of the ground wiring can be minimized. Wire the portion enclosed in the dotted line in Figure D-1 as follows to prevent influence of wiring capacitance.

- **Keep the wiring length as short as possible.**
- **Do not cross the wiring with the other signal lines. Keep a distance between the wiring and a line through which a high alternating current flows.**
- **Always keep the ground point of the capacitor of the oscillation circuit at the same potential as V_{SS}. Do not ground the wiring to a ground pattern through which a high current flows.**
- **Do not extract signals from the oscillation circuit.**

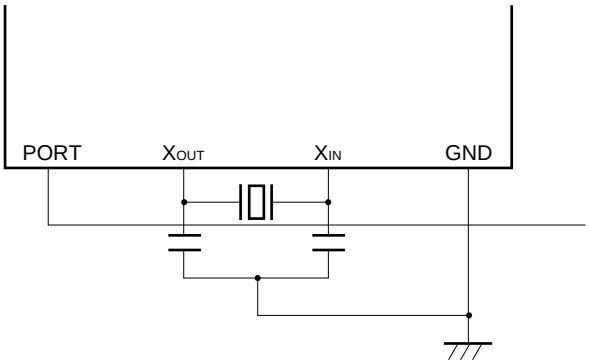
Figure D-2 shows an examples of incorrect oscillation circuits.

Figure D-2. Example of Incorrect Oscillation Circuits

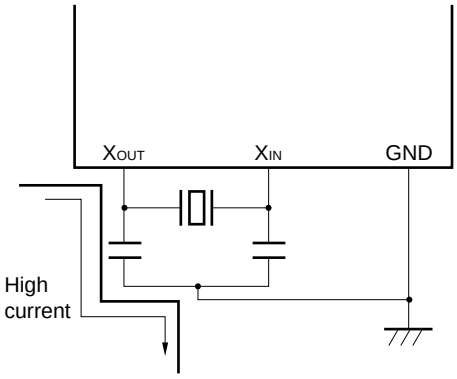
(a) Wiring length of circuit is too long.



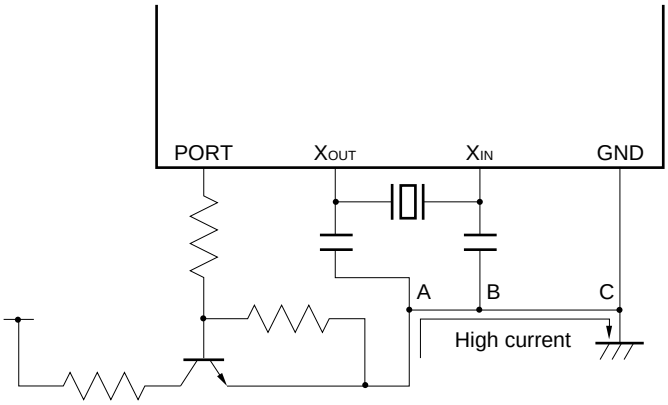
(b) Crossed signal lines



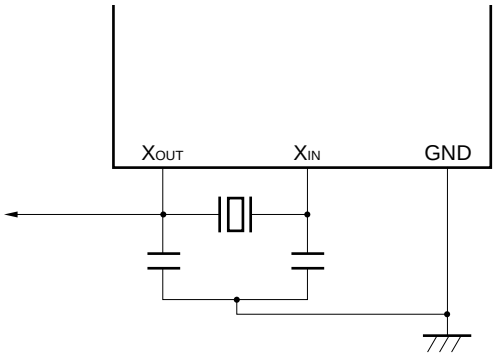
(c) Signal line close to high alternating current



(d) Current flowing through ground line of oscillation circuit (potential at points A and B changes in respect to point C)



(e) Signal is extracted



APPENDIX E INSTRUCTION LIST

E.1 INSTRUCTION LIST (by function)

[Addition Instructions]

ADD r, m ... 198
ADD m, #n4 ... 201
ADDC r, m ... 203
ADDC m, #n4 ... 205
INC AR ... 206
INC IX ... 208

[Subtraction Instructions]

SUB r, m ... 209
SUB m, #n4 ... 211
SUBC r, m ... 212
SUBC m, #n4 ... 214

[Logical Operation Instructions]

OR r, m ... 216
OR m, #n4 ... 216
AND r, m ... 217
AND m, #n4 ... 218
XOR r, m ... 219
XOR m, #n4 ... 220

[Judgment Instructions]

SKT m, #n ... 221
SKF m, #n ... 223

[Comparison Instructions]

SKE m, #n4 ... 223
SKNE m, #n4 ... 223
SKGE m, #n4 ... 224
SKLT m, #n4 ... 224

[Rotation Instructions]

RORC r ... 226

[Transfer Instructions]

LD r, m ... 227
ST m, r ... 228
MOV @r, m ... 231
MOV m, @r ... 231
MOV m, #n4 ... 233
MOVT DBF, @AR ... 234
PUSH AR ... 235
POP AR ... 237
PEEK WR, rf ... 238
POKE rf, WR ... 239
GET DBF, p ... 240
PUT p, DBF ... 241

[Branch Instructions]

BR addr ... 243
BR @AR ... 243

[Subroutine Instructions]

CALL addr ... 246
CALL @AR ... 247
RET ... 249
RETSK ... 249
RETI ... 250

[Interrupt Instructions]

EI ... 251
DI ... 252

[Other Instructions]

STOP s ... 253
HALT h ... 253
NOP ... 253

E.2 IINSTRUCTION LIST (alphabetical order)

[A]

ADD m, #n4 ... 201
ADD r, m ... 198
ADDC m, #n4 ... 205
ADDC r, m ... 203
AND m, #n4 ... 218
AND r, m ... 217

[B]

BR addr ... 243
BR @AR ... 243

[C]

CALL addr ... 246
CALL @AR ... 247

[D]

DI ... 252

[E]

EI ... 251

[G]

GET DBF, p ... 240

[H]

HALT h ... 253

[I]

INC AR ... 206
INC IX ... 208

[L]

LD r, m ... 227

[M]

MOV m, #n4 ... 233
MOV m, @r ... 231
MOV @r, m ... 231
MOVT DBF, @AR ... 234

[N]

NOP ... 253

[O]

OR m, #n4 ... 216
OR r, m ... 216

[P]

PEEK WR, rf ... 238
POKE rf, WR ... 239
POP AR ... 237
PUSH AR ... 235
PUT p, DBF ... 241

[R]

RET ... 249
RETI ... 250
RETSK ... 249
RORC r ... 226

[S]

SKE m, #n4 ... 223
SKF m, #n ... 221
SKGE m, #n4 ... 224
SKLT m, #n4 ... 224
SKNE m, #n4 ... 223
SKT m, #n ... 221
ST m, r ... 228
STOP s ... 253
SUB m, #n4 ... 211
SUB r, m ... 209
SUBC m, #n4 ... 214
SUBC r, m ... 212

[x]

XOR m, #n4 ... 220
XOR r, m ... 219

APPENDIX F ORDERING MASK ROM

After developing the program, place an order for the mask ROM version, according to the following procedure:

(1) Make reservation when ordering mask ROM.

Advise NEC of the schedule for placing an order for the mask ROM. If NEC is not informed in advance, on-time delivery may not be possible.

(2) Create ordering medium.

Use UV-EPROM to place an order for the mask ROM.
Add /PROM as an assemble option of the Assembler (AS17K), and create a mask ROM ordering HEX file (with extender for .PRO).
Next, write the mask ROM ordering HEX file into the UV-EPROM.
Create three UV-EPROMs with the same contents.

(3) Prepare necessary documents.

Fill out the following forms to place an order for the mask ROM:

- Mask ROM ordering sheet
- Mask ROM ordering check sheet

(4) Ordering

Submit the media created in (2) and documents prepared in (3) to NEC by the specified date.

Caution For details, refer to information document ROM Code Ordering Procedure (IEM-1366).

[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: +55-11-889-1689

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 Corporation
Semiconductor Solution Engineering Division
Technical Information Support Dept.
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	❑	❑	❑	❑

CS 96.8