NEC

User's Manual

78K/0S Series

8-Bit Single-Chip Microcontroller

Instructions

Common to 78K/0S Series

Document No. U11047EJ3V0UMJ1 (3rd edition)
Date Published November 2000 N CP(K)

© NEC Corporation 1996
Printed in Japan

[MEMO]

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.

EEPROM is a trademark of NEC Corporation.

The export of these products from Japan is regulated by the Japanese government. The export of some or all of these products may be prohibited without governmental license. To export or re-export some or all of these products from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

The following products are manufactured and sold based on a license contract with CP8 Transac regarding the EEPROM microcontroller patent.

These products cannot be used for an IC card (SMART CARD).

Applicable products: μPD789146, 789156, 789197AY, 789217AY Subseries

Purchase of NEC I^2C components conveys a license under the Philips I^2C Patent Rights to use these components in an I^2C system, provided that the system conforms to the I^2C Standard Specification as defined by Philips.

Applicable products: μPD789197AY, 789217AY Subseries

- **The information in this document is current as of March, 1999. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC's data sheets or data books, etc., for the most up-to-date specifications of NEC semiconductor products. Not all products and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without prior written consent of NEC. NEC assumes no responsibility for any errors that may appear in this document.
- NEC does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC semiconductor products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC endeavours to enhance the quality, reliability and safety of NEC semiconductor products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC semiconductor products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment, and anti-failure features.
- NEC semiconductor products are classified into the following three quality grades:
  "Standard", "Special" and "Specific". The "Specific" quality grade applies only to semiconductor products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of a semiconductor product depend on its quality grade, as indicated below. Customers must check the quality grade of each semiconductor product 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": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.
  The quality grade of NEC semiconductor products is "Standard" unless otherwise expressly specified in NEC's data sheets or data books, etc. If customers wish to use NEC semiconductor products in applications not intended by NEC, they must contact an NEC sales representative in advance to determine NEC's willingness to support a given application.
  (Note)
  (1) "NEC" as used in this statement means NEC Corporation and also includes its majority-owned subsidiaries.
  (2) "NEC semiconductor products" means any semiconductor product developed or manufactured by or for NEC (as defined above).

M8E 00.4

Regional Information

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

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

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

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

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

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

NEC Electronics Italiana s.r.l.
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.
Madrid Office
Madrid, Spain
Tel: 91-504-2787
Fax: 91-504-2860

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
Tel: 65-253-8311
Fax: 65-250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377
Fax: 02-2719-5951

NEC do Brasil S.A.
Electron Devices Division
Guarulhos-SP Brasil
Tel: 55-11-6462-6810
Fax: 55-11-6462-6829

J00.7

MAJOR REVISIONS IN THIS EDITION

Page	Contents
Throughout	• Addition of the following target products μPD789046, 789104, 789114, 789124, 789134, 789146, 789156, 789167, 789177, 789197AY, 789217AY, 789407A, 789417A, and 789842 Subseries
	• Deletion of the following target products μPD789407, 789417, and 789806Y Subseries
p. 52	Modification of MOV PSW, #byte instruction code
p. 52	Modification of MOVW rp, AX instruction code
p. 54	Modification of XOR A, r instruction code
p. 54	Modification of CMP A, r instruction code

The mark ★ shows major revised points.

INTRODUCTION

Readers

This manual is intended for users who wish to understand the functions of 78K/0S Series products and to design and develop its application systems and programs.

78K/0S Series products

- μPD789014 Subseries: μPD789011, 789012, 78P9014
- μPD789026 Subseries: μPD789022, 789024, 789025, 789026, 78F9026
- μPD789046 Subseries Note: μPD789046, 78F9046
- μPD789104 Subseries: μPD789101, 789102, 789104
- μPD789114 Subseries: μPD789111, 789112, 789114, 78F9116
- μPD789124 Subseries Note: μPD789121, 789122, 789124
- μPD789134 Subseries Note: μPD789131, 789132, 789134, 78F9136
- μPD789146 Subseries Note: μPD789144, 789146
- μPD789156 Subseries Note: μPD789154, 789156, 78F9156
- μPD789167 Subseries Note: μPD789166, 789167
- μPD789177 Subseries Note: μPD789176, 789177, 78F9177
- μPD789197AY Subseries Note: μPD789196AY, 789197AY, 78F9197AY
- μPD789217AY Subseries Note: μPD789216AY, 789217AY, 78F9217AY
- μPD789407A Subseries: μPD789405A, 789406A, 789407A
- μPD789417A Subseries: μPD789415A, 789416A, 789417A, 78F9418A
- μPD789800 Subseries: μPD789800, 78F9801
- μPD789842 Subseries Note: μPD789841, 789842, 78F9842

Note Under development

Purpose

This manual is intended for users to understand the instruction functions of 78K/0S Series products.

Organization

The contents of this manual are broadly divided into the following.

- CPU functions
- Instruction set
- Explanation of instructions

How to read this manual

It is assumed that the reader of this manual has general knowledge in the fields of electrical engineering, logic circuits, and microcontrollers.

- To check the details of the functions of an instruction whose mnemonic is known:
  → See **APPENDICES A** and **B INSTRUCTION INDEX**.
- To check an instruction whose mnemonic is not known but whose general function is known:
  → Check the mnemonic in **CHAPTER 4 INSTRUCTION SET**, then the functions in **CHAPTER 5 EXPLANATION OF INSTRUCTIONS**.
- To understand the overall functions of the 78K/0S Series products instructions in general:
  → Read this manual in the order of the **CONTENTS**.

- To learn the hardware functions of the 78K/0S Series products:
  → Refer to the user's manual for each product (see **Related documents**).

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numeral representation:	Binary.....×××× or ××××B
	Decimal.....××××
	Hexadecimal ....××××H

★ **Related Documents**

The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

○ **Document common to 78K/0S Series**

Document Name	Document Number	
	English	Japanese
User's Manual Instructions	This manual	U11047J

○ **Individual documents**

● **μPD789014 Subseries**

Document Name	Document Number	
	English	Japanese
μPD789011, 789012 Data Sheet	U11095E	U11095J
μPD78P9014 Data Sheet	U10912E	U10912J
μPD789014 Subseries User's Manual	U11187E	U11187J

● **μPD789026 Subseries**

Document Name	Document Number	
	English	Japanese
μPD789022, 789024, 789025, 789026 Data Sheet	U11715E	U11715J
μPD78F9026 Data Sheet	U11858E	U11858J
μPD789026 Subseries User's Manual	U11919E	U11919J

● **μPD789046 Subseries**

Document Name	Document Number	
	English	Japanese
μPD789046 Preliminary Product Information	U13380E	U13380J
μPD78F9046 Preliminary Product Information	U13546E	U13546J
μPD789046 Subseries User's Manual	U13600E	U13600J

• **μPD789104 Subseries**

Document Name	Document Number	
	English	Japanese
μPD789101, 789102, 789104 Data Sheet	To be prepared	U12815J
μPD789134 Subseries User's Manual	U13045E	U13045J

• **μPD789114 Subseries**

Document Name	Document Number	
	English	Japanese
μPD789111, 789112, 789114 Preliminary Product Information	U13013E	U13013J
μPD78F9116 Preliminary Product Information	U13037E	U13037J
μPD789134 Subseries User's Manual	U13045E	U13045J

• **μPD789124 Subseries**

Document Name	Document Number	
	English	Japanese
μPD789121, 789122, 789124 Preliminary Product Information	U13025E	U13025J
μPD789134 Subseries User's Manual	U13045E	U13045J

• **μPD789134 Subseries**

Document Name	Document Number	
	English	Japanese
μPD789131, 789132, 789134 Preliminary Product Information	U13015E	U13015J
μPD78F9136 Preliminary Product Information	U13036E	U13036J
μPD789134 Subseries User's Manual	U13045E	U13045J

• **μPD789146, 789156 Subseries**

Document Name	Document Number	
	English	Japanese
μPD789144, 789146, 789154, 789156 Preliminary Product Information	U13478E	U13478J
μPD78F9156 Preliminary Product Information	To be prepared	U13756J
μPD789146, 789156 Subseries User's Manual	U13651E	U13651J

• **μPD789167, 789177 Subseries**

Document Name	Document Number	
	English	Japanese
μPD789166, 789167, 789176, 789177 Preliminary Product Information	To be prepared	U14017J
μPD78F9177 Preliminary Product Information	To be prepared	U14022J
μPD789177 Subseries User's Manual	To be prepared	To be prepared

• **μPD789197AY Subseries**

Document Name	Document Number	
	English	Japanese
μPD789196AY, 789197AY Preliminary Product Information	U13853E	U13853J
μPD78F9197Y Preliminary Product Information	U13224E	U13224J
μPD789217Y Subseries User's Manual	U13186E	U13186J

• **μPD789217AY Subseries**

Document Name	Document Number	
	English	Japanese
μPD789216Y, 789217Y Preliminary Product Information	U13196E	U13196J
μPD78F9217Y Preliminary Product Information	U13205E	U13205J
μPD789217Y Subseries User's Manual	U13186E	U13186J

• **μPD789407A, 789417A Subseries**

Document Name	Document Number	
	English	Japanese
μPD789405A, 789406A, 789407A, 789415A, 789416A, 789417A Data Sheet	To be prepared	U14024J
μPD78F9418A Data Sheet	To be prepared	To be prepared
μPD789407A, 789417A Subseries User's Manual	To be prepared	U13952J

• **μPD789800 Subseries**

Document Name	Document Number	
	English	Japanese
μPD789800 Data Sheet	U12627E	U12627J
μPD78F9801 Preliminary Product Information	U12626E	U12626J
μPD789800 Subseries User's Manual	U12978E	U12978J

• **μPD789842 Subseries**

Document Name	Document Number	
	English	Japanese
μPD789841, 789842 Preliminary Product Information	U13790E	U13790J
μPD78F9842 Preliminary Product Information	U13901E	U13901J
μPD789842 Subseries User's Manual	U13776E	U13776J

Caution The above documents are subject to change without prior notice. Be sure to use the latest version document when starting design.

CONTENTS

CHAPTER 1 MEMORY SPACE 15

1.1 Memory Space 15
1.2 Internal Program Memory (Internal ROM) Space 15
1.3 Vector Table Area 17
1.4 CALLT Instruction Table Area 20
1.5 Internal Data Memory Space 20
1.6 Special Function Register (SFR) Area 22

CHAPTER 2 REGISTERS 23

2.1 Control Registers 23
2.1.1 Program counter (PC) 23
2.1.2 Program status word (PSW) 23
2.1.3 Stack pointer (SP) 24
2.2 General-Purpose Registers 25
2.3 Special Function Registers (SFRs) 27

CHAPTER 3 ADDRESSING 29

3.1 Addressing of Instruction Address 29
3.1.1 Relative addressing 29
3.1.2 Immediate addressing 30
3.1.3 Table indirect addressing 31
3.1.4 Register addressing 32
3.2 Addressing of Operand Address 33
3.2.1 Direct addressing 33
3.2.2 Short direct addressing 34
3.2.3 Special function register (SFR) addressing 35
3.2.4 Register addressing 36
3.2.5 Register indirect addressing 37
3.2.6 Based addressing 38
3.2.7 Stack addressing 38

CHAPTER 4 INSTRUCTION SET 39

4.1 Operation 40
4.1.1 Operand representation and description formats 40
4.1.2 Description of operation column 41
4.1.3 Description of flag column 41
4.1.4 Description of clock column 42
4.1.5 Operation list 43
4.1.6 Instruction list by addressing 48
4.2 Instruction Codes 51
4.2.1 Description of instruction code table 51
4.2.2 Instruction code list 52

CHAPTER 5 EXPLANATION OF INSTRUCTIONS 57

5.1 8-Bit Data Transfer Instructions 59
5.2 16-Bit Data Transfer Instructions 62
5.3 8-Bit Operation Instructions 65
5.4 16-Bit Operation Instructions 74
5.5 Increment/Decrement Instructions 78
5.6 Rotate Instructions 83
5.7 Bit Manipulation Instructions 88
5.8 CALL/RETURN Instructions 92
5.9 Stack Manipulation Instructions 97
5.10 Unconditional Branch Instruction 101
5.11 Conditional Branch Instructions 103
5.12 CPU Control Instructions 111

APPENDIX A INSTRUCTION INDEX (MNEMONIC: BY FUNCTION) 117

APPENDIX B INSTRUCTION INDEX (MNEMONIC: IN ALPHABETICAL ORDER) 119

APPENDIX C REVISION HISTORY 121

LIST OF FIGURES

Figure No.	Title	Page
2-1.	Format of Program Counter	23
2-2.	Format of Program Status Word	23
2-3.	Format of Stack Pointer	24
2-4.	Data to Be Saved to Stack Memory	25
2-5.	Data to Be Restored from Stack Memory	25
2-6.	General-Purpose Register Configuration	26

LIST OF TABLES

Table No.	Title	Page
1-1.	Internal ROM Space of 78K/0S Series Products	15
1-2.	Vector Table (0000H to 0013H) (μPD789014 Subseries)	17
1-3.	Vector Table (0000H to 002BH) (μPD789026 Subseries)	17
1-4.	Vector Table (0000H to 0019H) (μPD789046 Subseries)	17
1-5.	Vector Table (0000H to 0015H) (μPD789104, 789114, 789124, 789134 Subseries)	17
1-6.	Vector Table (0000H to 0019H) (μPD789146, 789156 Subseries)	18
1-7.	Vector Table (0000H to 0023H) (μPD789167, 789177 Subseries)	18
1-8.	Vector Table (0000H to 0027H) (μPD789197AY, 789217AY Subseries)	18
1-9.	Vector Table (0000H to 0023H) (μPD789407A and μPD789417A Subseries)	19
1-10.	Vector Table (0000H to 0019H) (μPD789800 Subseries)	19
1-11.	Vector Table (0000H to 0023H) (μPD789842 Subseries)	19
1-12.	Internal Data Memory Space of 78K/0S Series Products	20
4-1.	Operand Representation and Description Formats	40

[MEMO]

CHAPTER 1 MEMORY SPACE

1.1 Memory Space

The 78K/0S Series product program memory map varies depending on the internal memory capacity. For details of the memory mapped address area, refer to the User's Manual of each product.

1.2 Internal Program Memory (Internal ROM) Space

The 78K/0S Series product has internal ROM in the address space shown below. Program and table data, etc. are stored in ROM. This memory space is usually addressed by the program counter (PC).

★ **Table 1-1. Internal ROM Space of 78K/0S Series Products (1/2)**

Capacity / Address Space / Subseries Name	2 Kbytes	4 Kbytes	8 Kbytes	12 Kbytes	16 Kbytes	24 Kbytes	32 Kbytes
	0000H to 07FFH	0000H to 0FFFH	0000H to 1FFFH	0000H to 2FFFH	0000H to 3FFFH	0000H to 5FFFH	0000H to 7FFFH
μPD789014 Subseries	μPD789011	μPD789012	μPD78P9014				
μPD789026 Subseries		μPD789022	μPD789024	μPD789025	μPD789026 μPD78F9026		
μPD789046 Subseries					μPD789046 μPD78F9046		
μPD789104 Subseries	μPD789101	μPD789102	μPD789104				
μPD789114 Subseries	μPD789111	μPD789112	μPD789114		μPD78F9116		
μPD789124 Subseries	μPD789121	μPD789122	μPD789124				
μPD789134 Subseries	μPD789131	μPD789132	μPD789134		μPD78F9136		
μPD789146 Subseries			μPD789144		μPD789146		
μPD789156 Subseries			μPD789154		μPD789156 μPD78F9156		
μPD789167 Subseries					μPD789166	μPD789167	
μPD789177 Subseries					μPD789176	μPD789177 μPD78F9177	
μPD789197AY Subseries					μPD789196AY	μPD789197AY μPD78F9197AY	

★ **Table 1-1. Internal ROM Space of 78K/0S Series Products (2/2)**

Capacity / Address Space / Subseries Name	2 Kbytes	4 Kbytes	8 Kbytes	12 Kbytes	16 Kbytes	24 Kbytes	32 Kbytes
	0000H to 07FFH	0000H to 0FFFH	0000H to 1FFFH	0000H to 2FFFH	0000H to 3FFFH	0000H to 5FFFH	0000H to 7FFFH
μPD789217AY Subseries					μPD789216AY	μPD789217AY μPD78F9217AY	
μPD789407A Subseries				μPD789405A	μPD789406A	μPD789407A	
μPD789417A Subseries				μPD789415A	μPD789416A	μPD789417A	μPD78F9418A
μPD789800 Subseries			μPD789800		μPD78F9801		
μPD789842 Subseries			μPD789841		μPD789842 μPD78F9842		

1.3 Vector Table Area

The vector table area stores program start addresses to which execution branches when the $\overline{\text{RESET}}$ signal is input or when an interrupt request is generated. Of the 16-bit address, the lower 8 bits are stored in an even address, and the higher 8 bits are stored in an odd address.

Table 1-2. Vector Table (0000H to 0013H) (μPD789014 Subseries)

Vector Table Address	Interrupt Request	Vector Table Address	Interrupt Request
0000H	$\overline{\text{RESET}}$ input	000CH	INTSR/INTCSI0
0004H	INTWDT	000EH	INTST
0006H	INTP0	0010H	INTTM0
0008H	INTP1	0012H	INTTM1
000AH	INTP2		

Table 1-3. Vector Table (0000H to 002BH) (μPD789026 Subseries)

Vector Table Address	Interrupt Request	Vector Table Address	Interrupt Request
0000H	$\overline{\text{RESET}}$ input	000CH	INTSR/INTCSI0
0004H	INTWDT	000EH	INTST
0006H	INTP0	0010H	INTTM0
0008H	INTP1	0014H	INTTM2
000AH	INTP2	002AH	INTKR

★ **Table 1-4. Vector Table (0000H to 0019H) (μPD789046 Subseries)**

Vector Table Address	Interrupt Request	Vector Table Address	Interrupt Request
0000H	$\overline{\text{RESET}}$ input	000EH	INTST20
0004H	INTWDT	0010H	INTWT
0006H	INTP0	0012H	INTWTI
0008H	INTP1	0014H	INTTM80
000AH	INTP2	0016H	INTTM90
000CH	INTSR20/INTCSI20	0018H	INTKR00

★ **Table 1-5. Vector Table (0000H to 0015H) (μPD789104, 789114, 789124, 789134 Subseries)**

Vector Table Address	Interrupt Request	Vector Table Address	Interrupt Request
0000H	$\overline{\text{RESET}}$ input	000CH	INTSR20/INTCSI20
0004H	INTWDT	000EH	INTST20
0006H	INTP0	0010H	INTTM80
0008H	INTP1	0012H	INTTM20
000AH	INTP2	0014H	INTAD0

★ **Table 1-6. Vector Table (0000H to 0019H) (μPD789146, 789156 Subseries)**

Vector Table Address	Interrupt Request	Vector Table Address	Interrupt Request
0000H	$\overline{\text{RESET}}$ input	000EH	INTST20
0004H	INTWDT	0010H	INTTM80
0006H	INTP0	0012H	INTTM20
0008H	INTP1	0014H	INTAD0
000AH	INTP2	0016H	INTLVI0
000CH	INTSR20/INTCSI20	0018H	INTEE1

★ **Table 1-7. Vector Table (0000H to 0023H) (μPD789167, 789177 Subseries)**

Vector Table Address	Interrupt Request	Vector Table Address	Interrupt Request
0000H	$\overline{\text{RESET}}$ input	0012H	INTWT
0004H	INTWDT	0014H	INTWTI
0006H	INTP0	0016H	INTTM80
0008H	INTP1	0018H	INTTM81
000AH	INTP2	001AH	INTTM82
000CH	INTP3	001CH	INTTM90
000EH	INTSR20/INTCSI20	0022H	INTAD0
0010H	INTST20		

★ **Table 1-8. Vector Table (0000H to 0027H) (μPD789197AY, 789217AY Subseries)**

Vector Table Address	Interrupt Request	Vector Table Address	Interrupt Request
0000H	$\overline{\text{RESET}}$ input	0016H	INTTM80
0004H	INTWDT	0018H	INTTM81
0006H	INTP0	001AH	INTTM82
0008H	INTP1	001CH	INTTM90
000AH	INTP2	001EH	INTSMB0
000CH	INTP3	0020H	INTSMBOV0
000EH	INTSR20/INTCSI20	0022H	INTAD0
0010H	INTST20	0024H	INTLVI0
0012H	INTWT	0026H	INTEE1
0014H	INTWTI		

★ **Table 1-9. Vector Table (0000H to 0023H) (μPD789407A and μPD789417A Subseries)**

Vector Table Address	Interrupt Request	Vector Table Address	Interrupt Request
0000H	$\overline{\text{RESET}}$ input	0014H	INTWTI
0004H	INTWDT	0016H	INTTM00
0006H	INTP0	0018H	INTTM01
0008H	INTP1	001AH	INTTM02
000AH	INTP2	001CH	INTTM50
000CH	INTP3	001EH	INTKR00
000EH	INTSR00/INTCSI00	0020H	INTAD0
0010H	INTST00	0022H	INTCMP0
0012H	INTWT		

Table 1-10. Vector Table (0000H to 0019H) (μPD789800 Subseries)

Vector Table Address	Interrupt Request	Vector Table Address	Interrupt Request
0000H	$\overline{\text{RESET}}$ input	000EH	INTUSBRE
0004H	INTWDT	0010H	INTP0
0006H	INTUSBTM	0012H	INTCSI10
0008H	INTUSBRT	0014H	INTTM00
000AH	INTUSBRD	0016H	INTTM01
000CH	INTUSBST	0018H	INTKR00

★ **Table 1-11. Vector Table (0000H to 0023H) (μPD789842 Subseries)**

Vector Table Address	Interrupt Request	Vector Table Address	Interrupt Request
0000H	$\overline{\text{RESET}}$ input	0016H	INTST00
0004H	INTWDT	0018H	INTWT
0006H	INTP0	001AH	INTWTI
0008H	INTP1	001CH	INTTM80
000AH	INTTM7	001EH	INTTM81
000CH	INTSER00	0020H	INTTM82
000EH	INTSR00	0022H	INTAD

1.4 CALLT Instruction Table Area

In a 64-byte address area 0040H to 007FH, the subroutine entry address of a 1-byte call instruction (CALLT) can be stored.

1.5 Internal Data Memory Space

The 78K/0S Series products incorporate the following data memory:

(1) Internal high-speed RAM

The 78K/0S Series products incorporate internal high-speed RAM in the address space shown in Table 1-12.
The internal high-speed RAM is also used as a stack memory.

(2) LCD display RAM (μPD789407A and μPD789417A Subseries)

LCD display RAM is allocated in the area between FA00H and FA1BH.
The LCD display RAM can also be used as ordinary RAM.

(3) EEPROM™ (μPD789146, 789156, 789197AY, 789217AY Subseries)

Electrically erasable PROM (EEPROM) is allocated in the address space shown in Table 1-12.
Unlike ordinary RAM, EEPROM retains the data it contains even when the power is turned off. Also, unlike EPROM, the contents of EEPROM can be erased electrically, without the need to expose the chip to ultraviolet light.

★ **Table 1-12. Internal Data Memory Space of 78K/0S Series Products (1/2)**

Subseries Name	Product Name	High-Speed RAM	LCD Display RAM	EEPROM
μPD789014 Subseries	μPD789011	FE80H to FEFFH (128 bytes)	—	—
	μPD789012			
	μPD78P9014	FE00H to FEFFH (256 bytes)		
μPD789026 Subseries	μPD789022	FE00H to FEFFH (256 bytes)	—	—
	μPD789024			
	μPD789025	FD00H to FEFFH (512 bytes)		
	μPD789026			
	μPD78F9026			
μPD789046 Subseries	μPD789046	FD00H to FEFFH (512 bytes)	—	—
	μPD78F9046			
μPD789104 Subseries	μPD789101	FE00H to FEFFH (256 bytes)	—	—
	μPD789102			
	μPD789104			
μPD789114 Subseries	μPD789111	FE00H to FEFFH (256 bytes)	—	—
	μPD789112			
	μPD789114			
	μPD78F9116			

★ **Table 1-12. Internal Data Memory Space of 78K/0S Series Products (2/2)**

Subseries Name	Product Name	High-Speed RAM	LCD Display RAM	EEPROM
μPD789124 Subseries	μPD789121	FE00H to FEFFH (256 bytes)	—	—
	μPD789122			
	μPD789124			
μPD789134 Subseries	μPD789131	FE00H to FEFFH (256 bytes)	—	—
	μPD789132			
	μPD789134			
	μPD78F9136			
μPD789146 Subseries	μPD789144	FE00H to FEFFH (256 bytes)	—	F800H to F8FFH (256 bytes)
	μPD789146			
μPD789156 Subseries	μPD789154	FE00H to FEFFH (256 bytes)	—	F800H to F8FFH (256 bytes)
	μPD789156			
	μPD78F9156			
μPD789167 Subseries	μPD789166	FD00H to FEFFH (512 bytes)	—	—
	μPD789167			
μPD789177 Subseries	μPD789176	FD00H to FEFFH (512 bytes)	—	—
	μPD789177			
	μPD78F9177			
μPD789197AY Subseries	μPD789196AY	FD00H to FEFFH (512 bytes)	—	F800H to F87FH (128 bytes)
	μPD789197AY			
	μPD78F9197AY			
μPD789217AY Subseries	μPD789216AY	FD00H to FEFFH (512 bytes)	—	F800H to F87FH (128 bytes)
	μPD789217AY			
	μPD78F9217AY			
μPD789407A Subseries	μPD789405A	FD00H to FEFFH (512 bytes)	FA00H to FA1BH (28 bytes)	—
	μPD789406A			
	μPD789407A			
μPD789417A Subseries	μPD789415A	FD00H to FEFFH (512 bytes)	FA00H to FA1BH (28 bytes)	—
	μPD789416A			
	μPD789417A			
	μPD78F9418A			
μPD789800 Subseries	μPD789800	FE00H to FEFFH (256 bytes)	—	—
	μPD78F9801			
μPD789842 Subseries	μPD789841	FE00H to FEFFH (256 bytes)	—	—
	μPD789842			
	μPD78F9842			

1.6 Special Function Register (SFR) Area

Special-function registers (SFRs) of on-chip peripheral hardware are allocated to the area FF00H to FFFFH (refer to the User's Manual of each product).

CHAPTER 2 REGISTERS

2.1 Control Registers

The control registers have dedicated functions such as controlling the program sequence, statuses, and stack memory. The control registers include a program counter, program status word, and stack pointer.

2.1.1 Program counter (PC)

The program counter is a 16-bit register that holds the address information of the next program to be executed.

In normal operation, the PC is automatically incremented according to the number of bytes of the instruction to be fetched. When a branch instruction is executed, immediate data and register contents are set.

When the $\overline{\text{RESET}}$ signal is input, the program counter is set to the value of the reset vector table, which are located at addresses 0000H and 0001H.

Figure 2-1. Format of Program Counter

	15															0
PC	PC15	PC14	PC13	PC12	PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0

2.1.2 Program status word (PSW)

Program status word is an 8-bit register consisting of various flags to be set/reset by instruction execution.

The contents of program status word are automatically stacked when an interrupt request is generated or when the PUSH PSW instruction is executed and, are automatically reset when the RETI and POP PSW instruction are executed.

$\overline{\text{RESET}}$ input sets PSW to 02H.

Figure 2-2. Format of Program Status Word

7							0
IE	Z	0	AC	0	0	1	CY

(1) Interrupt enable flag (IE)

This flag controls interrupt request acknowledge operations of the CPU.

When IE = 0, all interrupts except non-maskable interrupts are disabled (DI status).

When IE = 1, interrupts are enabled (EI status). At this time, acknowledgment of interrupt requests is controlled by the interrupt mask flag for each interrupt source.

The IE flag is reset (0) when the DI instruction execution is executed or when an interrupt is acknowledged, and set (1) when the EI instruction is executed.

(2) Zero flag (Z)

When the operation result is zero, this flag is set (1); otherwise, it is reset (0).

(3) Auxiliary carry flag (AC)

If the operation result has a carry from bit 3 or a borrow to bit 3, this flag is set (1); otherwise, it is reset (0).

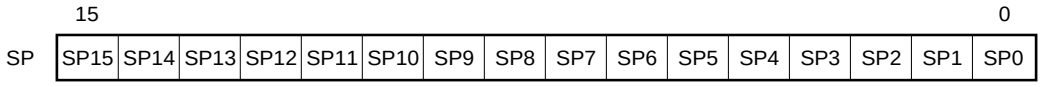
(4) Carry flag (CY)

This flag records an overflow or underflow upon add/subtract instruction execution. It also records the shift-out value upon rotate instruction execution, and functions as a bit accumulator during bit operation instruction execution.

2.1.3 Stack pointer (SP)

This is a 16-bit register that holds the first address of the stack area in the memory. Only the internal high-speed RAM area can be set as the stack area.

Figure 2-3. Format of Stack Pointer



The SP is decremented ahead of write (save) to the stack memory, and is incremented after read (reset) from the stack memory.

The data saved/restored as a result of each stack operation are as shown in Figures 2-4 and 2-5.

Caution Since $\overline{\text{RESET}}$ input makes the SP contents undefined, be sure to initialize the SP before executing an instruction.

Figure 2-4. Data to Be Saved to Stack Memory

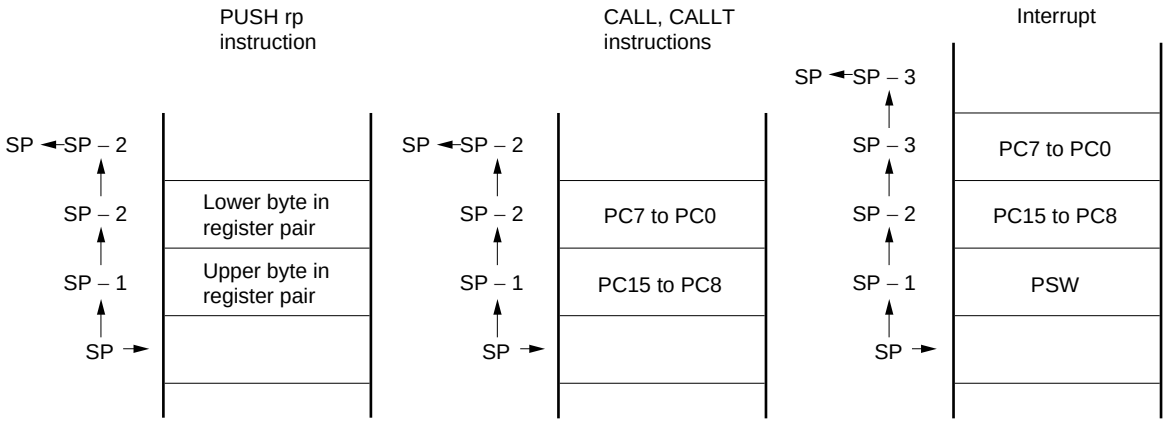
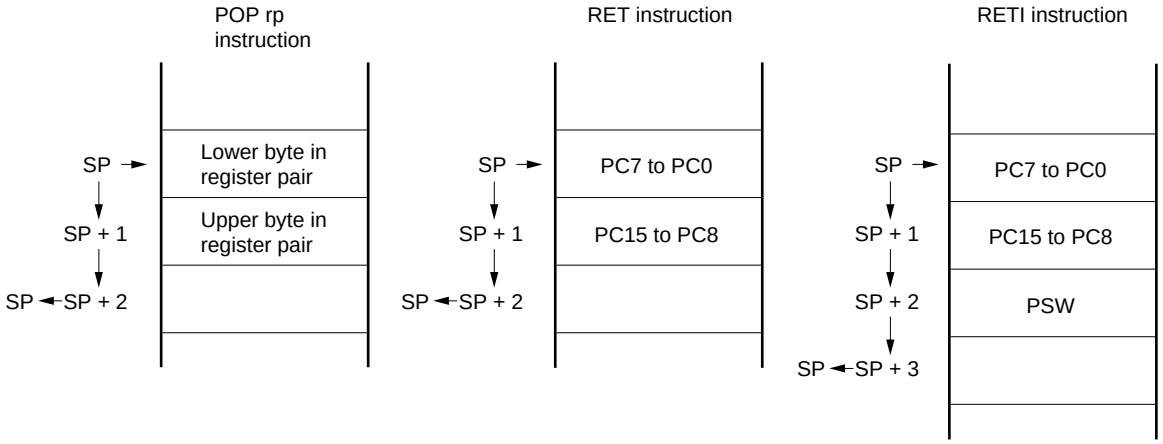


Figure 2-5. Data to Be Restored from Stack Memory



2.2 General-Purpose Registers

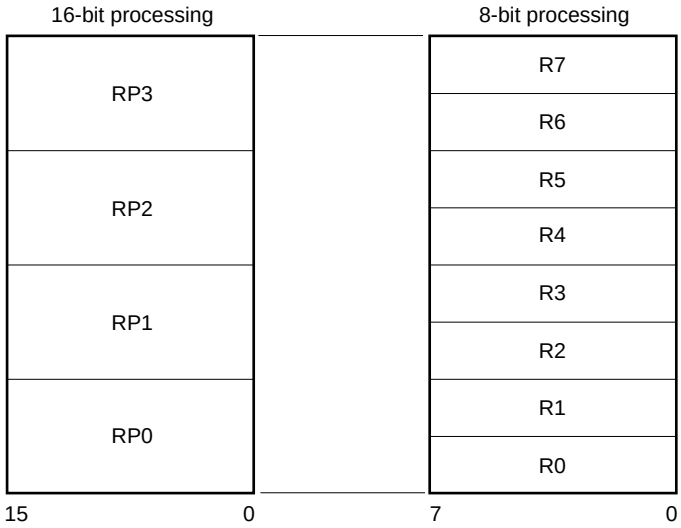
The general-purpose register consists of eight 8-bit registers (X, A, C, B, E, D, L, and H).

Each register can be used as an 8-bit register, or two 8-bit registers in pairs can be used as a 16-bit register (AX, BC, DE, and HL).

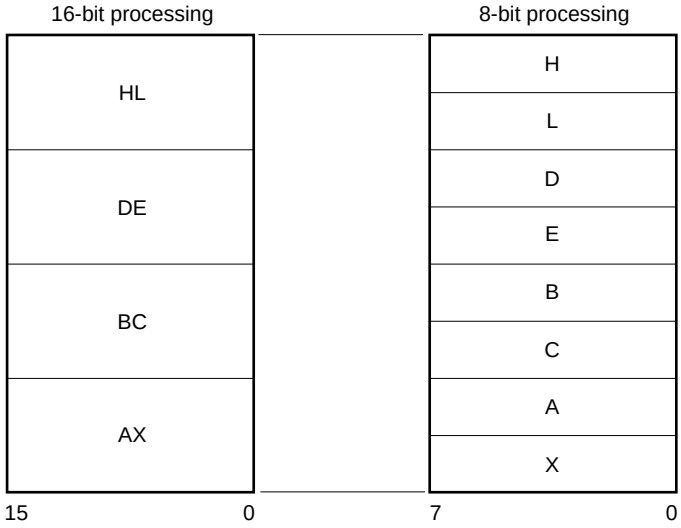
Registers can be described in terms of functional names (X, A, C, B, E, D, L, H, AX, BC, DE, and HL) and absolute names (R0 to R7 and RP0 to RP3).

Figure 2-6. General-Purpose Register Configuration

(a) Absolute name



(b) Functional name



2.3 Special Function Registers (SFRs)

Unlike general-purpose registers, special function registers have their own functions and are allocated to the 256-byte area FF00H to FFFFH.

A special function register can be manipulated, like a general-purpose register, by using operation, transfer, and bit manipulation instructions. The bit units in which one register is to be manipulated (1, 8, and 16) differ depending on the special function register type.

The bit unit for manipulation is specified as follows.

- 1-bit manipulation
  Describes a symbol reserved by the assembler for the 1-bit manipulation instruction operand (sfr.bit). This manipulation can also be specified with an address.

- 8-bit manipulation
  Describes a symbol reserved by the assembler for the 8-bit manipulation instruction operand (sfr). This manipulation can also be specified with an address.

- 16-bit manipulation
  Describes a symbol reserved by the assembler for the 16-bit manipulation instruction operand. When addressing an address, describe an even address.

For details of the special function registers, refer to the User's Manual of each product.

[MEMO]

CHAPTER 3 ADDRESSING

3.1 Addressing of Instruction Address

An instruction address is determined by the program counter (PC) contents. The PC contents are normally incremented (+1 per byte) automatically according to the number of bytes of an instruction to be fetched each time another instruction is executed. When a branch instruction is executed, the branch destination information is set in the PC and branched by the following addressing (For details of each instruction, see **CHAPTER 5 EXPLANATION OF INSTRUCTIONS**).

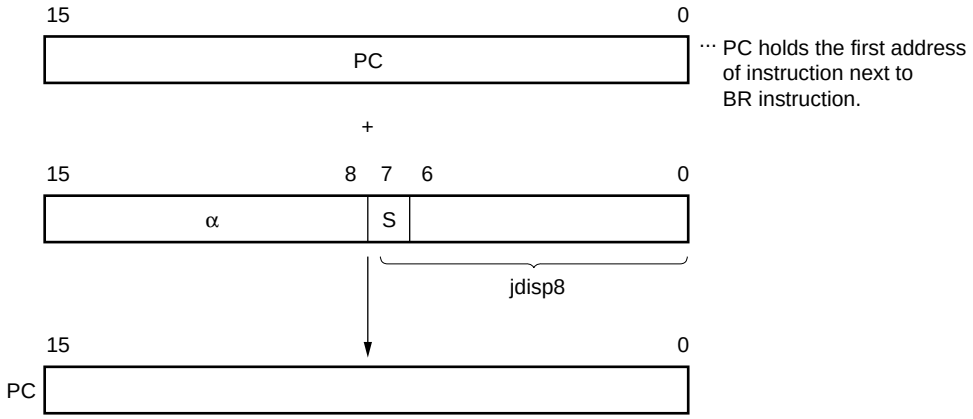
3.1.1 Relative addressing

[Function]

The value obtained by adding the 8-bit immediate data (displacement value: jdisp8) of an instruction code to the first address of the following instruction is transferred to the program counter (PC) and program branches. The displacement value is treated as signed two's complement data (–128 to +127) and bit 7 becomes a sign bit. Thus, relative addressing causes a branch to an address within the range of –128 to +127, relative to the first address of the next instruction.

This function is carried out when the BR $addr16 instruction or a conditional branch instruction is executed.

[Illustration]



When S = 0, all bits of α are 0.
When S = 1, all bits of α are 1.

3.1.2 Immediate addressing

[Function]

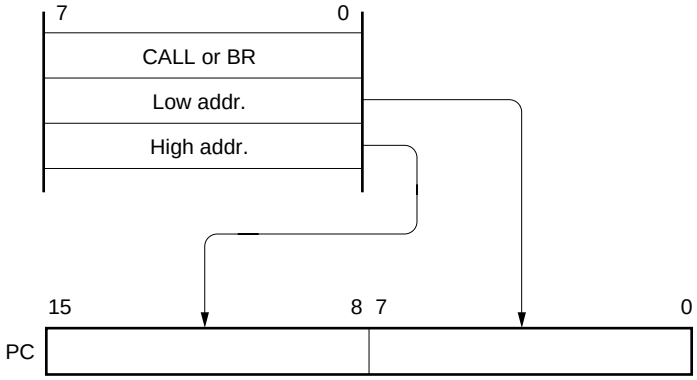
Immediate data in the instruction word is transferred to the program counter (PC) and program branches.

This function is carried out when the CALL !addr16 or BR !addr16 instruction is executed.

The CALL !addr16 and BR !addr16 instructions can be used to branch to any address within the memory spaces.

[Illustration]

In case of CALL !addr16 or BR !addr16 instruction



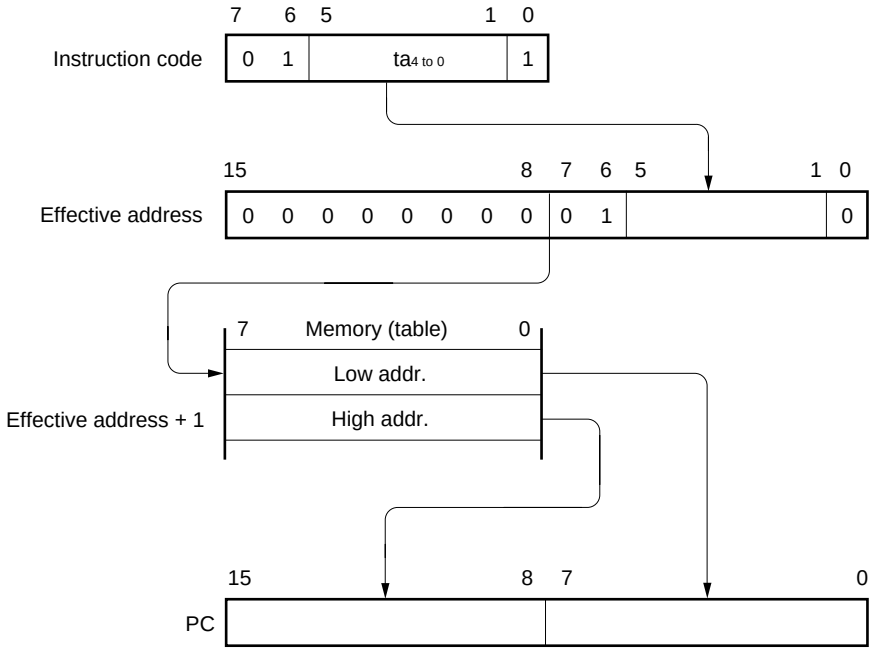
3.1.3 Table indirect addressing

[Function]

Table contents (branch destination address) of a particular location, addressed by the immediate data of bits 1 to 5 of an instruction code are transferred to the program counter (PC), and program branches.

Table indirect addressing is performed when the CALLT [addr5] instruction is executed. This instruction references the address stored in the memory table from 40H to 7FH, and allows branching to the entire memory space.

[Illustration]



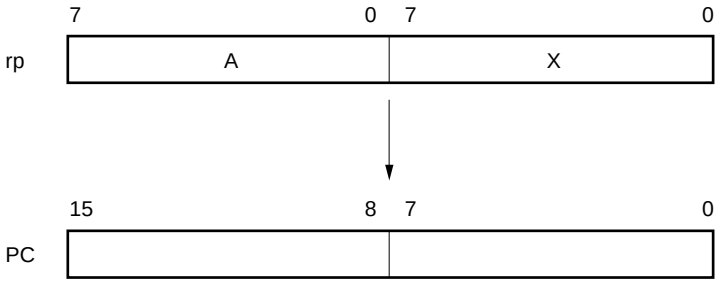
3.1.4 Register addressing

[Function]

Register pair (AX) contents specified with an instruction word are transferred to the program counter (PC) and program branches.

This function is carried out when the BR AX instruction is executed.

[Illustration]



3.2 Addressing of Operand Address

The following methods are available to specify the register and memory (addressing) which undergo manipulation during instruction execution.

3.2.1 Direct addressing

[Function]

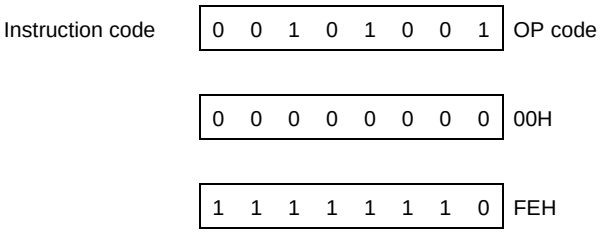
This addressing directly addresses a memory to be manipulated with immediate data in an instruction word.

[Operand format]

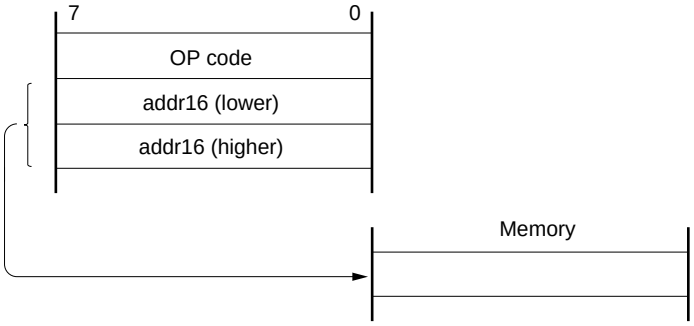
Operand	Description
addr16	Label or 16-bit immediate data

[Description example]

MOV A, !FE00H; When setting !addr16 to FE00H



[Illustration]



3.2.2 Short direct addressing

[Function]

This addressing directly addresses memory to be manipulated in the fixed space with the 8-bit data in an instruction word.

This addressing is applied to the 256-byte fixed space of FE20H to FF1FH. An internal high-speed RAM and special function registers (SFRs) are mapped at FE20H to FEFFH and FF00H to FF1FH, respectively.

The SFR area (FF00H-FF1FH) to which short direct addressing is applied constitutes only part of the overall SFR area. In this area, ports that are frequently accessed in a program and a compare register of the timer/event counter are mapped, and these SFRs can be manipulated with a small number of bytes and clocks.

When 8-bit immediate data is 20H to FFH, bit 8 of an effective address is set to 0. When it is 00H to 1FH, bit 8 is set to 1. See **Illustration** below.

[Operand format]

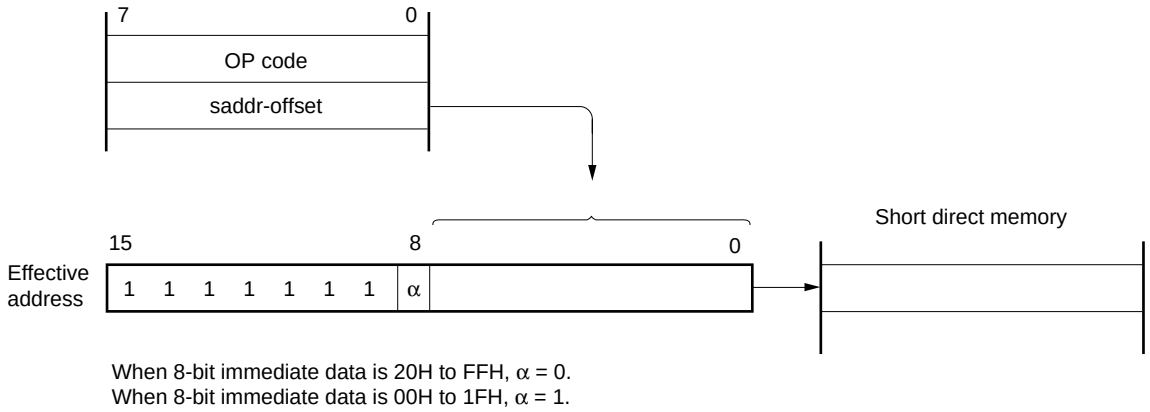
Operand	Description
saddr	Label or FE20H to FF1FH immediate data
saddrp	Label or FE20H to FF1FH immediate data (even address only)

[Description example]

MOV FE30H, #50H; When setting saddr to FE30H and the immediate data to 50H

Instruction code	1 1 1 1 0 1 0 1	OP code
	0 0 1 1 0 0 0 0	30H (saddr-offset)
	0 1 0 1 0 0 0 0	50H (immediate data)

[Illustration]



When 8-bit immediate data is 20H to FFH, α = 0.
When 8-bit immediate data is 00H to 1FH, α = 1.

3.2.3 Special function register (SFR) addressing

[Function]

This addressing is to address special function registers (SFRs) mapped to the memory with the 8-bit immediate data in an instruction word.

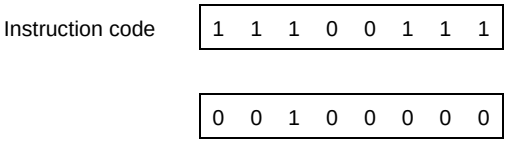
This addressing is applied to the 240-byte spaces of FF00H to FFCFH and FFE0H to FFFFH. However, the SFRs mapped at FF00H to FF1FH can also be accessed by means of short direct addressing.

[Operand format]

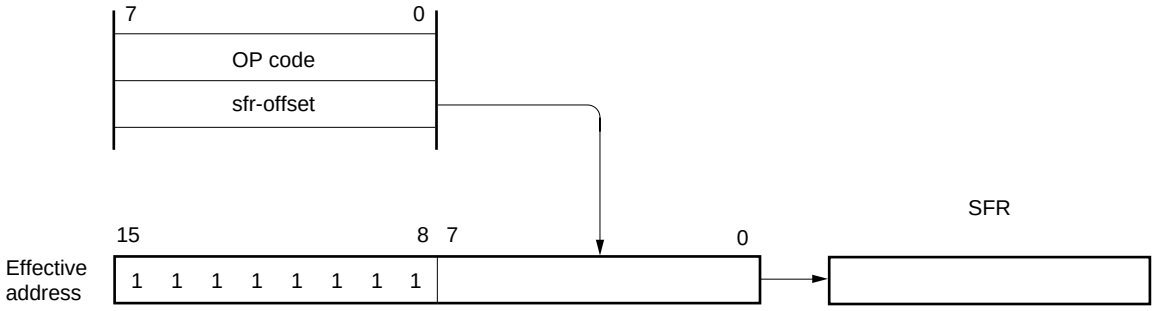
Operand	Description
sfr	Special function register name

[Description example]

MOV PM0, A; When selecting PM0 for sfr



[Illustration]



3.2.4 Register addressing

[Function]

This addressing is to access a general-purpose register by specifying it as an operand. The general-purpose register to be accessed is specified with a register specification code in an instruction code or function name.

Register addressing is carried out when an instruction with the following operand format is executed. When an 8-bit register is specified, one of the eight registers is specified with 3 bits (register specification code) in the instruction code.

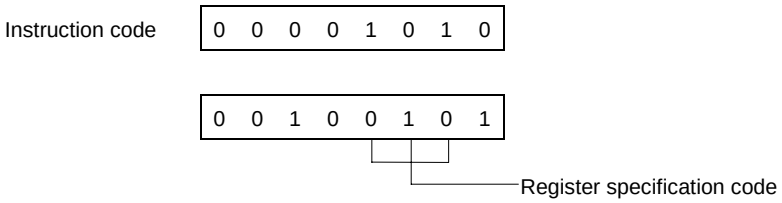
[Operand format]

Operand	Description
r	X, A, C, B, E, D, L, H
rp	AX, BC, DE, HL

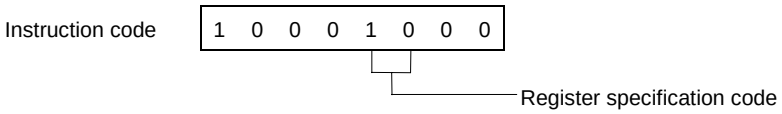
'r' and 'rp' can be described with absolute names (R0 to R7 and RP0 to RP3) as well as functional names (X, A, C, B, E, D, L, H, AX, BC, DE, and HL).

[Description example]

MOV A, C; When selecting the C register for r



INCW DE; When selecting the DE register pair for rp



3.2.5 Register indirect addressing

[Function]

This addressing is to address memory using the contents of the special register pair as an operand. The register pair to be accessed is specified with the register pair specification code in an instruction code. This addressing can be carried out for the entire memory space.

[Operand format]

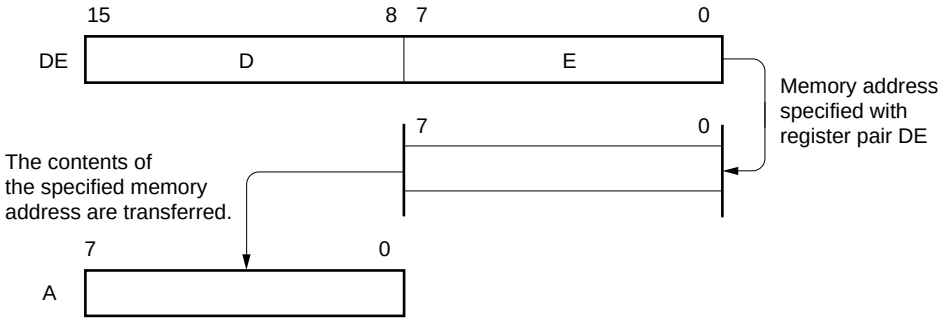
Operand	Description
—	[DE], [HL]

[Description example]

MOV A, [DE]; When selecting register pair [DE]

Instruction code	0	0	1	0	1	0	1	1

[Illustration]



3.2.6 Based addressing

[Function]

This addressing is to address the memory by using the result of adding 8-bit immediate data to the contents of the base register, i.e., the HL register pair. The addition is performed by expanding the offset data as a positive number to 16 bits. A carry from the 16th bit is ignored. This addressing can be carried out for the entire memory space.

[Operand format]

Operand	Description
—	[HL + byte]

[Description example]

MOV A, [HL+10H]; When setting “byte” to 10H

Instruction code	0	0	1	0	1	1	0	1
	0	0	0	1	0	0	0	0

3.2.7 Stack addressing

[Function]

This addressing is to indirectly address the stack area with the stack pointer (SP) contents.

This addressing method is automatically employed when the PUSH, POP, subroutine call, or RETURN instructions is executed or when the register is saved/restored upon generation of an interrupt request.

Stack addressing can address the internal high-speed RAM area only.

[Description example]

In the case of PUSH DE

Instruction code	1	0	1	0	1	0	1	0

CHAPTER 4 INSTRUCTION SET

This chapter lists the instruction set of the 78K/0S Series. The instructions are common to all 78K/0S Series products.

4.1 Operation

4.1.1 Operand representation and description formats

In the operand column of each instruction, an operand is described according to the description format for operand representation of that instruction (for details, refer to the assembler specifications). When there are two or more description methods, select one of them. Uppercase characters, #, !, $ and [ ] are keywords and must be described as is. Each symbol has the following meaning.

- # : Immediate data
- ! : Absolute address
- $: Relative address
- [ ] : Indirect address

In the case of immediate data, describe an appropriate numeric value or a label. When using a label, be sure to describe #, !, $, or [ ].

For operand register description formats, r and rp, either functional names (X, A, C, etc.) or absolute names (names in parentheses in the table below, R0, R1, R2, etc.) can be described.

Table 4-1. Operand Representation and Description Formats

Operand	Description Format
r rp sfr	X (R0), A (R1), C (R2), B (R3), E (R4), D (R5), L (R6), H (R7) AX (RP0), BC (RP1), DE (RP2), HL (RP3) Special function register symbol
saddr saddrp	FE20H to FF1FH Immediate data or labels FE20H to FF1FH Immediate data or labels (even addresses only)
addr16 addr5	0000H to FFFFH Immediate data or labels (only even addresses for 16-bit data transfer instructions) 0040H to 007FH Immediate data or labels (even addresses only)
word byte bit	16-bit immediate data or label 8-bit immediate data or label 3-bit immediate data or label

Remark Refer to the User's Manual of each product for symbols of special function registers.

4.1.2 Description of operation column

A:	A register; 8-bit accumulator
X:	X register
B:	B register
C:	C register
D:	D register
E:	E register
H:	H register
L:	L register
AX:	AX register pair; 16-bit accumulator
BC:	BC register pair
DE:	DE register pair
HL:	HL register pair
PC:	Program counter
SP:	Stack pointer
PSW:	Program status word
CY:	Carry flag
AC:	Auxiliary carry flag
Z:	Zero flag
IE:	Interrupt request enable flag
NMIS:	Non-maskable interrupt servicing flag
( ):	Memory contents indicated by address or register contents in parentheses
X_H, X_L:	Higher 8 bits and lower 8 bits of 16-bit register
$\wedge$:	Logical product (AND)
$\vee$:	Logical sum (OR)
$\veebar$:	Exclusive logical sum (exclusive OR)
$\overline{\quad}$:	Inverted data
addr16:	16-bit immediate data or label
jdisp8:	Signed 8-bit data (displacement value)

4.1.3 Description of flag column

(Blank):	Not affected
0:	Cleared to 0
1:	Set to 1
$\times$:	Set/cleared according to the result
R:	Previously saved value is restored

4.1.4 Description of clock column

The number of clock cycles during instruction execution is outlined as follows.

One instruction clock cycle is equal to one CPU clock cycle (f_{CPU}) selected by the processor clock control register (PCC).

The operation list is shown below.

4.1.5 Operation list

Mnemonic	Operand	Byte	Clock	Operation	Flag
					Z AC CY
MOV	r, #byte	3	6	r ← byte	
	saddr, #byte	3	6	(saddr) ← byte	
	sfr, #byte	3	6	sfr ← byte	
	A, r Note 1	2	4	A ← r	
	r, A Note 1	2	4	r ← A	
	A, saddr	2	4	A ← (saddr)	
	saddr, A	2	4	(saddr) ← A	
	A, sfr	2	4	A ← sfr	
	sfr, A	2	4	sfr ← A	
	A, !addr16	3	8	A ← (addr16)	
	!addr16, A	3	8	(addr16) ← A	
	PSW, #byte	3	6	PSW ← byte	× × ×
	A, PSW	2	4	A ← PSW	
	PSW, A	2	4	PSW ← A	× × ×
	A, [DE]	1	6	A ← (DE)	
	[DE], A	1	6	(DE) ← A	
	A, [HL]	1	6	A ← (HL)	
	[HL], A	1	6	(HL) ← A	
	A, [HL + byte]	2	6	A ← (HL + byte)	
	[HL + byte], A	2	6	(HL + byte) ← A	
XCH	A, X	1	4	A ↔ X	
	A, r Note 2	2	6	A ↔ r	
	A, saddr	2	6	A ↔ (saddr)	
	A, sfr	2	6	A ↔ sfr	
	A, [DE]	1	8	A ↔ (DE)	
	A, [HL]	1	8	A ↔ (HL)	
	A, [HL + byte]	2	8	A ↔ (HL + byte)	

Notes 1. Except r = A.

2. Except r = A, X.

Remark One instruction clock cycle is equal to one CPU clock (f_{CPU}) cycle selected by the processor clock control register (PCC).

Mnemonic	Operand	Byte	Clock	Operation	Flag		
					Z	AC	CY
MOVW	rp, #word	3	6	rp ← word			
	AX, saddrp	2	6	AX ← (saddrp)			
	saddrp, AX	2	8	(saddrp) ← AX			
	AX, rp Note	1	4	AX ← rp			
	rp, AX Note	1	4	rp ← AX			
XCHW	AX, rp Note	1	8	AX ↔ rp			
ADD	A, #byte	2	4	A, CY ← A + byte	×	×	×
	saddr, #byte	3	6	(saddr), CY ← (saddr) + byte	×	×	×
	A, r	2	4	A, CY ← A + r	×	×	×
	A, saddr	2	4	A, CY ← A + (saddr)	×	×	×
	A, !addr16	3	8	A, CY ← A + (addr16)	×	×	×
	A, [HL]	1	6	A, CY ← A + (HL)	×	×	×
	A, [HL + byte]	2	6	A, CY ← A + (HL + byte)	×	×	×
ADDC	A, #byte	2	4	A, CY ← A + byte + CY	×	×	×
	saddr, #byte	3	6	(saddr), CY ← (saddr) + byte + CY	×	×	×
	A, r	2	4	A, CY ← A + r + CY	×	×	×
	A, saddr	2	4	A, CY ← A + (saddr) + CY	×	×	×
	A, !addr16	3	8	A, CY ← A + (addr16) + CY	×	×	×
	A, [HL]	1	6	A, CY ← A + (HL) + CY	×	×	×
	A, [HL + byte]	2	6	A, CY ← A + (HL + byte) + CY	×	×	×
SUB	A, #byte	2	4	A, CY ← A – byte	×	×	×
	saddr, #byte	3	6	(saddr), CY ← (saddr) – byte	×	×	×
	A, r	2	4	A, CY ← A – r	×	×	×
	A, saddr	2	4	A, CY ← A – (saddr)	×	×	×
	A, !addr16	3	8	A, CY ← A – (addr16)	×	×	×
	A, [HL]	1	6	A, CY ← A – (HL)	×	×	×
	A, [HL + byte]	2	6	A, CY ← A – (HL + byte)	×	×	×

Note Only when rp = BC, DE, or HL.

Remark One instruction clock cycle is equal to one CPU clock (f_{CPU}) cycle selected by the processor clock control register (PCC).

Mnemonic	Operand	Byte	Clock	Operation	Flag		
					Z	AC	CY
SUBC	A, #byte	2	4	A, CY ← A – byte – CY	×	×	×
	saddr, #byte	3	6	(saddr), CY ← (saddr) – byte – CY	×	×	×
	A, r	2	4	A, CY ← A – r – CY	×	×	×
	A, saddr	2	4	A, CY ← A – (saddr) – CY	×	×	×
	A, !addr16	3	8	A, CY ← A – (addr16) – CY	×	×	×
	A, [HL]	1	6	A, CY ← A – (HL) – CY	×	×	×
	A, [HL + byte]	2	6	A, CY ← A – (HL + byte) – CY	×	×	×
AND	A, #byte	2	4	A ← A∧byte	×		
	saddr, #byte	3	6	(saddr) ← (saddr)∧byte	×		
	A, r	2	4	A ← A∧r	×		
	A, saddr	2	4	A ← A∧(saddr)	×		
	A, !addr16	3	8	A ← A∧(addr16)	×		
	A, [HL]	1	6	A ← A∧(HL)	×		
	A, [HL + byte]	2	6	A ← A∧(HL + byte)	×		
OR	A, #byte	2	4	A ← A∨byte	×		
	saddr, #byte	3	6	(saddr) ← (saddr)∨byte	×		
	A, r	2	4	A ← A∨r	×		
	A, saddr	2	4	A ← A∨(saddr)	×		
	A, !addr16	3	8	A ← A∨(addr16)	×		
	A, [HL]	1	6	A ← A∨(HL)	×		
	A, [HL + byte]	2	6	A ← A∨(HL + byte)	×		
XOR	A, #byte	2	4	A ← A$\veebar$byte	×		
	saddr, #byte	3	6	(saddr) ← (saddr)$\veebar$byte	×		
	A, r	2	4	A ← A$\veebar$r	×		
	A, saddr	2	4	A ← A$\veebar$(saddr)	×		
	A, !addr16	3	8	A ← A$\veebar$(addr16)	×		
	A, [HL]	1	6	A ← A$\veebar$(HL)	×		
	A, [HL + byte]	2	6	A ← A$\veebar$(HL + byte)	×		
CMP	A, #byte	2	4	A – byte	×	×	×
	saddr, #byte	3	6	(saddr) – byte	×	×	×
	A, r	2	4	A – r	×	×	×
	A, saddr	2	4	A – (saddr)	×	×	×
	A, !addr16	3	8	A – (addr16)	×	×	×
	A, [HL]	1	6	A – (HL)	×	×	×
	A, [HL + byte]	2	6	A – (HL + byte)	×	×	×

Remark One instruction clock cycle is equal to one CPU clock (f_{CPU}) cycle selected by the processor clock control register (PCC).

Mnemonic	Operand	Byte	Clock	Operation	Flag		
					Z	AC	CY
ADDW	AX, #word	3	6	AX, CY ← AX + word	×	×	×
SUBW	AX, #word	3	6	AX, CY ← AX – word	×	×	×
CMPW	AX, #word	3	6	AX – word	×	×	×
INC	r	2	4	r ← r + 1	×	×	
	saddr	2	4	(saddr) ← (saddr) + 1	×	×	
DEC	r	2	4	r ← r – 1	×	×	
	saddr	2	4	(saddr) ← (saddr) – 1	×	×	
INCW	rp	1	4	rp ← rp + 1			
DECW	rp	1	4	rp ← rp – 1			
ROR	A, 1	1	2	(CY, A_7 ← A_0, A_{m-1} ← A_m) × 1			×
ROL	A, 1	1	2	(CY, A_0 ← A_7, A_{m+1} ← A_m) × 1			×
RORC	A, 1	1	2	(CY ← A_0, A_7 ← CY, A_{m-1} ← A_m) × 1			×
ROLC	A, 1	1	2	(CY ← A_7, A_0 ← CY, A_{m+1} ← A_m) × 1			×
SET1	saddr.bit	3	6	(saddr.bit) ← 1			
	sfr.bit	3	6	sfr.bit ← 1			
	A.bit	2	4	A.bit ← 1			
	PSW.bit	3	6	PSW.bit ← 1	×	×	×
	[HL].bit	2	10	(HL).bit ← 1			
CLR1	saddr.bit	3	6	(saddr.bit) ← 0			
	sfr.bit	3	6	sfr.bit ← 0			
	A.bit	2	4	A.bit ← 0			
	PSW.bit	3	6	PSW.bit ← 0	×	×	×
	[HL].bit	2	10	(HL).bit ← 0			
SET1	CY	1	2	CY ← 1			1
CLR1	CY	1	2	CY ← 0			0
NOT1	CY	1	2	CY ← $\overline{CY}$			×
CALL	!addr16	3	6	(SP – 1) ← (PC + 3)$_H$, (SP – 2) ← (PC + 3)$_L$, PC ← addr16, SP ← SP – 2			
CALLT	[addr5]	1	8	(SP – 1) ← (PC + 1)$_H$, (SP – 2) ← (PC + 1)$_L$, PC_H ← (00000000, addr5 + 1), PC_L ← (00000000, addr5), SP ← SP – 2			

Remark One instruction clock cycle is equal to one CPU clock (f_{CPU}) cycle selected by the processor clock control register (PCC).

Mnemonic	Operand	Byte	Clock	Operation	Flag		
					Z	AC	CY
RET		1	6	$PC_H \leftarrow (SP + 1), PC_L \leftarrow (SP), SP \leftarrow SP + 2$			
RETI		1	8	$PC_H \leftarrow (SP + 1), PC_L \leftarrow (SP)$, $PSW \leftarrow (SP + 2), SP \leftarrow SP + 3, NMIS \leftarrow 0$	R	R	R
PUSH	PSW	1	2	$(SP - 1) \leftarrow PSW, SP \leftarrow SP - 1$			
	rp	1	4	$(SP - 1) \leftarrow rp_H, (SP - 2) \leftarrow rp_L, SP \leftarrow SP - 2$			
POP	PSW	1	4	$PSW \leftarrow (SP), SP \leftarrow SP + 1$	R	R	R
	rp	1	6	$rp_H \leftarrow (SP + 1), rp_L \leftarrow (SP), SP \leftarrow SP + 2$			
MOVW	SP,AX	2	8	$SP \leftarrow AX$			
	AX,SP	2	6	$AX \leftarrow SP$			
BR	!addr16	3	6	$PC \leftarrow$ addr16			
	$addr16	2	6	$PC \leftarrow PC + 2 +$ jdisp8			
	AX	1	6	$PC_H \leftarrow A, PC_L \leftarrow X$			
BC	$addr16	2	6	$PC \leftarrow PC + 2 +$ jdisp8 if CY = 1			
BNC	$addr16	2	6	$PC \leftarrow PC + 2 +$ jdisp8 if CY = 0			
BZ	$addr16	2	6	$PC \leftarrow PC + 2 +$ jdisp8 if Z = 1			
BNZ	$addr16	2	6	$PC \leftarrow PC + 2 +$ jdisp8 if Z = 0			
BT	saddr.bit, $addr16	4	10	$PC \leftarrow PC + 4 +$ jdisp8 if (saddr.bit) = 1			
	sfr.bit, $addr16	4	10	$PC \leftarrow PC + 4 +$ jdisp8 if sfr.bit = 1			
	A.bit, $addr16	3	8	$PC \leftarrow PC + 3 +$ jdisp8 if A.bit = 1			
	PSW.bit, $addr16	4	10	$PC \leftarrow PC + 4 +$ jdisp8 if PSW.bit = 1			
BF	saddr.bit, $addr16	4	10	$PC \leftarrow PC + 4 +$ jdisp8 if (saddr.bit) = 0			
	sfr.bit, $addr16	4	10	$PC \leftarrow PC + 4 +$ jdisp8 if sfr.bit = 0			
	A.bit, $addr16	3	8	$PC \leftarrow PC + 3 +$ jdisp8 if A.bit = 0			
	PSW.bit, $addr16	4	10	$PC \leftarrow PC + 4 +$ jdisp8 if PSW.bit = 0			
DBNZ	B, $addr16	2	6	$B \leftarrow B - 1$, then $PC \leftarrow PC + 2 +$ jdisp8 if $B \neq 0$			
	C, $addr16	2	6	$C \leftarrow C - 1$, then $PC \leftarrow PC + 2 +$ jdisp8 if $C \neq 0$			
	saddr, $addr16	3	8	(saddr) $\leftarrow$ (saddr) – 1, then $PC \leftarrow PC + 3 +$ jdisp8 if (saddr) $\neq 0$			
NOP		1	2	No Operation			
EI		3	6	IE $\leftarrow$ 1 (Enable Interrupt)			
DI		3	6	IE $\leftarrow$ 0 (Disable Interrupt)			
HALT		1	2	Set HALT Mode			
STOP		1	2	Set STOP Mode			

Remark One instruction clock cycle is equal to one CPU clock (f_{CPU}) cycle selected by the processor clock control register (PCC).

4.1.6 Instruction list by addressing

(1) 8-bit instructions

MOV, XCH, ADD, ADDC, SUB, SUBC, AND, OR, XOR, CMP, INC, DEC, ROR, ROL, RORC, ROLC, PUSH, POP, DBNZ

1st operand \ 2nd operand	#byte	A	r	sfr	saddr	!addr16	PSW	[DE]	[HL]	[HL + byte]	$addr16	1	None
A	ADD ADDC SUB SUBC AND OR XOR CMP		MOV[Note] XCH[Note] ADD ADDC SUB SUBC AND OR XOR CMP	MOV XCH	MOV XCH ADD ADDC SUB SUBC AND OR XOR CMP	MOV ADD ADDC SUB SUBC AND OR XOR CMP	MOV	MOV XCH	MOV XCH ADD ADDC SUB SUBC AND OR XOR CMP	MOV XCH ADD ADDC SUB SUBC AND OR XOR CMP		ROR ROL RORC ROLC	
r	MOV	MOV[Note]											INC DEC
B, C											DBNZ		
sfr	MOV	MOV											
saddr	MOV ADD ADDC SUB SUBC AND OR XOR CMP	MOV									DBNZ		INC DEC
!addr16		MOV											
PSW	MOV	MOV											PUSH POP
[DE]		MOV											
[HL]		MOV											
[HL + byte]		MOV											

Note Except r = A.

(2) 16-bit instructions

MOVW, XCHW, ADDW, SUBW, CMPW, PUSH, POP, INCW, DECW

2nd operand / 1st operand	#word	AX	rp[Note]	saddrp	SP	None
AX	ADDW SUBW CMPW		MOVW XCHW	MOVW	MOVW	
rp	MOVW	MOVW[Note]				INCW DECW PUSH POP
saddrp		MOVW				
SP		MOVW				

Note Only when rp = BC, DE, HL.

(3) Bit manipulation instructions

SET1, CLR1, NOT1, BT, BF

2nd operand / 1st operand	$saddr	None
A.bit	BT BF	SET1 CLR1
sfr.bit	BT BF	SET1 CLR1
saddr.bit	BT BF	SET1 CLR1
PSW.bit	BT BF	SET1 CLR1
[HL].bit		SET1 CLR1
CY		SET1 CLR1 NOT1

(4) Call instructions/branch instructions

CALL, CALLT, BR, BC, BNC, BZ, BNZ, BT, BF, DBNZ

2nd operand / 1st operand	AX	!addr16	[addr5]	$addr16
Basic instructions	BR	CALL BR	CALLT	BR BC BNC BZ BNZ
Compound instructions				DBNZ

(5) Other instructions

RET, RETI, NOP, EI, DI, HALT, STOP

4.2 Instruction Codes

4.2.1 Description of instruction code table

r

R_2	R_1	R_0	reg	
0	0	0	R0	X
0	0	1	R1	A
0	1	0	R2	C
0	1	1	R3	B
1	0	0	R4	E
1	0	1	R5	D
1	1	0	R6	L
1	1	1	R7	H

rp

P_1	P_0	reg-pair	
0	0	RP0	AX
0	1	RP1	BC
1	0	RP2	DE
1	1	RP3	HL

Bn: Immediate data corresponding to “bit”

Data: 8-bit immediate data corresponding to “byte”

Low/High byte: 16-bit immediate data corresponding to “word”

Saddr-offset: 16-bit address lower 8-bit offset data corresponding to “saddr”

Sfr-offset: sfr 16-bit address lower 8-bit offset data

Low/High addr: 16-bit immediate data corresponding to “addr16”

jdisp: Signed two's complement data (8 bits) of relative address distance between the start and branch addresses of the next instruction

$ta_{4\ to\ 0}$: 5 bits of immediate data corresponding to “addr5”

4.2.2 Instruction code list

Mnemonic	Operand	Instruction Code			
		B1	B2	B3	B4
MOV	r, #byte	0 0 0 0 1 0 1 0	1 1 1 1 R_2 R_1 R_0 1	Data	
	saddr, #byte	1 1 1 1 0 1 0 1	Saddr-offset	Data	
	sfr, #byte	1 1 1 1 0 1 1 1	Sfr-offset	Data	
	A, r Note 1	0 0 0 0 1 0 1 0	0 0 1 0 R_2 R_1 R_0 1		
	r, A Note 1	0 0 0 0 1 0 1 0	1 1 1 0 R_2 R_1 R_0 1		
	A, saddr	0 0 1 0 0 1 0 1	Saddr-offset		
	saddr, A	1 1 1 0 0 1 0 1	Saddr-offset		
	A, sfr	0 0 1 0 0 1 1 1	Sfr-offset		
	sfr, A	1 1 1 0 0 1 1 1	Sfr-offset		
	A, !addr16	0 0 1 0 1 0 0 1	Low addr	High addr	
	!addr16, A	1 1 1 0 1 0 0 1	Low addr	High addr	
★	PSW, #byte	1 1 1 1 0 1 0 1	0 0 0 1 1 1 1 0	Data	
	A, PSW	0 0 1 0 0 1 0 1	0 0 0 1 1 1 1 0		
	PSW, A	1 1 1 0 0 1 0 1	0 0 0 1 1 1 1 0		
	A, [DE]	0 0 1 0 1 0 1 1			
	[DE], A	1 1 1 0 1 0 1 1			
	A, [HL]	0 0 1 0 1 1 1 1			
	[HL], A	1 1 1 0 1 1 1 1			
	A, [HL + byte]	0 0 1 0 1 1 0 1	Data		
	[HL + byte], A	1 1 1 0 1 1 0 1	Data		
XCH	A, X	1 1 0 0 0 0 0 0			
	A, r Note 2	0 0 0 0 1 0 1 0	0 0 0 0 R_2 R_1 R_0 1		
	A, saddr	0 0 0 0 0 1 0 1	Saddr-offset		
	A, sfr	0 0 0 0 0 1 1 1	Sfr-offset		
	A, [DE]	0 0 0 0 1 0 1 1			
	A, [HL]	0 0 0 0 1 1 1 1			
	A, [HL + byte]	0 0 0 0 1 1 0 1	Data		
MOVW	rp, #word	1 1 1 1 P_1 P_0 0 0	Low byte	High byte	
	AX, saddrp	1 1 0 1 0 1 1 0	Saddr-offset		
	saddrp, AX	1 1 1 0 0 1 1 0	Saddr-offset		
	AX, rp Note 3	1 1 0 1 P_1 P_0 0 0			
★	rp, AX Note 3	1 1 1 0 P_1 P_0 0 0			
XCHW	AX, rp Note 3	1 1 0 0 P_1 P_0 0 0			

Notes 1. Except r = A.

2. Except r = A, X.

3. Only when rp = BC, DE, or HL.

Mnemonic	Operand	Instruction Code			
		B1	B2	B3	B4
ADD	A, #byte	1 0 0 0 0 0 1 1	Data		
	saddr, #byte	1 0 0 0 0 0 0 1	Saddr-offset	Data	
	A, r	0 0 0 0 1 0 1 0	1 0 0 0 R_2 R_1 R_0 1		
	A, saddr	1 0 0 0 0 1 0 1	Saddr-offset		
	A, !addr16	1 0 0 0 1 0 0 1	Low addr	High addr	
	A, [HL]	1 0 0 0 1 1 1 1			
	A, [HL + byte]	1 0 0 0 1 1 0 1	Data		
ADDC	A, #byte	1 0 1 0 0 0 1 1	Data		
	saddr, #byte	1 0 1 0 0 0 0 1	Saddr-offset	Data	
	A, r	0 0 0 0 1 0 1 0	1 0 1 0 R_2 R_1 R_0 1		
	A, saddr	1 0 1 0 0 1 0 1	Saddr-offset		
	A, !addr16	1 0 1 0 1 0 0 1	Low addr	High addr	
	A, [HL]	1 0 1 0 1 1 1 1			
	A, [HL + byte]	1 0 1 0 1 1 0 1	Data		
SUB	A, #byte	1 0 0 1 0 0 1 1	Data		
	saddr, #byte	1 0 0 1 0 0 0 1	Saddr-offset	Data	
	A, r	0 0 0 0 1 0 1 0	1 0 0 1 R_2 R_1 R_0 1		
	A, saddr	1 0 0 1 0 1 0 1	Saddr-offset		
	A, !addr16	1 0 0 1 1 0 0 1	Low addr	High addr	
	A, [HL]	1 0 0 1 1 1 1 1			
	A, [HL + byte]	1 0 0 1 1 1 0 1	Data		
SUBC	A, #byte	1 0 1 1 0 0 1 1	Data		
	saddr, #byte	1 0 1 1 0 0 0 1	Saddr-offset	Data	
	A, r	0 0 0 0 1 0 1 0	1 0 1 1 R_2 R_1 R_0 1		
	A, saddr	1 0 1 1 0 1 0 1	Saddr-offset		
	A, !addr16	1 0 1 1 1 0 0 1	Low addr	High addr	
	A, [HL]	1 0 1 1 1 1 1 1			
	A, [HL + byte]	1 0 1 1 1 1 0 1	Data		
AND	A, #byte	0 1 1 0 0 0 1 1	Data		
	saddr, #byte	0 1 1 0 0 0 0 1	Saddr-offset	Data	
	A, r	0 0 0 0 1 0 1 0	0 1 1 0 R_2 R_1 R_0 1		
	A, saddr	0 1 1 0 0 1 0 1	Saddr-offset		
	A, !addr16	0 1 1 0 1 0 0 1	Low addr	High addr	
	A, [HL]	0 1 1 0 1 1 1 1			
	A, [HL + byte]	0 1 1 0 1 1 0 1	Data		

Mnemonic	Operand	Instruction Code			
		B1	B2	B3	B4
OR	A, #byte	0 1 1 1 0 0 1 1	Data		
	saddr, #byte	0 1 1 1 0 0 0 1	Saddr-offset	Data	
	A, r	0 0 0 0 1 0 1 0	0 1 1 1 $R_2 R_1 R_0$ 1		
	A, saddr	0 1 1 1 0 1 0 1	Saddr-offset		
	A, !addr16	0 1 1 1 1 0 0 1	Low addr	High addr	
	A, [HL]	0 1 1 1 1 1 1 1			
	A, [HL + byte]	0 1 1 1 1 1 0 1	Data		
XOR	A, #byte	0 1 0 0 0 0 1 1	Data		
	saddr, #byte	0 1 0 0 0 0 0 1	Saddr-offset	Data	
★	A, r	0 0 0 0 1 0 1 0	0 1 0 0 $R_2 R_1 R_0$ 1		
	A, saddr	0 1 0 0 0 1 0 1	Saddr-offset		
	A, !addr16	0 1 0 0 1 0 0 1	Low addr	High addr	
	A, [HL]	0 1 0 0 1 1 1 1			
	A, [HL + byte]	0 1 0 0 1 1 0 1	Data		
CMP	A, #byte	0 0 0 1 0 0 1 1	Data		
	saddr, #byte	0 0 0 1 0 0 0 1	Saddr-offset	Data	
★	A, r	0 0 0 0 1 0 1 0	0 0 0 1 $R_2 R_1 R_0$ 1		
	A, saddr	0 0 0 1 0 1 0 1	Saddr-offset		
	A, !addr16	0 0 0 1 1 0 0 1	Low addr	High addr	
	A, [HL]	0 0 0 1 1 1 1 1			
	A, [HL + byte]	0 0 0 1 1 1 0 1	Data		
ADDW	AX, #word	1 1 0 1 0 0 1 0	Low byte	High byte	
SUBW	AX, #word	1 1 0 0 0 0 1 0	Low byte	High byte	
CMPW	AX, #word	1 1 1 0 0 0 1 0	Low byte	High byte	
INC	r	0 0 0 0 1 0 1 0	1 1 0 0 $R_2 R_1 R_0$ 1		
	saddr	1 1 0 0 0 1 0 1	Saddr-offset		
DEC	r	0 0 0 0 1 0 1 0	1 1 0 1 $R_2 R_1 R_0$ 1		
	saddr	1 1 0 1 0 1 0 1	Saddr-offset		
INCW	rp	1 0 0 0 $P_1 P_0$ 0 0			
DECW	rp	1 0 0 1 $P_1 P_0$ 0 0			
ROR	A, 1	0 0 0 0 0 0 0 0			
ROL	A, 1	0 0 0 1 0 0 0 0			
RORC	A, 1	0 0 0 0 0 0 1 0			
ROLC	A, 1	0 0 0 1 0 0 1 0			

Mnemonic	Operand	Instruction Code			
		B1	B2	B3	B4
SET1	saddr.bit	0 0 0 0 1 0 1 0	0 B_2 B_1 B_0 1 0 1 0	Saddr-offset	
	sfr.bit	0 0 0 0 1 0 1 0	0 B_2 B_1 B_0 0 1 1 0	Sfr-offset	
	A.bit	0 0 0 0 1 0 1 0	0 B_2 B_1 B_0 0 0 1 0		
	PSW.bit	0 0 0 0 1 0 1 0	0 B_2 B_1 B_0 1 0 1 0	0 0 0 1 1 1 1 0	
	[HL].bit	0 0 0 0 1 0 1 0	0 B_2 B_1 B_0 1 1 1 0		
CLR1	saddr.bit	0 0 0 0 1 0 1 0	1 B_2 B_1 B_0 1 0 1 0	Saddr-offset	
	sfr.bit	0 0 0 0 1 0 1 0	1 B_2 B_1 B_0 0 1 1 0	Sfr-offset	
	A.bit	0 0 0 0 1 0 1 0	1 B_2 B_1 B_0 0 0 1 0		
	PSW.bit	0 0 0 0 1 0 1 0	1 B_2 B_1 B_0 1 0 1 0	0 0 0 1 1 1 1 0	
	[HL].bit	0 0 0 0 1 0 1 0	1 B_2 B_1 B_0 1 1 1 0		
SET1	CY	0 0 0 1 0 1 0 0			
CLR1	CY	0 0 0 0 0 1 0 0			
NOT1	CY	0 0 0 0 0 1 1 0			
CALL	!addr16	0 0 1 0 0 0 1 0	Low addr	High addr	
CALLT	[addr5]	0 1 $ta_{4\ to\ 0}$ 0			
RET		0 0 1 0 0 0 0 0			
RETI		0 0 1 0 0 1 0 0			
PUSH	PSW	0 0 1 0 1 1 1 0			
	rp	1 0 1 0 P_1 P_0 1 0			
POP	PSW	0 0 1 0 1 1 0 0			
	rp	1 0 1 0 P_1 P_0 0 0			
MOVW	SP, AX	1 1 1 0 0 1 1 0	0 0 0 1 1 1 0 0		
	AX, SP	1 1 0 1 0 1 1 0	0 0 0 1 1 1 0 0		
BR	!addr16	1 0 1 1 0 0 1 0	Low addr	High addr	
	$addr16	0 0 1 1 0 0 0 0	jdisp		
	AX	1 0 1 1 0 0 0 0			
BC	$addr16	0 0 1 1 1 0 0 0	jdisp		
BNC	$addr16	0 0 1 1 1 0 1 0	jdisp		
BZ	$addr16	0 0 1 1 1 1 0 0	jdisp		
BNZ	$addr16	0 0 1 1 1 1 1 0	jdisp		
BT	saddr.bit, $addr16	0 0 0 0 1 0 1 0	1 B_2 B_1 B_0 1 0 0 0	Saddr-offset	jdisp
	sfr.bit, $addr16	0 0 0 0 1 0 1 0	1 B_2 B_1 B_0 0 1 0 0	Sfr-offset	jdisp
	A.bit, $addr16	0 0 0 0 1 0 1 0	1 B_2 B_1 B_0 0 0 0 0	jdisp	
	PSW.bit, $addr16	0 0 0 0 1 0 1 0	1 B_2 B_1 B_0 1 0 0 0	0 0 0 1 1 1 1 0	jdisp

Mnemonic	Operand	Instruction Code			
		B1	B2	B3	B4
BF	saddr.bit, $addr16	0 0 0 0 1 0 1 0	0 B_2 B_1 B_0 1 0 0 0	Saddr-offset	jdisp
	sfr.bit, $addr16	0 0 0 0 1 0 1 0	0 B_2 B_1 B_0 0 1 0 0	Sfr-offset	jdisp
	A.bit, $addr16	0 0 0 0 1 0 1 0	0 B_2 B_1 B_0 0 0 0 0	jdisp	
	PSW.bit, $addr16	0 0 0 0 1 0 1 0	0 B_2 B_1 B_0 1 0 0 0	0 0 0 1 1 1 1 0	jdisp
DBNZ	B, $addr16	0 0 1 1 0 1 1 0	jdisp		
	C, $addr16	0 0 1 1 0 1 0 0	jdisp		
	saddr, $addr16	0 0 1 1 0 0 1 0	Saddr-offset	jdisp	
NOP		0 0 0 0 1 0 0 0			
EI		0 0 0 0 1 0 1 0	0 1 1 1 1 0 1 0	0 0 0 1 1 1 1 0	
DI		0 0 0 0 1 0 1 0	1 1 1 1 1 0 1 0	0 0 0 1 1 1 1 0	
HALT		0 0 0 0 1 1 0 0			
STOP		0 0 0 0 1 1 1 0			

CHAPTER 5 EXPLANATION OF INSTRUCTIONS

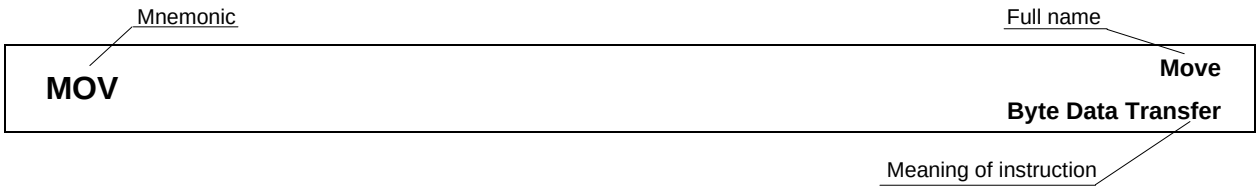
This chapter explains the instructions of 78K/0S Series. Each instruction is described in the unit of mnemonic, including description of multiple operands.

The basic configuration of instruction descriptions is shown on the next page.

For the number of instruction bytes and operation codes, refer to **CHAPTER 4 INSTRUCTION SET**.

All the instructions are common to 78K/0S Series products.

DESCRIPTION EXAMPLE



[Instruction format] MOV dst, src: Indicates the basic description format of the instruction.

[Operation] dst ← src: Indicates instruction operation using symbols.

[Operand] Indicates operands that can be specified with this instruction. Refer to **4.1 Operation** for a description of each operand symbol.

Mnemonic	Operand (dst, src)
MOV	r, #byte
	A, saddr
	saddr, A
	PSW, #byte

Mnemonic	Operand (dst, src)
MOV	A, PSW
	[HL], A
	A, [HL + byte]
	[HL + C], A

[Flag] Indicates the operation of the flag that changes by instruction execution.
Each flag operation symbol is shown in the legend**.**

Z	AC	CY

Legend

Symbol	Description
Blank	Unchanged
0	Cleared to 0
1	Set to 1
×	Set or cleared according to the result
R	Previously saved value is restored

[Description] Describes the instruction operation in detail.

- The contents of the source operand (src) specified by the 2nd operand are transferred to the destination operand (dst) specified by the 1st operand.

[Description example]

MOV A, #4DH; 4DH is transferred to A register.

5.1 8-Bit Data Transfer Instructions

The following instructions are 8-bit data transfer instructions.

MOV ... 60
XCH ... 61

MOV

Move

Byte Data Transfer

[Instruction format] MOV dst, src

[Operation] dst ← src

[Operand]

Mnemonic	Operand (dst, src)
MOV	r, #byte
	saddr, #byte
	sfr, #byte
	A, r Note
	r, A Note
	A, saddr
	saddr, A
	A, sfr
	sfr, A
	A, !addr16

Mnemonic	Operand (dst, src)
MOV	!addr16, A
	PSW, #byte
	A, PSW
	PSW, A
	A, [DE]
	[DE], A
	A, [HL]
	[HL], A
	A, [HL + byte]
	[HL + byte], A

Note Except r = A

[Flag]

PSW, #byte and PSW, A operands

Z	AC	CY
×	×	×

All other operand combinations

Z	AC	CY

[Description]

- The contents of the source operand (src) specified by the 2nd operand are transferred to the destination operand (dst) specified by the 1st operand.
- No interrupts are acknowledged between the “MOV PSW, #byte” instruction or the “MOV PSW, A” instruction and the subsequent instruction.

[Description example]

MOV A, #4DH; 4DH is transferred to A register.

XCH

Exchange

Byte Data Exchange

[Instruction format] XCH dst, src

[Operation] dst ↔ src

[Operand]

Mnemonic	Operand (dst, src)
XCH	A, X
	A, r Note
	A, saddr
	A, sfr
	A, [DE]
	A, [HL]
	A, [HL + byte]

Note Except r = A, X

[Flag]

Z	AC	CY

[Description]

- The 1st and 2nd operand contents are exchanged.

[Description example]

XCH A, 0FEBCH; The A register contents and address FEBCH contents are exchanged.

5.2 16-Bit Data Transfer Instructions

The following instructions are 16-bit data transfer instructions.

MOVW ... 63
XCHW ... 64

MOVW

Move Word

Word Data Transfer

[Instruction format] MOVW dst, src

[Operation] dst ← src

[Operand]

Mnemonic	Operand (dst, src)
MOVW	rp, #word
	AX, saddrp
	saddrp, AX
	AX, rp Note
	rp, AX Note

Note Only when rp = BC, DE or HL

[Flag]

Z	AC	CY

[Description]

- The contents of the source operand (src) specified by the 2nd operand are transferred to the destination operand (dst) specified by the 1st operand.

[Description example]

MOVW AX, HL; The HL register contents are transferred to the AX register.

[Caution]

★ Only an even address can be specified to saddrp. An odd address cannot be specified.

XCHW

Exchange Word
Word Data Exchange

[Instruction format] XCHW dst, src

[Operation] dst ↔ src

[Operand]

Mnemonic	Operand (dst, src)
XCHW	AX, rp Note

Note Only when rp = BC, DE or HL

[Flag]

Z	AC	CY

[Description]

- The 1st and 2nd operand contents are exchanged.

[Description example]

XCHW AX, BC; The memory contents of AX register are exchanged with those of the BC register.

5.3 8-Bit Operation Instructions

The following are 8-bit operation instructions.

ADD ... 66
ADDC ... 67
SUB ... 68
SUBC ... 69
AND ... 70
OR ... 71
XOR ... 72
CMP ... 73

ADD

Add

Byte Data Addition

[Instruction format] ADD dst, src

[Operation] dst, CY ← dst + src

[Operand]

Mnemonic	Operand (dst, src)
ADD	A, #byte
	saddr, #byte
	A, r
	A, saddr

Mnemonic	Operand (dst, src)
ADD	A, !addr16
	A, [HL]
	A, [HL + byte]

[Flag]

Z	AC	CY
×	×	×

[Description]

- The destination operand (dst) specified with the 1st operand is added to the source operand (src) specified with the 2nd operand and the result is stored in the CY flag and the destination operand (dst).
- If the addition result shows that dst is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the addition generates a carry from bit 7, the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- If the addition generates a carry from bit 3 to bit 4, the AC flag is set (1). In all other cases, the AC flag is cleared (0).

[Description example]

ADD CR10, #56H; 56H is added to the CR10 register and the result is stored in the CR10 register.

ADDC

Add with Carry

Addition of Byte Data with Carry

[Instruction format] ADDC dst, src

[Operation] dst, CY ← dst + src + CY

[Operand]

Mnemonic	Operand (dst, src)
ADDC	A, #byte
	saddr, #byte
	A, r
	A, saddr

Mnemonic	Operand (dst, src)
ADDC	A, !addr16
	A, [HL]
	A, [HL + byte]

[Flag]

Z	AC	CY
×	×	×

[Description]

- The destination operand (dst) specified with the 1st operand, the source operand (src) specified with the 2nd operand, and the CY flag are added and the result is stored in the destination operand (dst) and the CY flag. The CY flag is added to the least significant bit. This instruction is mainly used to add two or more bytes.
- If the addition result shows that dst is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the addition generates a carry from bit 7, the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- If the addition generates a carry from bit 3 to bit 4, the AC flag is set (1). In all other cases, the AC flag is cleared (0).

[Description example]

ADDC A, [HL]; The A register contents, the contents at address (HL register), and the CY flag are added and the result is stored in the A register.

SUB

Subtract

Byte Data Subtraction

[Instruction format] SUB dst, src

[Operation] dst, CY ← dst – src

[Operand]

Mnemonic	Operand (dst, src)
SUB	A, #byte
	saddr, #byte
	A, r
	A, saddr

Mnemonic	Operand (dst, src)
SUB	A, !addr16
	A, [HL]
	A, [HL + byte]

[Flag]

Z	AC	CY
×	×	×

[Description]

- The source operand (src) specified with the 2nd operand is subtracted from the destination operand (dst) specified with the 1st operand and the result is stored in the destination operand (dst) and the CY flag.
  The destination operand can be cleared to 0 by equalizing the source operand (src) and the destination operand (dst).
- If the subtraction shows that dst is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the subtraction generates a borrow at bit 7, the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- If the subtraction generates a borrow from bit 4 to bit 3, the AC flag is set (1). In all other cases, the AC flag is cleared (0).

[Description example]

SUB A, D; The D register is subtracted from the A register and the result is stored in the A register.

SUBC

Subtract with Carry

Subtraction of Byte Data with Carry

[Instruction format] SUBC dst, src

[Operation] dst, CY ← dst – src – CY

[Operand]

Mnemonic	Operand (dst, src)
SUBC	A, #byte
	saddr, #byte
	A, r
	A, saddr

Mnemonic	Operand (dst, src)
SUBC	A, !addr16
	A, [HL]
	A, [HL + byte]

[Flag]

Z	AC	CY
×	×	×

[Description]

- The source operand (src) specified with the 2nd operand and the CY flag are subtracted from the destination operand (dst) specified with the 1st operand and the result is stored in the destination operand (dst).
  The CY flag is subtracted from the least significant bit. This instruction is mainly used for subtraction of two or more bytes.
- If the subtraction shows that dst is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the subtraction generates a borrow at bit 7, the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- If the subtraction generates a borrow from bit 4 to bit 3, the AC flag is set (1). In all other cases, the AC flag is cleared (0).

[Description example]

SUBC A, [HL]; The (HL register) address contents and the CY flag are subtracted from the A register and the result is stored in the A register.

AND

And

Logical Product of Byte Data

[Instruction format] AND dst, src

[Operation] dst ← dst ∧ src

[Operand]

Mnemonic	Operand (dst, src)
AND	A, #byte
	saddr, #byte
	A, r
	A, saddr

Mnemonic	Operand (dst, src)
AND	A, !addr16
	A, [HL]
	A, [HL + byte]

[Flag]

Z	AC	CY
×		

[Description]

- The destination operand (dst) specified with the 1st operand and the source operand (src) specified with the 2nd operand are ANDed bit wise, and the result is stored in the destination operand (dst).
- If the logical product shows that all bits are 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).

[Description example]

AND 0FEBAH, #11011100B; The FEBAH contents and 11011100B are ANDed bit wise and the result is stored at FEBAH.

OR

Or

Logical Sum of Byte Data

[Instruction format] OR dst, src

[Operation] dst ← dst ∨ src

[Operand]

Mnemonic	Operand (dst, src)
OR	A, #byte
	saddr, #byte
	A, r
	A, saddr

Mnemonic	Operand (dst, src)
OR	A, !addr16
	A, [HL]
	A, [HL + byte]

[Flag]

Z	AC	CY
×		

[Description]

- The destination operand (dst) specified with the 1st operand and the source operand (src) specified with the 2nd operand are ORed bit wise, and the result is stored in the destination operand (dst).
- If the logical sum shows that all bits are 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).

[Description example]

OR A, 0FE98H; The A register and FE98H are ORed bit wise and the result is stored in the A register.

XOR

Exclusive Or

Exclusive Logical Sum of Byte Data

[Instruction format] XOR dst, src

[Operation] dst ← dst ⊻ src

[Operand]

Mnemonic	Operand (dst, src)
XOR	A, #byte
	saddr, #byte
	A, r
	A, saddr

Mnemonic	Operand (dst, src)
XOR	A, !addr16
	A, [HL]
	A, [HL + byte]

[Flag]

Z	AC	CY
×		

[Description]

- The destination operand (dst) specified with the 1st operand and the source operand (src) specified with the 2nd operand are XORed bit wise, and the result is stored in the destination operand (dst).
  Logical negation of all bits of the destination operand (dst) is possible with this instruction by selecting #0FFH for the source operand (src).
- If the exclusive logical sum shows that all bits are 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).

[Description example]

XOR A, L; The A and L registers are XORed bit wise and the result is stored in the A register.

CMP

Compare

Byte Data Comparison

[Instruction format] CMP dst, src

[Operation] dst – src

[Operand]

Mnemonic	Operand (dst, src)
CMP	A, #byte
	saddr, #byte
	A, r
	A, saddr

Mnemonic	Operand (dst, src)
CMP	A, !addr16
	A, [HL]
	A, [HL + byte]

[Flag]

Z	AC	CY
×	×	×

[Description]

- The source operand (src) specified with the 2nd operand is subtracted from the destination operand (dst) specified with the 1st operand.
  The subtraction result is not stored anywhere and only the Z, AC, and CY flags are changed.
- If the subtraction result is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the subtraction generates a borrow at bit 7, the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- If the subtraction generates a borrow from bit 4 to bit 3, the AC flag is set (1). In all other cases, the AC flag is cleared (0).

[Description example]

CMP 0FE38H, #38H; 38H is subtracted from the contents at address FE38H and only the Z, AC, and CY flags are changed (comparison of contents at address FE38H and the immediate data).

5.4 16-Bit Operation Instructions

The following are 16-bit operation instructions.

ADDW ... 75
SUBW ... 76
CMPW ... 77

ADDW

Add Word
Word Data Addition

[Instruction format] ADDW dst, src

[Operation] dst, CY ← dst + src

[Operand]

Mnemonic	Operand (dst, src)
ADDW	AX, #word

[Flag]

Z	AC	CY
×	×	×

[Description]

- The destination operand (dst) specified with the 1st operand is added to the source operand (src) specified with the 2nd operand and the result is stored in the destination operand (dst).
- If the addition result shows that dst is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the addition generates a carry from bit 15, the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- As a result of addition, the AC flag becomes undefined.

[Description example]

```
ADDW AX, #0ABCDH;   ABCDH is added to the AX register and the result is stored in the AX register.
```

SUBW

Subtract Word

Word Data Subtraction

[Instruction format] SUBW dst, src

[Operation] dst, CY ← dst – src

[Operand]

Mnemonic	Operand (dst, src)
SUBW	AX, #word

[Flag]

Z	AC	CY
×	×	×

[Description]

- The source operand (src) specified with the 2nd operand is subtracted from the destination operand (dst) specified with the 1st operand and the result is stored in the destination operand (dst) and the CY flag.
  The destination operand can be cleared to 0 by equalizing the source operand (src) and the destination operand (dst).
- If the subtraction shows that dst is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the subtraction generates a borrow at bit 15, the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- As a result of subtraction, the AC flag becomes undefined.

[Description example]

SUBW AX, #0ABCDH; ABCDH is subtracted from the AX register contents and the result is stored in the AX register.

CMPW

Compare Word
Word Data Comparison

[Instruction format] CMPW dst, src

[Operation] dst – src

[Operand]

Mnemonic	Operand (dst, src)
CMPW	AX, #word

[Flag]

Z	AC	CY
×	×	×

[Description]

- The source operand (src) specified with the 2nd operand is subtracted from the destination operand (dst) specified with the 1st operand.
  The subtraction result is not stored anywhere and only the Z, AC, and CY flags are changed.
- If the subtraction result is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the subtraction generates a borrow at bit 15, the CY flag is set (1). In all other cases, the CY flag is cleared (0).
- As a result of subtraction, the AC flag becomes undefined.

[Description example]

```
CMPW AX, #0ABCDH;  ABCDH is subtracted from the AX register and only the Z, AC, and CY flags are
                   changed (comparison of the AX register and the immediate data).
```

5.5 Increment/Decrement Instructions

The following are increment/decrement instructions.

INC ... 79
DEC ... 80
INCW ... 81
DECW ... 82

INC

Increment

Byte Data Increment

[Instruction format] INC dst

[Operation] dst ← dst + 1

[Operand]

Mnemonic	Operand (dst)
INC	r
	saddr

[Flag]

Z	AC	CY
×	×	

[Description]

- The destination operand (dst) contents are incremented by only one.
- If the increment result is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the increment generates a carry from bit 3 to bit 4, the AC flag is set (1). In all other cases, the AC flag is cleared (0).
- Because this instruction is frequently used for a counter for repeated operations, the CY flag contents are not changed (to hold the CY flag contents in multiple-byte operation).

[Description example]

INC B; The B register is incremented.

DEC

Decrement

Byte Data Decrement

[Instruction format] DEC dst

[Operation] dst ← dst – 1

[Operand]

Mnemonic	Operand (dst)
DEC	r
	saddr

[Flag]

Z	AC	CY
×	×	

[Description]

- The destination operand (dst) contents are decremented by only one.
- If the decrement result is 0, the Z flag is set (1). In all other cases, the Z flag is cleared (0).
- If the decrement generates a carry from bit 4 to bit 3, the AC flag is set (1). In all other cases, the AC flag is cleared (0).
- Because this instruction is frequently used for a counter for repeated operations, the CY flag contents are not changed (to hold the CY flag contents in multiple-byte operation).
- If dst is the B or C register or saddr, and it is not desired to change the AC and CY flag contents, the DBNZ instruction can be used.

[Description example]

DEC 0FE92H ; The contents at address FE92H are decremented.

INCW

Increment Word
Word Data Increment

[Instruction format] INCW dst

[Operation] dst ← dst + 1

[Operand]

Mnemonic	Operand (dst)
INCW	rp

[Flag]

Z	AC	CY

[Description]

- The destination operand (dst) contents are incremented by only one.
- Because this instruction is frequently used for increment of a register (pointer) used for addressing, the Z, AC, and CY flag contents are not changed.

[Description example]

INCW HL ; The HL register is incremented.

DECW

Decrement Word
Word Data Decrement

[Instruction format] DECW dst

[Operation] dst ← dst – 1

[Operand]

Mnemonic	Operand (dst)
DECW	rp

[Flag]

Z	AC	CY

[Description]

- The destination operand (dst) contents are decremented by only one.
- Because this instruction is frequently used for decrement of a register (pointer) used for addressing, the Z, AC, and CY flag contents are not changed.

[Description example]

DECW DE ; The DE register is decremented.

5.6 Rotate Instructions

The following are rotate instructions.

ROR ... 84
ROL ... 85
RORC ... 86
ROLC ... 87

ROR

Rotate Right

Byte Data Rotation to the Right

[Instruction format] ROR dst, cnt

[Operation] (CY, $dst_7 \leftarrow dst_0$, $dst_{m-1} \leftarrow dst_m$) × one time

[Operand]

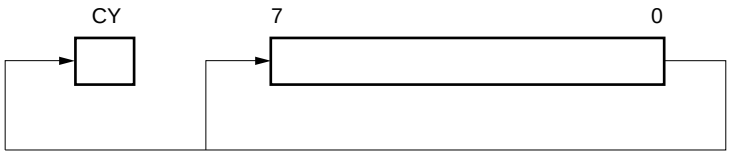
Mnemonic	Operand (dst, cnt)
ROR	A, 1

[Flag]

Z	AC	CY
		×

[Description]

- The destination operand (dst) contents specified with the 1st operand are rotated to the right just once.
- The LSB (bit 0) contents are simultaneously rotated to MSB (bit 7) and transferred to the CY flag.



[Description example]

ROR A, 1; The A register contents are rotated one bit to the right.

ROL

Rotate Left

Byte Data Rotation to the Left

[Instruction format] ROL dst, cnt

[Operation] (CY, $dst_0 \leftarrow 0$, dst_7, $dst_{m+1} \leftarrow dst_m$) × one time

[Operand]

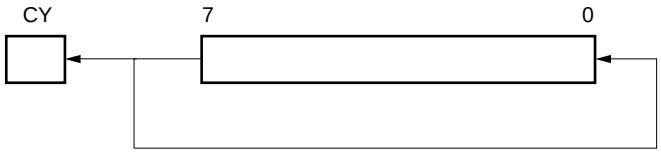
Mnemonic	Operand (dst, cnt)
ROL	A, 1

[Flag]

Z	AC	CY
		×

[Description]

- The destination operand (dst) contents specified with the 1st operand are rotated to the left just once.
- The MSB (bit 7) contents are simultaneously rotated to LSB (bit 0) and transferred to the CY flag.



[Description example]

ROL A, 1; The A register contents are rotated to the left by one bit.

RORC

Rotate Right with Carry

Byte Data Rotation to the Right with Carry

[Instruction format] RORC dst, cnt

[Operation] (CY ← dst_0, dst_7 ← CY, dst_{m-1} ← dst_m) × one time

[Operand]

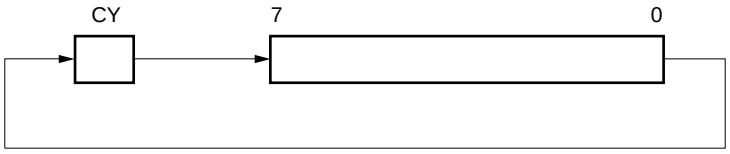
Mnemonic	Operand (dst, cnt)
RORC	A, 1

[Flag]

Z	AC	CY
		×

[Description]

- The destination operand (dst) contents specified with the 1st operand are rotated just once to the right including the CY flag.



[Description example]

RORC A, 1; The A register contents are rotated to the right by one bit including the CY flag.

ROLC

Rotate Left with Carry

Byte Data Rotation to the Left with Carry

[Instruction format] ROLC dst, cnt

[Operation] (CY ← dst_7, dst_0 ← CY, dst_{m+1} ← dst_m) × one time

[Operand]

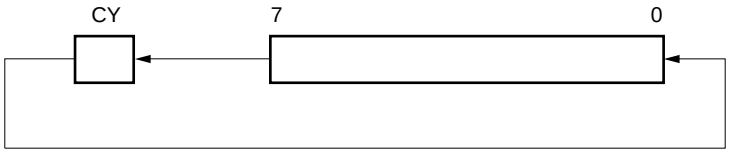
Mnemonic	Operand (dst, cnt)
ROLC	A, 1

[Flag]

Z	AC	CY
		×

[Description]

- The destination operand (dst) contents specified with the 1st operand are rotated just once to the left including the CY flag.



[Description example]

ROLC A, 1; The A register contents are rotated to the left by one bit including the CY flag.

5.7 Bit Manipulation Instructions

The following are bit manipulation instructions.

SET1 ... 89
CLR1 ... 90
NOT1 ... 91

SET1

Set Single Bit (Carry Flag)
1 Bit Data Set

[Instruction format] SET1 dst

[Operation] dst ← 1

[Operand]

Mnemonic	Operand (dst)
SET1	saddr.bit
	sfr.bit
	A.bit
	PSW.bit
	[HL].bit
	CY

[Flag]

dst = PSW.bit

Z	AC	CY
×	×	×

dst = CY

Z	AC	CY
		1

In all other cases

Z	AC	CY

[Description]

- The destination operand (dst) is set (1).
- When the destination operand (dst) is CY or PSW.bit, only the corresponding flag is set (1).

[Description example]

SET1 0FE55H.1; Bit 1 of FE55H is set (1).

CLR1

Clear Single Bit (Carry Flag)
1 Bit Data Clear

[Instruction format] CLR1 dst

[Operation] dst ← 0

[Operand]

Mnemonic	Operand (dst)
CLR1	saddr.bit
	sfr.bit
	A.bit
	PSW.bit
	[HL].bit
	CY

[Flag]

dst = PSW.bit

Z	AC	CY
×	×	×

dst = CY

Z	AC	CY
		0

In all other cases

Z	AC	CY

[Description]

- The destination operand (dst) is cleared (0).
- When the destination operand (dst) is CY or PSW.bit, only the corresponding flag is cleared (0).

[Description example]

CLR1 P3.7; Bit 7 of port 3 is cleared (0).

NOT1

Not Single Bit (Carry Flag)
1 Bit Data Logical Negation

[Instruction format] NOT1 dst

[Operation] dst ← $\overline{\text{dst}}$

[Operand]

Mnemonic	Operand (dst)
NOT1	CY

[Flag]

Z	AC	CY
		×

[Description]

- The CY flag is inverted.

[Description example]

NOT1 CY; The CY flag is inverted.

5.8 CALL/RETURN Instructions

The following are call/return instructions.

CALL ... 93
CALLT ... 94
RET ... 95
RETI ... 96

CALL

Call

Subroutine Call (16 Bit Direct)

[Instruction format] CALL target

[Operation]

(SP – 1) ← (PC + 3)$_H$,
(SP – 2) ← (PC + 3)$_L$,
SP ← SP – 2,
PC ← target

[Operand]

Mnemonic	Operand (target)
CALL	!addr16

[Flag]

Z	AC	CY

[Description]

- This is a subroutine call with a 16-bit absolute address or a register indirect address.
- The next instruction's start address (PC + 3) is saved in the stack and is branched to the address specified with the target operand (target).

[Description example]

CALL !3059H; Subroutine call to 3059H

CALLT

Call Table

Subroutine Call (Call Table Reference)

[Instruction format] CALLT [addr5]

[Operation]

(SP – 1) ← (PC + 1)$_H$,
(SP – 2) ← (PC + 1)$_L$,
SP ← SP – 2,
PC$_H$ ← (00000000,addr5 + 1)
PC$_L$ ← (00000000,addr5)

[Operand]

Mnemonic	Operand ([addr5])
CALLT	[addr5]

[Flag]

Z	AC	CY

[Description]

- This is a subroutine call for call table reference.
- The next instruction's start address (PC + 1) is saved in the stack and is branched to the address indicated with the word data of a call table (the higher 8 bits of address are fixed to 00000000B and the following 5 bits are specified with addr5).

[Description example]

CALLT [40H]; Subroutine call to the addresses indicated by word data of 0040H and 0041H.

RET

Return

Return from Subroutine

[Instruction format] RET

[Operation]

$PC_L \leftarrow (SP)$,
$PC_H \leftarrow (SP + 1)$,
$SP \leftarrow SP + 2$

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- This is a return instruction from the subroutine call made with the CALL and CALLT instructions.
- The word data saved in the stack returns to the PC, and the program returns from the subroutine.

RETI

Return from Interrupt
Return from Hardware Vectored Interrupt

[Instruction format] RETI

[Operation]

$PC_L \leftarrow (SP)$,
$PC_H \leftarrow (SP + 1)$,
PSW $\leftarrow (SP + 2)$,
SP $\leftarrow SP + 3$,
NMIS $\leftarrow 0$

[Operand]

None

[Flag]

Z	AC	CY
R	R	R

[Description]

- This is a return instruction from the vectored interrupt.
- The data saved in the stack returns to the PC and PSW, and the program returns from the interrupt service routine.
- None of interrupts are acknowledged between this instruction and the next instruction to be executed.
- The NMIS flag is set to 1 by acknowledgment of a non-maskable interrupt, and cleared to 0 by the RETI instruction.

[Caution]

When the return from non-maskable interrupt servicing is performed by an instruction other than the RETI instruction, the NMIS flag is not cleared to 0, and therefore no interrupts (including non-maskable interrupts) can be acknowledged.

5.9 Stack Manipulation Instructions

The following are stack manipulation instructions.

PUSH ... 98
POP ... 99
MOVW SP, AX ... 100
MOVW AX, SP ... 100

PUSH

Push

Push

[Instruction format] PUSH src

[Operation]

When src = rp

(SP – 1) ← src_H,
(SP – 2) ← src_L,
SP ←SP – 2

When src = PSW

(SP – 1) ← src
SP ← SP – 1

[Operand]

Mnemonic	Operand (src)
PUSH	PSW
	rp

[Flag]

Z	AC	CY

[Description]

- The data of the register specified with the source operand (src) is saved in the stack.

[Description example]

PUSH AX; AX register contents are saved in the stack.

POP

Pop

Pop

[Instruction format] POP dst

[Operation]

When dst = rp

$dst_L \leftarrow (SP)$,
$dst_H \leftarrow (SP + 1)$,
$SP \leftarrow SP + 2$

When dst = PSW

$dst \leftarrow (SP)$
$SP \leftarrow SP + 1$

[Operand]

Mnemonic	Operand (dst)
POP	PSW
	rp

[Flag]

dst = rp

Z	AC	CY

PSW

Z	AC	CY
R	R	R

[Description]

- Data is returned from the stack to the register specified with the destination operand (dst).
- When the operand is PSW, each flag is replaced with stack data.
- No interrupts are acknowledged between the POP PSW instruction and the subsequent instruction.

[Description example]

POP AX; The stack data is returned to the AX register.

MOVW SP, AX
MOVW AX, SP

Move Word

Word Data Transfer with Stack Pointer

[Instruction format] MOVW dst, src

[Operation] dst ← src

[Operand]

Mnemonic	Operand (dst, src)
MOVW	SP, AX
	AX, SP

[Flag]

Z	AC	CY

[Description]

- This is an instruction to manipulate the stack pointer contents.
- The source operand (src) specified with the 2nd operand is stored in the destination operand (dst) specified with the 1st operand.

[Description example]

MOVW SP, AX; AX register contents are stored in the stack pointer.

5.10 Unconditional Branch Instruction

The following is an unconditional branch instruction.

BR ... 102

BR

Branch

Unconditional Branch

[Instruction format] BR target

[Operation] PC ← target

[Operand]

Mnemonic	Operand (target)
BR	!addr16
	AX
	$addr16

[Flag]

Z	AC	CY

[Description]

- This is an instruction to branch unconditionally.
- The word data of the target address operand (target) is transferred to PC and program branches.

[Description example]

BR AX; The AX register contents are regarded as an address to which the program branches.

5.11 Conditional Branch Instructions

The following are conditional branch instructions.

BC ... 104
BNC ... 105
BZ ... 106
BNZ ... 107
BT ... 108
BF ... 109
DBNZ ... 110

BC

Branch if Carry

Conditional Branch with Carry Flag (CY = 1)

[Instruction format] BC $addr16

[Operation] PC ← PC + 2 + jdisp8 if CY = 1

[Operand]

Mnemonic	Operand ($addr16)
BC	$addr16

[Flag]

Z	AC	CY

[Description]

- When CY = 1, program branches to the address specified with the operand.
  When CY = 0, no processing is carried out and the subsequent instruction is executed.

[Description example]

BC $300H; When CY = 1, program branches to 0300H (with the start of this instruction set in the range of addresses 027FH to 037EH).

BNC

Branch if Not Carry

Conditional Branch with Carry Flag (CY = 0)

[Instruction format] BNC $addr16

[Operation] PC ← PC + 2 + jdisp8 if CY = 0

[Operand]

Mnemonic	Operand ($addr16)
BNC	$addr16

[Flag]

Z	AC	CY

[Description]

- When CY = 0, program branches to the address specified with the operand.
  When CY = 1, no processing is carried out and the subsequent instruction is executed.

[Description example]

BNC $300H; When CY = 0, program branches to 0300H (with the start of this instruction set in the range of addresses 027FH to 037EH).

BZ

Branch if Zero

Conditional Branch with Zero Flag (Z = 1)

[Instruction format] BZ $addr16

[Operation] PC ← PC + 2 + jdisp8 if Z = 1

[Operand]

Mnemonic	Operand ($addr16)
BZ	$addr16

[Flag]

Z	AC	CY

[Description]

- When Z = 1, program branches to the address specified with the operand.
  When Z = 0, no processing is carried out and the subsequent instruction is executed.

[Description example]

```
    DEC B
    BZ $3C5H;  When the B register is 0, program branches to 03C5H (with the start of this instruction set in the
               range of addresses 0344H to 0443H).
```

BNZ

Branch if Not Zero

Conditional Branch with Zero Flag (Z = 0)

[Instruction format] BNZ $addr16

[Operation] PC ← PC + 2 + jdisp8 if Z = 0

[Operand]

Mnemonic	Operand ($addr16)
BNZ	$addr16

[Flag]

Z	AC	CY

[Description]

- When Z = 0, program branches to the address specified with the operand.
  When Z = 1, no processing is carried out and the subsequent instruction is executed.

[Description example]

```
CMP A, #55H
BNZ $0A39H;   If the A register is not 0055H, program branches to 0A39H (with the start of this instruction set in
              the range of addresses 09B8H to 0AB7H).
```

BT

Branch if True

Conditional Branch by Bit Test (Byte Data Bit = 1)

[Instruction format] BT bit, $addr16

[Operation] PC ← PC + b + jdisp8 if bit = 1

[Operand]

Mnemonic	Operand (bit, $addr16)	b (Number of bytes)
BT	saddr.bit, $addr16	4
	sfr.bit, $addr16	4
	A.bit, $addr16	3
	PSW.bit, $addr16	4

[Flag]

Z	AC	CY

[Description]

- If the 1st operand (bit) contents have been set (1), program branches to the address specified with the 2nd operand ($addr16).

  If the 1st operand (bit) contents have not been set (1), no processing is carried out and the subsequent instruction is executed.

[Description example]

BT 0FE47H.3, $55CH; When bit 3 at address FE47H is 1, program branches to 055CH (with the start of this instruction set in the range of addresses 04DAH to 05D9H).

BF

Branch if False

Conditional Branch by Bit Test (Byte Data Bit = 0)

[Instruction format] BF bit, $addr16

[Operation] PC ← PC + b + jdisp8 if bit = 0

[Operand]

Mnemonic	Operand (bit, $addr16)	b (Number of bytes)
BF	saddr.bit, $addr16	4
	sfr.bit, $addr16	4
	A.bit, $addr16	3
	PSW.bit, $addr16	4

[Flag]

Z	AC	CY

[Description]

- If the 1st operand (bit) contents have been cleared (0), program branches to the address specified with the 2nd operand ($addr16).
  If the 1st operand (bit) contents have not been cleared (0), no processing is carried out and the subsequent instruction is executed.

[Description example]

BF P2.2, $1549H; When bit 2 of port 2 is 0, program branches to address 1549H (with the start of this instruction set in the range of addresses 14C6H to 15C5H).

DBNZ

Decrement and Branch if Not Zero
Conditional Loop (R1 ≠ 0)

[Instruction format] DBNZ dst, $addr16

[Operation] dst ← dst – 1,
then PC ← PC + b + jdisp16 if dst R1 ≠ 0

[Operand]

Mnemonic	Operand (dst, $addr16)	b (Number of bytes)
DBNZ	B, $addr16	2
	C, $addr16	2
	saddr, $addr16	3

[Flag]

Z	AC	CY

[Description]

- One is subtracted from the destination operand (dst) contents specified with the 1st operand and the subtraction result is stored in the destination operand (dst).
- If the subtraction result is not 0, program branches to the address indicated with the 2nd operand ($addr16). When the subtraction result is 0, no processing is carried out and the subsequent instruction is executed.
- The flag remains unchanged.

[Description example]

DBNZ B, $1215H; The B register contents are decremented. If the result is not 0, program branches to 1215H (with the start of this instruction set in the range of addresses 1194H to 1293H).

5.12 CPU Control Instructions

The following are CPU control instructions.

NOP ... 112
EI ... 113
DI ... 114
HALT ... 115
STOP ... 116

NOP

No Operation

No Operation

[Instruction format] NOP

[Operation] no operation

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- No processing is performed and only time is consumed.

EI

Enable Interrupt

Interrupt Enabled

[Instruction format] EI

[Operation] IE ← 1

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- The maskable interrupt acknowledge-enable status is set (by setting the interrupt enable flag (IE) (1)).
- Interrupts are acknowledged immediately after this instruction is executed.
- If this instruction is executed, vectored interrupt acknowledgment with another source can be disabled. For details, refer to **"Interrupt Functions"** in the User's Manual of each product.

DI

Disable Interrupt

Interrupt Disabled

[Instruction format] DI

[Operation] IE ← 0

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- Maskable interrupt acknowledgment with vectored interrupt is disabled (with the interrupt enable flag (IE) cleared (0)).
- No interrupts are acknowledged between this instruction and the subsequent instruction.
- For details of interrupt servicing, refer to **"Interrupt Functions"** in the User's Manual of each product.

HALT

Halt

HALT Mode Set

[Instruction format] HALT

[Operation] Set HALT Mode

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- This instruction is used to set the HALT mode to stop the CPU operation clock. Total power consumption of the system can be reduced with intermittent operations through combination with the normal operation mode.

STOP

Stop
Stop Mode Set

[Instruction format] STOP

[Operation] Set STOP Mode

[Operand]

None

[Flag]

Z	AC	CY

[Description]

- This instruction is used to set the STOP mode to stop the main system clock oscillator and to stop the whole system. Power dissipation can be minimized to an ultra-low leakage current level only.

APPENDIX A INSTRUCTION INDEX (MNEMONIC: BY FUNCTION)

[8-bit data transfer instructions]

MOV ... 60
XCH ... 61

[16-bit data transfer instructions]

MOVW ... 63
XCHW ... 64

[8-bit operation instructions]

ADD ... 66
ADDC ... 67
SUB ... 68
SUBC ... 69
AND ... 70
OR ... 71
XOR ... 72
CMP ... 73

[16-bit operation instructions]

ADDW ... 75
SUBW ... 76
CMPW ... 77

[Increment/decrement instructions]

INC ... 79
DEC ... 80
INCW ... 81
DECW ... 82

[Rotate instructions]

ROR ... 84
ROL ... 85
RORC ... 86
ROLC ... 87

[Bit manipulation instructions]

SET1 ... 89
CLR1 ... 90
NOT1 ... 91

[Call/return instructions]

CALL ... 93
CALLT ... 94
RET ... 95
RETI ... 96

[Stack manipulation instructions]

PUSH ... 98
POP ... 99
MOVW SP, AX ... 100
MOVW AX, SP ... 100

[Unconditional branch instruction]

BR ... 102

[Conditional branch instructions]

BC ... 104
BNC ... 105
BZ ... 106
BNZ ... 107
BT ... 108
BF ... 109
DBNZ ... 110

[CPU control instructions]

NOP ... 112
EI ... 113
DI ... 114
HALT ... 115
STOP ... 116

[MEMO]

APPENDIX B INSTRUCTION INDEX (MNEMONIC: IN ALPHABETICAL ORDER)

[A]

ADD ... 66
ADDC ... 67
ADDW ... 75
AND ... 70

[B]

BC ... 104
BF ... 109
BNC ... 105
BNZ ... 107
BR ... 102
BT ... 108
BZ ... 106

[C]

CALL ... 93
CALLT ... 94
CLR1 ... 90
CMP ... 73
CMPW ... 77

[D]

DBNZ ... 110
DEC ... 80
DECW ... 82
DI ... 114

[E]

EI ... 113

[H]

HALT ... 115

[I]

INC ... 79
INCW ... 81

[M]

MOV ... 60
MOVW ... 63
MOVW AX, SP ... 100
MOVW SP, AX ... 100

[N]

NOP ... 112
NOT1 ... 91

[O]

OR ... 71

[P]

POP ... 99
PUSH ... 98

[R]

RET ... 95
RETI ... 96
ROL ... 85
ROLC ... 87
ROR ... 84
RORC ... 86

[S]

SET1 ... 89
STOP ... 116
SUB ... 68
SUBC ... 69
SUBW ... 76

[X]

XCH ... 61
XCHW ... 64
XOR ... 72

[MEMO]

APPENDIX C REVISION HISTORY

A history of the revisions up to this edition is shown below. “Applied to:” indicates the chapters to which the revision was applied.

Edition	Contents	Applied to:
2nd	Addition of the following target products μPD789026, 789407, 789417, 789800, and 789806Y Subseries	Throughout
	Modification of the format of the table of the internal data memory space of the 78K/0S Series products	**CHAPTER 1 MEMORY SPACE**
3rd	Addition of the following target products μPD789046, 789104, 789114, 789124, 789134, 789146, 789156, 789167, 789177, 789197AY, 789217AY, 789407A, 789417A, and 789842 Subseries	Throughout
	Deletion of the following target products μPD789407, 789417, and 789806Y Subseries	
	Modification of MOV PSW, #byte instruction code	**CHAPTER 4 INSTRUCTION SET**
	Modification of MOVW rp, AX instruction code	
	Modification of XOR A, r instruction code	
	Modification of CMP A, r instruction code	

[MEMO]

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288
1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6465-6829

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

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

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: 044-435-9608

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	❑	❑	❑	❑