

User's Manual

V25TM, V35TM Family

16-/8- and 16-Bit Single-Chip Microcontrollers

Instructions

Target device

V25

V35

V25+TM

V35+TM

[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.

Intertool is a trademark of Intermetrics Microsystems Software.

V20, V30, V25, V35, V25+, V35+, and V Series are trademarks of NEC Corporation.

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

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

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

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

NEC devices are classified into the following three quality grades:

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

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

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

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

The quality grade of NEC devices is "Standard" unless otherwise specified in NEC's Data Sheets or Data Books. If customers intend to use NEC devices for applications other than those specified for Standard quality grade, they should contact an NEC sales representative in advance.

Anti-radioactive design is not implemented in this product.

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.

Spain Office
Madrid, Spain
Tel: 01-504-2787
Fax: 01-504-2860

NEC Electronics (Germany) GmbH

Scandinavia Office
Taebby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

NEC Electronics Hong Kong Ltd.

Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.

Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.

United Square, Singapore 1130
Tel: 253-8311
Fax: 250-3583

NEC Electronics Taiwan Ltd.

Taipei, Taiwan
Tel: 02-719-2377
Fax: 02-719-5951

NEC do Brasil S.A.

Cumbica-Guarulhos-SP, Brasil
Tel: 011-6465-6810
Fax: 011-6465-6829

Major Revisions in This Edition

Page	Description
Introduction	The following product names are deleted : μ PD70320(A), μ PD70330(A), μ PD70325(A), and μ PD70335(A)

The mark ★ shows major revised points.

INTRODUCTION

Readers: This manual is intended for the user engineers who understand the functions of the V25/V35 Family microcontrollers and design application systems using the microcontrollers.

★

Product name	Nickname
μ PD70320	V25
μ PD70330	V35
μ PD70325	V25+
μ PD70335	V35+

Purpose: This manual is intended to deepen the understanding of the user on the instruction functions of the above V25/V35 Family microcontrollers.

Organization: Two manuals are available for the V25/V35 Family products:
Hardware Manual and Instruction Manual (this manual).

Hardware

Instruction

General

Pin Function

CPU Function

Internal Block Function

Bus Control Function

Interrupt Function

Standby Function

Reset Function

Others

General

Instruction Description

Additional instructions of V20/V30

Instruction Map

Correspondence to Mnemonic of μ PD8086, 8088

Number of program execution clocks

Program development by 86 C compiler and assembler

How to Read This Manual: It is assumed that the readers of this manual have general knowledge on electric engineering, logic circuits, and microcontrollers. Unless otherwise specified, information contained in this manual commonly applies to all the models in the V series. Throughout this manual, the nicknames of the products are used instead of the product names.

To find the functional details of an instruction whose mnemonic is known,

→ Refer to **CHAPTER 2 INSTRUCTIONS** (instructions are presented in alphabetic order of mnemonics)

To understand the functions of all the instructions,

→ Read through the manual according to the Contents.

To understand the hardware functions of each product,

→ Refer to the **User's Manual – Hardware** (separately available).

Legend:

Data significance	:	Higher digits on left, lower on right
Active low	:	\overline{xxx} (top bar over pin or signal name)
Memory map address	:	top: higher, bottom: lower
Address notation	:	x indicates segment value, while y indicates offset value in following case: x : yH
Note	:	Footnote
Caution	:	Important points
Remarks	:	Supplement
Numeric notation	:	binary ... xxxx or xxxxB decimal ... xxxx hexadecimal ... xxxxH

Related Documents

Part number	Document name	User's manual		Application note
	Data sheet	Hardware	Instruction	
V25	U10090E	IEM-1220	This manual	IEA-1256 IEA-604 IEA-701
V35	U10170E			IEA-709
V25+	U12850J	IEU-1427		—
V35+	U12884J			IEA-709

Remark The contents of the above related documents are subject to change without notice. If you place an order for a document, be sure to confirm that the document is the latest edition.

[MEMO]

CONTENTS

CHAPTER 1 GENERAL	15
1.1 Functional Instruction List	16
1.2 Format of Instruction Words	17
1.3 Instruction Outline	17
1.3.1 Data transfer	17
1.3.2 Block manipulation	17
1.3.3 Bit field manipulation	17
1.3.4 I/O	18
1.3.5 Arithmetic operation	18
1.3.6 BCD operation	18
1.3.7 BCD adjustment	18
1.3.8 Data conversion	19
1.3.9 Bit manipulation	19
1.3.10 Shift and rotate	19
1.3.11 Stack manipulation	19
1.3.12 Program branch	20
1.3.13 CPU control	20
1.3.14 Register bank switching	20
CHAPTER 2 INSTRUCTIONS	21
2.1 Description of Instructions (in alphabetical order of mnemonic)	21
2.2 Number of Clocks Instruction Execution	201
CHAPTER 3 ADDITIONAL INSTRUCTIONS V20/V30	217
APPENDIX A INSTRUCTION MAP	221
APPENDIX B CORRESPONDENCE OF MNEMONICS TO μPD8086 AND 8088	225
APPENDIX C NUMBER OF PROGRAM EXECUTION CLOCKS	227
C.1 Synchronous Pipeline	227
C.2 Forced Prefetch Cycle	228
APPENDIX D PROGRAM DEVELOPMENT WITH 86 C COMPILER AND ASSEMBLER	229
D.1 C Language	229
D.2 Assembler	230
D.3 Example of Include File/Macro File	230
APPENDIX E INSTRUCTION INDEX (mnemonic: by function)	243
APPENDIX F INSTRUCTION INDEX (mnemonic: in alphabetical order)	245

[MEMO]

LIST OF FIGURES

Figure No.	Title	Page
1-1	Relationships between Common Instructions and Dedicated Instructions of Each Product	15
1-2	Instruction Word Format	17
1-3	Operation of ALU on Execution of Arithmetic Instruction	18
2-1	Coding Example	27

LIST OF TABLES

Table No.	Title	Page
1-1	Functional Instruction List	16
2-1	Legend of Flag Operations	21
2-2	Legend of Operand Types	22
2-3	Legend of Instruction Words	23
2-4	Legend of Instruction Formats and Operation Descriptions	24
2-5	Memory Addressing	26
2-6	Selecting 8-/16-Bit General-Purpose Registers	26
2-7	Selecting Segment Registers	26
2-8	Number of Instruction Execution Clocks	202
A-1	Instruction Map	222
A-2	Group1, Group2, Imm, Shift Code Table	223
A-3	Group3 Code Table	223
B-1	Mnemonic Correspondence with μ PD8086 and 8088	226

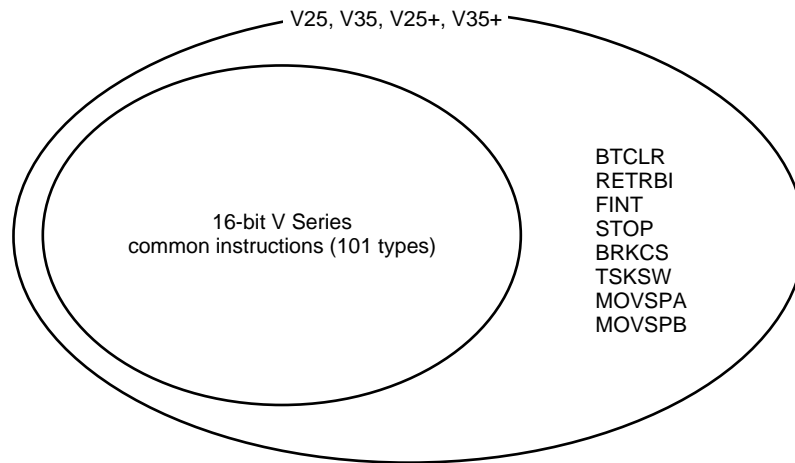
[MEMO]

CHAPTER 1 GENERAL

The 16-bit V Series™ microcontrollers have 101 common instructions that are completely software-compatible, so that your software resources can be effectively used.

In addition to these common instructions, eight instructions are provided to the V25, V35, V25+, and V35+.

Figure 1-1. Relationships between Common Instructions and Dedicated Instructions of Each Product



1.1 Functional Instruction List

The instructions of the V25/V35 Family microcontrollers can be broadly divided into the following 26 types by classification of function:

Table 1-1. Functional Instruction List (1/2)

Instructions	Mnemonic (in alphabetical order)
Data transfer	LDEA, MOV, MOVSPA, MOVSPB, TRANS, TRANSB, XCH
Repeat prefix	REP, REPC, REPE, REPNC, REPNE, REPNZ, REPZ
Primitive block transfer	CMPBK, CMPBKB, CMPBKW, CMPM, CMPMB, CMPMW, LDM, LDMB, LDMW, MOVBK, MOVBKB, MOVBKW, STM, STMB, STMW
Bit field manipulation	EXT, INS
I/O	IN, OUT
Primitive I/O	INM, OUTM
Add/subtract	ADD, ADDC, SUB, SUBC
BCD operation	ADD4S, CMP4S, ROL4, ROR4, SUB4S
Increment/decrement	DEC, INC
Multiplication/division	DIV, DIVU, MUL, MULU
BCD adjustment	ADJ4A, ADJ4S, ADJBA, ADJBS
Data conversion	CVTBD, CVTBW, CVTDB, CVTWL
Comparison	CMP
Complement operation	NEG, NOT
Logical operation	AND, OR, TEST, XOR
Bit manipulation	CLR1, NOT1, SET1, TEST1
Shift	SHL, SHR, SHRA
Rotate	ROL, ROLC, ROR, RORC
Subroutine control	CALL, RET
Stack manipulation	DISPOSE, POP, PREPARE, PUSH
Branch	BR
Conditional branch	BC, BCWZ, BE, BGE, BGT, BH, BL, BLE, BLT, BN, BNC, BNE, BNH, BNL, BNV, BNZ, BP, BPE, BPO, BTCLR, BZ, BV, DBNZ, DBNZE, DBNZNE
Interrupt	BRK, BRKV, CHKIND, FINT, RETI, RETRBI
CPU control	BUSLOCK, DI, EI, FPO1, FPO2, HALT, NOP, POLL, STOP
Segment override prefix	DS0:, DS1:, PS:, SS:
Register bank switching instructions	BRKCS, TSKSW

1.2 Format of Instruction Words

An instruction word (object code) is basically represented in the following format:

Figure 1-2. Instruction Word Format



Remark op code : 8-bit code indicating type of instruction
 Operand : 0- to 5-byte field indicating register or memory address subject to instruction processing

1.3 Instruction Outline

1.3.1 Data transfer

The data transfer instruction transfers data between registers or between a register and a memory address without data being manipulated. The following five types of data transfer instructions are available:

General-purpose data transfer (MOV):

Transfers byte/word from the second operand to the first operand. Can also transfer a numeric value directly to a register or memory address.

To transfer data in register bank (MOVSPA, MOVSPB):

Data is transferred between SS and SP before and after register bank has been switched.

Effective address transfer (LDEA):

Transfers the offset address (effective address) specified by the second operand to the first operand.

Conversion table transfer (TRANS/TRANSB):

Transfers 1 byte of the conversion table.

General-purpose data exchange (XCH):

Exchanges the contents of the first and second operands.

1.3.2 Block manipulation

The repeat prefix and primitive block transfer instructions transfer or compare byte or word (consecutive data) blocks.

As the primitive block transfer instructions, instructions that transfer data, compare a value, and scan are available, like instructions that transfer data in block units with the accumulator. If a 1-byte repeat prefix is prefixed, repetitive processing can be performed by hardware, so that data can be manipulated successively.

1.3.3 Bit field manipulation

The bit field manipulation instruction can transfer data of a specified length between a specified bit field and the AW register, regarding a contiguous memory area as a bit field.

This instruction updates the word offset (IX or IY register) and bit offset (8-bit general-purpose register) and automatically specifies contiguous bit field data at the end of execution, and is useful for applications such as computer graphics and high-level languages and can support, for example, the data structure of packed array or record type of Pascal.

1.3.4 I/O

The I/O and primitive I/O instructions can read/write I/O devices. When these instructions are executed, the I/O devices transfer data with the CPU through the data bus.

1.3.5 Arithmetic operation

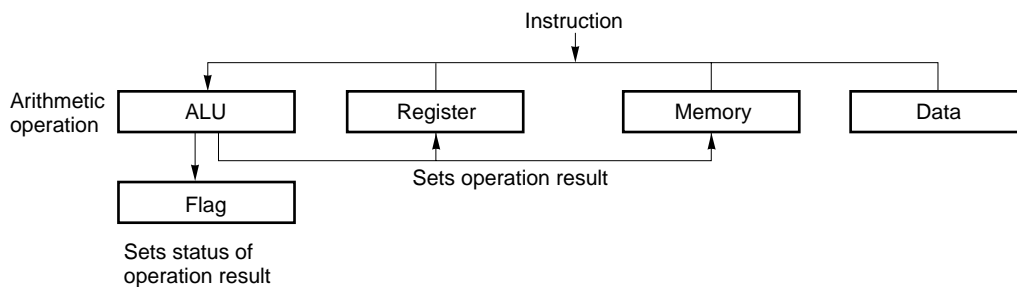
Arithmetic operation of 8-/16-bit data can be executed by the following instructions:

Add/subtract, increment/decrement, multiplication, division, comparison, complement operation, and logical operation

In addition, the 8-/16-bit data of a general-purpose register or memory area can be incremented (+1) or decremented (-1) by the increment or decrement instruction.

Each arithmetic operation instruction is not executed in a register or memory subject to the operation, but by the internal ALU. The result of the operation is set (to 1) or reset (to 0) in the program status word (PSW).

Figure 1-3. Operation of ALU on Execution of Arithmetic Instruction



1.3.6 BCD operation

The BCD operation instruction can express a decimal number by using a hexadecimal number for calculation.

This instruction can also execute arithmetic operations or comparison of BCD strings in memory.

This instruction supports rotate of BCD string.

There is no operand that specifies a packed BCD string because the registers used for the operation and comparison instructions are fixed.

The first address of the source string (address of byte data including LSD) is specified by the contents of the IX register in data segment 0 (DS0).

The first address of the destination string (address of byte data including LSD) is specified by the contents of the IY register in data segment 1 (DS1).

The number of digits is specified by the contents of the CL register.

The destination string and source string must be of the same length (number of digits). If the length of one string is different from that of the other, the shorter string is expanded to the longer length with 0s.

1.3.7 BCD adjustment

BCD operation is supported by executing a BCD adjustment instruction before or after execution of an arithmetic operation.

The BCD adjustment instruction is executed in respect to the AL register and has no operand. Addition and subtraction can be adjusted by both the packed or unpacked BCD representation, but multiplication and division can only be adjusted by an operation of unpacked BCD representation.

1.3.8 Data conversion

The data conversion instruction can convert the type and word length of a binary or decimal number.

The CVTBD and CVTDB instructions convert the type of binary and 2-digit unpacked BCD.

The CVTBW and CVTWL instructions extend the sign in a register.

1.3.9 Bit manipulation

The bit manipulation instruction can execute a logical operation to the bit data of a general-purpose register or memory.

The operand of this instruction is in the format of “reg, bit” or “mem, bit”.

The first operand reg or mem specifies 8-/16-bit that includes bit data, and a general-purpose register or effective address is specified as this operand.

The second operand bit specifies the in-byte/word address of the bit data, and is specified by the contents of CL or 8-bit immediate data.

However, if reg or mem is 8-bit data, only the lower 3 bits specify the valid bit address, and the higher bits are ignored.

If reg or mem is 16-bit data, only the lower 4 bits are valid.

1.3.10 Shift and rotate

The shift and rotate instructions shift or rotate 1 or more bits (0-255) of the 8-/16-bit data of a general-purpose register or memory.

Arithmetic shift or logical shift can be executed. The number of digits to be shifted is usually 1, but it can be changed by the value of the CL register each time the shift instruction has been executed, if so specified by the count operand of the instruction (255 max.). When arithmetic shift is executed, 0 is inserted to the LSB of data when the data is shifted 1 bit to the left, and to the MSB when the data is shifted 1 bit to the right. The value of the LSB or MSB does not change even if logical shift is executed and the data is shifted 1 bit.

To rotate data, the number of digits by which the data is to be rotated is specified by the value stored to the CL register by the count operand. As a result of executing the rotate instruction, only the CY and V flags are changed. To the CY flag, the bit pushed out of the data always enters the CY flag. The V flag is always undefined if data of 2 digits or more is rotated. If data of 1 digit is rotated, and if the destination MSB (extension) is changed as a result, the V flag is set to 1; if the MSB is not changed, the V flag is reset to 0. The CY flag can be used as the extension of the destination when the ROLC or RORC instruction is used.

1.3.11 Stack manipulation

The stack in memory can be manipulated by using the stack manipulation instruction.

The following four types of stack manipulation instructions are available:

PUSH : Saves data to stack.

POP : Restores data from stack.

PREPARE: Generates a stack frame and copies the frame pointer so that the area of local variables can be reserved and that global variables can be referenced.

DISPOSE : Returns the stack pointer (SP) and base pointer (BP) to the status immediately before execution of the PREPARE instruction.

1.3.12 Program branch

The branch instruction allows program execution to branch to a specified address.

The following four types of branch instructions are available:

- Subroutine control : Saves the contents of the program counter (PC) to the stack (CALL) or restores the contents of the PC from the stack (RET).
- Branch : Passes the flow of instruction to another address.
- Conditional branch : Passes the flow of instruction execution to another address depending on the value of a flag.
- Interrupt : Temporarily stops program execution if an interrupt request is issued from an external device or if an operation error occurs, and controls program execution through software interrupt.

1.3.13 CPU control

The CPU control instruction can manipulate flags, synchronize external devices, and transfer data. An instruction that causes the CPU to do nothing (NOP) is also available.

1.3.14 Register bank switching

These instructions select a register bank and are used for high-speed subroutine call.

CHAPTER 2 INSTRUCTIONS

2.1 Description of Instructions (in alphabetical order of mnemonic)

This section describes the following items of each instruction:

[Format]
[Operation]
[Operand]
[Flag]
[Description]
[Example]
[Bytes]
[Word Format]

In [Format], [Operation], and [Operand], many identifiers are used for description. Tables 2-2 through 2-4 show the meanings of these identifiers, and Tables 2-5 through 2-7 show memory addressing mode, general-purpose registers, and segment registers.

In [Flag], the operation of a flag that is changed as a result of instruction execution is shown by an identifier. The legend of each flag operation is shown in Table 2-1.

Table 2-1. Legend of Flag Operations

Identifier	Description
Blank	No change
0	Reset to 0
1	Set to 1
×	Set to 1 or reset to 0 depending on result
U	Undefined
R	Previously saved value is restored

Table 2-2. Legend of Operand Types

Identifier	Description
reg	8-/16-bit general-purpose register (destination register of instruction using two 8-/16-bit general-purpose registers)
reg'	Source register of instruction using two 8-/16-bit general-purpose registers
reg8	8-bit general-purpose register (destination register of instruction using two 8-bit general-purpose registers)
reg8'	Source register of instruction using two 8-bit general-purpose registers
reg16	16-bit general-purpose register (destination register of instruction using two 16-bit general-purpose registers)
reg16'	Source register of instruction using two 16-bit general-purpose registers
mem	8-/16-bit memory address
mem8	8-bit memory address
mem16	16-bit memory address
mem32	32-bit memory address
sfr	8-bit special function register location
dmem	16-bit direct memory address
imm	8-/16-bit immediate data
imm3	3-bit immediate data
imm4	4-bit immediate data
imm8	8-bit immediate data
imm16	16-bit immediate data
acc	Accumulator (AW or AL)
sreg	Segment register
src-table	Name of 256-byte conversion table
src-block	Name of source block addressed by IX register
dst-block	Name of destination block addressed by IY register
near-proc	Procedure in current program segment
far-proc	Procedure in another program segment
near-label	Label in current program segment
short-label	Label in range of -128 to +127 bytes from end of instruction
far-label	Label in another program segment
regptr16	16-bit general-purpose register having offset of call address in current program segment
memptr16	16-bit memory address having offset of call address in current program segment
memptr32	32-bit memory address having offset and segment data of call address in another program segment
pop-value	Number of bytes discarded from stack (0-64K, usually, even number)
fp-op	Immediate value identifying op code of floating-point coprocessor
R	Register set (AW, BW, CW, DW, SP, BP, IX, IY)
DS1-spec	Segment name/group name ASSUME to DS1 or DS1
Seg-spec	Segment name/group name ASSUME to segment register or any segment register name
[]	Can be omitted

Table 2-3. Legend of Instruction Words

Identifier	Description
W	Byte/word field (0, 1)
reg	Register field (000-111)
reg'	Register field (000-111) (source register of instruction having two registers)
mod,mem	Memory addressing specification bit (mod: 00-10, mem: 000-111)
(disp-low)	Lower byte of 16-bit displacement of option
(disp-high)	Higher byte of 16-bit displacement of option
disp-low	Lower byte of 16-bit displacement of PC relative for addition
disp-high	Higher byte of 16-bit displacement of PC relative for addition
imm3	3-bit immediate data
imm4	4-bit immediate data
imm8	8-bit immediate data
imm16-low	Lower byte of 16-bit immediate data
imm16-high	Higher byte of 16-bit immediate data
addr-low	Lower byte of 16-bit direct address
addr-high	Higher byte of 16-bit direct address
sreg	Segment register specification bit (00-11)
s	Sign extension specification bit (1: sign extended, 0: sign not extended)
offset-low	Lower byte of 16-bit offset data loaded to PC
offset-high	Higher byte of 16-bit offset data loaded to PC
seg-low	Lower byte of 16-bit segment loaded to PS
seg-high	Higher byte of 16-bit segment loaded to PS
pop-value-low	Lower byte of 16-bit data specifying number of bytes discarded from stack
pop-value-high	Higher byte of 16-bit data specifying number of bytes discarded from stack
disp8	8-bit displacement relatively added to PC
X	} op code of floating-point coprocessor
XXX	
YYY	
ZZZ	

Table 2-4. Legend of Instruction Formats and Operation Descriptions (1/2)

Identifier	Description
dst	Destination operand
dst1	Destination operand
dst2	Destination operand
src	Source operand
src1	Source operand
src2	Source operand
target	Target operand
AW	Accumulator (16 bits)
AH	Accumulator (higher byte)
AL	Accumulator (lower byte)
BW	BW register (16 bits)
CW	CW register (16 bits)
CL	CW register (lower byte)
DW	DW register (16 bits)
BP	Base pointer (16 bits)
SP	Stack pointer (16 bits)
PC	Program counter (16 bits)
PSW	Program status word (16 bits)
IX	Index register (source) (16 bits)
IY	Index register (destination) (16 bits)
PS	Program segment register (16 bits)
SS	Stack segment register (16 bits)
DS0	Data segment 0 register (16 bits)
DS1	Data segment 1 register (16 bits)
AC	Auxiliary carry flag
CY	Carry flag
P	Parity flag
S	Sign flag
Z	Zero flag
DIR	Direction flag
IE	Interrupt enable flag
V	Overflow flag
BRK	Break flag
IBRK	I/O break flag
RB0	Register bank flag
RB1	Register bank flag
RB2	Register bank flag
F0	User flag
F1	User flag

Table 2-4. Legend of Instruction Formats and Operation Descriptions (2/2)

Identifier	Description
(...)	Contents indicated in ()
disp	Displacement (8/16 bits)
temp	Temporary register (8/16/32 bits)
temp1	Temporary register (16 bits)
temp2	Temporary register (16 bits)
TA	Temporary register A (16 bits)
TB	Temporary register B (16 bits)
TC	Temporary register C (16 bits)
ext-disp8	16-bits of sign-extended 8-bit displacement
seg	Immediate segment data (16 bits)
offset	Immediate offset data (16 bits)
←	Transfer direction
+	Addition
-	Subtraction
×	Multiplication
÷	Division
%	Modulo
^	Logical product (AND)
∨	Logical sum (OR)
⊕	Exclusive logical sum (XOR)
xxH	2-digit hexadecimal number
xxxxH	4-digit hexadecimal number

Table 2-5. Memory Addressing

mem \ mod	00	01	10
000	BW + IX	BW + IX + disp8	BW + IX + disp16
001	BW + IY	BW + IY + disp8	BW + IY + disp16
010	BP + IX	BP + IX + disp8	BP + IX + disp16
011	BP + IY	BP + IY + disp8	BP + IY + disp16
100	IX	IX + disp8	IX + disp16
101	IY	IY + disp8	IY + disp16
110	Direct address	BP + disp8	BP + disp16
111	BW	BW + disp8	BW + disp16

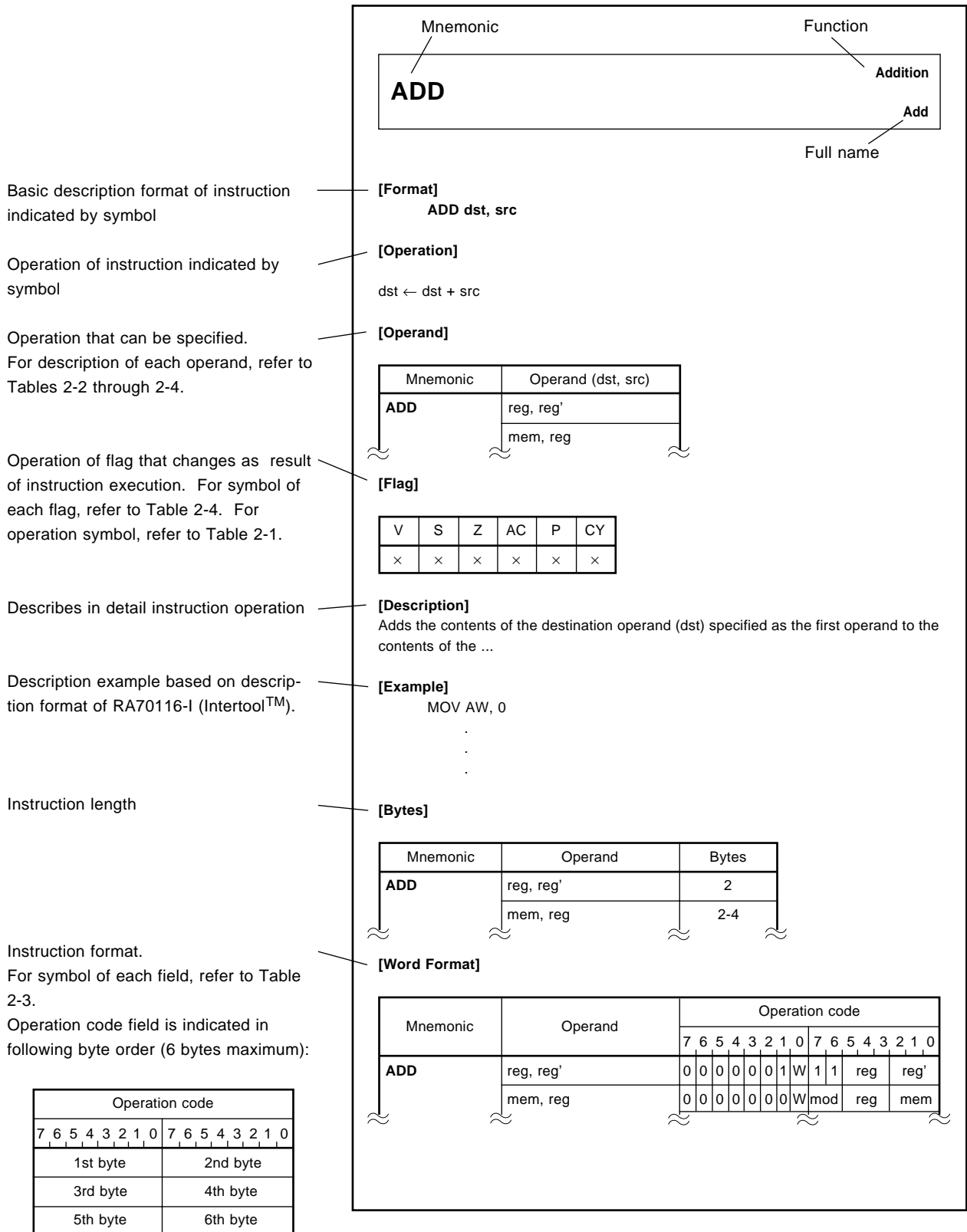
Table 2-6. Selecting 8-/16-Bit General-Purpose Registers

reg, reg'	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	IX
111	BH	IY

Table 2-7. Selecting Segment Registers

sreg	
00	DS1
01	PS
10	SS
11	DS0

Figure 2-1. Coding Example



ADD**Addition****Add****[Format]** **ADD dst, src****[Operand, operation]**

Mnemonic	Operand (dst, src)	Operation
ADD	reg, reg'	dst ← dst + src
	mem, reg	
	reg, mem	
	reg, imm	
	mem, imm	
	acc, imm	[When W = 0] AL ← AL + imm8 [When W = 1] AW ← AW + imm16

[Flag]

V	S	Z	AC	P	CY
×	×	×	×	×	×

[Description] Adds the contents of the destination operand (dst) specified as the first operand to the contents of the source operand (src) specified as the second operand, and stores the result to the destination operand (dst).

[Example] To add the contents of memory 0:50H (word data) to the contents of the DW register and store the result to 0:50H

```
MOV AW, 0
MOV DS1, AW
MOV IY, 50H
ADD DS1: WORD PTR [IY], DW
```

[Bytes]

Mnemonic	Operand	Bytes
ADD	reg, reg'	2
	mem, reg	2-4
	reg, mem	2-4
	reg, imm	3, 4
	mem, imm	3-6
	acc, imm	2, 3

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
ADD	reg, reg'	0	0	0	0	0	0	1	W	1	1	reg			reg'		
	mem, reg	0	0	0	0	0	0	0	W	mod		reg			mem		
		(disp-low)								(disp-high)							
	reg, mem	0	0	0	0	0	0	1	W	mod		reg			mem		
		(disp-low)								(disp-high)							
	reg, imm	1	0	0	0	0	0	s	W	1	1	0	0	0	reg		
		imm8 or imm16-low								imm16-high							
	mem, imm	1	0	0	0	0	0	s	W	mod		0	0	0	mem		
		(disp-low)								(disp-high)							
		imm8 or imm16-low								imm16-high							
	acc, imm	0	0	0	0	0	1	0	W	imm8 or imm16-low							
		imm16-high								—							

ADD4S	Decimal addition Add Nibble String
--------------	---------------------------------------

[Format] **ADD4S** [DS1-spec:] dst-string, [Seg-spec:] src-string
ADD4S

[Operation] BCD-string (IY, CL) ← BCD-string (IY, CL) + BCD-string (IX, CL)

[Operand]

Mnemonic	Operand (dst, src)
ADD4S	[DS1-spec:] dst-string, [Seg-spec:] src-string
	None

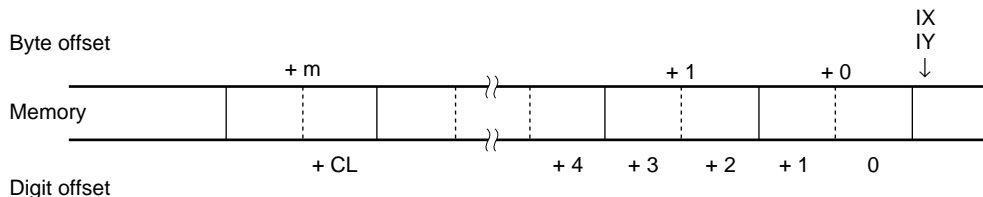
[Flag]

V	S	Z	AC	P	CY
U	U	×	U	U	×

[Description] Adds the packed BCD string addressed by the IX register to the packed BCD string addressed by the IY register, and stores the result to the string addressed by the IY register. The string length (number of BCD digits) is determined by the CL register (if the contents of CL are d, d digits) and can be set to 1 to 254 digits.

The destination string must be always located in a segment specified by the DS1 register, and segment override is not allowed. The default segment register of the source register is DS0 and segment override is allowed, so that the source register can be located in a segment specified by any segment register.

The format of the packed BCD string is as follows:



Caution: The BCD string instruction always operates in units of even number digits. If an even number is specified as the number of digits, therefore, the result of the operation and flags are normal. If an odd number is specified, however, the operation is executed with the even number of digits (= odd number + 1).

Consequently, the result and flags indicate the even number of digits. To specify an odd number, clear the higher 4 bits of the most significant byte before executing the BCD add instruction.

As a result, the carry is indicated by bit 4 of the most significant byte, and is not reflected on flags.

[Example] MOV IX, OFFSET VAR_1
 MOV IY, OFFSET VAR_2
 MOV CL, 4
 ADD4S

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
ADD4S	[DS1-spec:] dst-string, [Seg-spec:] src-string	0	0	0	0	1	1	1	1	0	0	1	0	0	0	0	0
	None																

ADDC

Addition with carry

Add with Carry

[Format] **ADDC dst, src****[Operand, operation]**

Mnemonic	Operand (dst, src)	Operation
ADDC	reg, reg'	dst ← dst + src + CY
	mem, reg	
	reg, mem	
	reg, imm	
	mem, imm	
	acc, imm	[When W = 0] AL ← AL + imm8 + CY [When W = 1] AW ← AW + imm16 + CY

[Flag]

V	S	Z	AC	P	CY
×	×	×	×	×	×

[Description] Adds the contents of the destination operand (dst) specified as the first operand to the contents of the source operand (src) specified as the second operand, including the carry, and stores the result to the destination operand (dst).

[Example] SET1 CY ; Sets CY flag to 1
 XOR AW, AW ; AW = 0
 MOV BW, 0FFH ; BW = 0FFH
 ADDC AW, BW ; Contents of AW register = 100H

[Bytes]

Mnemonic	Operand	Bytes
ADD	reg, reg'	2
	mem, reg	2-4
	reg, mem	2-4
	reg, imm	3, 4
	mem, imm	3-6
	acc, imm	2, 3

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
ADDC	reg, reg'	0	0	0	1	0	0	1	W	1	1	reg			reg'		
	mem, reg	0	0	0	1	0	0	0	W	mod		reg			mem		
		(disp-low)								(disp-high)							
	reg, mem	0	0	0	1	0	0	1	W	mod		reg			mem		
		(disp-low)								(disp-high)							
	reg, imm	1	0	0	0	0	0	s	W	1	1	0	1	0	reg		
		imm8 or imm16-low								imm16-high							
	mem, imm	1	0	0	0	0	0	s	W	mod		0	1	0	mem		
		(disp-low)								(disp-high)							
		imm8 or imm16-low								imm16-high							
	acc, imm	0	0	0	1	0	1	0	W	imm8 or imm16-low							
		imm16-high								—							

ADJ4A

Packed decimal adjustment of addition result

Adjust Nibble Add

[Format] ADJ4A**[Operation]** When $AL \wedge 0FH > 9$ or $AC = 1$ $AL \leftarrow AL + 6$ $AC \leftarrow 1$ When $AL > 9FH$ or $CY = 1$ $AL \leftarrow AL + 60H$ $CY \leftarrow 1$ **[Operand]**

Mnemonic	Operand
ADJ4A	None

[Flag]

V	S	Z	AC	P	CY
U	×	×	×	×	×

[Description] Adjusts the result of addition of two packed decimal numbers stored in the AL register to one packed decimal number.**[Example]** ADJ4A**[Bytes]** 1**[Word Format]**

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
ADJ4A	None	0	0	1	0	0	1	1	1

ADJ4S

Packed decimal adjustment of subtraction result
Adjust Nibble Subtract

[Format] ADJ4S

[Operation] When $AL \wedge 0FH > 9$ or $AC = 1$
 $AL \leftarrow AL - 6$
 $AC \leftarrow 1$
 When $AL > 9FH$ or $CY = 1$
 $AL \leftarrow AL - 60H$
 $CY \leftarrow 1$

[Operand]

Mnemonic	Operand
ADJ4S	None

[Flag]

V	S	Z	AC	P	CY
U	×	×	×	×	×

[Description] Adjusts the result of subtraction of two packed decimal numbers stored in the AL register to one packed decimal number.

[Example] SUB AW, BW
ADJ4S

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
ADJ4S	None	0	0	1	0	1	1	1	1

ADJBA

Unpacked decimal adjustment of addition result

Adjust Byte Add

[Format] ADJBA**[Operation]** When $AL \wedge 0FH > 9$ or $AC = 1$ $AL \leftarrow AL + 6$ $AH \leftarrow AH + 1$ $AC \leftarrow 1$ $CY \leftarrow AC$ $AL \leftarrow AL \wedge 0FH$ **[Operand]**

Mnemonic	Operand
ADJBA	None

[Flag]

V	S	Z	AC	P	CY
U	U	U	×	U	×

[Description] Adjusts the result of addition of two unpacked decimal numbers stored in the AL register to one unpacked decimal number. The higher 4 bits become 0.**[Example]** ADJBA**[Bytes]** 1**[Word Format]**

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
ADJBA	None	0	0	1	1	0	1	1	1

ADJBS

Unpacked decimal adjustment of subtraction result

Adjust Byte Subtract

[Format] ADJBS**[Operation]** When $AL \wedge 0FH > 9$ or $AC = 1$ $AL \leftarrow AL - 6$ $AH \leftarrow AH - 1$ $AC \leftarrow 1$ $CY \leftarrow AC$ $AL \leftarrow AL \wedge 0FH$ **[Operand]**

Mnemonic	Operand
ADJBS	None

[Flag]

V	S	Z	AC	P	CY
U	U	U	×	U	×

[Description] Adjusts the result of subtraction of two unpacked decimal numbers stored in the AL register to one unpacked decimal number. The higher 4 bits become 0.**[Example]** SUB AW, BW
ADJBS**[Bytes]** 1**[Word Format]**

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
ADJBS	None	0	0	1	1	1	1	1	1

AND

Logical product

And

[Format] **AND dst, src****[Operand, operation]**

Mnemonic	Operand (dst, src)	Operation
AND	reg, reg'	$dst \leftarrow dst \wedge src$
	mem, reg	
	reg, mem	
	reg, imm	
	mem, imm	
	acc, imm	[When W = 0] $AL \leftarrow AL \wedge imm8$ [When W = 1] $AW \leftarrow AW \wedge imm16$

[Flag]

V	S	Z	AC	P	CY
0	×	×	U	×	0

[Description] ANDs the destination operand (dst) specified as the first operand with the source operand (src) specified as the second operand, and saves the result to the destination operand (dst).**[Example]** MOV DW, IY
AND DW, 7FFFH**[Bytes]**

Mnemonic	Operand	Bytes
AND	reg, reg'	2
	mem, reg	2-4
	reg, mem	2-4
	reg, imm	3, 4
	mem, imm	3-6
	acc, imm	2, 3

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
AND	reg, reg'	0	0	1	0	0	0	1	W	1	1	reg			reg'		
	mem, reg	0	0	1	0	0	0	0	W	mod		reg			mem		
		(disp-low)								(disp-high)							
	reg, mem	0	0	1	0	0	0	1	W	mod		reg			mem		
		(disp-low)								(disp-high)							
	reg, imm ^{Note}	1	0	0	0	0	0	0	W	1	1	1	0	0	reg		
		imm8 or imm16-low								imm16-high							
	mem, imm	1	0	0	0	0	0	0	W	mod		1	0	0	mem		
		(disp-low)								(disp-high)							
		imm8 or imm16-low								imm16-high							
	acc, imm	0	0	1	0	0	1	0	W	imm8 or imm16-low							
		imm16-high								—							

Note With some assemblers and compilers, the codes shown below may be generated.

Operation code															
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	W	1	1	1	0	0	reg		
imm8								—							

Even in this case, instructions are executed normally. Take precautions, however, since some emulators do not support the disassembly function or line assembly function for this instruction.

BC
BL

Conditional branch with CY = 1

Branch if Carry

Branch if Lower

[Format] **BC short-label**
 BL short-label

[Operation] When CY = 1: PC ← PC + ext-disp8

[Operand]

Mnemonic	Operand
BC	short-label
BL	

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the CY flag is 1.
Execution can branch in a segment where this instruction is placed and in an address range of -128 to +127 bytes.

[Example] TEST AL, BL
 BC SHORT LP4 ; LP4 = label
 :
 TEST AL, BL
 BL SHORT LP5 ; LP5 = label
 :
LP4:

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BC	short-label	0	1	1	1	0	0	1	0	disp8							
BL																	

BCWZ

Conditional branch with CW = 0

Branch if CW equals Zero

[Format] BCWZ short-label**[Operation]** When CW = 0: PC ← PC + ext-disp8**[Operand]**

Mnemonic	Operand
BCWZ	short-label

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the value of the CW register is 0.

Execution can branch in a segment where this instruction is placed and in an address range of -128 to +127 bytes.

If the condition is not satisfied, the next instruction is executed.

[Example] LP22:
 ⋮
 ADD AL, BL
 BCWZ SHORT LP22 ; LP22 = label

[Bytes] 2**[Word Format]**

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BCWZ	short-label	1	1	1	0	0	0	1	1	disp8							

BE
BZ

Conditional branch with Z = 1

Branch if Equal

Branch if Zero

[Format] **BE short-label**
 BZ short-label

[Operation] When Z = 1: PC ← PC + ext-disp8

[Operand]

Mnemonic	Operand
BE	short-label
BZ	

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the Z flag is 1. Execution can branch in a segment where this instruction is placed and in an address range of -128 to +127 bytes.

[Example] AND AL, 2
 BE SHORT LOOP; LOOP = label
 :
 OR AH, BH
 BZ SHORT LOOP1; LOOP1 = label
 :
 LOOP:

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BE	short-label	0	1	1	1	0	1	0	0	disp8							
BZ																	

BGE

Conditional branch with $S \nabla V = 0$
Branch if Greater Than or Equal

[Format] BGE short-label

[Operation] When $S \nabla V = 0$: $PC \leftarrow PC + \text{ext-disp8}$

[Operand]

Mnemonic	Operand
BGE	Short-label

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the result of exclusive OR (XOR) of the S flag with the V flag is 0. Execution can branch in a segment where this instruction is placed and in an address range of -128 to +127 bytes. If the condition is not satisfied, the next instruction is executed.

[Example]

```

SHL AL, 1
BGE SHORT LP16; LP16 = label
:
LP16:
    
```

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BGE	short-label	0	1	1	1	1	1	0	1	disp8							

BGT

Conditional branch with $(S \nabla V) \vee Z = 0$
 Branch if Greater Than

[Format] BGT short-label

[Operation] When $(S \nabla V) \vee Z = 0$: $PC \leftarrow PC + \text{ext-disp8}$

[Operand]

Mnemonic	Operand
BGT	short-label

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the result of OR between the Z flag and the result of exclusive OR (XOR) between the S flag and V flag is 0.
 Execution can branch in a segment where this instruction is placed and in an address range of -128 to +127 bytes.
 If the condition is not satisfied, the next instruction is executed.

[Example] LP18:
 :
 SHL AL, 1
 BGT LP18

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BGT	short-label	0	1	1	1	1	1	1	1	disp8							

BH

Conditional branch with $CY \vee Z = 0$

Branch if Higher

[Format] BH short-label

[Operation] When $CY \vee Z = 0$: $PC \leftarrow PC + \text{ext-disp8}$

[Operand]

Mnemonic	Operand
BH	short-label

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the result of OR between the CY flag and Z flag is 0. Execution can branch in a segment where this instruction is placed and in an address range of -128 to +127 bytes.

[Example]

```

ROL AL, 1
BH SHORT LP10; LP10 = label
:
LP10:
    
```

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BH	short-label	0	1	1	1	0	1	1	1	disp8							

BLE

Conditional branch with $(S \nabla V) \vee Z = 1$
 Branch if Less than or Equal

[Format] BLE short-label

[Operation] When $(S \nabla V) \vee Z = 1$: $PC \leftarrow PC + \text{ext-disp8}$

[Operand]

Mnemonic	Operand
BLE	short-label

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the result of OR between the Z flag with the result of exclusive OR (XOR) between the S flag and V flag is 1.

Execution can branch in a segment where this instruction is placed and in an address range of -128 to +127 bytes.

If the condition is not satisfied, the next instruction is executed.

[Example] LP17:
 ⋮
 SHR AL, 1
 BLE SHORT LP17

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BLE	short-label	0	1	1	1	1	1	1	0	disp8							

BLT

Conditional branch with $S \nabla V = 1$
 Branch if Less Than

[Format] BLT short-label

[Operation] When $S \nabla V = 1$: $PC \leftarrow PC + \text{ext-disp8}$

[Operand]

Mnemonic	Operand
BLT	short-label

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the result of exclusive XOR between the S flag and V flag is 1. Execution can branch in a segment where this instruction is placed and in an address range of -128 to +127 bytes. If the condition is not satisfied, the next instruction is executed.

[Example]

```

    ADD AL, BL
    BLT SHORT LP15; LP15 = label
    ⋮
    LP15:
```

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BLT	short-label	0	1	1	1	1	1	0	0	disp8							

BN

Conditional branch with S = 1
Branch if Negative

[Format] BN short-label

[Operation] When S = 1: PC ← PC + ext-disp8

[Operand]

Mnemonic	Operand
BN	short-label

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the S flag is 1.
Execution can branch in a segment where this instruction is placed and in an address range of -128 to +127 bytes.

[Example]

```

ADD AL, BL
BN LP11; LP11 = label
:
LP11:
    
```

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BN	short-label	0	1	1	1	1	0	0	0	disp8							

BNC
BNL

Conditional branch with CY = 0
Branch if Not Carry
Branch if Not Lower

[Format] **BNC short-label**
 BNL short-label

[Operation] When CY = 0: PC ← PC + ext-disp8

[Operand]

Mnemonic	Operand
BNC	short-label
BNL	

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the CY flag is 0.
Execution can branch in a segment where this instruction is placed and in an address range of -128 to +127 bytes.

[Example] ROR AL, 1
 BNC SHORT LP6; LP6 = label
 :
 ROR AL, 1
 BNL SHORT LP7; LP7 = label
 :
LP6:

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BNC	short-label	0	1	1	1	0	0	1	1	disp8							
BNL																	

BNE
BNZ

Conditional branch with Z = 0
Branch if Not Equal
Branch if Not Zero

[Format] **BNE short-label**
 BNZ short-label

[Operation] When Z = 0: PC ← PC + ext-disp8

[Operand]

Mnemonic	Operand
BNE	short-label
BNZ	

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the Z flag is 0.
Execution can branch in a segment where this instruction is placed and in an address range of -128 to +127 bytes.

[Example] OR AL, BL
 BNE SHORT LP8; LP8 = label
 :
 AND SH, BH
 BNZ SHORT LP9; LP9 = label
 :
LP8:

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BNE	short-label	0	1	1	1	0	1	0	1	disp8							
BNZ																	

BNH

Conditional branch with $CY \vee Z = 1$

Branch if Not Higher

[Format] BNH short-label

[Operation] When $CY \vee Z = 1$: $PC \leftarrow PC + \text{ext-disp8}$

[Operand]

Mnemonic	Operand
BNH	short-label

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the result of OR between the CY flag and Z flag is 1. Execution can branch in a segment where this instruction is placed and in an address range of -128 to +127 bytes.

[Example] ROR AL, 1
 BNH SHORT LP9; LP9 = label
 ⋮
 LP9:

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BNH	short-label	0	1	1	1	0	1	1	0	disp8							

BNV

Conditional branch with V = 0
Branch if Not Overflow

[Format] BNV short-label

[Operation] When V = 0: PC ← PC + ext-disp8

[Operand]

Mnemonic	Operand
BNV	short-label

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the V flag is 0. Execution can branch in a segment where this instruction is placed and in an address range of -128 to +127 bytes.

[Example]

```
ROR AL, 1
BNV LP3
:
LP3:
```

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BNV	short-label	0	1	1	1	0	0	0	1	disp8							

BP

Conditional branch with S = 0
Branch if Positive

[Format] BP short-label

[Operation] When S = 0: $PC \leftarrow PC + \text{ext-disp8}$

[Operand]

Mnemonic	Operand
BP	short-label

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the S flag is 0. Execution can branch in a segment where this instruction is placed and in an address range of -128 to +127 bytes.

[Example]

```
SHR AL, 1
BP SHORT LP12; LP12 = label
:
LP12:
```

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BP	short-label	0	1	1	1	1	0	0	1	disp8							

BPE

Conditional branch with P = 1
Branch if Parity Even

[Format] BPE short-label

[Operation] When P = 1: PC ← PC + ext-disp8

[Operand]

Mnemonic	Operand
BPE	short-label

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the P flag is 1. Execution can branch in a segment where this instruction is placed and in an address range of -128 to +127 bytes.

[Example] ADD AL, BL
BPE SHORT LP13; LP13 = label
:
LP13:

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BPE	short-label	0	1	1	1	1	0	1	0	disp8							

BPO

Conditional branch with P = 0
Branch if Parity Odd

[Format] BPO short-label

[Operation] When P = 0: PC ← PC + ext-disp8

[Operand]

Mnemonic	Operand
BPO	short-label

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the P flag is 0. Execution can branch in a segment where this instruction is placed and in an address range of -128 to +127 bytes.

[Example] ADD AL, BL
BPO SHORT LP14; LP14 = label
:
LP14:

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BPO	short-label	0	1	1	1	1	0	1	1	disp8							

BRUnconditional branch
Branch**[Format]** BR target**[Operand, operation]**

Mnemonic	Operand (target)	Operation
BR	near-label	$PC \leftarrow PC + \text{disp}$
	short-label	$PC \leftarrow PC + \text{ext-disp8}$
	regptr16	$PC \leftarrow \text{target}$
	memptr16	
	far-label	$PS \leftarrow \text{seg}$ $PC \leftarrow \text{offset}$
	memptr32	$PS \leftarrow (\text{memptr32} + 3, \text{memptr32} + 2)$ $PC \leftarrow (\text{memptr32} + 1, \text{memptr32})$

[Flag]

V	S	Z	AC	P	CY

[Description]

- When target = near-label
Transfers the current PC value plus 16-bit displacement (disp) to the PC.
If the branch address is in the segment where this instruction is placed, the assembler automatically executes this instruction.
- When target = short-label
Transfers the current PC value plus 8-bit displacement (actually, sign-extended 16 bits (ext-disp8)) to the PC.
If the branch address is in the segment where this instruction is placed and within a range of ± 127 bytes, the assembler automatically executes this instruction.
- When target = regptr16 or target = memptr16
Transfers the contents of the target operand (target) to the PC.
Execution can branch to any address in the segment where this instruction is placed.
- When target = far-label
Transfers 16-bit offset data, which is the second and third bytes of the instruction, to the PC, and 16-bit segment data, which is the fourth and fifth bytes of the instruction, to the PS.
Execution can branch to any address in any segment.

- When target = memptr32
Loads the higher 2 bytes of 32-bit memory to the PS, and the lower 2 bytes to the PC.
Execution can branch to any address in any segment.

[Example] BR \$ - 8

[Bytes]

Mnemonic	Operand	Bytes
BR	near-label	3
	short-label	2
	regptr16	2
	memptr16	2-4
	far-label	5
	memptr32	2-4

[Word Format]

Mnemonic	Operand	Operation code																
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
BR	near-label	1	1	1	0	1	0	0	1	disp-low								
		disp-high								—								
	short-label	1	1	1	0	1	0	1	1	disp8								
	regptr16	1	1	1	1	1	1	1	1	1	1	1	0	0	reg			
	memptr16	1	1	1	1	1	1	1	1	mod	1	0	0	mem				
		(disp-low)								(disp-high)								
	far-label	1	1	1	0	1	0	1	0	offset-low								
		offset-high								seg-low								
seg-high								—										
memptr32	1	1	1	1	1	1	1	1	mod	1	0	1	mem					
	(disp-low)								(disp-high)									

BRK

Software trap

Break

[Format] BRK target**[Operand, operation]**

Mnemonic	Operand (target)	Operation
BRK	3	$TA \leftarrow (00DH, 00CH)$ $TC \leftarrow (00FH, 00EH)$ $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PSW$ $IE \leftarrow 0, BRK \leftarrow 0$ $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PS$ $PS \leftarrow TC$ $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PC$ $PC \leftarrow TA$
	imm8($\neq 3$)	$TA \leftarrow (imm8 \times 4 + 1, imm8 \times 4)$ $TC \leftarrow (imm8 \times 4 + 3, imm8 \times 4 + 2)$ $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PSW$ $IE \leftarrow 0, BRK \leftarrow 0$ $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PS$ $PS \leftarrow TC$ $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PC$ $PC \leftarrow TA$

[Flag]

V	IE	BRK	S	Z	AC	P	CY
	0	0					

[Description] Saves the values of the PSW, PS, and PC to the stack and resets the IE and BRK flags to 0. Next, when target = 3, loads the lower 2 bytes of vector 3 in the interrupt vector table to the PC, and the higher 2 bytes to the PS.

When target = imm8, loads the lower 2 bytes of the interrupt vector table specified by the 8-bit immediate data to the PC, and the higher 2 bytes to the PS.

[Example]

- BRK 3
- BRK 5

[Bytes]

Mnemonic	Operand	Bytes
BRK	3	1
	imm8	2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BRK	3	1	1	0	0	1	1	0	0	—							
	imm8	1	1	0	0	1	1	0	1	imm8							

BRKCS [Added to V20/V30]

Register bank switching
Break Context Switch

[Format] BRKCS reg16

[Operation] RB2-0 ← lower 3 bits of reg16
PSW save ← PSW, PC save ← PC
IE ← 0, BRK ← 0
PC ← vector PC

[Operand]

Mnemonic	Operand
BRKCS	reg16

[Flag]

RB2	RB1	RB0	V	DIR	IE	BRK	S	Z	F1	AC	F0	P	$\overline{\text{IBRK}}$	CY
×	×	×			0	0								

[Description] This instruction selects a register bank and is used for high-speed subroutine call. To restore from a newly selected register bank, use the RETRBI instruction. At this time, it is not necessary to execute the FINT instruction.

[Example] BRKCS AW

[Bytes] 3

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BRKCS	reg16	0	0	0	0	1	1	1	1	0	0	1	0	1	1	0	1
		1	1	0	0	0	reg			—							

BRKV

Overflow exception

Break if Overflow

[Format] BRKV

[Operation] When V = 1,
 $TA \leftarrow (011H, 010H)$
 $TC \leftarrow (013H, 012H)$
 $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PSW$
 $IE \leftarrow 0, BRK \leftarrow 0$
 $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PS$
 $PS \leftarrow TC$
 $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PC$
 $PC \leftarrow TA$

[Operand]

Mnemonic	Operand
BRKV	None

[Flag]

V	IE	BRK	S	Z	AC	P	CY
	0	0					

[Description] Saves the values of the PSW, PS, and PC to the stack if the V flag is set to 1, and resets the IE and BRK flags to 0.
 Next, if target = 3, loads the lower 2 bytes of vector 4 of the interrupt vector table to the PC, and the higher 2 bytes to the PS.
 The next instruction is executed if the V flag is reset to 0.

[Example] BRKV**[Bytes]** 1**[Word Format]**

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
BRKV	None	1	1	0	0	1	1	1	0

BTCLR [Added to V20/V30]

Conditional branch
Branch if True and Clear

[Format] BTCLR sfr, imm3, short-label

[Operation] When bit No. imm3 of (sfr) = 1
 $PC \leftarrow PC + \text{ext-disp8}$
 Bit No. imm3 of (sfr) $\leftarrow 0$

[Operand]

Mnemonic	Operand
BTCLR	sfr, imm3, short-label

[Flag]

V	S	Z	AC	P	CY

[Description] If a specified bit of a special function register is 1, clears the bit and loads the sum of the current PC value and an 8-bit displacement (actually, sign-extended 16 bits) to the PC. Execution can branch in the segment where this instruction is placed and in an address range of -128 to +127 bytes.

[Example] BTCLR EXIC0, 7, 45

[Bytes] 5

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BTCLR	sfr, imm3, short-label	0	0	0	0	1	1	1	1	1	0	0	1	1	1	0	0
		sfr								imm3							
		disp8								—							

BUSLOCK

Bus lock prefix
Bus Lock Prefix

[Format] BUSLOCK

[Operation] Bus Lock Prefix

[Operand]

Mnemonic	Operand
BUSLOCK	None

[Flag]

V	S	Z	AC	P	CY

[Description] Disables the hold request while an instruction following this instruction is executed. This instruction is used not to accept the hold request while the block processing described earlier is in progress.

- Cautions:**
1. Do not place this instruction immediately before the POLL instruction.
 2. Hardware interrupts (maskable interrupt and non-maskable interrupt) and single-stepbreak are not accepted in between this instruction and the next instruction.

[Example] BUSLOCK REP MOV BKB

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
BUSLOCK	None	1	1	1	1	0	0	0	0

BV

Conditional branch with V = 1

Branch if Overflow

[Format] **BV short-label****[Operation]** When V = 1: PC ← PC + ext-disp8**[Operand]**

Mnemonic	Operand
BV	short-label

[Flag]

V	S	Z	AC	P	CY

[Description] Loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits) when the V flag is 1.

Execution can branch in the segment where this instruction is placed and in an address range of -128 to +127 bytes.

[Example] LP2:
 :
 SHL AL, 1
 BV SHORT LP2

[Bytes] 2**[Word Format]**

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
BV	short-label	0	1	1	1	0	0	0	0	disp8							

CALL

Subroutine call
Call

[Format] CALL target

[Operand, operation]

Mnemonic	Operand (target)	Operation
CALL	near-proc	$SP \leftarrow SP - 2$ $(SP + 1, SP) \leftarrow PC$ $PC \leftarrow PC + disp$
	regptr16	$SP \leftarrow SP - 2$ $(SP + 1, SP) \leftarrow PC$ $PC \leftarrow regptr16$
	memptr16	$TA \leftarrow (memptr16 + 1, memptr16)$ $SP \leftarrow SP - 2$ $(SP + 1, SP) \leftarrow PC$ $PC \leftarrow TA$
	far-proc	$SP \leftarrow SP - 2$ $(SP + 1, SP) \leftarrow PS$ $PS \leftarrow seg$ $SP \leftarrow SP - 2$ $(SP + 1, SP) \leftarrow PC$ $PS \leftarrow offset$
	memptr32	$TA \leftarrow (memptr32 + 1, memptr32)$ $TB \leftarrow (memptr32 + 3, memptr32 + 2)$ $SP \leftarrow SP - 2$ $(SP + 1, SP) \leftarrow PS$ $PS \leftarrow TB$ $SP \leftarrow SP - 2$ $(SP + 1, SP) \leftarrow PC$ $PC \leftarrow TA$

[Flag]

V	S	Z	AC	P	CY

- [Description]**
- When target = near-proc, or target = regptr16
 The value of PC is saved to the stack and the next contents of the target operand (target) are transferred to PC.
 When target = near-proc : 16-bit relative address
 When target = regptr16 : Value of 16-bit register (offset)
 - When target = memptr16
 The value of PC is saved to the stack, and the contents of the 16-bit memory (offset) addressed by the target operand (target) are transferred to PC.
 Any address in the segment where this instruction is placed can be called.
 - When target = far-proc
 The values of PC and PS are saved to the stack, and the second and third bytes are transferred to PC, and the fourth and fifth bytes are transferred to PS.
 This instruction can call any address in any segment.
 - When target = memptr32
 The values of PC and PS are saved to the stack, and the higher 2 bytes of the 32-bit memory addressed by the target operand (target) are transferred to the PS, and the lower 2 bytes are transferred to PC.
 This instruction can call any address in any segment.

- [Example]**
- CALL \$ + 10
 - CALL SUB1; SUB1 is label

[Bytes]

Mnemonic	Operand	Bytes
CALL	near-proc	3
	regptr16	2
	memptr16	2-4
	far-proc	5
	memptr32	2-4

[Word Format]

Mnemonic	Operand	Operation code																
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
CALL	near-proc	1	1	1	0	1	0	0	0	disp-low								
		disp-high								—								
	regptr16	1	1	1	1	1	1	1	1	1	1	0	1	0	reg			
	memptr16	1	1	1	1	1	1	1	1	mod	0	1	0	mem				
		(disp-low)								(disp-high)								
	far-proc	1	0	0	1	1	0	1	0	offset-low								
		offset-high								seg-low								
		seg-high								—								
	memptr32	1	1	1	1	1	1	1	1	mod	0	1	1	mem				
		(disp-low)								(disp-high)								

CHKIND

Index value check

Check Index

[Format] **CHKIND reg16, mem32**

[Operation] When $(mem32) > reg16$ or $(mem32 + 2) < reg16$
 $TA \leftarrow (015H, 014H)$
 $TC \leftarrow (017H, 016H)$
 $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PSW$
 $IE \leftarrow 0, BRK \leftarrow 0$
 $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PS$
 $PS \leftarrow TC$
 $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PC$
 $PC \leftarrow TA$

[Operand]

Mnemonic	Operand
CHKIND	reg16, mem32

[Flag] If the interrupt condition is satisfied

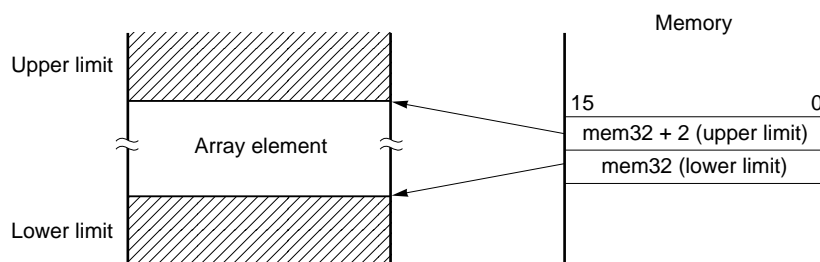
V	IE	BRK	S	Z	AC	P	CY
						0	0

If the interrupt condition is not satisfied

V	IE	BRK	S	Z	AC	P	CY

[Description] This instruction checks whether an index value specifying an element of array type data structure is in a defined area. If the index exceeds the defined area, the BRK 5 instruction is started. The defined area value is set in advance in 2 words (lower-limit value for the first word and the upper-limit value for the second word) in memory.

The index value is for a register (any 16-bit register) used for an array manipulation program.



[Example] CHKIND AW, DWORD_VAR

[Bytes] 2-4

[Word Format]

Mnemonic	Operand	Operation code														
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1
CHKIND	reg16, mem32	0	1	1	0	0	0	1	0	mod	reg			mem		
		(disp-low)							(disp-high)							

CLR1

Reset bit

Clear Bit

[Format] <1> CLR1 dst, src
 <2> CLR1 dst

[Operation] Format <1> : bit n of dst (n is specified by src) ← 0
 Format <2> : dst ← 0

[Operand] Format <1>

Mnemonic	Operand (dst, src)
CLR1	reg8, CL
	mem8, CL
	reg16, CL
	mem16, CL
	reg8, imm3
	mem8, imm3
	reg16, imm4
	mem16, imm4

Format <2>

Mnemonic	Operand (dst)
CLR1	CY
	DIR

[Flag] Format <1>

V	S	Z	AC	P	CY

Format <2> (when dst = CY)

V	S	Z	AC	P	CY
					0

Format <2> (when dst = DIR)

V	DIR	S	Z	AC	P	CY
	0					

[Description] Format <1> : Resets bit n (n is the contents of the source operand (src) specified by the second operand) of the destination operand (dst) specified by the first operand, and stores the result to the destination operand (dst).
 When the operand is reg8, CL or mem8, CL, only the lower 3 bits (0-7) of the CL value are valid.
 When the operand is reg16, CL or mem16, CL, only the lower 4 bits of the CL value (0-15) are valid.
 When the operand is reg8, imm3, only the lower 3 bits of the immediate data at the fourth byte position of the instruction are valid.
 When the operand is mem8, imm3, only the lower 3 bits of the immediate data at the end byte position of the instruction are valid.
 When the operand is reg16, imm4, only the lower 4 bits of the immediate data at the fourth byte position of the instruction are valid.
 When the operand is mem16, imm4, only the lower 4 bits of the immediate data at the end byte position of the instruction are valid.

Format <2> : Resets the CY flag to 0 when dst = CY.
 When dst = DIR, resets the DIR flag to 0. In addition, sets so that the index registers (IX, IY) are auto-incremented when each of the MOVBK, CMPBK, CMPM, LDM, STM, INM, and OUTM instructions are executed.

[Example] CLR1 CY
 SHL AL, 1
 BC \$ + 6

[Bytes]

Mnemonic	Operand	Bytes
CLR1	reg8, CL	3
	mem8, CL	3-5
	reg16, CL	3
	mem16, CL	3-5
	reg8, imm3	4
	mem8, imm3	4-6
	reg16, imm4	4
	mem16, imm4	4-6
	CY	1
	DIR	1

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
CLR1	reg8, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	0
		1	1	0	0	0	reg			—							
	mem8, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	0
		mod	0	0	0	mem			(disp-low)								
		(disp-high)								—							
	reg16, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1
		1	1	0	0	0	reg			—							
	mem16, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1
		mod	0	0	0	mem			(disp-low)								
		(disp-high)								—							
	reg8, imm3	0	0	0	0	1	1	1	1	0	0	0	1	1	0	1	0
		1	1	0	0	0	reg			imm3							
	mem8, imm3	0	0	0	0	1	1	1	1	0	0	0	1	1	0	1	0
		mod	0	0	0	mem			(disp-low)								
		(disp-high)								imm3							
	reg16, imm4	0	0	0	0	1	1	1	1	0	0	0	1	1	0	1	1
		1	1	0	0	0	reg			imm4							
	mem16, imm4	0	0	0	0	1	1	1	1	0	0	0	1	1	0	1	1
		mod	0	0	0	mem			(disp-low)								
		(disp-high)								imm4							
CY		1	1	1	1	1	0	0	0	—							
DIR		1	1	1	1	1	1	0	0	—							

CMP	Compare Compare
------------	--------------------

[Format] **CMP dst, src**

[Operand, operation]

Mnemonic	Operand (dst, src)	Operation
CMP	reg, reg'	dst – src
	mem, reg	
	reg, mem	
	reg, imm	
	mem, imm	
	acc, imm	[When W = 0] AL – imm8 [When W = 1] AW – imm16

[Flag]

V	S	Z	AC	P	CY
×	×	×	×	×	×

[Description] Subtracts the source operand specified as the second operand (src) from the destination operand specified as the first operand (dst).

The result of subtraction is not saved to anywhere, and only the flags are changed.

- [Example]**
- CMP BL, BYTE PTR [IX]
 - CMP CW, [BP + 4]

[Bytes]

Mnemonic	Operand	Bytes
CMP	reg, reg'	2
	mem, reg	2-4
	reg, mem	
	reg, imm	3, 4
	mem, imm	3-6
	acc, imm	2, 3

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
CMP	reg, reg'	0	0	1	1	1	0	1	W	1	1		reg				reg'
	mem, reg	0	0	1	1	1	0	0	W	mod			reg				mem
		(disp-low)							(disp-high)								
	reg, mem	0	0	1	1	1	0	1	W	mod			reg				mem
		(disp-low)							(disp-high)								
	reg, imm	1	0	0	0	0	0	s	W	1	1	1	1	1			reg
		imm8 or imm16-low							imm16-high								
	mem, imm	1	0	0	0	0	0	s	W	mod	1	1	1				mem
		(disp-low)							(disp-high)								
		imm8 or imm16-low							imm16-high								
	acc, imm	0	0	1	1	1	1	0	W	imm8 or imm16-low							
		imm16-high							—								

CMP4S

Decimal compare
Compare Nibble String

[Format] CMP4S [DS1-spec:] dst-string, [Seg-spec:] src-string
CMP4S

[Operation] BCD string (IY, CL) – BCD string (IX, CL)

[Operand]

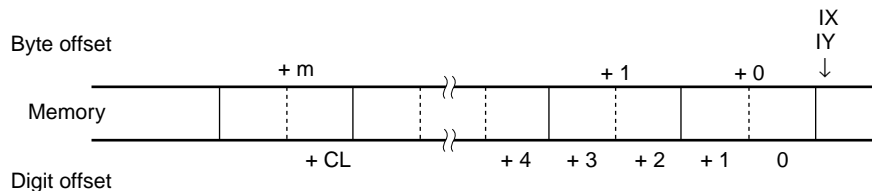
Mnemonic	Operand (dst, src)
CMP4S	[DS1-spec:] dst-string, [Seg-spec:] src-string
	None

[Flag]

V	S	Z	AC	P	CY
U	U	×	U	U	×

[Description] Subtracts the packed BCD string addressed by the IX register from the packed BCD string address by the IY register. The result is not stored but only the flags are affected. The string length (number of BCD digits) is determined by the CL register (if the contents of CL are d, d digits) and can be set to 1 to 254 digits.

The destination string must be always located in a segment specified by the DS1 register, and segment override is not allowed. The default segment register of the source register is DS0 and segment override is allowed, so that the source register can be located in a segment specified by any segment register. The format of the packed BCD string is as follows:



Caution The BCD string instruction always operates in units of even number digits. If an even number is specified as the number of digits, therefore, the result of the operation and flags are normal. If an odd number is specified, however, the operation is executed with the even number of digits (= odd number + 1). Consequently, the result and flags indicate the even number of digits. To specify an odd number, clear the higher 4 bits of the most significant byte before executing the BCD compare instruction.

[Example] MOV IX, OFFSET VAR_1
 MOV IY, OFFSET VAR_2
 MOV CL, 4
 CMP4S

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
CMP4S	[DS1-spec:] dst-string, [Seg-spec:] src-string	0	0	0	0	1	1	1	1	0	0	1	0	0	1	1	0
	None																

CMPBK

CMPBKB

CMPBKW

Block compare
 Compare Block
 Compare Block Byte
 Compare Block Word

[Format] (repeat) **CMPBK** [Seg-spec:] src-block, [DS1-spec:] dst-block
 (repeat) **CMPBKB**
 (repeat) **CMPBKW**

[Operation] [When W = 0] $(IX) - (IY)$
 DIR = 0: $IX \leftarrow IX + 1, IY \leftarrow IY + 1$
 DIR = 1: $IX \leftarrow IX - 1, IY \leftarrow IY - 1$
 [When W = 1] $(IX + 1, IX) - (IY + 1, IY)$
 DIR = 0: $IX \leftarrow IX + 2, IY \leftarrow IY + 2$
 DIR = 1: $IX \leftarrow IX - 2, IY \leftarrow IY - 2$

[Operand]

Mnemonic	Operand
CMPBK	[Seg-spec:] src-block, [DS1-spec:] dst-block
CMPBKB	None
CMPBKW	

[Flag]

V	S	Z	AC	P	CY
×	×	×	×	×	×

[Description] Repeatedly subtracts the block addressed by the IY register from the block addressed by the IX register in byte or word units, and reflects the result on the flags.

The IX and IY registers are automatically incremented (+1/+2) or decremented (-1/-2) for the next byte/word processing each time 1-byte/word data has been processed. The block direction is determined by the status of the DIR flag.

Whether subtraction is executed in byte or word units is determined by the attribute of the operand when the CMPBK instruction is used. When the CMPBKB or CMPBKW instructions are used, byte and word types are directly specified.

The destination block must be always located in a segment specified by the DS1 register, and segment override is not allowed. The default segment register of the source block is DS0 and segment override is allowed, so that the source block can be located in a segment specified by any segment register.

[Example] `CMPBK BYTE_VAR1, BYTE_VAR2`

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
CMPBK	[Seg-spec:] src-block, [DS1-spec:] dst-block	1	0	1	0	0	1	1	W
CMPBKB	None								
CMPBKW									

CMPM

CMPMB

CMPMW

Block compare with accumulator

Compare Multiple

Compare Multiple Byte

Compare Multiple Word

[Format] (repeat) **CMPM** [DS1-spec:] dst-block
 (repeat) **CMPMB**
 (repeat) **CMPMW**

[Operation] [When W = 0] AL – (IY)
 DIR = 0: IY ← IY + 1
 DIR = 1: IY ← IY – 1
 [When W = 1] AW – (IY + 1, IY)
 DIR = 0: IY ← IY + 2
 DIR = 1: IY ← IY – 2

[Operand]

Mnemonic	Operand (dst, src)
CMPM	[DS1-spec:] dst-block
CMPMB	None
CMPMW	

[Flag]

V	S	Z	AC	P	CY
×	×	×	×	×	×

[Description] Repeatedly subtracts the block addressed by the IY register from the accumulator (AL/AW) in byte or word units, and reflects the result on the flags.
 The IY register is automatically incremented (+1/+2) or decremented (–1/–2) for the next byte/word processing each time 1-byte/word data has been processed. The block direction is determined by the status of the DIR flag.
 Whether subtraction is executed in byte or word units is determined by the attribute of the operand when the CMPM instruction is used. When the CMPMB or CMPMW instructions are used, byte and word types are directly specified.
 The destination block must be always located in a segment specified by the DS1 register, and segment override is not allowed.

- [Example]**
- MOV AW, 5555H
 - MOV BW, 1000H
 - MOV IY, BW
 - REPC CMPM WORD PTR [IY]
 - REPNC CMPMW
 - REPZ CMPMB

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
CMPM	[DS1-spec:] dst-block	1	0	1	0	1	1	1	W
CMPMB	None								
CMPMW									

CVTBD

Binary-to-unpacked decimal conversion

Convert Binary to Decimal

[Format] CVTBD**[Operation]** AH ← AL ÷ 0AH
AL ← AL%0AH**[Operand]**

Mnemonic	Operand
CVTBD	None

[Flag]

V	S	Z	AC	P	CY
U	×	×	U	×	U

[Description] Converts the 8-bit binary number of the AL register to a 2-digit unpacked decimal number. The contents of the AH register are replaced with the quotient resulted from dividing the value of the AL register by 10, and the contents of the AL register are replaced by the remainder.**[Example]** MOV AL, 30H
CVTBD**[Bytes]** 2**[Word Format]**

Mnemonic	Operand	Operation code																
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
CVTBD	None	1	1	0	1	0	1	0	0	0	0	0	0	0	1	0	1	0

CVTBW

Word sign expansions

Convert Byte to Word

[Format] CVTBW**[Operation]** When AL < 80H: AH ← 0
When AL ≥ 80H: AH ← FFH**[Operand]**

Mnemonic	Operand
CVTBW	None

[Flag]

V	S	Z	AC	P	CY

[Description] Extends the sign of the byte in the AL register to the AH register. This instruction is useful for obtaining dividend of double length (word) from a certain byte before execution of byte division.**[Example]** MOV AL, BUF1; BUF1 is byte variable
CVTBW
MOV DL, 60
DIV DL**[Bytes]** 1**[Word Format]**

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
CVTBW	None	1	0	0	1	1	0	0	0

CVTDB

Unpacked decimal-to-binary conversion

Convert Decimal to Binary

[Format] CVTDB**[Operation]** $AL \leftarrow AH \times 0AH + AL$
 $AH \leftarrow 0$ **[Operand]**

Mnemonic	Operand
CVTDB	None

[Flag]

V	S	Z	AC	P	CY
U	×	×	U	×	U

[Description] Converts the 2-digit unpacked decimal number of the AH and AL registers to a binary number. The contents of the AH register are replaced with a value resulting from adding the value of the AL register to the result of multiplying the value of the AH register by 10, and the contents of the AH register are replaced with 0.**[Example]** MOV AW, [BW]
CVTDB**[Bytes]** 2**[Word Format]**

Mnemonic	Operand	Operation code																
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
CVTDB	None	1	1	0	1	0	1	0	1	0	0	0	0	0	1	0	1	0

CVTWL

Double word sign expansion
Convert Word to Long Word

[Format] CVTWL

[Operation] When $AW < 8000H$: $DW \leftarrow 0$
When $AW \geq 8000H$: $DW \leftarrow FFFFH$

[Operand]

Mnemonic	Operand
CVTWL	None

[Flag]

V	S	Z	AC	P	CY

[Description] Extends the sign of the word in the AW register to the DW register. This instruction is useful for obtaining dividend of double length (double word) from a certain word before execution of word division.

[Example] MOV AW, BUFFER
CVTWL
DIV CW

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
CVTWL	None	1	0	0	1	1	0	0	1

DBNZ

Conditional loop with CW ≠ 0
Decrement and Branch if Not Zero

[Format] DBNZ short-label

[Operation] CW ← CW – 1
 When CW ≠ 0: PC ← PC + ext-disp8

[Operand]

Mnemonic	Operand
DBNZ	short-label

[Flag]

V	S	Z	AC	P	CY

[Description] Decrements the value of the CW register (–1). If the value of the CW register is not 0 as a result, loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits). Execution can branch in the segment where this instruction is placed and within an address range of –128 to +127 bytes. The next instruction is executed if the condition is not satisfied.

[Example] LP21:
 ⋮
 SHL AL, 1
 DBNZ LP21; LP21 = label

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
DBNZ	short-label	1	1	1	0	0	0	1	0	disp8							

DBNZE

Conditional loop with $CW \neq 0$ and $Z = 1$
Decrement and Branch if Not Zero and Equal

[Format] **DBNZE short-label**

[Operation] $CW \leftarrow CW - 1$
 When $CW \neq 0$ and $Z = 1$: $PC \leftarrow PC + \text{ext-disp8}$

[Operand]

Mnemonic	Operand
DBNZE	short-label

[Flag]

V	S	Z	AC	P	CY

[Description] Decrements the value of the CW register (-1). If the value of the CW register is not 0 and the Z flag is set to 1 as a result, loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits).

Execution can branch in the segment where this instruction is placed and within an address range of -128 to +127 bytes.

The next instruction is executed if the condition is not satisfied.

[Example] LP20:
 :
 AND AL, BL
 DBNZE LP20; LP20 = label

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
DBNZE	short-label	1	1	1	0	0	0	0	1	disp8							

DBNZNE

Conditional loop with $CW \neq 0$ and $Z = 0$
Decrement and Branch if Not Zero and Not Equal

[Format] **DBNZNE short-label**

[Operation] $CW \leftarrow CW - 1$
 When $CW \neq 0$ and $Z = 0$: $PC \leftarrow PC + \text{ext-disp8}$

[Operand]

Mnemonic	Operand
DBNZNE	short-label

[Flag]

V	S	Z	AC	P	CY

[Description] Decrements the value of the CW register (-1). If the value of the CW register is not 0 and the Z flag is cleared to 0 as a result, loads the current PC value plus 8-bit displacement (actually, sign-extended 16 bits).

Execution can branch in the segment where this instruction is placed and within an address range of -128 to +127 bytes.

If the condition is not satisfied the next instruction is executed.

[Example] LP19:
 :
 AND AL, 0FFH
 DBNZNE SHORT LP19; LP19 = label

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
DBNZNE	short-label	1	1	1	0	0	0	0	0	disp8							

DEC

Decrement
Decrement

[Format] **DEC dst**

[Operation] $dst \leftarrow dst - 1$

[Operand]

Mnemonic	Operand
DEC	reg8
	mem
	reg16

[Flag]

V	S	Z	AC	P	CY
×	×	×	×	×	

[Description] Decrements (-1) the contents of the destination operand (dst).

- [Example]**
- DEC BW
 - DEC BP
 - DEC IX
 - DEC IY

[Bytes]

Mnemonic	Operand	Bytes
DEC	reg8	2
	mem	2-4
	reg16	1

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
DEC	reg8	1	1	1	1	1	1	1	0	1	1	0	0	1			reg
	mem	1	1	1	1	1	1	1	W	mod	0	0	1			mem	
			(disp-low)							(disp-high)							
	reg16	0	1	0	0	1			reg	—							

DI

Disable maskable interrupt
Disable Interrupt

[Format] DI

[Operation] IE ← 0

[Operand]

Mnemonic	Operand
DI	None

[Flag]

V	IE	S	Z	AC	P	CY
	0					

[Description] Resets the IE flag to 0 and disables maskable interrupt requests. This instruction does not disable non-maskable interrupt requests and software interrupt requests.

[Example] DI
PUSH R

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
DI	None	1	1	1	1	1	0	1	0

DISPOSE

Dispose stack frame
Dispose a Stack Frame

[Format] DISPOSE

[Operation] SP ← BP
BP ← (SP + 1, SP)
SP ← SP + 2

[Operand]

Mnemonic	Operand
DISPOSE	None

[Flag]

V	S	Z	AC	P	CY

[Description] This instruction releases one stack frame created by the PREPARE instruction. A pointer value indicating the preceding frame is loaded to BP, and a pointer value indicating the lowest frame is loaded to SP.

[Example] DISPOSE

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
DISPOSE	None	1	1	0	0	1	0	0	1

DIV**Signed division****Divide Signed****[Format] DIV dst****[Operand, operation]**

Mnemonic	Operand (dst)	Operation
DIV	reg8	temp ← AW When temp ÷ reg8 > 0 and temp ÷ reg8 ≤ 7FH, or temp ÷ reg8 < 0 and temp ÷ reg8 > 0 - 7FH - 1, AH ← temp%dst AL ← temp ÷ dst When temp ÷ reg8 > 0 and temp ÷ reg8 > 7FH, or temp ÷ reg8 < 0 and temp ÷ reg8 ≤ 0 - 7FH - 1,
	mem8	quotient and remainder undefined TA ← (001H, 000H) TC ← (003H, 002H) SP ← SP - 2, (SP + 1, SP) ← PSW IE ← 0, BRK ← 0 SP ← SP - 2, (SP + 1, SP) ← PS PS ← TC SP ← SP - 2, (SP + 1, SP) ← PC PC ← TA
	reg16	temp ← DW, AW When temp ÷ dst > 0 and temp ÷ dst ≤ 7FFFH, or temp ÷ dst < 0 and temp ÷ dst > 0 - 7FFFH - 1, DW ← temp%dst AW ← temp ÷ dst When temp ÷ dst > 0 and temp ÷ dst > 7FFFH, or temp ÷ dst < 0 and temp ÷ dst ≤ 0 - 7FFFH - 1,
	mem16	quotient and remainder undefined TA ← (001H, 000H) TC ← (003H, 002H) SP ← SP - 2, (SP + 1, SP) ← PSW IE ← 0, BRK ← 0

Mnemonic	Operand (dst)	Operation
DIV	mem16	$SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PS$ $PS \leftarrow TC$ $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PC$ $PC \leftarrow TA$

[Flag]

V	S	Z	AC	P	CY
U	U	U	U	U	U

[Description]

- When `src = reg8` or `src = mem8`
 Divides the value of the AW register by the contents of the destination operand (dst) with sign. The quotient is saved to the AL register, and the remainder is saved to the AH register. The maximum positive quotient is +127 (7FH) and the minimum negative quotient is -127 (81H). If the quotient is positive and exceeds the maximum value, or if the quotient is negative and falls below the minimum value, vector interrupt 0 occurs, and the quotient and remainder are undefined (in particular, the interrupt occurs when `src = 00H`). Non-integer quotients are rounded to be integers, and the remainder has the same sign as that of the dividend.
- When `src = reg16` or `src = mem16`
 Divides the values of the AW and DW registers by the contents of the destination operand (dst) with sign. The quotient is saved to the AW register, and the remainder is saved to the DW register. The maximum positive quotient is +32767 (7FFFH) and the minimum negative quotient is -32767 (8001H). If the quotient is positive and exceeds the maximum value, or if the quotient is negative and falls below the minimum value, vector interrupt 0 occurs, and the quotient and remainder are undefined (in particular, the interrupt occurs when `src = 0000H`). Non-integer quotients are rounded to be integers, and the remainder has the same sign as that of the dividend.

[Example]

To divide 32-bit data DW: AW by memory contents 0:50
`MOV BW, 0`
`MOV DS0, BW`
`MOV IX, 50H`
`DIV DS0: WORD PTR [IX]`

[Bytes]

Mnemonic	Operand	Bytes
DIV	reg8	2
	mem8	2-4
	reg16	2
	mem16	2-4

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
DIV	reg8	1	1	1	1	0	1	1	0	1	1	1	1	1			reg
	mem	1	1	1	1	0	1	1	0	mod	1	1	1				mem
		(disp-low)								(disp-high)							
	reg16	1	1	1	1	0	1	1	1	1	1	1	1	1			reg
	mem16	1	1	1	1	0	1	1	1	mod	1	1	1				mem
		(disp-low)								(disp-high)							

DIVU

Unsigned division

Divide Unsigned

[Format] DIVU dst

[Operand, operation]

Mnemonic	Operand (dst)	Operation
DIVU	reg8	temp ← AW When temp ÷ dst ≤ FFH: AH ← temp%dst AL ← temp ÷ dst When temp ÷ dst > FFH:
	mem8	TA ← (001H, 000H) TC ← (003H, 002H) SP ← SP - 2, (SP + 1, SP) ← PSW IE ← 0, BRK ← 0 SP ← SP - 2, (SP + 1, SP) ← PS RS ← TC SP ← SP - 2, (SP + 1, SP) ← PC PC ← TA
	reg16	temp ← DW, AW When temp ÷ dst ≤ FFFFH: DW ← temp%dst AW ← temp ÷ dst When temp ÷ dst > FFFFH:
	mem16	TA ← (001H, 000H) TC ← (003H, 002H) SP ← SP - 2, (SP + 1, SP) ← PSW IE ← 0, BRK ← 0 SP ← SP - 2, (SP + 1, SP) ← PS RS ← TC SP ← SP - 2, (SP + 1, SP) ← PC PC ← TA

[Flag]

V	S	Z	AC	P	CY
U	U	U	U	U	U

[Description]

- When src = reg8 or src = mem8
Divides the value of the AW register by the contents of the destination operand (dst) without sign. The quotient is saved to the AL register, and the remainder is saved to the AH register. If the quotient exceeds the AL register value (FFH), vector interrupt 0 occurs, and the quotient and remainder are undefined (especially, the interrupt occurs when src = 00H). Non-integer quotients are rounded to be integers.
- When src = reg16 or src = mem16
Divides the values of the AW and DW registers by the contents of the destination operand (dst) without sign. The quotient is saved to the AW register, and the remainder is saved to the DW register. If the quotient is exceeds the AW register value (FFFFH), vector interrupt 0 occurs, and the quotient and remainder are undefined (in particular, the interrupt occurs when src = 0000H). Non-integer quotients are rounded to be integers.

[Example]

To divide 5 by 3
 MOV AW, 5
 MOV DL, 3
 DIVU DL
 ; AH = 2 AL = 1

[Bytes]

Mnemonic	Operand	Bytes
DIVU	reg8	2
	mem8	2-4
	reg16	2
	mem16	2-4

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
DIVU	reg8	1	1	1	1	0	1	1	0	1	1	1	1	0	reg		
	mem8	1	1	1	1	0	1	1	0	mod	1	1	0	mem			
		(disp-low)						(disp-high)									
	mem16	reg16	1	1	1	1	0	1	1	1	1	1	1	1	0	reg	
mem16		1	1	1	1	0	1	1	1	mod	1	1	0	mem			
	(disp-low)						(disp-high)										

DS0:	Segment override prefix
DS1:	Data Segment 0
PS:	Data Segment 1
SS:	Program Segment
	Stack Segment

[Format] **DS0:**
DS1:
PS:
SS:

[Operation] Segment override prefix

[Operand]

Mnemonic	Operand
DS0:	None
DS1:	
PS:	
SS:	

[Flag]

V	S	Z	AC	P	CY

[Description] When a memory operand whose segment can be overridden is accessed, specifies a register to be used by appending an operand. Even if this instruction is not directly used, the assembler follows the segment override specification when the assembler directive ASSUME is used.

Caution **The hardware interrupt (maskable interrupt and non-maskable interrupt) requests and single-step break are not accepted after this instruction is executed and before the next instruction is executed.**

[Example] MOV DW, DS1: [BW]; Default segment register is DS0.

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
DS0:	None	0	0	1	sreg		1	1	0
DS1:									
PS:									
SS:									

EI

Enable maskable interrupt

Enable Interrupt

[Format] EI

[Operation] IE ← 1

[Operand]

Mnemonic	Operand
EI	None

[Flag]

V	IE	S	Z	AC	P	CY
	1					

[Description] Sets the IE flag to 1 and enables the maskable interrupt request. However, the interrupt is actually enabled after one instruction following the EI instruction has been executed.

[Example] POP R
EI

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
EI	None	1	1	1	1	1	0	1	1

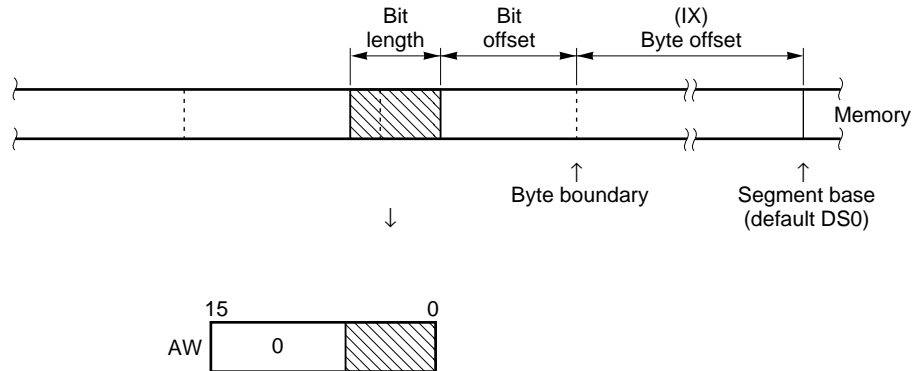
EXT

Extraction of bit field

Extract Bit Field

[Format] EXT dst, src

[Operation] AW ← 16-bit field



[Operand]

Mnemonic	Operand (dst, src)
EXT	reg8, reg8'
	reg8, imm4

[Flag]

V	S	Z	AC	P	CY
U	U	U	U	U	U

[Description] Loads bit field data of bit length specified by the source operand (src) from a memory area determined by the byte offset addressed by the IX register and bit offset specified by an 8-bit register described as the first operand, to the AW register. At this time, 0 is loaded to the remaining higher bits of the AW register.

After transfer, the IX register and the 8-bit register specified by the first operand are automatically updated as follows to indicate the next bit field:

```

reg8 ← reg8 + src + 1
if reg8 > 15 then
{
  reg8 ← reg8 - 16
  IX ← IX + 2
}
    
```

Only values from 0 to 15 are valid as the value of the 8-bit register described as the first operand to specify the bit offset (15 bits maximum). As the value of the source operand (src) that specifies the bit length (16 bits maximum), only 0 to 15 are valid, where 0 indicates 1-bit length and 15 indicates 16-bit length.

The bit field data can extend over a byte boundary of the memory.

As the source bit field, the default segment register is DS0. A segment override can be performed so that the bit field can be located in a segment specified by any register.

Caution: Clear the higher 4 bits of reg8 or reg8' to 0.

[Example]

- EXT CL, DL
- EXT CL, 8

[Bytes]

Mnemonic	Operand	Bytes
EXT	reg8, reg8'	3
	reg8, imm4	4

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
EXT	reg8, reg8'	0	0	0	0	1	1	1	1	0	0	1	1	0	0	1	1
		1	1	reg'				reg				—					
	reg8, imm4	0	0	0	0	1	1	1	1	0	0	1	1	1	0	1	1
		1	1	0	0	0	reg				imm4						

FINT [Added to V20/V30]

Interrupt
Finish Interrupt

[Format] FINT

[Operation] Indicates that the interrupt processing routine for the internal interrupt controller of the CPU has been terminated.

[Operand]

Mnemonic	Operand
FINT	None

[Flag]

V	S	Z	AC	P	CY

[Description] Resets the least significant bit of the ISPR register bits that are set, so that the internal interrupt controller is informed that interrupts except NMI, INT, and software interrupt have been terminated. Use this instruction immediately before an interrupt processing program, except when NMI, INT, or a software interrupt is terminated (i.e., immediately before the RETI or RETRBI instruction). Do not use this instruction in any other cases.

[Example] FINT

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
FINT	None	0	0	0	0	1	1	1	1	1	0	0	1	0	0	1	0

FPO1

Floating-point coprocessor control

Floating Point Operation 1

[Format] <1> FPO1 fp-op
 <2> FPO1 fp-op, mem

[Operand, operation]

Format <1>, <2>

Mnemonic	Operand	Operation
FPO1	fp-op	(SP - 1, SP - 2) ← PSW (SP - 3, SP - 4) ← PS (SP - 5, SP - 6) ← PC - x ^{Note} SP ← SP - 6
	fp-op, mem	IE ← 0, BRK ← 0 PC ← (01DH, 01CH) PS ← (01FH, 01EH)

Note x indicates the number of instruction bytes + number of prefixes.

[Flag]

V	IE	BRK	S	Z	AC	P	CY
	0	0					

[Description] Saves PSW, PS, and PC - x to the stack and resets the IE and BRK flags to 0. Next, loads the lower 2 bytes of the vector 7 of the interrupt vector table to the PC and the higher 2 bytes to the PS.

This instruction is provided to maintain compatibility with the V20/V30. With the V25/V35 family, this instruction has no functional meaning but only generates an interrupt.

- [Example]
- FPO1 010101010B
 - FPO1 0FFH
 - FPO1 6, BYTE PTR [IX]
 - FPO1 4, WORD_VAR

[Bytes]

Mnemonic	Operand	Bytes
FPO1	fp-op	2
	fp-op, mem	2-4

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
FPO1	fp-op	1	1	0	1	1	X	X	X	1	1	Y	Y	Y	Z	Z	Z
	fp-op, mem	1	1	0	1	1	X	X	X	mod	Y	Y	Y	mem			
		(disp-low)						(disp-high)									

FPO2

Floating-point coprocessor control

Floating Point Operation 2

[Format] <1> FPO2 fp-op
 <2> FPO2 fp-op, mem

[Operand, operation]

Format <1>, <2>

Mnemonic	Operand	Operation
FPO2	fp-op	(SP - 1, SP - 2) ← PSW (SP - 3, SP - 4) ← PS (SP - 5, SP - 6) ← PC - x ^{Note} SP ← SP - 6
	fp-op, mem	IE ← 0, BRK ← 0 PC ← (01DH, 01CH) PS ← (01FH, 01EH)

Note x indicates the number of instruction bytes + number of prefixes.

[Flag]

V	IE	BRK	S	Z	AC	P	CY
	0	0					

[Description] Saves PSW, PS, and PC - x to the stack and resets the IE and BRK flags to 0. Next, loads the lower 2 bytes of the vector 7 of the interrupt vector table to the PC and the higher 2 bytes to the PS.

This instruction is provided to maintain compatibility with the V20/V30. With the V25/V35 family, this instruction has no functional meaning but only generates an interrupt.

[Example]

- FPO2 010101010B
- FPO2 0FFH
- FPO2 0101B, BYTE PTR [IY]
- FPO2 1010B, WORD_VAR

[Bytes]

Mnemonic	Operand	Bytes
FPO2	fp-op	2
	fp-op, mem	2-4

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
FPO2	fp-op	0	1	1	0	0	1	1	X	1	1	Y	Y	Y	Z	Z	Z
	fp-op, mem	0	1	1	0	0	1	1	X	mod	Y	Y	Y	mem			
		(disp-low)						(disp-high)									

HALT	Halt Halt
-------------	----------------------------

[Format] **HALT**

[Operation] CPU Halt

[Operand]

Mnemonic	Operand
HALT	None

[Flag]

V	S	Z	AC	P	CY

[Description] Stops the CPU's clock and sets the CPU to standby mode.
Standby mode is released for the following reasons:

- Reset input
- Maskable interrupt request input
- Non-maskable interrupt request input

[Example] **HALT**

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
HALT	None	1	1	1	1	0	1	0	0

IN	Data input from I/O device Input
----	-------------------------------------

[Format] IN dst, src

[Operand, operation]

When $\overline{\text{IBRK}} = 1$

Mnemonic	Operand (dst, src)	Operation
IN	acc, imm8	[When W = 0] AL ← (imm8) [When W = 1] AH ← (imm8 + 1), AL ← (imm8)
	acc, DW	[When W = 0] AL ← (DW) [When W = 1] AH ← (DW + 1), AL ← (DW)

When $\overline{\text{IBRK}} = 0$

Mnemonic	Operand (dst, src)	Operation
IN	acc, imm8	(SP - 1, SP - 2) ← PSW (SP - 3, SP - 4) ← PS (SP - 5, SP - 6) ← PC - x ^{Note}
	acc, DW	IE ← 0, BRK ← 0, $\overline{\text{IBRK}} \leftarrow 1$ PC ← (04DH, 04CH) PS ← (04FH, 04EH)

Note x indicates the number of instruction bytes + number of prefixes.

[Flag] When $\overline{\text{IBRK}} = 1$

V	IE	BRK	S	Z	AC	P	$\overline{\text{IBRK}}$	CY

When $\overline{\text{IBRK}} = 0$

V	IE	BRK	S	Z	AC	P	$\overline{\text{IBRK}}$	CY
	0	0					1	

[Description] Transfers the contents of the I/O device specified by the source operand (src) to the accumulator (AL or AW register) specified by the destination operand (dst) when $\overline{\text{IBRK}} = 1$.

When $\overline{\text{IBRK}} = 0$, saves PSW, PS, and PC – x to the stack, resets the IE and BRK flags to 0, and sets IBRK to 1.

Next, loads the lower 2 bytes of vector 19 of the interrupt vector table to the PC, and the higher 2 bytes to the PS.

This function is provided to facilitate transplantation of software.

[Example] To transfer the contents of the port address 0DAH to the AL register
 MOV DW, 0DAH
 IN AL, DW

[Bytes]

Mnemonic	Operand	Bytes
IN	acc, imm8	2
	acc, DW	1

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
IN	acc, imm8	1	1	1	0	0	1	0	W	imm8							
	acc, DW	1	1	1	0	1	1	0	W	—							

<h1 style="margin: 0;">INC</h1>	Increment Increment
---------------------------------	------------------------

[Format] **INC dst**

[Operation] $dst \leftarrow dst + 1$

[Operand]

Mnemonic	Operand (dst)
INC	reg8
	mem
	reg16

[Flag]

V	S	Z	AC	P	CY
×	×	×	×	×	

[Description] Increments (+1) the contents of the destination operand (dst).

- [Example]**
- INC DW
 - INC BP
 - INC SP

[Bytes]

Mnemonic	Operand	Bytes
INC	reg8	2
	mem	2-4
	reg16	1

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
INC	reg8	1	1	1	1	1	1	1	0	1	1	0	0	0	reg		
	mem	1	1	1	1	1	1	1	W	mod	0	0	0	mem			
		(disp-low)							(disp-high)								
	reg16	0	1	0	0	0	reg		—								

INM

I/O-memory block transfer
Input Multiple

[Format] (repeat) INM [DS1-spec:] dst-block, DW

[Operation] When $\overline{\text{IBRK}} = 1$ [When $W = 0$] $(\text{IY}) \leftarrow (\text{DW})$
 DIR = 0: $\text{IY} \leftarrow \text{IY} + 1$
 DIR = 1: $\text{IY} \leftarrow \text{IY} - 1$
 [When $W = 1$] $(\text{IY} + 1, \text{IY}) \leftarrow (\text{DW} + 1, \text{DW})$
 DIR = 0: $\text{IY} \leftarrow \text{IY} + 2$
 DIR = 1: $\text{IY} \leftarrow \text{IY} - 2$

When $\overline{\text{IBRK}} = 0$ $(\text{SP} - 1, \text{SP} - 2) \leftarrow \text{PSW}$
 $(\text{SP} - 3, \text{SP} - 4) \leftarrow \text{PS}$
 $(\text{SP} - 5, \text{SP} - 6) \leftarrow \text{PC} - x^{\text{Note}}$
 $\text{IE} \leftarrow 0, \text{BRK} \leftarrow 0, \overline{\text{IBRK}} \leftarrow 1$
 $\text{PC} \leftarrow (04\text{DH}, 04\text{CH})$
 $\text{PS} \leftarrow (04\text{FH}, 04\text{EH})$

Note: x indicates the number of instruction bytes + number of prefixes.

[Operand]

Mnemonic	Operand
INM	[DS1-spec:] dst-block, DW

[Flag] When $\overline{\text{IBRK}} = 1$

V	IE	BRK	S	Z	AC	P	$\overline{\text{IBRK}}$	CY

When $\overline{\text{IBRK}} = 0$

V	IE	BRK	S	Z	AC	P	$\overline{\text{IBRK}}$	CY
	0	0					1	

[Description] Transfers the contents of the register of the I/O device addressed by the DW register to the memory addressed by the IY register when $\overline{\text{IBRK}} = 1$. The number of times the transfer is repeated is controlled by the repeat prefix of the REP instruction, which is used in conjunction with this instruction. Although the contents of the DW register (address of I/O device) are fixed, the contents of the IY register are automatically incremented (+1/+2) or decremented (-1/-2) for the next byte/word transfer each time 1-byte/word data has been transferred. The direction of the block is determined by the status of the DIR flag.

Whether data is transferred in byte or word units is determined by the attribute of the operand. The INM instruction is used in conjunction with the REP instruction of the repeat prefix. The destination block must always be located in the segment specified by the DS1 register. Segment overrides are not allowed.

When $\overline{\text{IBRK}} = 0$, saves PSW, PS, and PC - x to the stack, resets the IE and BRK flags to 0, and sets IBRK to 1.

Next, loads the lower 2 bytes of vector 19 of the interrupt vector table to the PC, and the higher 2 bytes to the PS.

This function is provided to facilitate transplantation of software.

- [Example]**
- To load the contents of port address 0DAH (byte data) to memory work area


```
MOV AW, 0
MOV DS1, AW
MOV IY, 50H
MOV DW, 0DAH
INM DS1: BYTE PTR [IY], DW
```
 - To load the contents of port address 0DAH (byte data) to memory 0:0H-0:FFH


```
MOV AW, 0
MOV DS1, AW
MOV IY, 0
MOV DW, 0DAH
MOV CW, 0FFH
REP INM DS1: BYTE PTR [IY], DW
```

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
INM	[DS1-spec:] dst-block, DW	0	1	1	0	1	1	0	W

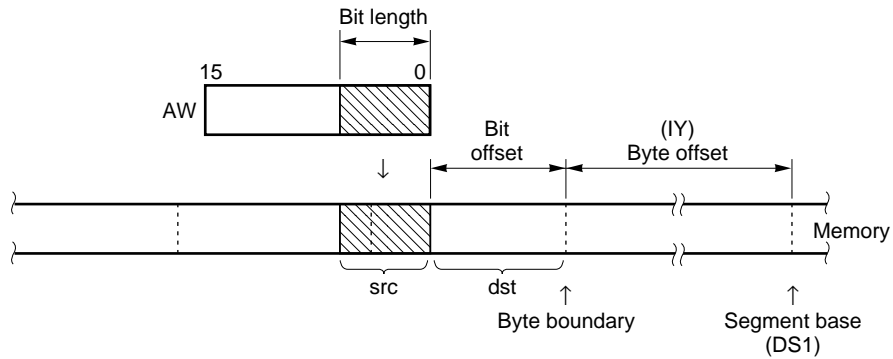
INS

Insertion of bit field

Insert Bit Field

[Format] **INS dst, src**

[Operation] 16-bit field ← AW



[Operand]

Mnemonic	Operand (dst, src)
INS	reg8, reg8'
	reg8, imm4

[Flag]

V	S	Z	AC	P	CY
U	U	U	U	U	U

[Description] Of the 16-bit data of the AW register, transfers the lower bit data of the length specified by the source operand (src) to a memory area determined by the byte offset addressed by the DS1 and IY registers and bit offset specified by an 8-bit register described as the first operand.

After completion of the transfer, the IY register and the 8-bit register specified by the first operand are automatically updated as follows to indicate the next bit field:

```

reg8 ← reg8 + src + 1
if reg8 > 15 then
{
  reg8 ← reg8 - 16
  IY ← IY + 2
}
    
```


Only the values 0 to 15 are valid as the value of the 8-bit register described as the first operand to specify the bit offset (15 bits maximum). As the value of the source operand (src) that specifies the bit length (16 bits maximum), only 0 to 15 are valid, where 0 indicates 1-bit length and 15 indicates 16-bit length.

The bit field data can extend over a byte boundary of the memory.

The destination bit field must always be located in a segment specified by the DS1 register.

Segment overrides are not allowed.

Caution: Clear the higher 4 bits of reg8 or reg8' to 0.

- [Example]**
- INS DL, CL
 - INS DL, 12

[Bytes]

Mnemonic	Operand	Bytes
INS	reg8, reg8'	3
	reg8, imm4	4

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
INS	reg8, reg8'	0	0	0	0	1	1	1	1	0	0	1	1	0	0	0	1
		1	1	reg'				reg				—					
	reg8, imm4	0	0	0	0	1	1	1	1	0	0	1	1	1	0	0	1
		1	1	0	0	0	reg			imm4							

LDEA

Load effective address
Load Effective Address

[Format] LDEA reg16, mem16

[Operation] reg16 ← mem16

[Operand]

Mnemonic	Operand
LDEA	reg16, mem16

[Flag]

V	S	Z	AC	P	CY

[Description] Loads an effective address (offset) generated by the second operand to a 16-bit general-purpose register specified by the first operand.

This instruction is used to set the first value of an operand address to a register automatically used to specify an operand by the TRANS or primitive block transfer instruction.

[Example] To load the offset of the effective address of procedure INT_PROC to AW register

```
LDEA AW, INT_PROC
LDEA AW, [BP] VAR01 + 2
```

[Bytes] 2-4

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
LDEA	reg16, mem16	1	0	0	0	1	1	0	1	mod			reg				mem
		(disp-low)							(disp-high)								

LDM
LDMB
LDMW

Block load
Load Multiple
Load Multiple Byte
Load Multiple Word

[Format] (repeat) LDM [Seg-spec:] src-block
 (repeat) LDMB
 (repeat) LDMW

[Operation] [When W = 0] $AL \leftarrow (IX)$
 DIR = 0: $IX \leftarrow IX + 1$
 DIR = 1: $IX \leftarrow IX - 1$
 [When W = 1] $AW \leftarrow (IX + 1, IX)$
 DIR = 0: $IX \leftarrow IX + 2$
 DIR = 1: $IX \leftarrow IX - 2$

[Operand]

Mnemonic	Operand
LDM	[Seg-spec:] src-block
LDMB	None
LDMW	

[Flag]

V	S	Z	AC	P	CY

[Description] Repeatedly transfer a block addressed by the IX register to the accumulator (AL/AW) in byte or word units.

The contents of the IX register are automatically incremented (+1/+2) or decremented (−1/−2) for the next byte/word each time 1-byte/word data has been transferred. The direction of the block is determined by the status of the DIR flag.

Whether the transfer is carried out in byte or word units is specified by the attribute of the operand when the LDM instruction is used. When the LDMB or LDMW instruction is used, the byte or word type is directly specified.

As the source block, the default segment register is DS0. However, segment overrides are allowed, and the source block can be located in a segment specified by any segment register.

[Example]

- REP LDM DS1: BYTE_VAR ; DS1 segment
- REP LDMB ; DS0 segment

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
LDM	[Seg-spec:] src-block	1	0	1	0	1	1	0	W
LDMB	None								
LDMW									

MOV

Data transfer
Move

[Format] <1> MOV dst, src
 <2> MOV dst1, dst2, src

[Operand, operation]

Format <1>

Mnemonic	Operand (dst, src)	Operation
MOV	reg, reg'	dst ← src
	mem, reg	
	reg, mem	
	mem, imm	
	reg, imm	
	acc, dmem	[When W = 0] AL ← (dmem) [When W = 1] AH ← (dmem + 1) AL ← (dmem)
	dmem, acc	[When W = 0] (dmem) ← AL [When W = 1] (dmem + 1) ← AH, (dmem) ← AL
	sreg, reg16	dst ← src
	sreg, mem16	
	reg16, sreg	
	mem16, sreg	
	AH, PSW	AH ← S, Z, F1, AC, F0, P, $\overline{\text{IBRK}}$, CY
	PSW, AH	S, Z, F1, AC, F0, P, $\overline{\text{IBRK}}$, CY ← AH

Format <2>

Mnemonic	Operand (dst1, dst2, src)	Operation
MOV	DS0, reg16, mem32	reg16 ← (mem32) DS0 ← (mem32 + 2)
	DS1, reg16, mem32	reg16 ← (mem32) DS1 ← (mem32 + 2)

[Flag] When operand is PSW, AH

V	S	Z	F1	AC	F0	P	$\overline{\text{IBRK}}$	CY
	x	x	x	x	x	x	x	x

Other than above

V	S	Z	F1	AC	F0	P	$\overline{\text{IBRK}}$	CY

[Description] Format <1> : Transfers the contents of the source operand (src) specified as the second operand to the destination operand (dst) specified as the first operand.
 If the operand is AH, PSW, the S, Z, F1, AC, F0, P, $\overline{\text{IBRK}}$, and CY flags are transferred to the AH register.
 If the operand is PSW, AH, bits 0-7 of the AH register are transferred to the S, Z, F1, AC, F0, P, $\overline{\text{IBRK}}$ and CY flags of the PSW, respectively.

Caution When $\text{dst} = \text{sreg}$ or $\text{src} = \text{sreg}$, the hardware interrupt requests (maskable interrupt and non-maskable interrupt) and single-step breaks cannot be accepted after this instruction is executed and before the next instruction is executed.

Format <2> : Transfers the lower 16 bits (offset word of a 32-bit pointer variable) of the 32-bit memory addressed by the source operand (src) to a 16-bit register specified by the destination operand 2 (dst2), and the higher 16 bits (segment word) to a segment register (DS0 or DS1) specified by the destination operand 1 (dst1).

[Example] To write 55H to memory 0:50H
 MOV AW, 0
 MOV DS1, AW
 MOV IY, 50H
 MOV DL, 55H
 MOV DS1: [IY], DL

[Bytes]

Mnemonic	Operand	Bytes
MOV	reg, reg'	2
	mem, reg	2-4
	reg, mem	
	mem, imm	3-6
	reg, imm	2, 3
	acc, dmem	3
	dmem, acc	
	sreg, reg16	2
	sreg, mem16	2-4
	reg16, sreg	2
	mem16, sreg	2-4
	DS0, reg16, mem32	
	DS1, reg16, mem32	
	AH, PSW	1
	PSW, AH	

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
MOV	reg, reg'	1	0	0	0	1	0	1	W	1	1		reg		reg'		
	mem, reg	1	0	0	0	1	0	0	W	mod		reg		mem			
		(disp-low)								(disp-high)							
	reg, mem	1	0	0	0	1	0	1	W	mod		reg		mem			
		(disp-low)								(disp-high)							
	mem, imm	1	1	0	0	0	1	1	W	mod	0	0	0		mem		
		(disp-low)								(disp-high)							
		imm8 or imm16-low								imm16-high							
	reg, imm	1	0	1	1	W			reg	imm8 or imm16-low							
		imm16-high								—							
	acc, dmem	1	0	1	0	0	0	0	W	addr-low							
		addr-high								—							
	dmem, acc	1	0	1	0	0	0	1	W	addr-low							
		addr-high								—							
	sreg, reg16	1	0	0	0	1	1	1	0	1	1	0	sreg		reg		
	sreg, mem16	1	0	0	0	1	1	1	0	mod	0	sreg		mem			
		(disp-low)								(disp-high)							
	sreg16, sreg	1	0	0	0	1	1	0	0	1	1	0	sreg		reg		
	mem16, sreg	1	0	0	0	1	1	0	0	mod	0	sreg		mem			
		(disp-low)								(disp-high)							
DS0, reg16, mem32	1	1	0	0	0	1	0	1	mod		reg		mem				
	(disp-low)								(disp-high)								
DS1, reg16, mem32	1	1	0	0	0	1	0	0	mod		reg		mem				
	(disp-low)								(disp-high)								
AH, PSW	1	0	0	1	1	1	1	1	—								
PSW, AH	1	0	0	1	1	1	1	0	—								

MOVBK

MOVBKB

MOVBKW

Block transfer
Move Block
Move Block Byte
Move Block Word

[Format] (repeat) **MOVBK** [DS1-spec:] **dst-block**, [Seg-spec:] **src-block**
 (repeat) **MOVBKB**
 (repeat) **MOVBKW**

[Operation] [When W = 0] $(IY) \leftarrow (IX)$
 DIR = 0: $IX \leftarrow IX + 1, IY \leftarrow IY + 1$
 DIR = 1: $IX \leftarrow IX - 1, IY \leftarrow IY - 1$
 [When W = 1] $(IY + 1, IY) \leftarrow (IX + 1, IX)$
 DIR = 0: $IX \leftarrow IX + 2, IY \leftarrow IY + 2$
 DIR = 1: $IX \leftarrow IX - 2, IY \leftarrow IY - 2$

[Operand]

Mnemonic	Operand
MOVBK	[DS1-spec:] dst-block , [Seg-spec:] src-block
MOVBKB	None
MOVBKW	None

[Flag]

V	S	Z	AC	P	CY

[Description] Repeatedly transfers a block addressed by the IX register to a block addressed by the IY register in byte or word units.

The contents of the IX and IY registers are automatically incremented (+1/+2) or decremented (-1/-2) for the next byte/word transfer each time 1-byte/word data has been transferred. The direction of the block is determined by the status of the DIR flag.

Whether a transfer is carried out in byte or word units is specified by the attribute of the operand when the MOVBK instruction is used. When the MOVBKB or MOVBKW instruction is used, the byte or word type is directly specified.

The destination block must always be located in a segment specified by the DS1 register. Segment overrides are not allowed.

As the source block, the default segment register is DS0. Overrides are allowed, and the source block can be located in a segment specified by any segment register.

[Example] **MOVBK** BYTE_VAR1, BYTE_VAR2
MOVBK WORD_VAR1, MORD_VAR2

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
MOVBK	[DS1-spec:] dst-block [Seg-spec:] src-block	1	0	1	0	0	1	0	W
MOVBKB	None								
MOVBKW									

MOVSPA [Added to V20/V30]

Data transfer

Move Stack Pointer After Context Switch

[Format] MOVSPA**[Operation]** SS, SP of new register bank ← SS, SP of old register bank**[Operand]**

Mnemonic	Operand
MOVSPA	None

[Flag]

V	S	Z	AC	P	CY

[Description] Transfers the value of SS before the register bank is switched to SS of the newly selected register bank. Likewise, the value of SP is transferred.

This instruction is used when the register bank is switched by the BRKCS instruction or an interrupt request, or when the stack is to be successively used immediately after register bank switching.

[Example] MOVSPA**[Bytes]** 2**[Word Format]**

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
MOVSPA	None	0	0	0	0	1	1	1	1	0	0	1	0	0	1	0	1

MOVSPB [Added to V20/V30]

Data transfer

Move Stack Pointer Before Task Switch

[Format] MOVSPB reg16

[Operation] SS, SP of new register bank indicated by reg16 ← SS, SP of old register bank

[Operand]

Mnemonic	Operand
MOVSPB	reg16

[Flag]

V	S	Z	AC	P	CY

[Description] Transfers the value of SS in the current register bank to SS of the new register bank indicated by the lower 3 bits of the 16-bit register described as the operand. Likewise, the value of SP is transferred.

This instruction is used to switch the register bank with the TSKSW instruction, or to use the stack successively before and after switching.

[Example] MOVSPB AW

[Bytes] 3

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
MOVSPB	reg16	0	0	0	0	1	1	1	1	1	0	0	1	0	1	0	1
		1	1	1	1	1	reg			—							

MUL**Signed multiplication****Multiply Signed**

- [Format]**
- <1> **MUL src**
 - <2> **MUL dst, src**
 - <3> **MUL dst, src1, src2**

[Operand, operation]

Format <1>

Mnemonic	Operand	Operation
MUL	reg8	$AW \leftarrow AL \times src$
	mem8	AH = AL sign extension: $CY \leftarrow 0, V \leftarrow 0$ AH \neq AL sign extension: $CY \leftarrow 1, V \leftarrow 1$
	reg16	$DW, AW \leftarrow AW \times src$
	mem16	DW = AW sign extension: $CY \leftarrow 0, V \leftarrow 0$ DW \neq AW sign extension: $CY \leftarrow 1, V \leftarrow 1$

Format <2>

Mnemonic	Operand	Operation
MUL	reg16, imm8	$dst \leftarrow dst \times src$
	reg16, imm16	Product \leq 16 bits: $CY \leftarrow 0, V \leftarrow 0$ Product > 16 bits: $CY \leftarrow 1, V \leftarrow 1$

Format <3>

Mnemonic	Operand	Operation
MUL	reg16, reg16', imm8	$dst \leftarrow src1 \times src2$
	reg16, mem16, imm8	Product \leq 16 bits: $CY \leftarrow 0, V \leftarrow 0$
	reg16, reg16', imm16	Product > 16 bits: $CY \leftarrow 1, V \leftarrow 1$
	reg16, mem16, imm16	

[Flag]

V	S	Z	AC	P	CY
×	U	U	U	U	×

- [Description]** Format <1> :
- When src = reg8 or src = mem8
Multiplies the value of the AL register by the source operand (src) with sign, and stores the double-length result to the AW register. At this time, if the higher half of the result (AH register) is not the sign extension of the lower half (AL register), the CY and V flags are set to 1. The AH register is an extended register.
 - When src = reg16 or src = mem16
Multiplies the value of the AW register by the source operand (src) with sign, and stores the double-length result to the AW and DW registers. At this time, if the higher half of the result (DW register) is not the sign extension of the lower half (AW register), the CY and V flags are set to 1. The DW register is an extended register.

Format <2> : Multiplies the destination operand (dst) by the source operand (src) with sign, and stores the result to the destination operand (dst).

Format <3> : Multiplies the first source operand (src1) by the second source operand (src2) with sign, and stores the result to the destination operand (dst).

[Example] To multiply the value of the AW register by the contents of memory 0:50H (word data)

```
MOV BW, 0
MOV DS0, BW
MOV IX, 50H
MUL WORD PTR [IX]
```

[Bytes]

Mnemonic	Operand	Bytes
MUL	reg8	2
	mem8	2-4
	reg16	2
	mem16	2-4
	reg16, imm8	3
	reg16, imm16	4
	reg16, reg16', imm8	3
	reg16, mem16, imm8	3-5
	reg16, reg16', imm16	4
	reg16, mem16, imm16	4-6

[Word Format]

Mnemonic	Operand	Operation code																
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
MUL	reg8	1	1	1	1	0	1	1	0	1	1	1	0	1			reg	
	mem8	1	1	1	1	0	1	1	0	mod	1	0	1				mem	
		(disp-low)									(disp-high)							
	reg16	1	1	1	1	0	1	1	1	1	1	1	0	1			reg	
	mem16	1	1	1	1	0	1	1	1	mod	1	0	1				mem	
		(disp-low)									(disp-high)							
	reg16, imm8	0	1	1	0	1	0	1	1	1	1			reg			reg'	
		imm8									—							
	reg16, imm16	0	1	1	0	1	0	0	1	1	1			reg			reg'	
		imm16-low									imm16-high							
	reg16, reg16', imm8	0	1	1	0	1	0	1	1	1	1			reg			reg'	
		imm8									—							
	reg16, mem16, imm8	0	1	1	0	1	0	1	1	mod			reg				mem	
		(disp-low)									(disp-high)							
		imm8									—							
	reg16, reg16', imm16	0	1	1	0	1	0	0	1	1	1			reg			reg'	
imm16-low										imm16-high								
reg16, mem16, imm16	0	1	1	0	1	0	0	1	mod			reg				mem		
	(disp-low)									(disp-high)								
	imm16-low									imm16-high								

MULU

Unsigned multiplication

Multiply Unsigned

[Format] MULU src**[Operand, operation]**

Mnemonic	Operand (src)	Operation
MULU	reg8	$AW \leftarrow AL \times src$
	mem8	AH = 0: $CY \leftarrow 0, V \leftarrow 0$ AH \neq 0: $CY \leftarrow 1, V \leftarrow 1$
	reg16	$DW, AW \leftarrow AW \times src$
	mem16	DW = 0: $CY \leftarrow 0, V \leftarrow 0$ DW \neq 0: $CY \leftarrow 1, V \leftarrow 1$

[Flag]

V	S	Z	AC	P	CY
×	U	U	U	U	×

[Description]

- When src = reg8 or src = mem8
Multiplies the value of the AL register by the source operand (src) without sign, and stores the double-length result to the AW register. If the higher half of the result (AH register) is not zero at this time, the CY and V flags are set to 1. The AH register is an extended register.
- When src = reg16 or src = mem16
Multiplies the value of the AW register by the source operand (src) without sign, and stores the double-length result to the AW and DW registers. If the higher half of the result (DW register) is not zero at this time, the CY and V flags are set to 1. The DW register is an extended register.

[Example]

To multiply contents of AL register by contents of CL register
MULU CL

[Bytes]

Mnemonic	Operand	Bytes
MULU	reg8	2
	mem8	2-4
	reg16	2
	mem16	2-4

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
MULU	reg8	1	1	1	1	0	1	1	0	1	1	1	0	0			reg
	mem8	1	1	1	1	0	1	1	0	mod	1	0	0			mem	
		(disp-low)							(disp-high)								
	reg16	1	1	1	1	0	1	1	1	1	1	1	0	0		reg	
	mem16	1	1	1	1	0	1	1	1	mod	1	0	0			mem	
		(disp-low)							(disp-high)								

NEG

2's complement

Negate

[Format] **NEG dst**

[Operation] $dst \leftarrow \overline{dst} + 1$

[Operand]

Mnemonic	Operand (dst)
NEG	reg
	mem

[Flag]

V	S	Z	AC	P	CY
×	×	×	×	×	Note

Note CY = 1. CY = 0 when dst is 0 before execution.

[Description] Obtains 2's complement of the contents of the destination operand (dst).

- [Example]
- NEG DL
 - NEG CW
 - NEG IX
 - NEG BP

[Bytes]

Mnemonic	Operand	Bytes
NEG	reg	2
	mem	2-4

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
NEG	reg	1	1	1	1	0	1	1	W	1	1	0	1	1			reg
	mem	1	1	1	1	0	1	1	W	mod	0	1	1			mem	
			(disp-low)							(disp-high)							

NOP

No operation
No Operation

[Format] **NOP**

[Operation] No operation

[Operand]

Mnemonic	Operand
NOP	None

[Flag]

V	S	Z	AC	P	CY

[Description] Executes nothing but takes 4 clocks.

[Example] NOP

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
NOP	None	1	0	0	1	0	0	0	0

<h1>NOT</h1>	Logical not Not
--------------	--------------------------------------

[Format] NOT dst

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

[Operand]

Mnemonic	Operand (dst)
NOT	reg
	mem

[Flag]

V	S	Z	AC	P	CY

[Description] Inverts a bit specified by the destination operand (dst) (logical NOT), and stores the result to the destination operand (dst).

- [Example]**
- NOT AL
 - NOT CW
 - NOT IX

[Bytes]

Mnemonic	Operand	Bytes
NOT	reg	2
	mem	2-4

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
NOT	reg	1	1	1	1	0	1	1	W	1	1	0	1	0			reg
	mem	1	1	1	1	0	1	1	W	mod	0	1	0			mem	
	(disp-low)								(disp-high)								

NOT1

Inverts bit
Not Bit

[Format] <1> NOT1 dst, src
<2> NOT1 dst

[Operation] Format <1> : bit n of dst (n is specified by src) \leftarrow $\overline{\text{bit n of dst (n is specified by src)}}$
Format <2> : $\text{dst} \leftarrow \overline{\text{dst}}$

[Operand] Format <1>

Mnemonic	Operand (dst, src)
NOT1	reg8, CL
	mem8, CL
	reg16, CL
	mem16, CL
	reg8, imm3
	mem8, imm3
	reg16, imm4
	mem16, imm4

Format <2>

Mnemonic	Operand (dst)
NOT1	CY

[Flag] Format <1>

V	S	Z	AC	P	CY

Format <2>

V	S	Z	AC	P	CY
					x

[Description] Format <1> : Logically NOTs bit n (n is the contents of the source operand (src) specified by the second operand) of the destination operand (dst) specified by the first operand, and stores the result to the destination operand (dst).
 When the operands are reg8, CL or mem8, CL, only the lower 3 bits (0-7) of the value of CL are valid.
 When the operands are reg16, CL or mem16, CL, only the lower 4 bits (0-15) of the value of CL are valid.
 When the operand is reg8, imm3, only the lower 3 bits of the immediate data at the fourth byte position of the instruction are valid.
 When the operand is mem8, imm3, only the lower 3 bits of the immediate data at the end byte position of the instruction are valid.
 When the operand is reg16, imm4, only the lower 4 bits of the immediate data at the fourth byte position of the instruction are valid.
 When the operand is mem16, imm4, only the lower 4 bits of the immediate data at the end byte position of the instruction are valid.

Format <2> : Logically NOTs the content of the CY flag and stores the result to the CY flag.

[Example] IN AL, 0
 NOT1 AL, 7

[Bytes]

Mnemonic	Operand	Bytes
NOT1	reg8, CL	3
	mem8, CL	3-5
	reg16, CL	3
	mem16, CL	3-5
	reg8, imm3	4
	mem8, imm3	4-6
	reg16, imm4	4
	mem16, imm4	4-6
	CY	1

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
NOT1	reg8, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	1	1	0
		1	1	0	0	0	reg			—							
	mem8, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	1	1	0
		mod	0	0	0	mem			(disp-low)								
		(disp-high)								—							
	reg16, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	1	1	1
		1	1	0	0	0	reg			—							
	mem16, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	1	1	1
		mod	0	0	0	mem			(disp-low)								
		(disp-high)								—							
	reg8, imm3	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0
		1	1	0	0	0	reg			imm3							
	mem8, imm3	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0
		mod	0	0	0	mem			(disp-low)								
		(disp-high)								imm3							
	reg16, imm4	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	1
		1	1	0	0	0	reg			imm4							
	mem16, imm4	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	1
		mod	0	0	0	mem			(disp-low)								
		(disp-high)								imm4							
	CY	—															

OR	Logical sum Or
-----------	-------------------

[Format] **OR dst, src**

[Operand, operation]

Mnemonic	Operand (dst, src)	Operation
OR	reg, reg'	$dst \leftarrow dst \vee src$
	mem, reg	
	reg, mem	
	reg, imm	
	mem, imm	
	acc, imm	[When W = 0] $AL \leftarrow AL \vee imm8$ [When W = 1] $AW \leftarrow AW \vee imm16$

[Flag]

V	S	Z	AC	P	CY
0	×	×	U	×	0

[Description] ORs the destination operation (dst) specified by the first operand with the source operand (src) specified by the second operand, and stores the result to the destination operand (dst).

[Example] OR AW, WORD PTR [IX]

[Bytes]

Mnemonic	Operand	Bytes
OR	reg, reg'	2
	mem, reg	2-4
	reg, mem	2-4
	reg, imm	3, 4
	mem, imm	3-6
	acc, imm	2, 3

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
OR	reg, reg'	0	0	0	0	1	0	1	W	1	1	reg			reg'		
	mem, reg	0	0	0	0	1	0	0	W	mod		reg			mem		
		(disp-low)								(disp-high)							
	reg, mem	0	0	0	0	1	0	1	W	mod		reg			mem		
		(disp-low)								(disp-high)							
	reg, imm ^{Note}	1	0	0	0	0	0	0	W	1	1	0	0	1	reg		
		imm8 or imm16-low								imm16-high							
	mem, imm	1	0	0	0	0	0	0	W	mod		0	0	1	mem		
		(disp-low)								(disp-high)							
		imm8 or imm16-low								imm16-high							
	acc, imm	0	0	0	0	1	1	0	W	imm8 or imm16-low							
		imm16-high								—							

Note With some assemblers and compilers, the codes shown below may be generated.

Operation code															
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	W	1	1	0	0	1	reg		
imm8								—							

Even in this case, instructions are executed normally. Take precautions, however, since some emulators do not support the disassembly function or line assembly function for this instruction.

<h1>OUT</h1>	Data output to I/O device Output
--------------	--

[Format] **OUT dst, src**

[Operand, operation]

When $\overline{\text{IBRK}} = 1$

Mnemonic	Operand (dst, src)	Operation
OUT	imm8, acc	[When W = 0] (imm8) ← AL [When W = 1] (imm8 + 1) ← AH, (imm8) ← AL
	DW, acc	[When W = 0] (DW) ← AL [When W = 1] (DW + 1) ← AH, (DW) ← AL

When $\overline{\text{IBRK}} = 0$

Mnemonic	Operand (dst, src)	Operation
OUT	imm8, acc	(SP - 1, SP - 2) ← PSW (SP - 3, SP - 4) ← PS (SP - 5, SP - 6) ← PC - x ^{Note}
	DW, acc	IE ← 0, BRK ← 0, $\overline{\text{IBRK}} \leftarrow 1$ PC ← (04DH, 04CH) PS ← (04FH, 04EH)

Note: x indicates the number of instruction bytes + number of prefixes.

[Flag] When $\overline{\text{IBRK}} = 1$

V	IE	BRK	S	Z	AC	P	$\overline{\text{IBRK}}$	CY

When $\overline{\text{IBRK}} = 0$

V	IE	BRK	S	Z	AC	P	$\overline{\text{IBRK}}$	CY
	0	0					1	

[Description] Transfers the contents of the accumulator (AL or AW register) to the register of an I/O device specified by the destination operand (dst) when $\overline{\text{IBRK}} = 1$.

When $\overline{\text{IBRK}} = 0$, saves PSW, PS, and PC – x to the stack, resets the IE and BRK flags to 0, and sets IBRK to 1.

Next, loads the lower 2 bytes of vector 19 of the interrupt vector table to the PC, and the higher 2 bytes to the PS.

This function is provided to facilitate transplantation of software.

[Example] To transfer the contents of the AL register to port address 0D8H
 MOV DW, 0D8H
 OUT DW, AL

[Bytes]

Mnemonic	Operand	Bytes
OUT	imm8, acc	2
	DW, acc	1

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
OUT	imm8, acc	1	1	1	0	0	1	1	W	imm8							
	DW, acc	1	1	1	0	1	1	1	W	—							

OUTM

Memory-I/O block transfer

Output Multiple

[Format] (repeat) **OUTM DW, [Seg-spec:] src-block**

[Operation] When $\overline{\text{IBRK}} = 1$ [When $W = 0$] $(\text{DW}) \leftarrow (\text{IX})$
 $\text{DIR} = 0: \text{IX} \leftarrow \text{IX} + 1$
 $\text{DIR} = 1: \text{IX} \leftarrow \text{IX} - 1$
 [When $W = 1$] $(\text{DW} + 1, \text{DW}) \leftarrow (\text{IX} + 1, \text{IX})$
 $\text{DIR} = 0: \text{IX} \leftarrow \text{IX} + 2$
 $\text{DIR} = 1: \text{IX} \leftarrow \text{IX} - 2$

When $\overline{\text{IBRK}} = 0$ $(\text{SP} - 1, \text{SP} - 2) \leftarrow \text{PSW}$
 $(\text{SP} - 3, \text{SP} - 4) \leftarrow \text{PS}$
 $(\text{SP} - 5, \text{SP} - 6) \leftarrow \text{PC} - x^{\text{Note}}$
 $\text{IE} \leftarrow 0, \text{BRK} \leftarrow 0, \overline{\text{IBRK}} \leftarrow 1$
 $\text{PC} \leftarrow (04\text{DH}, 04\text{CH})$
 $\text{PS} \leftarrow (04\text{FH}, 04\text{EH})$

Note x indicates the number of instruction bytes + number of prefixes.

[Operand]

Mnemonic	Operand
OUTM	DW, [Seg-spec:] src-block

[Flag] When $\overline{\text{IBRK}} = 1$

V	IE	BRK	S	Z	AC	P	$\overline{\text{IBRK}}$	CY

When $\overline{\text{IBRK}} = 0$

V	IE	BRK	S	Z	AC	P	$\overline{\text{IBRK}}$	CY
	0	0					1	

[Description] Transfers the contents of the memory addressed by the IX register to the register of an I/O device addressed by the DW register when $\overline{\text{IBRK}} = 1$.

The number of times a transfer is to be executed is controlled by the repeat prefix REP instruction, which is used in conjunction with this instruction. When data is transferred repeatedly, the contents of the DW register (address of the I/O device) are fixed, but the contents of the IX register are automatically incremented (+1/+2) or decremented (-1/-2) for the next byte/word transfer each time 1-byte/word data has been transferred. The direction of the block is determined by the DIR flag. Whether data is transferred in byte or word units is specified by the attribute of the operand. The OUTM instruction is used in pairs with the repeat prefix REP instruction. As the source block, the default segment register is DS0. Segment overrides are allowed, and the source block can be located in a segment specified by any segment register.

When $\overline{\text{IBRK}} = 0$, saves PSW, PS, and PC - x to the stack, resets the IE and BRK flags to 0, and sets $\overline{\text{IBRK}}$ to 1.

Next, loads the lower 2 bytes of vector 19 of the interrupt vector table to the PC, and the higher 2 bytes to the PS.

This function is provided to facilitate transplantation of software.

[Example]

- To transfer the contents of memory 0:50H to port address 0D8H (byte data)


```
MOV AW, 0
MOV DS0, AW
MOV IX, 50H
MOV DW, 0D8H
OUTM DW, DS0: WORD PTR [IX]
```
- To transfer the contents of memory 0:0H to 0FFH to port address 0D8H (byte data)


```
MOV AW, 0
MOV DS0, AW
MOV IX, 0H
MOV DW, 0D8H
MOV CW, 0FFH
REP OUTM DW, DS0: PTR [IX]
```

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
OUTM	DW, [Seg-spec:] src-block	0	1	1	0	1	1	1	W

POLL

Floating-point coprocessor wait

Poll and Wait

[Format] POLL**[Operation]** POLL and wait**[Operand]**

Mnemonic	Operand
POLL	None

[Flag]

V	S	Z	AC	P	CY

[Description] Places the CPU in the wait status until the $\overline{\text{POLL}}$ pin becomes active (low level).

- Cautions**
1. The BUSLOCK instruction must not be placed immediately before this instruction.
 2. The POLL instruction is valid when P14 is set in the input port mode, in which case, it is assumed that the $\overline{\text{POLL}}$ pin is always active (low level).

[Example] POLL**[Bytes]** 1**[Word Format]**

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
POLL	None	1	0	0	1	1	0	1	1

POP

Restore from stack

Pop

[Format] POP dst

[Operand, operation]

Mnemonic	Operand (dst)	Operation
POP	mem16	$SP \leftarrow SP + 2$ $(mem16) \leftarrow (SP - 1, SP - 2)$
	reg16	$SP \leftarrow SP + 2$
	sreg	$dst \leftarrow (SP - 1, SP - 2)$
	PSW	
	R	$IY \leftarrow (SP + 1, SP)$ $IX \leftarrow (SP + 3, SP + 2)$ $BP \leftarrow (SP + 5, SP + 4)$ $BW \leftarrow (SP + 9, SP + 8)$ $DW \leftarrow (SP + 11, SP + 10)$ $CW \leftarrow (SP + 13, SP + 12)$ $AW \leftarrow (SP + 15, SP + 14)$ $SP \leftarrow SP + 16$

[Flag]

- When dst = PSW

V	DIR	IE	BRK	S	Z	F1	AC	F0	P	\overline{IBRK}	CY
R	R	R	R	R	R	R	R	R	R	R	R

Remark The RB0-2 flags are not affected.

- When other than dst = PSW

V	S	Z	AC	P	CY

[Description]

Transfers the contents of the stack to the destination operand (dst) (however, the contents are not transferred to PS when dst = sreg).

Caution When dst = sreg, hardware interrupt request (maskable interrupt and non-maskable interrupt) and single-step breaks are not accepted in between this instruction and the next instruction.

[Example]

- POP AW
- POP BW
- POP IY
- POP SP
- MOV BP, SP

[Bytes]

Mnemonic	Operand	Bytes
POP	mem16	2-4
	reg16	1
	sreg	
	PSW	
	R	1

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
POP	mem16	1	0	0	0	1	1	1	1	mod	0	0	0	mem			
		(disp-low)							(disp-high)								
	reg16	0	1	0	1	1	reg			—							
	sreg	0	0	0	sreg			1	1	1	—						
	PSW	1	0	0	1	1	1	0	1	—							
R	0	1	1	0	0	0	0	1	—								

PREPARE

Creates stack frame
Prepare New Stack Frame

[Format] **PREPARE imm16, imm8**

[Operation] $(SP - 1, SP - 2) \leftarrow BP$
 $SP \leftarrow SP - 2$
temp \leftarrow after SP,
if imm8 > 0, repeats the following operation “imm8 – 1”
times:
 $(SP - 1, SP - 2) \leftarrow (BP - 1, BP - 2)$
 $SP \leftarrow SP - 2$
 $BP \leftarrow BP - 2$ } *1
then executes the following:
 $(SP - 1, SP - 2) \leftarrow temp$
 $SP \leftarrow SP - 2$ } *2
After that, executes the following processing:
 $BP \leftarrow temp$
 $SP \leftarrow SP - imm16$
When imm8 = 1, repeating operation *1 is not executed.
When imm8 = 0, operations *1 and *2 are not executed.

[Operand]

Mnemonic	Operand
PREPARE	imm16, imm8

[Flag]

V	S	Z	AC	P	CY

[Description] This instruction is used to create “stack frames” necessary for high-level languages with block structures (such as Pascal and Ada). The stack frame includes pointers indicating a group of frames including variables that can be referenced from the procedure, and an area of local variables. This instruction copies a frame pointer to make it possible to reserve an area of local variables and to reference global variables. The 16-bit immediate data specified as the first operand specifies the size of the area to be reserved for local variables (in byte units), and the 8-bit immediate data described as the second operand specifies the depth of a procedure block (called lexical level). The base address of the frame created by this instruction is set to BP. First, BP is saved to the stack. This is because the BP of the calling procedure is to be restored at the end of the called procedure. Next, a frame pointer (saved BP) in a range that can be referenced by the called procedure is pushed to the stack. The range that can be referenced is the value of the lexical level of the procedure minus 1. If the lexical level is greater than 1, the frame pointer of the calling procedure is also pushed to the stack. This is to copy the frame pointer of the calling procedure when the frame pointer is to be copied from the procedure called by a procedure. Next, the value of the new frame pointer is set to BP, and an area of the local variables to be used by the procedure is reserved on the stack. Consequently, SP is decremented by the number of the local variables.

[Example]

```

MOV    SP, 60H
MOV    BP, SP
CALL   CHK
PREPARE 0006, 04
MOV    AW, [BP + 0FAH]
ADD    AW, [BP + 0F8A]
MOV    [BP + 0FCH], AW
    
```

[Bytes] 4

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
PREPARE	imm16, imm8	1	1	0	0	1	0	0	0	imm16-low							
		imm16-high								imm8							

PUSH

Save to stack

Push

[Format] **PUSH src****[Operand, operation]**

Mnemonic	Operand (src)	Operation
PUSH	mem16	SP ← SP - 2 (SP + 1, SP) ← (mem16 + 1, mem16)
	reg16	SP ← SP - 2
	sreg	(SP + 1, SP) ← src
	PSW	
	R	TEMP ← SP (SP - 1, SP - 2) ← AW (SP - 3, SP - 4) ← CW (SP - 5, SP - 6) ← DW (SP - 7, SP - 8) ← BW (SP - 9, SP - 10) ← TEMP (SP - 11, SP - 12) ← BP (SP - 13, SP - 14) ← IX (SP - 15, SP - 16) ← IY SP ← SP - 16
	imm8	(SP - 1, SP - 2) ← sign extension of imm8 SP ← SP - 2
	imm16	(SP - 1, SP - 2) ← imm16 SP ← SP - 2

[Flag]

V	S	Z	AC	P	CY

[Description]

Saves the contents of the source operand (src) to the stack.

If 8-bit immediate data (imm8) is described as the operand, imm8 is sign-extended and is saved to the stack addressed by SP as 16-bit data.

[Example]

- PUSH DS0
- PUSH SS
- PUSH DS1

[Bytes]

Mnemonic	Operand	Bytes
PUSH	mem16	2-4
	reg16	1
	sreg	
	PSW	
	R	1
	imm8	2
	imm16	3

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
PUSH	mem16	1	1	1	1	1	1	1	1	mod	1	1	0	mem			
		(disp-low)							(disp-high)								
	reg16	0	1	0	1	0	reg			—							
	sreg	0	0	0	sreg			1	1	0	—						
	PSW	1	0	0	1	1	1	0	0	—							
	R	0	1	1	0	0	0	0	0	—							
	imm8	0	1	1	0	1	0	1	0	imm8							
	imm16	imm16-low							imm16-low								
imm16-high							—										

REP
REPE
REPZ

Repeat prefix with Z = 1
Repeat
Repeat while Equal
Repeat while Zero

[Format] **REP**
 REPE
 REPZ

[Operation] [When $CW \neq 0$] PS: executes byte instruction of $PC + 1$
 $CW: \leftarrow CW - 1$
 When $Z \neq 1$: $PC \leftarrow PC + 2$
 When $Z = 1$: executes again
[When $CW = 0$] $PC \leftarrow PC + 2$

[Operand]

Mnemonic	Operand
REP	None
REPE	
REPZ	

[Flag]

V	S	Z	AC	P	CY

[Description] Executes the block transfer/compare/input/output instruction that follows this instruction and decrements the value of the CW register ($- 1$) while CW is not equal to 0.
REP is used with MOV BK, LDM, STM, OUTM, or INM, and performs repetitive processing while CW is not equal to 0, independently of the value of the Z flag.
REPZ and REPE are used with CMP BK and CMPM and allow execution to exit from the loop if Z is not equal to 1 as a result of comparison by each block instruction, or if $CW = 0$.
The CW register is checked immediately before execution of each block instruction, i.e., execution of REP, REPE, or REPZ. If REP, REPE, or REPZ is executed with $CW = 0$, then the following block instruction is not executed at all, but the next instruction is executed.
The Z flag is checked as a result of execution of a block instruction, and its contents before execution of REPE or REPZ is independent of execution.

Caution Hardware interrupt requests (maskable interrupt and non-maskable interrupt) and single-step breaks are not accepted in between this instruction and the next instruction.

- [Example]**
- REP MOVBKW
 - REPZ CMPBKW

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
REP	None	1	1	1	1	0	0	1	1
REPE									
REPZ									

REPC

Repeat prefix with CY = 1

Repeat while Carry

[Format] REPC

[Operation] [When CW ≠ 0] PS: executes byte instruction of PC + 1
 CW: ← CW - 1
 When CY ≠ 1: PC ← PC + 2
 When CY = 1: executes again
 [When CW = 0] PC ← PC + 2

[Operand]

Mnemonic	Operand
REPC	None

[Flag]

V	S	Z	AC	P	CY

[Description] Executes the block comparison instruction (CMPBK or CMPM) that follows this instruction and decrements the value of the CW register (-1) while CW is not equal to 0. Execution exits from the loop if CY is not equal to 1 as a result of executing the block comparison instruction.

The CW register is checked immediately before execution of the block comparison instruction, i.e., execution of REPC. If REPC is executed with CW = 0, then the following block instruction is not executed at all, but the next instruction is executed.

The CY flag is checked as a result of execution of the following block instruction, and its contents before the execution of REPC are independent of execution.

Caution **Hardware interrupt requests (maskable interrupt and non-maskable interrupt) and single-step breaks are not accepted in between this instruction and the next instruction.**

[Example] REPC CMPBKW**[Bytes]** 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
REPC	None	0	1	1	0	0	1	0	1

REPNC

Repeat prefix with CY = 0

Repeat while Not Carry

[Format] REPNC

[Operation] [When CW ≠ 0] PS: executes byte instruction of PC + 1
 CW: ← CW – 1
 When CY ≠ 1: executes again
 When CY = 1: PC ← PC + 2
 [When CW = 0] PC ← PC + 2

[Operand]

Mnemonic	Operand
REPNC	None

[Flag]

V	S	Z	AC	P	CY

[Description] Executes the block comparison instruction (CMPBK or CMPM) that follows this instruction and decrements the value of the CW register (– 1) while CW is not equal to 0. Execution exits from the loop if CY is equal to 1 as a result of executing the block comparison instruction.

The CW register is checked immediately before execution of the block comparison instruction, i.e., execution of REPNC. If REPNC is executed with CW = 0, then the following block instruction is not executed at all, but the next instruction is executed.

The CY flag is checked as a result of execution of the following block instruction, and its contents before execution of REPNC are independent of execution.

Caution **Hardware interrupt requests (maskable interrupt and non-maskable interrupt) and single-step breaks are not accepted in between this instruction and the next instruction.**

[Example] REPNC CMPMB**[Bytes]** 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
REPNC	None	0	1	1	0	0	1	0	0

REPNE REPZ

Repeat prefix with Z = 0
 Repeat while Not Equal
 Repeat while Not Zero

[Format] **REPNE**
 REPZ

[Operation] [When $CW \neq 0$] PS: executes byte instruction of $PC + 1$
 $CW: \leftarrow CW - 1$
 When $Z \neq 1$: executes again
 When $Z = 1$: $PC \leftarrow PC + 2$
 [When $CW = 0$] $PC \leftarrow PC + 2$

[Operand]

Mnemonic	Operand
REPNE	None
REPZ	

[Flag]

V	S	Z	AC	P	CY

[Description] Executes the block comparison (CMPBK or CPM) instruction that follows this instruction and decrements the value of the CW register (-1) while CW is not equal to 0. Execution exits from the loop if Z is not equal to 0 or $CW = 0$ as a result of execution of the block comparison instruction.

The CW register is checked immediately before execution of each block instruction, i.e., execution of REPNE or REPZ. If REPNE or REPZ is executed with $CW = 0$, then the following block instruction is not executed at all, but the next instruction is executed.

The Z flag is checked as a result of execution of a block instruction, and its contents before execution of REPNE or REPZ are independent of execution.

Caution **Hardware interrupt requests (maskable interrupt and non-maskable interrupt) and single-step breaks are not accepted in between this instruction and the next instruction.**

[Example]

- REPNE CMPMB
- REPZ CMPBKW

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
REPNE	None	1	1	1	1	0	0	1	0
REPZ									

RET

Return from subroutine

Return from Procedure

[Format] <1> RET
 <2> RET pop-value

[Operand, operation]

- When returning from call in segment

Mnemonic	Operand	Operation
RET	None	PC ← (SP + 1, SP) SP ← SP + 2
	pop-value	PC ← (SP + 1, SP) SP ← SP + 2 SP ← SP + pop-value

- When returning from call out of segment

Mnemonic	Operand	Operation
RET	None	PC ← (SP + 1, SP) PS ← (SP + 3, SP + 2) SP ← SP + 4
	pop-value	PC ← (SP + 1, SP) PS ← (SP + 3, SP + 2) SP ← SP + 4 SP ← SP + pop-value

[Flag]

V	S	Z	AC	P	CY

[Description] • Returning from call in segment
 PC is restored from the stack. When pop-value is described as the operand, the 16-bit pop-value is added to SP (if the parameters saved to the stack following PC are not necessary, this instruction is useful for skipping the value of SP by the number of unnecessary parameters). This instruction is automatically distinguished from the RET instruction from calls out of a segment by the assembler.

- Returning from a call out of segment
 PC and PS are restored from the stack. When pop-value is described as the operand, the 16-bit pop-value is added to SP (if the parameters saved to the stack following PC and PS are not necessary, this instruction is useful for skipping the value of SP by the number of unnecessary parameters).
 This instruction is automatically distinguished from the RET instruction from a call in a segment by the assembler.

[Example] POP R
 RET

[Bytes]

Mnemonic	Operand	Bytes
RET	None	1
	pop-value	3

[Word Format] • Return from call in segment

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
RET	None	1	1	0	0	0	0	1	1	—							
	pop-value	1	1	0	0	0	0	1	0	pop-value-low							
		pop-value-high								—							

- Return from call out of segment

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
RET	None	1	1	0	0	1	0	1	1	—							
	pop-value	1	1	0	0	1	0	1	0	pop-value-low							
		pop-value-high								—							

RETI

Return from interrupt

Return from Interrupt

[Format] RETI

[Operation] $PC \leftarrow (SP + 1, SP)$
 $PS \leftarrow (SP + 3, SP + 2)$
 $PSW \leftarrow (SP + 5, SP + 4)$
 $SP \leftarrow SP + 6$

[Operand]

Mnemonic	Operand
RETI	None

[Flag]

RB2	RB1	RB0	V	DIR	IE	BRK	S	Z	F1	AC	F0	P	$\overline{\text{IBRK}}$	CY
R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

[Description] Restores the contents of the stack to PC, PS, and PSW. This instruction is used to return from interrupt processing.

[Example] POP R
 RETI

[Bytes] 1**[Word Format]**

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
RETI	None	1	1	0	0	1	1	1	1

RETRBI [Added to V20/V30]

Return from register bank interrupt or BRKCS instruction

Return from Register Bank Switching

[Format] RETRBI

[Operation] PC ← PC save
PSW ← PSW save

[Operand]

Mnemonic	Operand
RETRBI	None

[Flag]

RB2	RB1	RB0	V	DIR	IE	BRK	S	Z	F1	AC	F0	P	IBRK	CY
R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

[Description] This instruction is used to return from an interrupt using the register bank switching function or the BRKCS instruction. Execution cannot return from this interrupt by using the RETI instruction. This instruction cannot be used to return from any other interrupts.

To return from an interrupt using the register bank switching function, the FINT instruction must be executed before this instruction.

[Example] RETRBI

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
RETRBI	None	0	0	0	0	1	1	1	1	1	0	0	1	0	0	0	1

ROL

Rotate left
Rotate Left

[Format] ROL dst, src

[Operation]



[Operand]

Mnemonic	Operand (dst, src)
ROL	reg, 1
	mem, 1
	reg, CL
	mem, CL
	reg, imm8
	mem, imm8

[Flag]

When src = 1

V	S	Z	AC	P	CY
×					×

Other than left

V	S	Z	AC	P	CY
U					×

[Description]

- When src = 1
Shifts the contents of the destination operand (dst) specified by the first operand 1 bit to the left. The MSB, which is the contents of dst (bit 7 or 15), is shifted to the LSB (bit 0), and at the same time transferred to the CY flag.
If the MSB is changed as a result of shifting, the V flag is set to 1. If the MSB is not changed, the V flag is reset to 0.
- When src = CL or src = imm8
Shifts the contents of the destination operand (dst) specified by the first operand to the left the number of bits of the contents of the source operand specified by the second operand. The MSB, which is the content of dst (bit 7 or 15), is shifted to the LSB (bit 0), and at the same time transferred to the CY flag.

[Example]

```
MOV [IX], BL
ROL BYTE PTR [IX], 1
```

[Bytes]

Mnemonic	Operand	Bytes
ROL	reg, 1	2
	mem, 1	2-4
	reg, CL	2
	mem, CL	2-4
	reg, imm8	3
	mem, imm8	3-5

[Word Format]

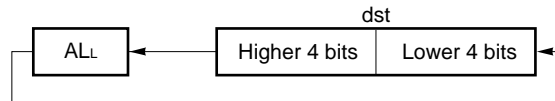
Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
ROL	reg, 1	1	1	0	1	0	0	0	W	1	1	0	0	0	reg		
	mem, 1	1	1	0	1	0	0	0	W	mod		0	0	0	mem		
		(disp-low)								(disp-high)							
	reg, CL	1	1	0	1	0	0	1	W	1	1	0	0	0	reg		
	mem, CL	1	1	0	1	0	0	1	W	mod		0	0	0	mem		
		(disp-low)								(disp-high)							
	reg, imm8	1	1	0	0	0	0	0	W	1	1	0	0	0	reg		
		imm8								—							
	mem, imm8	1	1	0	0	0	0	0	W	mod		0	0	0	mem		
		(disp-low)								(disp-high)							
		imm8								—							

ROL4

Rotate left nibble
Rotate Left Nibble

[Format] ROL4 dst

[Operation]



[Operand]

Mnemonic	Operand (dst)
ROL4	reg8
	mem8

[Flag]

V	S	Z	AC	P	CY

[Description] Treats the contents of the destination operand (dst) as a 2-digit packed BCD and rotates it one digit to the left via the lower 4 bits of the AL register (AL_L). As a result, the higher 4 bits of the AL register are not guaranteed.

[Example]

- MOV AL, 24H
 ROL4 AL
- MOV AL, BYTE PTR [IX]
 ROL4 AL

[Bytes]

Mnemonic	Operand	Bytes
ROL4	reg8	3
	mem8	3-5

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
ROL4	reg8	0	0	0	0	1	1	1	1	0	0	1	0	1	0	0	0
		1	1	0	0	0	reg			—							
	mem8	0	0	0	0	1	1	1	1	0	0	1	0	1	0	0	0
		mod	0	0	0	mem			(disp-low)								
		(disp-high)								—							

ROLC

Rotate left with carry
 Rotate Left with Carry

[Format] ROLC dst, src

[Operation]



[Operand]

Mnemonic	Operand (dst, src)
ROLC	reg, 1
	mem, 1
	reg, CL
	mem, CL
	reg, imm8
	mem, imm8

[Flag]

When src = 1

Other than left

V	S	Z	AC	P	CY
×					×

V	S	Z	AC	P	CY
U					×

[Description]

- When src = 1
 Shifts the contents of the destination operand (dst) specified by the first operand 1 bit to the left via the CY flag. The MSB, which is the contents of dst (bit 7 or 15), is shifted to the CY flag, and the data of the CY flag is transferred to the LSB (bit 0).
 If the MSB is changed as a result of shifting, the V flag is set to 1. If the MSB is not changed, the V flag is reset to 0.
- When src = CL or src = imm8
 Shifts via the CY flag, the contents of the destination operand (dst) specified by the first operand to the left, the number of bits of the contents of the source operand specified by the second operand. The MSB, which is the contents of dst (bit 7 or 15) is shifted to the CY flag, and the data of the CY flag is transferred to the LSB (bit 0).

- [Example]**
- ROLC AL, 1
 - ROLC CL, 1
 - ROLC DW, 1
 - ROLC AW, 1

[Bytes]

Mnemonic	Operand	Bytes
ROLC	reg, 1	2
	mem, 1	2-4
	reg, CL	2
	mem, CL	2-4
	reg, imm8	3
	mem, imm8	3-5

[Word Format]

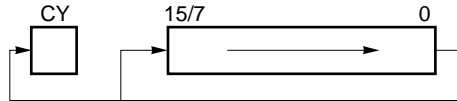
Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
ROLC	reg, 1	1	1	0	1	0	0	0	W	1	1	0	1	0	reg		
	mem, 1	1	1	0	1	0	0	0	W	mod	0	1	0	mem			
		(disp-low)						(disp-high)									
	reg, CL	1	1	0	1	0	0	1	W	1	1	0	1	0	reg		
	mem, CL	1	1	0	1	0	0	1	W	mod	0	1	0	mem			
		(disp-low)						(disp-high)									
	reg, imm8	1	1	0	0	0	0	0	W	1	1	0	1	0	reg		
		imm8						—									
	mem, imm8	1	1	0	0	0	0	0	W	mod	0	1	0	mem			
		(disp-low)						(disp-high)									
		imm8						—									

ROR

Rotate right
Rotate Right

[Format] ROR dst, src

[Operation]



[Operand]

Mnemonic	Operand (dst, src)
ROR	reg, 1
	mem, 1
	reg, CL
	mem, CL
	reg, imm8
	mem, imm8

[Flag]

When src = 1

V	S	Z	AC	P	CY
×					×

Other than left

V	S	Z	AC	P	CY
U					×

[Description]

- When src = 1
Shifts the contents of the destination operand (dst) specified by the first operand 1 bit to the right. The LSB, which is the contents of dst (bit 0), is shifted to the MSB (bit 7 or 15), and at the same time transferred to the CY flag. If the MSB is changed as a result of shifting, the V flag is set to 1. If the MSB is not changed, the V flag is reset to 0.
- When src = CL or src = imm8
Shifts the contents of the destination operand (dst) specified by the first operand to the right the number of bits of the content of the source operand (src) specified by the second operand. The LSB, which is the contents of dst (bit 0) is shifted to the MSB (bit 7 or 15), and at the same time transferred to the CY flag.

- [Example]**
- ROR AL, 3
 - ROR CW, 6
 - ROR IY, 2

[Bytes]

Mnemonic	Operand	Bytes
ROR	reg, 1	2
	mem, 1	2-4
	reg, CL	2
	mem, CL	2-4
	reg, imm8	3
	mem, imm8	3-5

[Word Format]

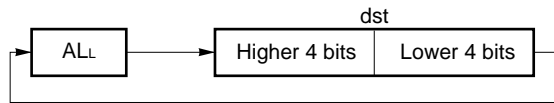
Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
ROR	reg, 1	1	1	0	1	0	0	0	W	1	1	0	0	1			reg
	mem, 1	1	1	0	1	0	0	0	W	mod	0	0	1			mem	
		(disp-low)							(disp-high)								
	reg, CL	1	1	0	1	0	0	1	W	1	1	0	0	1		reg	
	mem, CL	1	1	0	1	0	0	1	W	mod	0	0	1			mem	
		(disp-low)							(disp-high)								
	reg, imm8	1	1	0	0	0	0	0	W	1	1	0	0	1		reg	
	mem, imm8	imm8								—							
		1	1	0	0	0	0	0	W	mod	0	0	1			mem	
		(disp-low)							(disp-high)								
		imm8								—							

ROR4

Rotate right nibble
Rotate Right Nibble

[Format] ROR4 dst

[Operation]



[Operand]

Mnemonic	Operand (dst)
ROR4	reg8
	mem8

[Flag]

V	S	Z	AC	P	CY

[Description] Treats the contents of the destination operand (dst) as a 2-digit packed BCD and rotates it one digit to the right via the lower 4 bits of the AL register (AL_L). As a result, the higher 4 bits of the AL register are not guaranteed.

- [Example]**
- MOV AL, 24H
ROR4 AL
 - MOV AL, BYTE PTR [IX]
ROR4 AL

[Bytes]

Mnemonic	Operand	Bytes
ROR4	reg8	3
	mem8	3-5

[Word Format]

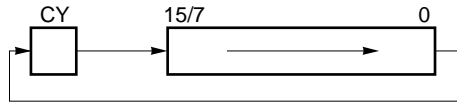
Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
ROR4	reg8	0	0	0	0	1	1	1	1	0	0	1	0	1	0	1	0
		1	1	0	0	0	reg			—							
	mem8	0	0	0	0	1	1	1	1	0	0	1	0	1	0	1	0
		mod	0	0	0	mem			(disp-low)								
		(disp-high)								—							

RORC

Rotate right with carry
Rotate Right with Carry

[Format] RORC dst, src

[Operation]



[Operand]

Mnemonic	Operand (dst, src)
RORC	reg, 1
	mem, 1
	reg, CL
	mem, CL
	reg, imm8
	mem, imm8

[Flag]

When src = 1

V	S	Z	AC	P	CY
×					×

Other than left

V	S	Z	AC	P	CY
U					×

[Description]

- When src = 1
Shifts the contents of the destination operand (dst) specified by the first operand 1 bit to the right via the CY flag. The LSB, which is the contents of dst (bit 0), is shifted to the CY flag, and the data of the CY flag is transferred to the MSB (bit 7 or 15).
If the MSB is changed as a result of shifting, the V flag is set to 1. If the MSB is not changed, the V flag is reset to 0.
- When src = CL or src = imm8
Shifts via the CY flag, the contents of the destination operand (dst) specified by the first operand to the right, the number of bits of the content of the source operand (src) specified by the second operand. The LSB, which is the contents of dst (bit 0), is shifted to the CY flag, and the data of the CY flag is transferred to the MSB (bit 7 or 15).

- [Example]**
- RORC AL, 1
 - RORC BL, 1
 - RORC CW, 1
 - RORC IX, 1

[Bytes]

Mnemonic	Operand	Bytes
RORC	reg, 1	2
	mem, 1	2-4
	reg, CL	2
	mem, CL	2-4
	reg, imm8	3
	mem, imm8	3-5

[Word Format]

Mnemonic	Operand	Operation code																
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
RORC	reg, 1	1	1	0	1	0	0	0	W	1	1	0	1	1	reg			
	mem, 1	1	1	0	1	0	0	0	W	mod		0	1	1	mem			
		(disp-low)								(disp-high)								
	reg, CL	1	1	0	1	0	0	1	W	1	1	0	1	1	reg			
	mem, CL	1	1	0	1	0	0	1	W	mod		0	1	1	mem			
		(disp-low)								(disp-high)								
	reg, imm8	1	1	0	0	0	0	0	W	1	1	0	1	1	reg			
		imm8								—								
	mem, imm8	1	1	0	0	0	0	0	W	mod		0	1	1	mem			
		(disp-low)								(disp-high)								
		imm8								—								

SET1

Set bit

Set Bit

[Format] <1> SET1 dst, src
 <2> SET1 dst

[Operation] Format <1> : bit n of dst (n is specified by src) ← 1
 Format <2> : dst ← 1

[Operand] Format <1>

Mnemonic	Operand (dst, src)
SET1	reg8, CL
	mem8, CL
	reg16, CL
	mem16, CL
	reg8, imm3
	mem8, imm3
	reg16, imm4
	mem16, imm4

Format <2>

Mnemonic	Operand (dst)
SET1	CY
	DIR

[Flag] Format <1>

V	S	Z	AC	P	CY

Format <2> (when dst = CY)

V	S	Z	AC	P	CY
					1

Format <2> (when dst = DIR)

V	DIR	S	Z	AC	P	CY
	1					

[Description] Format <1> : Sets bit n (n is the contents of the source operand (src) specified by the second operand) of the destination operand (dst) specified by the first operand, and stores the result to the destination operand (dst).

When the operands are reg8, CL or mem8, CL, only the lower 3 bits (0-7) of the CL value are valid.

When the operands are reg16, CL or mem16, CL, only the lower 4 bits of the CL value (0-15) are valid.

When the operands are reg8, imm3, only the lower 3 bits of the immediate data at the fourth byte position of the instruction are valid.

When the operands are mem8, imm3, only the lower 3 bits of the immediate data at the end byte position of the instruction are valid.

When the operands are reg16, imm4, only the lower 4 bits of the immediate data at the fourth byte position of the instruction are valid.

When the operands are mem16, imm4, only the lower 4 bits of the immediate data at the end byte position of the instruction are valid.

Format <2> : Sets the CY flag to 1 when dst = CY.

When dst = DIR, sets the DIR flag to 1. In addition, sets such that the index registers (IX, IY) are auto-decremented when each of the MOVBK, CMPBK, CMPM, LDM, STM, INM, and OUTM instructions is executed.

[Example]
 MOV CL, 4
 SET1 AL, CL
 OUT 0DAH, AL

[Bytes]

Mnemonic	Operand	Bytes
SET1	reg8, CL	3
	mem8, CL	3-5
	reg16, CL	3
	mem16, CL	3-5
	reg8, imm3	4
	mem8, imm3	4-6
	reg16, imm4	4
	mem16, imm4	4-6
	CY	1
	DIR	1

[Word Format]

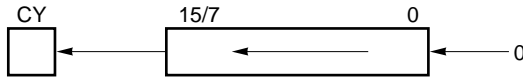
Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
SET1	reg8, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	1	0	0
		1	1	0	0	0	reg			—							
	mem8, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	1	0	0
		mod	0	0	0	mem			(disp-low)								
		(disp-high)								—							
	reg16, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	1	0	1
		1	1	0	0	0	reg			—							
	mem16, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	1	0	1
		mod	0	0	0	mem			(disp-low)								
		(disp-high)								—							
	reg8, imm3	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	0
		1	1	0	0	0	reg			imm3							
	mem8, imm3	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	0
		mod	0	0	0	mem			(disp-low)								
		(disp-high)								imm3							
	reg16, imm4	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	1
		1	1	0	0	0	reg			imm4							
	mem16, imm4	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	1
		mod	0	0	0	mem			(disp-low)								
		(disp-high)								imm4							
CY		1	1	1	1	1	0	0	1	—							
DIR		1	1	1	1	1	1	0	1	—							

SHL

Shift left
Shift Left

[Format] SHL dst, src

[Operation]



[Operand]

Mnemonic	Operand (dst, src)
SHL	reg, 1
	mem, 1
	reg, CL
	mem, CL
	reg, imm8
	mem, imm8

[Flag] When src = 1

Other than left

V	S	Z	AC	P	CY
×	×	×	U	×	×

V	S	Z	AC	P	CY
U	×	×	U	×	×

[Description]

- When src = 1
Shifts the contents of the destination operand (dst) specified by the first operand 1 bit to the left. Zero is shifted to the LSB, which is the contents of dst (bit 0), and the data of the MSB (bit 7 or bit 15) is set to the CY flag.
The V flag is cleared if the sign bit (bit 7 or 15) is not changed as a result of shifting.
- When src = CL or src = imm8
Shifts the contents of the destination operand (dst) specified by the first operand to the left the number of bits of the contents of the source operand specified by the second operand. Zero is shifted to the LSB (bit 0), which is the contents of dst, each time the data has been shifted 1 bit, and the data of the MSB (bit 7 or 15) is set to the CY flag.

[Example]

```
IN    AW, 0C8H
MOV   [IY], AW
SHL   WORD PTR [IY], 12
```


[Bytes]

Mnemonic	Operand	Bytes
SHL	reg, 1	2
	mem, 1	2-4
	reg, CL	2
	mem, CL	2-4
	reg, imm8	3
	mem, imm8	3-5

[Word Format]

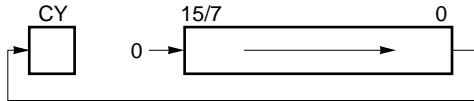
Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
SHL	reg, 1	1	1	0	1	0	0	0	W	1	1	1	0	0	reg		
	mem, 1	1	1	0	1	0	0	0	W	mod	1	0	0	mem			
		(disp-low)							(disp-high)								
	reg, CL	1	1	0	1	0	0	1	W	1	1	1	0	0	reg		
	mem, CL	1	1	0	1	0	0	1	W	mod	1	0	0	mem			
		(disp-low)							(disp-high)								
	reg, imm8	1	1	0	0	0	0	0	W	1	1	1	0	0	reg		
		imm8							—								
	mem, imm8	1	1	0	0	0	0	0	W	mod	1	0	0	mem			
		(disp-low)							(disp-high)								
	imm8							—									

SHR

Shift right
Shift Right

[Format] **SHR dst, src**

[Operation]



[Operand]

Mnemonic	Operand (dst, src)
SHL	reg, 1
	mem, 1
	reg, CL
	mem, CL
	reg, imm8
	mem, imm8

[Flag]

When src = 1

Other than left

V	S	Z	AC	P	CY
×	×	×	U	×	×

V	S	Z	AC	P	CY
U	×	×	U	×	×

[Description]

- When src = 1
Shifts the contents of the destination operand (dst) specified by the first operand 1 bit to the right. Zero is shifted to the MSB, which is the contents of dst (bit 7 or 15), and the data of the LSB (bit 0) is set to the CY flag.
The V flag is cleared if the sign bit (bit 7 or 15) is not changed as a result of shifting.
- When src = CL or src = imm8
Shifts the contents of the destination operand (dst) specified by the first operand to the right the number of bits of the contents of the source operand (src) specified by the second operand. Zero is shifted to the MSB (bit 7 or 15), which is the contents of dst, each time the data has been shifted 1 bit, and the data of the LSB (bit 0) is set to the CY flag.

[Example]

- RCV: IN AL, 0DAH
SHR AL, 3
BC RCV
- SHR CW, 8

[Bytes]

Mnemonic	Operand	Bytes
SHR	reg, 1	2
	mem, 1	2-4
	reg, CL	2
	mem, CL	2-4
	reg, imm8	3
	mem, imm8	3-5

[Word Format]

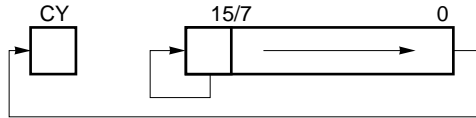
Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
SHR	reg, 1	1	1	0	1	0	0	0	W	1	1	1	0	1	reg		
	mem, 1	1	1	0	1	0	0	0	W	mod	1	0	1	mem			
		(disp-low)						(disp-high)									
	reg, CL	1	1	0	1	0	0	1	W	1	1	1	0	1	reg		
	mem, CL	1	1	0	1	0	0	1	W	mod	1	0	1	mem			
		(disp-low)						(disp-high)									
	reg, imm8	1	1	0	0	0	0	0	W	1	1	1	0	1	reg		
	mem, imm8	imm8						—									
		1	1	0	0	0	0	0	W	mod	1	0	1	mem			
		(disp-low)						(disp-high)									
		imm8						—									

SHRA

Arithmetic shift right
Shift Right Arithmetic

[Format] SHRA dst, src

[Operation]



[Operand]

Mnemonic	Operand (dst, src)
SHRA	reg, 1
	mem, 1
	reg, CL
	mem, CL
	reg, imm8
	mem, imm8

[Flag]

When src = 1

V	S	Z	AC	P	CY
0	x	x	U	x	x

Other than left

V	S	Z	AC	P	CY
U	x	x	U	x	x

[Description]

- When src = 1
Arithmetically shifts the contents of the destination operand (dst) specified by the first operand 1 bit to the right.
The original value is shifted to the MSB, which is the contents of dst (bit 7 or 15), and the sign is not changed after shifting. The data of the LSB (bit 0) is set to the CY flag.
- When src = CL or src = imm8
Shifts the contents of the destination operand (dst) specified by the first operand to the right the number of bits of the contents of the source operand (src) specified by the second operand. The original value is shifted to the MSB (bit 7 or 15), which is the contents of dst, and the sign is not changed after shifting. The data of the LSB (bit 0) is set to the CY flag.

[Example]

- MOV CL, 2
SHRA BL, CL
- MOV CL, 9
SHRA DW, CL

[Bytes]

Mnemonic	Operand	Bytes
SHRA	reg, 1	2
	mem, 1	2-4
	reg, CL	2
	mem, CL	2-4
	reg, imm8	3
	mem, imm8	3-5

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
SHRA	reg, 1	1	1	0	1	0	0	0	W	1	1	1	1	1			reg
	mem, 1	1	1	0	1	0	0	0	W	mod		1	1	1		mem	
		(disp-low)							(disp-high)								
	reg, CL	1	1	0	1	0	0	1	W	1	1	1	1	1		reg	
	mem, CL	1	1	0	1	0	0	1	W	mod		1	1	1		mem	
		(disp-low)							(disp-high)								
	reg, imm8	1	1	0	0	0	0	0	W	1	1	1	1	1		reg	
		imm8							—								
	mem, imm8	1	1	0	0	0	0	0	W	mod		1	1	1		mem	
		(disp-low)							(disp-high)								
	imm8							—									

STM STMB STMW	Block store
	Store Multiple
	Store Multiple Byte
	Store Multiple Word

[Format] (repeat) **STM** [DS1-spec:] dst-block
 (repeat) **STMB**
 (repeat) **STMW**

[Operation] [When W = 0] (IX) ← AL
 DIR = 0: IY ← IY + 1
 DIR = 1: IY ← IY - 1
 [When W = 1] (IY + 1, IY) ← AW
 DIR = 0: IY ← IY + 2
 DIR = 1: IY ← IY - 2

[Operand]

Mnemonic	Operand
STM	[DS1-spec:] dst-block
STMB	None
STMW	

[Flag]

V	S	Z	AC	P	CY

[Description] Repeatedly transfers the value of the AL register or AW register to a block addressed by the IY register in byte or word units.
 The value of the IY register is automatically incremented (+1/+2) or decremented (-1/-2) for the next byte/word process each time 1-byte/word data has been transferred. The direction of the block is determined by the status of the DIR flag.
 Whether a transfer is carried out in byte or word units is specified by the attribute of the operand when the STM instruction is used. When the STMB or STMW instruction is used, the byte or word type is specified directly.
 The destination block must always be located in a segment specified by the DS1 register. Segment overrides are not allowed.

[Example]

- REP STM DS1 : WORD_VAR ; DS1 segment
- REP STMB ; DS1 segment

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
STM	[DS1-spec:] dst-block	1	0	1	0	1	0	1	W
STMB	None								
STMW									

STOP [Added to V20/V30]

CPU control

Stop

[Format] STOP

[Operation] CPU Stop

[Operand]

Mnemonic	Operand
STOP	None

[Flag]

V	S	Z	AC	P	CY

[Description] Places the CPU in the STOP status.

[Example] STOP

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
STOP	None	0	0	0	0	1	1	1	1	1	0	0	1	1	1	1	0

SUB

Subtraction

Subtract

[Format] **SUB dst, src****[Operand, operation]**

Mnemonic	Operand (dst, src)	Operation
SUB	reg, reg'	dst ← dst – src
	mem, reg	
	reg, mem	
	reg, imm	
	mem, imm	
	acc, imm	[When W = 0] AL ← AL – imm8 [When W = 1] AW ← AW – imm16

[Flag]

V	S	Z	AC	P	CY
×	×	×	×	×	×

[Description] Subtracts the contents of the source operand (src) specified by the first operand from the contents of the destination operand (dst) specified as the first operand, and stores the result to the destination operand (dst).

[Example] To subtract the contents of memory 0:50H (word data) from the contents of the DL register and store the result to the DL register:

```
MOV  AW, 0
MOV  DS0, AW
MOV  IX, 50H
SUB  DL, DS0: BYTE PTR [IX]
```

[Bytes]

Mnemonic	Operand	Bytes
SUB	reg, reg'	2
	mem, reg	2-4
	reg, mem	2-4
	reg, imm	3, 4
	mem, imm	3-6
	acc, imm	2, 3

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
SUB	reg, reg'	0	0	1	0	1	0	1	W	1	1	reg			reg'		
	mem, reg	0	0	1	0	1	0	0	W	mod		reg			mem		
		(disp-low)								(disp-high)							
	reg, mem	0	0	1	0	1	0	1	W	mod		reg			mem		
		(disp-low)								(disp-high)							
	reg, imm	1	0	0	0	0	0	s	W	1	1	1	0	1	reg		
		imm8 or imm16-low								imm16-high							
	mem, imm	1	0	0	0	0	0	s	W	mod		1	0	1	mem		
		(disp-low)								(disp-high)							
		imm8 or imm16-low								imm16-high							
acc, imm	0	0	1	0	1	1	0	W	imm8 or imm16-low								
	imm16-high								—								

SUB4S

Decimal subtraction
Subtract Nibble String

[Format] SUB4S [DS1-spec:] dst-string, [Seg-spec:] src-string
SUB4S

[Operation] BCD string (IY, CL) ← BCD string (IY, CL) - BCD string (IX, CL)

[Operand]

Mnemonic	Operand (dst, src)
SUB4S	[DS1-spec:] dst-string, [Seg-spec:] src-string
	None

[Flag]

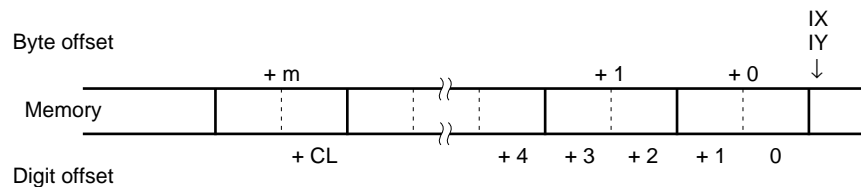
V	S	Z	AC	P	CY
U	U	×	U	U	×

[Description] Subtracts the packed BCD string addressed by the IX register from the packed BCD string addressed by the IY register, and stores the result to the string addressed by the IY register. The string length (number of BCD digits) is determined by the CL register (if the contents of CL are d, d digits) and can be set to 1 to 254 digits.

The destination string must be always located in a segment specified by the DS1 register, and segment overrides are not allowed.

The default segment register of the source register is DS0 and segment overrides are allowed, so that the source register can be located in a segment specified by any segment register.

The format of the packed BCD string is as follows:



Caution The BCD string instruction always operates in units of even-number digits. If an even number is specified as the number of digits, then the result of the operation and flags are normal. If an odd number is specified, however, the operation is executed with the even number of digits (= odd number + 1). Consequently, the result and flags indicate the even number of digits. To specify an odd number, clear the higher 4 bits of the most significant byte before executing the BCD subtract instruction. If a borrow occurs as a result, the higher 4 bits of the most significant byte indicate “9”.

[Example] MOV IX, OFFSET VAR_1
 MOV IY, OFFSET VAR_2
 MOV CL,4
 SUB4S

[Bytes] 2

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
SUB4S	[DS-spec:] dst-string, [Seg-spec:] src-string	0	0	0	0	1	1	1	1	0	0	1	0	0	0	1	0
	None																

SUBC

Subtraction with carry

Subtract with Carry

[Format] SUBC dst, src**[Operand, operation]**

Mnemonic	Operand (dst, src)	Operation
SUBC	reg, reg'	$dst \leftarrow dst - src - CY$
	mem, reg	
	reg, mem	
	reg, imm	
	mem, imm	
	acc, imm	[When W = 0] $AL \leftarrow AL - imm8 - CY$ [When W = 1] $AW \leftarrow AW - imm16 - CY$

[Flag]

V	S	Z	AC	P	CY
×	×	×	×	×	×

[Description] Subtracts the contents of the source operand specified by the second operand from the contents of the destination operand (dst) specified as the first operand, including the carry, and stores the result to the destination operand (dst).

[Example] SUBC DL, BYTE PTR [IX]

[Bytes]

Mnemonic	Operand	Bytes
SUBC	reg, reg'	2
	mem, reg	2-4
	reg, mem	2-4
	reg, imm	3, 4
	mem, imm	3-6
	acc, imm	2, 3

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
SUBC	reg, reg'	0	0	0	1	1	0	1	W	1	1	reg			reg'		
	mem, reg	0	0	0	1	1	0	0	W	mod	reg			mem			
		(disp-low)								(disp-high)							
	reg, mem	0	0	0	1	1	0	1	W	mod	reg			mem			
		(disp-low)								(disp-high)							
	reg, imm	1	0	0	0	0	0	s	W	1	1	0	1	1	reg		
		imm8 or imm16-low								imm16-high							
	mem, imm	1	0	0	0	0	0	s	W	mod	0	1	1	mem			
		(disp-low)								(disp-high)							
		imm8 or imm16-low								imm16-high							
acc, imm	0	0	0	1	1	1	0	W	imm8 or imm16-low								
	imm16-high								—								

TEST	Test Test
------	--------------

[Format] **TEST dst, src**

[Operand, operation]

Mnemonic	Operand (dst, src)	Operation
TEST	reg, reg'	$dst \wedge src$
	mem, reg	
	reg, mem	
	reg, imm	
	mem, imm	
	acc, imm	[When W = 0] $AL \wedge imm8$ [When W = 1] $AW \wedge imm16$

[Flag]

V	S	Z	AC	P	CY
0	×	×	U	×	0

[Description] ANDs the destination operand (dst) specified by the first operand with the source operand (src) specified by the second operand.
The result is not stored anywhere, and only the flags are changed.

[Example] IN AL 0D8H
TEST AL, 'A'

[Bytes]

Mnemonic	Operand	Bytes
TEST	reg, reg'	2
	mem, reg	2-4
	reg, mem	
	reg, imm	3, 4
	mem, imm	3-6
	acc, imm	2, 3

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
TEST	reg, reg'	1	0	0	0	0	1	0	W	1	1	reg'			reg		
	mem, reg	1	0	0	0	0	1	0	W	mod	reg			mem			
		(disp-low)								(disp-high)							
	reg, mem	1	0	0	0	0	1	0	W	mod	reg			mem			
		(disp-low)								(disp-high)							
	reg, imm	1	1	1	1	0	1	1	W	1	1	0	0	0	reg		
		imm8 or imm16-low								imm16-high							
	mem, imm	1	1	1	1	0	1	1	W	mod	0	0	0	mem			
		(disp-low)								(disp-high)							
		imm8 or imm16-low								imm16-high							
	acc, imm	1	0	1	0	1	0	0	W	imm8 or imm16-low							
		imm16-high								—							

TEST1**Test bit****Test Bit****[Format]** **TEST1 dst, src**

[Operation] When bit *n* of *dst* = 0 (*n* is specified by *src*): $Z \leftarrow 1$
 When bit *n* of *dst* = 1 (*n* is specified by *src*): $Z \leftarrow 0$

[Operand]

Mnemonic	Operand (dst, src)
TEST1	reg8, CL
	mem8, CL
	reg16, CL
	mem16, CL
	reg8, imm3
	mem8, imm3
	reg16, imm4
	mem16, imm4

[Flag]

V	S	Z	AC	P	CY
0	U	×	U	U	0

[Description] Sets the Z flag to 1 if bit *n* of the destination operand (*dst*) specified by the first operand (*n* is the content of the source operand (*src*) specified by the second operand), and resets the flag to 0 if bit *n* is 1.

When the operands are reg8, CL or mem8, CL, only the lower 3 bits (0-7) of the CL value are valid.

When the operands are reg16, CL or mem16, CL, only the lower 4 bits of the CL value (0-15) are valid.

When the operands are reg8, imm3, only the lower 3 bits of the immediate data at the fourth byte position of the instruction are valid.

When the operands are mem8, imm3, only the lower 3 bits of the immediate data at the end byte position of the instruction are valid.

When the operands are reg16, imm4, only the lower 4 bits of the immediate data at the fourth byte position of the instruction are valid.

When the operands are mem16, imm4, only the lower 4 bits of the immediate data at the end byte position of the instruction are valid.

[Example] MOV CL, 01
 IN AL, 0DAH
 TEST1 AL, CL; tests bit 1

[Bytes]

Mnemonic	Operand	Bytes
TEST1	reg8, CL	3
	mem8, CL	3-5
	reg16, CL	3
	mem16, CL	3-5
	reg8, imm3	4
	mem8, imm3	4-6
	reg16, imm4	4
	mem16, imm4	4-6

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
TEST1	reg8, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	0	0
		1	1	0	0	0	reg		—								
	mem8, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	0	0
		mod	0	0	0	mem		(disp-low)									
		(disp-high)								—							
	reg16, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	0	1
		1	1	0	0	0	reg		—								
	mem16, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	0	1
		mod	0	0	0	mem		(disp-low)									
		(disp-high)								—							
	reg8, imm3	0	0	0	0	1	1	1	1	0	0	0	1	1	0	0	0
		1	1	0	0	0	reg		imm3								
	mem8, imm3	0	0	0	0	1	1	1	1	0	0	0	1	1	0	0	0
		mod	0	0	0	mem		(disp-low)									
		(disp-high)								imm3							
	reg16, imm4	0	0	0	0	1	1	1	1	0	0	0	1	1	0	0	1
		1	1	0	0	0	reg		imm4								
	mem16, imm4	0	0	0	0	1	1	1	1	0	0	0	1	1	0	0	1
		mod	0	0	0	mem		(disp-low)									
		(disp-high)								imm4							

TRANS TRANSB

Conversion table transfer

Translate

Translate Byte

[Format] **TRANS src-table**
TRANS
TRANSB

[Operation] $AL \leftarrow (BW + AL)$

[Operand]

Mnemonic	Operand
TRANS	src-table
	None
TRANSB	None

[Flag]

V	S	Z	AC	P	CY

[Description] Transfers 1 byte of a 256-byte conversion table addressed by the BW and AL registers to the AL register. At this time, the BW register specifies the first address of the table, and the AL register specifies an offset value in a 256-byte area from the first address.

[Example] TRANS SIN_TBL

[Bytes] 1

[Word Format]

Mnemonic	Operand	Operation code							
		7	6	5	4	3	2	1	0
TRANS	src-table	1	1	0	1	0	1	1	1
	None								
TRANSB	None								

TSKSW [Added to V20/V30]

Register bank switching

Task Switch

[Format] TSKSW reg16

[Operation] RB0-2 ← lower 3 bits of reg16
 PSW save, PC save of old register bank ← PSW, PC
 PSW, PC ← PSW save, PC save of new register bank

[Operand]

Mnemonic	Operand
TSKSW	reg16

[Flag]

RB2	RB1	RB0	V	DIR	IE	BRK	S	Z	F1	AC	F0	P	$\overline{\text{BRK}}$	CY
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

[Description] This instruction switches the register bank and is used for high-speed task switching. It saves the contents of the PC to the PC save area in the current register bank. In the same manner, the contents of the PSW are saved to the PSW save area.

[Example] TSKSW BP

[Bytes] 3

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
TSKSW	reg16	0	0	0	0	1	1	1	1	1	0	0	1	0	1	0	0
		1	1	1	1	1	reg			—							

XCHData exchange
Exchange**[Format]** XCH dst, src**[Operation]** dst ↔ src**[Operand]**

Mnemonic	Operand (dst, src)
XCH	reg, reg'
	mem, reg
	reg, mem
	AW, reg16
	reg16, AW

[Flag]

V	S	Z	AC	P	CY

[Description] Exchanges the contents of the destination operand (dst) specified by the first operand with the contents of the source operand (src) specified by the second operand.

[Example] MOV AW, 100H
 MOV BW, 50H
 XCH AW, BW
 ; AW = 50H, BW = 100H

[Bytes]

Mnemonic	Operand	Bytes
XCH	reg, reg'	2
	mem, reg	2-4
	reg, mem	
	AW, reg16	1
	reg16, AW	

[Word Format]

Mnemonic	Operand	Operation code															
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
XCH	reg, reg'	1	0	0	0	0	1	1	W	1	1	reg			reg'		
	mem, reg	1	0	0	0	0	1	1	W	mod		reg			mem		
		(disp-low)								(disp-high)							
	reg, mem	1	0	0	0	0	1	1	W	mod		reg			mem		
		(disp-low)								(disp-high)							
	AW, reg16	1	0	0	1	0	reg			—							
reg16, AW	1	0	0	1	0	reg			—								

XOR

Exclusive logical sum

Exclusive Or

[Format] XOR dst, src**[Operand, operation]**

Mnemonic	Operand (dst, src)	Operation
XOR	reg, reg'	dst ← dst ∨ src
	mem, reg	
	reg, mem	
	reg, imm	
	mem, imm	
	acc, imm	[When W = 0] AL ← AL ∨ imm8 [When W = 1] AW ← AW ∨ imm16

[Flag]

V	S	Z	AC	P	CY
0	×	×	U	×	0

[Description] Exclusive-ORs the destination operand (dst) specified by the first operand with the source operand (src) specified by the second operand, and stores the result to the destination operand (dst).**[Example]**

- XOR CL, DL
- XOR CW, CW ; clears CW register
- XOR AW, DW

[Bytes]

Mnemonic	Operand	Bytes
XOR	reg, reg'	2
	mem, reg	2-4
	reg, mem	2-4
	reg, imm	3, 4
	mem, imm	3-6
	acc, imm	2, 3

[Word Format]

Mnemonic	Operand	Operation code																
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
XOR	reg, reg'	0	0	1	1	0	0	1	W	1	1	reg				reg'		
	mem, reg	0	0	1	1	0	0	0	W	mod		reg				mem		
		(disp-low)								(disp-high)								
	reg, mem	0	0	1	1	0	0	1	W	mod		reg				mem		
		(disp-low)								(disp-high)								
	reg, imm ^{Note}	1	0	0	0	0	0	0	W	1	1	1	1	0	reg			
		imm8 or imm16-low								imm16-high								
	mem, imm	1	0	0	0	0	0	0	W	mod		1	1	0	mem			
		(disp-low)								(disp-high)								
		imm8 or imm16-low								imm16-high								
acc, imm	0	0	1	1	0	1	0	W	imm8 or imm16-low									
	imm16-high								—									

Note With some assemblers and compilers, the codes shown below may be generated.

Operation code															
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	W	1	1	1	1	0	reg		
imm8								—							

Even in this case, instructions are executed normally. Take precautions, however, since some emulators do not support the disassembly function or line assembly function for this instruction.

2.2 Number of Clocks Instruction Execution

Table 2-8 shows the number of execution clocks and the number of word transfers of each instruction in the alphabetical order of mnemonics.

(1) Number of clocks

The values shown in the table are the time required for the execution unit to execute an instruction under the following conditions:

- (a) Does not include prefetch time, predecode time, and wait time for bus.
- (b) The number of clocks may increase depending on the fetch status of the op code and the processing status of the pipeline.

For details, refer to **APPENDIX C NUMBER OF PROGRAM EXECUTION CLOCKS**.

- (c) I/O access is assumed to have 0 wait.
- (d) Primitive block transfer instructions and primitive I/O instructions include repeat prefix.
- (e) Bus cycle is started two times in the case of word data to an odd address.
- (f) External data bus width is as follows:

- V25, V25+ : 8 bits
- V35, V35+ : 16 bits

- (g) If a memory operand is specified, the number of clocks differs depending on the addressing mode. "EA" takes the following value:

mem \ mod	00		01		10	
		Clocks		Clocks		Clocks
000	BW+IX	3	BW+IX+disp8	3	BW+IX+disp16	4
001	BW+IY	3	BW+IY+disp8	3	BW+IY+disp16	4
010	BP+IX	3	BP+IX+disp8	3	BP+IX+disp16	4
011	BP+IY	3	BP+IY+disp8	3	BP+IY+disp16	4
100	IX	3	IX+disp8	3	IX+disp16	4
101	IY	3	IY+disp8	3	IY+disp16	4
110	Direct address	3	BP+disp8	3	BP+disp16	4
111	BW	3	BW+disp8	3	BW+disp16	4

- (h) "T" indicates the number of wait states. Select the desired number of wait states, starting from "0" (no wait).
- (i) The number of clocks of the branch instruction is indicated as follows:

On left of / : Number of clocks when branch occurs
 On right of / : Number of clocks when branch does not occur

(2) Number of word transfers

The number of word transfers is the number of times the bus is accessed for word data (16 bits) generated as a result of executing an instruction.

Table 2-8. Number of Instruction Execution Clocks (1/14)

Mnemonic	Operand	Number of word transfers	V25, V25+				V35, V35+			
			Byte processing		Word processing		Byte processing		Word processing	
			Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled
ADD	reg, reg'	0	2	2	2	2	2	2	2	2
	mem, reg	2	EA+8+2•T	EA+6+T	EA+12+4•T	EA+8+2•T	EA+10+2•T	EA+7+T	EA+10+2•T	EA+7+T
	reg, mem	1	EA+6+T	EA+6+T	EA+8+2•T	EA+8+2•T	EA+7+T	EA+7+T	EA+7+T	EA+7+T
	reg, imm	0	5	5	6	6	5	5	6	6
	mem, imm	2	EA+9+2•T	EA+7+2•T	EA+14+4•T	EA+10+4•T	EA+11+2•T	EA+9+2•T	EA+12+2•T	EA+8+2•T
	acc, imm	0	5	5	6	6	5	5	6	6
ADD4S ^{Note}	[DS1-spec:]dst-string, [Seg-spec:]src-string	0	22+(27+3•T)•m	22+(25+3•T)•m	–	–	22+(30+3•T)•m	22+(28+3•T)•m	–	–
	None	0	22+(27+3•T)•m	22+(25+3•T)•m	–	–	22+(30+3•T)•m	22+(28+3•T)•m	–	–
ADDC	reg, reg'	0	2	2	2	2	2	2	2	2
	mem, reg	2	EA+8+2•T	EA+6+T	EA+12+4•T	EA+8+2•T	EA+10+2•T	EA+7+T	EA+10+2•T	EA+7+T
	reg, mem	1	EA+6+T	EA+6+T	EA+8+2•T	EA+8+2•T	EA+7+T	EA+7+T	EA+7+T	EA+7+T
	reg, imm	0	5	5	6	6	5	5	6	6
	mem, imm	2	EA+9+2•T	EA+7+2•T	EA+14+4•T	EA+10+4•T	EA+10+2•T	EA+9+2•T	EA+12+2•T	EA+8+2•T
	acc, imm	0	5	5	6	6	5	5	6	6
ADJ4A	None	0	9	9	–	–	9	9	–	–
ADJ4S	None	0	9	9	–	–	9	9	–	–
ADJBA	None	0	17	17	–	–	17	17	–	–
ADJBS	None	0	17	17	–	–	17	17	–	–
AND	reg, reg'	0	2	2	2	2	2	2	2	2
	mem, reg	2	EA+8+2•T	EA+6+T	EA+12+4•T	EA+8+2•T	EA+10+2•T	EA+7+T	EA+10+2•T	EA+7+T
	reg, mem	1	EA+6+T	EA+6+T	EA+8+2•T	EA+8+2•T	EA+7+T	EA+7+T	EA+7+T	EA+7+T
	reg, imm	0	5	5	6	6	5	5	6	6
	mem, imm	2	EA+9+2•T	EA+7+2•T	EA+14+4•T	EA+10+4•T	EA+11+2•T	EA+9+2•T	EA+12+2•T	EA+8+2•T
	acc, imm	0	5	5	6	6	5	5	6	6

Note m : number of BCD digits × 1/2

Table 2-8. Number of Instruction Execution Clocks (2/14)

Mnemonic	Operand	Number of word transfers	V25, V25+				V35, V35+			
			Byte processing		Word processing		Byte processing		Word processing	
			Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled
BC	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BCWZ	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BE	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BGE	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BGT	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BH	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BL	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BLE	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BLT	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BN	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BNC	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BNE	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BNH	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BNL	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BNV	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BNZ	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BP	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BPE	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BPO	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BR	near-label	0	–	–	12	12	–	–	12	12
	short-label	0	–	–	12	12	–	–	12	12
	regptr16	0	–	–	13	13	–	–	13	13
	memptr16	1	–	–	EA+17+2•T	EA+17+2•T	–	–	EA+16+T	EA+16+T
	far-label	0	–	–	15	15	–	–	15	15
	memptr32	2	–	–	EA+25+4•T	EA+25+4•T	–	–	EA+23+2•T	EA+23+2•T

Table 2-8. Number of Instruction Execution Clocks (3/14)

Mnemonic	Operand	Number of word transfers	V25, V25+				V35, V35+			
			Byte processing		Word processing		Byte processing		Word processing	
			Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled
BRK	3	5	–	–	55+10•T	43+10•T	–	–	50+5•T	38+5•T
	imm8(≠3)	5	2	2	56+10•T	44+10•T	2	2	51+5•T	39+5•T
BRKCS	reg16	0	–	–	15	15	–	–	15	15
BRKV	None	5	–	–	55+10•T	43+10•T	–	–	50+5•T	38+5•T
BTCLR	sfr, imm3, short-label	0	29/21	29/21	–	–	29/21	29/21	–	–
BUSLOCK	None	0	2	2	2	2	2	2	2	2
BV	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
BZ	short-label	0	–	–	15/8	15/8	–	–	15/8	15/8
CALL	near-proc	1	–	–	22+2•T	18+2•T	–	–	21+T	17+T
	regptr16	1	–	–	22+2•T	18+2•T	–	–	21+T	17+T
	memptr16	2	–	–	EA+26+4•T	EA+24+4•T	–	–	EA+24+2•T	EA+22+2•T
	far-proc	2	–	–	38+4•T	34+4•T	–	–	36+2•T	32+2•T
	memptr32	4	–	–	EA+36+8•T	EA+24+8•T	–	–	EA+32+4•T	EA+20+4•T
CHKIND	reg16, mem32 (when interrupt condition is satisfied)	7	–	–	EA+26+4•T	EA+26+4•T	–	–	EA+24+2•T	EA+24+2•T
	reg16, mem32 (when interrupt condition is not satisfied)	2	–	–	EA+26+4•T	EA+26+4•T	–	–	EA+24+2•T	EA+24+2•T
CLR1	reg8, CL	0	8	8	–	–	8	8	–	–
	mem8, CL	0	EA+14+2•T	EA+12+T	–	–	EA+16+2•T	EA+13+T	–	–
	reg16, CL	0	–	–	8	8	–	–	8	8
	mem16, CL	2	–	–	EA+18+4•T	EA+14+2•T	–	–	EA+16+2•T	EA+13+T
	reg8, imm3	0	7	7	–	–	7	7	–	–
	mem8, imm3	0	EA+11+2•T	EA+9+T	–	–	EA+13+2•T	EA+10+T	–	–
	reg16, imm4	0	–	–	7	7	–	–	7	7
	mem16, imm4	3	–	–	EA+15+4•T	EA+10+2•T	–	–	EA+13+2•T	EA+9+T
	CY	0	2	2	2	2	2	2	2	2
DIR	0	2	2	2	2	2	2	2	2	

Table 2-8. Number of Instruction Execution Clocks (4/14)

Mnemonic	Operand	Number of word transfers	V25, V25+				V35, V35+				
			Byte processing		Word processing		Byte processing		Word processing		
			Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	
CMP	reg, reg'	0	2	2	2	2	2	2	2	2	2
	mem, reg	1	EA+6+T	EA+6+T	EA+8+2•T	EA+8+2•T	EA+7+T	EA+7+T	EA+7+T	EA+7+T	EA+7+T
	reg, mem	1	EA+6+T	EA+6+T	EA+8+2•T	EA+8+2•T	EA+7+T	EA+7+T	EA+7+T	EA+7+T	EA+7+T
	reg, imm	0	5	5	6	6	5	5	6	6	6
	mem, imm	1	EA+7+T	EA+7+T	EA+10+2•T	EA+10+2•T	EA+8+T	EA+8+T	EA+9+T	EA+9+T	EA+9+T
	acc, imm	0	5	5	6	6	5	5	6	6	6
CMP4S Note 1	[DS1-spec:]dst-string, [Seg-spec:]src-string	0	22+(23+3•T)•m	22+(23+2•T)•m	–	–	22+(25+2•T)•m	22+(25+2•T)•m	–	–	–
	None	0	22+(23+3•T)•m	22+(23+2•T)•m	–	–	22+(25+2•T)•m	22+(25+2•T)•m	–	–	–
CMPBK Note 2	[Seg-spec:]src-block, [DS1-spec:]dst-block	2 × rep	16+(21+2•T)•n	16+(21+2•T)•n	16+(25+4•T)•n	16+(25+4•T)•n	16+(23+2•T)•n	16+(23+2•T)•n	16+(23+2•T)•n	16+(23+2•T)•n	16+(23+2•T)•n
		(2) Note 3	23+2•T	19+2•T	27+4•T	21+4•T	25+2•T	21+2•T	25+2•T	19+2•T	19+2•T
CMPBKB Note 2	None	2 × rep	16+(21+2•T)•n	16+(21+2•T)•n	16+(25+4•T)•n	16+(25+4•T)•n	16+(23+2•T)•n	16+(23+2•T)•n	16+(23+2•T)•n	16+(23+2•T)•n	16+(23+2•T)•n
		(2) Note 3	23+2•T	19+2•T	27+4•T	21+4•T	25+2•T	21+2•T	25+2•T	19+2•T	19+2•T
CMPBKW Note 2	None	2 × rep	16+(21+2•T)•n	16+(21+2•T)•n	16+(25+4•T)•n	16+(25+4•T)•n	16+(23+2•T)•n	16+(23+2•T)•n	16+(23+2•T)•n	16+(23+2•T)•n	16+(23+2•T)•n
		(2) Note 3	23+2•T	19+2•T	27+4•T	21+4•T	25+2•T	21+2•T	25+2•T	19+2•T	19+2•T
CMPM Note 2	[DS1-spec:]dst-block	1 × rep	16+(15+T)•n	16+(15+T)•n	16+(17+2•T)•n	16+(17+2•T)•n	16+(16+T)•n	16+(16+T)•n	16+(16+T)•n	16+(16+T)•n	16+(16+T)•n
		(1) Note 3	17+T	17+T	19+2•T	19+2•T	18+T	18+T	19+2•T	19+2•T	19+2•T
CMPMB Note 2	None	1 × rep	16+(15+T)•n	16+(15+T)•n	16+(17+2•T)•n	16+(17+2•T)•n	16+(16+T)•n	16+(16+T)•n	16+(16+T)•n	16+(16+T)•n	16+(16+T)•n
		(1) Note 3	17+T	17+T	19+2•T	19+2•T	18+T	18+T	19+2•T	19+2•T	19+2•T
CMPMW Note 2	None	1 × rep	16+(15+T)•n	16+(15+T)•n	16+(17+2•T)•n	16+(17+2•T)•n	16+(16+T)•n	16+(16+T)•n	16+(16+T)•n	16+(16+T)•n	16+(16+T)•n
		(1) Note 3	17+T	17+T	19+2•T	19+2•T	18+T	18+T	19+2•T	19+2•T	19+2•T
CVTBD	None	0	19	19	–	–	19	19	–	–	–
CVTBW	None	0	3	3	–	–	3	3	–	–	–
CVTDB	None	0	20	20	–	–	20	20	–	–	–
CVTWL	None	0	–	–	8	8	–	–	8	8	8

- Notes**
1. m : number of BCD digits × 1/2
 2. n : number of repeats (n ≥ 1)
 3. () : applicable to only one processing

Table 2-8. Number of Instruction Execution Clocks (5/14)

Mnemonic	Operand	Number of word transfers	V25, V25+				V35, V35+			
			Byte processing		Word processing		Byte processing		Word processing	
			Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled
DBNZ	short-label	0	–	–	17/8	17/8	–	–	17/8	17/8
DBNZE	short-label	0	–	–	17/8	17/8	–	–	17/8	17/8
DBNZNE	short-label	0	–	–	17/8	17/8	–	–	17/8	17/8
DEC	reg8	0	5	5	–	–	5	5	–	–
	mem	2	EA+11+2•T	EA+9+2•T	EA+15+4•T	EA+11+4•T	EA+13+2•T	EA+11+2•T	EA+13+2•T	EA+9+2•T
	reg16	0	–	–	2	2	–	–	2	2
DI	None	0	4	4	4	4	4	4	4	4
DISPOSE	None	1	–	–	12+2•T	12+2•T	–	–	11+T	11+T
DIV	reg8	0	45 – 56	45 – 56	–	–	45 – 56	45 – 56	–	–
	mem8	0	EA+48+T to EA+58+T	EA+48+T to EA+58+T	–	–	EA+49+T to EA+59+T	EA+49+T to EA+59+T	–	–
	reg16	0	–	–	54 – 64	54 – 64	–	–	54 – 64	54 – 64
	mem16	1	–	–	EA+58+2•T to EA+68+2•T	EA+58+2•T to EA+68+2•T	–	–	EA+57+T to EA+67+T	EA+57+T to EA+67+T
DIVU	reg8	0	31	31	–	–	31	31	–	–
	mem8	0	EA+33+T	EA+33+T	–	–	EA+34+T	EA+34+T	–	–
	reg16	0	–	–	39	39	–	–	39	39
	mem16	1	–	–	EA+43+2•T	EA+43+2•T	–	–	EA+43+2•T	EA+43+2•T
DS0:	None	0	2	2	2	2	2	2	2	2
DS1:	None	0	2	2	2	2	2	2	2	2

Table 2-8. Number of Instruction Execution Clocks (6/14)

Mnemonic	Operand	Number of word transfers	V25, V25+				V35, V35+			
			Byte processing		Word processing		Byte processing		Word processing	
			Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled
EI	None	0	12	12	12	12	12	12	12	12
EXT	reg8, reg8'	1 or 2	41 – 121 (differs depending on bit length)							
	reg8, imm4	1 or 2	42 – 122 (differs depending on bit length)							
FINT	None	0	2	2	2	2	2	2	2	2
FPO1	fp-op	5	–	–	60+10•T	48+10•T	–	–	55+5•T	43+5•T
	fp-op, mem	5	–	–	60+10•T	48+10•T	–	–	55+5•T	43+5•T
FPO2	fp-op	5	–	–	60+10•T	48+10•T	–	–	55+5•T	43+5•T
	fp-op, mem	5	–	–	60+10•T	48+10•T	–	–	55+5•T	43+5•T
HALT	None	0	–	–	–	–	–	–	–	–
IN ^{Note 1}	acc, imm8	1	14+T	14+T	16+2•T	16+2•T	15+T	15+T	15+T	15+T
	acc, DW	1	13+T	13+T	15+2•T	15+2•T	14+T	14+T	14+T	14+T
INC	reg8	0	5	5	–	–	5	5	–	–
	mem	2	EA+11+2•T	EA+9+2•T	EA+15+4•T	EA+11+4•T	EA+13+2•T	EA+11+2•T	EA+13+2•T	EA+9+2•T
	reg16	0	–	–	2	2	–	–	2	2
INM ^{Note 1,2}	[DS1-spec:]dst-block, DW	2 × rep	18+(13+2•T)•n	18+(11+2•T)•n	18+(15+4•T)•n	18+(11+4•T)•n	18+(15+2•T)•n	18+(13+2•T)•n	18+(13+2•T)•n	18+(9+2•T)•n
		(2) ^{Note 3}	19+2•T	17+2•T	21+4•T	17+4•T	21+2•T	19+2•T	19+2•T	15+2•T
INS	reg8, reg8'	2 or 4	63 – 155 (differs depending on bit length)							
	reg8, imm4	2 or 4	64 – 156 (differs depending on bit length)				–	–	64 – 156	64 – 156
LDEA	reg16, mem16	0	–	–	EA+2	EA+2	–	–	EA+2	EA+2
LDM ^{Note 2}	[Seg-spec:]src-block	1 × rep	16+(10+T)•n	16+(10+T)•n	16+(12+2•T)•n	16+(12+2•T)•n	16+(11+T)•n	16+(11+T)•n	16+(11+T)•n	16+(11+T)•n
		(1) ^{Note 3}	12+T	12+T	14+2•T	14+2•T	13+T	13+T	13+T	13+T
LDMB ^{Note 2}	None	1 × rep	16+(10+T)•n	16+(10+T)•n	16+(12+2•T)•n	16+(12+2•T)•n	16+(11+T)•n	16+(11+T)•n	16+(11+T)•n	16+(11+T)•n
		(1) ^{Note 3}	12+T	12+T	14+2•T	14+2•T	13+T	13+T	13+T	13+T
LDMW ^{Note 2}	None	1 × rep	16+(10+T)•n	16+(10+T)•n	16+(12+2•T)•n	16+(12+2•T)•n	16+(11+T)•n	16+(11+T)•n	16+(11+T)•n	16+(11+T)•n
		(1) ^{Note 3}	12+T	12+T	14+2•T	14+2•T	13+T	13+T	13+T	13+T

- Notes**
1. When $\overline{\text{IBRK}} = 1$
 2. n : number of repeats ($n \geq 1$)
 3. () : applicable to only one processing

Table 2-8. Number of Instruction Execution Clocks (7/14)

Mnemonic	Operand	Number of word transfers	V25, V25+				V35, V35+			
			Byte processing		Word processing		Byte processing		Word processing	
			Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled
MOV	reg, reg'	0	2	2	2	2	2	2	2	2
	mem, reg	1	EA+4+T	EA+2	EA+6+2•T	EA+2	EA+5+T	EA+2	EA+5+T	EA+2
	reg, mem	1	EA+6+T	EA+6+T	EA+8+2•T	EA+8+2•T	EA+7+T	EA+7+T	EA+7+T	EA+7+T
	mem, imm	1	EA+5+T	EA+5+T	EA+5+2•T	EA+5+2•T	EA+6+T	EA+6+T	EA+6+T	EA+6+T
	reg, imm	0	5	5	6	6	5	5	6	6
	acc, dmem	1	9+T	9+T	11+2•T	11+2•T	10+T	10+T	10+T	10+T
	dmem, acc	1	7+T	5	9+2•T	5	8+T	5	8+T	5
	sreg, reg16	0	–	–	4	4	–	–	4	4
	sreg, mem16	1	–	–	EA+10+2•T	EA+10+2•T	–	–	EA+9+T	EA+9+T
	reg16, sreg	0	–	–	3	3	–	–	3	3
	mem16, sreg	1	–	–	EA+7+2•T	EA+3	–	–	EA+6+T	EA+3
	DS0, reg16, mem32	2	–	–	EA+19+4•T	EA+19+4•T	–	–	EA+17+2•T	EA+17+2•T
	DS1, reg16, mem32	2	–	–	EA+19+4•T	EA+19+4•T	–	–	EA+17+2•T	EA+17+2•T
	AH, PSW	0	2	2	–	–	2	2	–	–
PSW, AH	0	3	3	–	–	3	3	–	–	
MOVBK Note 1	[DS1-spec:]dst-block, [Seg-spec:]src-block	2 × rep	16+(16+2•T)•n	16+(12+T)•n	16+(20+4•T)•n	16+(12+2•T)•n	16+(18+2•T)•n	16+(13+T)•n	16+(18+2•T)•n	16+(10+T)•n
		(2) Note 2	20+2•T	16+T	24+4•T	20+2•T	22+2•T	17+T	22+2•T	19+T
MOVBKB Note 1	None	2 × rep	16+(16+2•T)•n	16+(12+T)•n	16+(20+4•T)•n	16+(12+2•T)•n	16+(18+2•T)•n	16+(13+T)•n	16+(18+2•T)•n	16+(10+T)•n
		(2) Note 2	20+2•T	16+T	24+4•T	20+2•T	22+2•T	17+T	22+2•T	19+T
MOVBKW Note 1	None	2 × rep	16+(16+2•T)•n	16+(12+T)•n	16+(20+4•T)•n	16+(12+2•T)•n	16+(18+2•T)•n	16+(13+T)•n	16+(18+2•T)•n	16+(10+T)•n
		(2) Note 2	20+2•T	16+T	24+4•T	20+2•T	22+2•T	17+T	22+2•T	19+T
MOVSPA	None	0	–	–	16	16	–	–	16	16
MOVSPB	reg16	0	–	–	11	11	–	–	11	11

- Notes**
1. n : number of repeats ($n \geq 1$)
 2. () : applicable to only one processing

Table 2-8. Number of Instruction Execution Clocks (8/14)

Mnemonic	Operand	Number of word transfers	V25, V25+				V35, V35+			
			Byte processing		Word processing		Byte processing		Word processing	
			Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled
MUL	reg8	0	31 – 40	31 – 40	–	–	31 – 40	31 – 40	–	–
	mem8	0	EA+33+T to EA+42+T	EA+33+T to EA+42+T	–	–	EA+34+T to EA+43+T	EA+34+T to EA+43+T	–	–
	reg16	0	–	–	39 – 48	39 – 48	–	–	39 – 48	39 – 48
	mem16	1	–	–	EA+43+2•T to EA+52+2•T	EA+43+2•T to EA+52+2•T	–	–	EA+42+T to EA+51+T	EA+42+T to EA+51+T
	reg16, imm8	0	–	–	39 – 49	39 – 49	–	–	39 – 49	39 – 49
	reg16, imm16	0	–	–	40 – 50	40 – 50	–	–	40 – 50	40 – 50
	reg16, reg16', imm8	0	–	–	39 – 49	39 – 49	–	–	39 – 49	39 – 49
	reg16, mem16, imm8	1	–	–	EA+43+2•T to EA+53+2•T	EA+43+2•T to EA+53+2•T	–	–	EA+42+T to EA+52+T	EA+42+T to EA+52+T
	reg16, reg16', imm16	0	–	–	40 – 50	40 – 50	–	–	40 – 50	40 – 50
reg16, mem16, imm16	1	–	–	EA+44+2•T to EA+54+2•T	EA+44+2•T to EA+54+2•T	–	–	EA+43+T to EA+53+T	EA+43+T to EA+53+T	
MULU	reg8	0	24	24	–	–	24	24	–	–
	mem8	1	EA+26+T	EA+26+T	–	–	EA+27+T	EA+27+T	–	–
	reg16	0	–	–	32	32	–	–	32	32
	mem16	1	–	–	EA+34+2•T	EA+34+2•T	–	–	EA+33+T	EA+33+T
NEG	reg	0	5	5	5	5	5	5	5	5
	mem	2	EA+11+2•T	EA+9+T	EA+15+4•T	EA+11+2•T	EA+13+2•T	EA+10+T	EA+13+2•T	EA+10+T
NOP	None	0	4	4	4	4	4	4	4	4
NOT	reg	0	5	5	5	5	5	5	5	5
	mem	2	EA+11+2•T	EA+9+T	EA+15+4•T	EA+11+2•T	EA+10+2•T	EA+10+T	EA+13+2•T	EA+10+T
NOT1	reg8, CL	0	7	7	–	–	7	7	–	–
	mem8, CL	0	EA+13+2•T	EA+11+2•T	–	–	EA+15+2•T	EA+12+T	–	–
	reg16, CL	0	–	–	7	7	–	–	7	7
	mem16, CL	2	–	–	EA+17+4•T	EA+13+2•T	–	–	EA+15+2•T	EA+12+T
	reg8, imm3	0	6	6	–	–	6	6	–	–
	mem8, imm3	0	EA+10+2•T	EA+8+T	–	–	EA+12+2•T	EA+9+T	–	–
	reg16, imm4	0	–	–	6	6	–	–	6	6
	mem16, imm4	2	–	–	EA+14+4•T	EA+10+2•T	–	–	EA+12+2•T	EA+9+T
CY	0	2	2	2	2	2	2	2	2	

Table 2-8. Number of Instruction Execution Clocks (9/14)

Mnemonic	Operand	Number of word transfers	V25, V25+				V35, V35+			
			Byte processing		Word processing		Byte processing		Word processing	
			Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled
OR	reg, reg'	0	2	2	2	2	2	2	2	2
	mem, reg	2	EA+8+2•T	EA+6+T	EA+12+4•T	EA+8+2•T	EA+10+2•T	EA+7+T	EA+10+2•T	EA+7+T
	reg, mem	1	EA+6+T	EA+6+T	EA+8+2•T	EA+8+2•T	EA+7+T	EA+7+T	EA+7+T	EA+7+T
	reg, imm	0	5	5	6	6	5	5	6	6
	mem, imm	2	EA+9+2•T	EA+7+2•T	EA+14+4•T	EA+10+4•T	EA+11+2•T	EA+9+2•T	EA+12+12•T	EA+8+2•T
	acc, imm	0	5	5	6	6	5	5	6	6
OUT ^{Note 1}	imm8, acc	1	10+T	10+T	10+2•T	10+2•T	11+T	11+T	9+T	9+T
	DW, acc	1	9+T	9+T	9+2•T	9+2•T	9+T	9+T	9+2•T	9+2•T
OUTM ^{Note 2}	DW, [Seg-spec]src-block	2 × rep	18+(13+2•T)•n	18+(11+2•T)•n	18+(15+4•T)•n	18+(11+4•T)•n	18+(15+2•T)•n	18+(13+2•T)•n	18+(13+2•T)•n	18+(9+2•T)•n
		(2) ^{Note 3}	19+2•T	17+2•T	21+4•T	17+4•T	21+2•T	19+2•T	19+2•T	15+2•T
POLL	None	0	–	–	–	–	–	–	–	–
POP	mem16	2	–	–	EA+16+4•T	EA+12+2•T	–	–	EA+14+2•T	EA+11+T
	reg16	1	–	–	12+2•T	12+2•T	–	–	11+T	11+T
	sreg	1	–	–	13+2•T	13+2•T	–	–	12+T	12+T
	PSW	1	–	–	14+2•T	14+2•T	–	–	13+T	13+T
	R	7	–	–	82+16•T	58	–	–	74+8•T	58
PREPARE	imm16, imm8 (when imm8 = 0)	0	27+2•T				26+T			
	imm16, imm8 (when imm8 = 1)	1+2	39+4•T				37+2•T			
	imm16, imm8 (imm8 ≥ n, when n > 1)	(imm8–1)	46+19 (n–1)+4•T				44+19 (n–1)+2•T			
PS:	None	0	2	2	2	2	2	2	2	2
PUSH	mem16	2	–	–	EA+18+4•T	EA+14+4•T	–	–	EA+16+2•T	EA+12+2•T
	reg16	1	–	–	10+2•T	6	–	–	13+T	9+T
	sreg	1	–	–	11+2•T	7	–	–	10+T	7
	PSW	1	–	–	10+2•T	6	–	–	9+T	6
	R	8	–	–	82+16•T	50	–	–	74+8•T	50
	imm8	1	–	–	13+2•T	9	–	–	12+T	9
	imm16	1	–	–	14+2•T	10	–	–	13+T	10

- Notes**
1. When $\overline{\text{IBRK}} = 1$
 2. n : number of repeats ($n \geq 1$)
 3. () : applicable to only one processing

Table 2-8. Number of Instruction Execution Clocks (10/14)

Mnemonic	Operand	Number of word transfers	V25, V25+				V35, V35+				
			Byte processing		Word processing		Byte processing		Word processing		
			Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	
REP	None	0	2	2	2	2	2	2	2	2	2
REPC	None	0	2	2	2	2	2	2	2	2	2
REPE	None	0	2	2	2	2	2	2	2	2	2
REPNC	None	0	2	2	2	2	2	2	2	2	2
REPNE	None	0	2	2	2	2	2	2	2	2	2
REPZ	None	0	2	2	2	2	2	2	2	2	2
REPZ	None	0	2	2	2	2	2	2	2	2	2
RET	None (in-segment call)	1	–	–	20+2•T	20+2•T	–	–	19+T	19+T	19+T
	pop-value (in-segment call)	1	–	–	20+2•T	20+2•T	–	–	19+T	19+T	19+T
	None (out-segment call)	2	–	–	29+4•T	29+4•T	–	–	27+2•T	27+2•T	27+2•T
	pop-value (out-segment call)	2	–	–	30+4•T	30+4•T	–	–	28+2•T	28+2•T	28+2•T
RETI	None	3	–	–	45+6•T	37+2•T	–	–	42+3•T	36+T	36+T
RETRBI	None	0	–	–	12	12	–	–	12	12	12
ROL ^{Note}	reg, 1	0	8	8	8	8	8	8	8	8	8
	mem, 1	2	EA+14+2•T	EA+12+T	EA+18+4•T	EA+14+2•T	EA+16+2•T	EA+13+T	EA+16+2•T	EA+13+T	EA+13+T
	reg, CL	0	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n
	mem, CL	2	EA+17+2•T+2•n	EA+15+T+2•n	EA+21+4•T+2•n	EA+17+2•T+2•n	EA+19+2•T+2•n	EA+16+T+2•n	EA+19+2•T+2•n	EA+16+T+2•n	EA+16+T+2•n
	reg, imm8	0	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n
	mem, imm8	2	EA+13+2•T+2•n	EA+11+T+2•n	EA+17+4•T+2•n	EA+13+2•T+2•n	EA+15+2•T+2•n	EA+12+T+2•n	EA+15+2•T+2•n	EA+12+T+2•n	EA+12+T+2•n
ROL4	reg8	0	17	17	–	–	17	17	–	–	–
	mem8	0	EA+18+2•T	EA+16+2•T	–	–	EA+20+2•T	EA+18+2•T	–	–	–
	reg, 1	0	8	8	8	8	8	8	8	8	8
	mem, 1	2	EA+14+2•T	EA+12+T	EA+18+4•T	EA+14+2•T	EA+16+2•T	EA+13+T	EA+16+2•T	EA+13+T	EA+13+T
	reg, CL	0	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n
	mem, CL	2	EA+17+2•T+2•n	EA+15+T+2•n	EA+21+4•T+2•n	EA+17+2•T+2•n	EA+19+2•T+2•n	EA+16+T+2•n	EA+19+2•T+2•n	EA+16+T+2•n	EA+16+T+2•n
	reg, imm8	0	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n
	mem, imm8	2	EA+13+2•T+2•n	EA+11+T+2•n	EA+17+4•T+2•n	EA+13+2•T+2•n	EA+15+2•T+2•n	EA+12+T+2•n	EA+15+2•T+2•n	EA+12+T+2•n	EA+12+T+2•n

Note n : shift amount

Table 2-8. Number of Instruction Execution Clocks (11/14)

Mnemonic	Operand	Number of word transfers	V25, V25+				V35, V35+			
			Byte processing		Word processing		Byte processing		Word processing	
			Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled
ROR <i>Note</i>	reg, 1	0	8	8	8	8	8	8	8	8
	mem, 1	2	EA+14+2•T	EA+12+T	EA+18+4•T	EA+14+2•T	EA+16+2•T	EA+13+T	EA+16+2•T	EA+13+T
	reg, CL	0	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n
	mem, CL	2	EA+17+2•T+2•n	EA+15+T+2•n	EA+21+4•T+2•n	EA+17+2•T+2•n	EA+19+2•T+2•n	EA+16+T+2•n	EA+19+2•T+2•n	EA+16+T+2•n
	reg, imm8	0	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n
	mem, imm8	2	EA+13+2•T+2•n	EA+11+T+2•n	EA+17+4•T+2•n	EA+13+2•T+2•n	EA+15+2•T+2•n	EA+12+T+2•n	EA+15+2•T+2•n	EA+12+T+2•n
ROR4	reg8	0	21	21	–	–	21	21	–	–
	mem8	0	EA+24+2•T	EA+22+2•T	–	–	EA+26+2•T	EA+24+2•T	–	–
RORC <i>Note</i>	reg, 1	0	8	8	8	8	8	8	8	8
	mem, 1	2	EA+14+2•T	EA+12+2•T	EA+18+4•T	EA+14+2•T	EA+16+2•T	EA+13+T	EA+16+2•T	EA+13+T
	reg, CL	0	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n
	mem, CL	2	EA+17+2•T+2•n	EA+15+T+2•n	EA+21+4•T+2•n	EA+17+2•T+2•n	EA+19+2•T+2•n	EA+16+T+2•n	EA+19+2•T+2•n	EA+16+T+2•n
	reg, imm8	0	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n
	mem, imm8	2	EA+13+2•T+2•n	EA+11+T+2•n	EA+17+4•T+2•n	EA+13+2•T+2•n	EA+15+2•T+2•n	EA+12+T+2•n	EA+15+2•T+2•n	EA+12+T+2•n
SET1	reg8, CL	0	7	7	–	–	7	7	–	–
	mem8, CL	0	EA+13+2•T	EA+11+T	–	–	EA+15+2•T	EA+12+T	–	–
	reg16, CL	0	–	–	7	7	–	–	7	7
	mem16, CL	2	–	–	EA+17+4•T	EA+13+2•T	–	–	EA+15+2•T	EA+12+T
	reg8, imm3	0	6	6	–	–	6	6	–	–
	mem8, imm3	0	EA+10+2•T	EA+8+T	–	–	EA+12+2•T	EA+9+T	–	–
	reg16, imm4	0	–	–	6	6	–	–	6	6
	mem16, imm4	2	–	–	EA+14+4•T	EA+10+2•T	–	–	EA+12+2•T	EA+9+T
	CY	0	2	2	2	2	2	2	2	2
DIR	0	2	2	2	2	2	2	2	2	
SHL <i>Note</i>	reg, 1	0	8	8	8	8	8	8	8	8
	mem, 1	2	EA+14+2•T	EA+12+2•T	EA+18+4•T	EA+14+2•T	EA+16+2•T	EA+13+T	EA+16+2•T	EA+13+T
	reg, CL	0	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n
	mem, CL	2	EA+17+2•T+2•n	EA+15+T+2•n	EA+21+4•T+2•n	EA+17+2•T+2•n	EA+19+2•T+2•n	EA+16+T+2•n	EA+19+2•T+2•n	EA+16+T+2•n
	reg, imm8	0	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n
	mem, imm8	2	EA+13+2•T+2•n	EA+11+T+2•n	EA+17+4•T+2•n	EA+13+2•T+2•n	EA+15+2•T+2•n	EA+12+T+2•n	EA+15+2•T+2•n	EA+12+T+2•n

Note n : shift amount

Table 2-8. Number of Instruction Execution Clocks (12/14)

Mnemonic	Operand	Number of word transfers	V25, V25+				V35, V35+			
			Byte processing		Word processing		Byte processing		Word processing	
			Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled
SHR ^{Note 1}	reg, 1	0	8	8	8	8	8	8	8	8
	mem, 1	2	EA+14+2•T	EA+12+T	EA+18+4•T	EA+14+2•T	EA+16+2•T	EA+13+T	EA+16+2•T	EA+13+T
	reg, CL	0	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n
	mem, CL	2	EA+17+2•T+2•n	EA+15+T+2•n	EA+21+4•T+2•n	EA+17+2•T+2•n	EA+19+2•T+2•n	EA+16+T+2•n	EA+19+2•T+2•n	EA+16+2•T+2•n
	reg, imm8	0	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n
	mem, imm8	2	EA+13+2•T+2•n	EA+11+T+2•n	EA+17+4•T+2•n	EA+13+2•T+2•n	EA+15+2•T+2•n	EA+12+T+2•n	EA+15+2•T+2•n	EA+12+T+2•n
SHRA ^{Note 1}	reg, 1	0	8	8	8	8	8	8	8	8
	mem, 1	2	EA+14+2•T	EA+12+T	EA+18+4•T	EA+14+2•T	EA+16+2•T	EA+13+T	EA+16+2•T	EA+13+T
	reg, CL	0	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n	11+2•n
	mem, CL	2	EA+17+2•T+2•n	EA+15+T+2•n	EA+21+4•T+2•n	EA+17+2•T+2•n	EA+19+2•T+2•n	EA+16+T+2•n	EA+19+2•T+2•n	EA+16+2•T+2•n
	reg, imm8	0	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n	9+2•n
	mem, imm8	2	EA+13+2•T+2•n	EA+11+T+2•n	EA+17+4•T+2•n	EA+13+2•T+2•n	EA+15+2•T+2•n	EA+12+T+2•n	EA+15+2•T+2•n	EA+12+T+2•n
SS:	None	0	2	2	2	2	2	2	2	2
STM ^{Note 2}	[DS1-spec:]dst-block	1 × rep	16+(8+T)•n	16+(6+T)•n	16+(10+2•T)•n	16+(6+2•T)•n	16+(9+T)•n	16+(7+T)•n	16+(9+T)•n	16+(5+T)•n
		(1) ^{Note 3}	12+T	10	14+2•T	10	13+T	10	13+T	10
STMB ^{Note 2}	None	1 × rep	16+(8+T)•n	16+(6+T)•n	16+(10+2•T)•n	16+(6+2•T)•n	16+(9+T)•n	16+(7+T)•n	16+(9+T)•n	16+(5+T)•n
		(1) ^{Note 3}	12+T	10	14+2•T	10	13+T	10	13+T	10
STMW ^{Note 2}	None	1 × rep	16+(8+T)•n	16+(6+T)•n	16+(10+2•T)•n	16+(6+2•T)•n	16+(9+T)•n	16+(7+T)•n	16+(9+T)•n	16+(5+T)•n
		(1) ^{Note 3}	12+T	10	14+2•T	10	13+T	10	13+T	10
STOP	None	0	–	–	–	–	–	–	–	–
SUB	reg, reg'	0	2	2	2	2	2	2	2	2
	mem, reg	2	EA+8+2•T	EA+6+T	EA+12+4•T	EA+8+2•T	EA+10+2•T	EA+7+T	EA+10+2•T	EA+7+T
	reg, mem	1	EA+6+T	EA+6+T	EA+8+2•T	EA+8+2•T	EA+7+T	EA+7+T	EA+7+T	EA+7+T
	reg, imm	0	5	5	6	6	5	5	6	6
	mem, imm	2	EA+9+2•T	EA+7+2•T	EA+14+4•T	EA+10+4•T	EA+11+2•T	EA+9+2•T	EA+12+2•T	EA+8+2•T
	acc, imm	0	5	5	6	6	5	5	6	6
SUB4S ^{Note 4}	[DS1-spec:]dst-string, [Seg-spec:]src-string	0	22+(27+3•T)•m	22+(25+3•T)•m	–	–	22+(30+3•T)•m	22+(28+3•T)•m	–	–
	None	0	22+(27+3•T)•m	22+(25+3•T)•m	–	–	22+(30+3•T)•m	22+(28+3•T)•m	–	–

- Notes**
1. n : shift amount
 2. n : number of repeats (n ≥ 1)
 3. () : applicable to only one processing
 4. m : number of BCD digits × 1/2

Table 2-8. Number of Instruction Execution Clocks (13/14)

Mnemonic	Operand	Number of word transfers	V25, V25+				V35, V35+			
			Byte processing		Word processing		Byte processing		Word processing	
			Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled
SUBC	reg, reg'	0	2	2	2	2	2	2	2	2
	mem, reg	2	EA+8+2•T	EA+6+T	EA+12+4•T	EA+8+2•T	EA+10+2•T	EA+7+T	EA+10+2•T	EA+7+T
	reg, mem	1	EA+6+T	EA+6+T	EA+8+2•T	EA+8+2•T	EA+7+T	EA+7+T	EA+7+T	EA+7+T
	reg, imm	0	5	5	6	6	5	5	6	6
	mem, imm	2	EA+9+2•T	EA+7+2•T	EA+14+4•T	EA+10+4•T	EA+11+2•T	EA+9+2•T	EA+12+2•T	EA+8+2•T
	acc, imm	0	5	5	6	6	5	5	6	6
TEST	reg, reg'	0	4	4	4	4	4	4	4	4
	mem, reg	1	EA+8+T	EA+8+T	EA+10+2•T	EA+10+2•T	EA+12+T	EA+12+T	EA+11+2•T	EA+11+2•T
	reg, mem	1	EA+8+T	EA+8+T	EA+10+2•T	EA+10+2•T	EA+12+T	EA+12+T	EA+11+2•T	EA+11+2•T
	reg, imm	0	7	7	8	8	7	7	8	8
	mem, imm	1	EA+11+T	EA+11+T	EA+11+2•T	EA+11+2•T	EA+9+T	EA+9+T	EA+10+T	EA+10+T
	acc, imm	0	5	5	6	6	5	5	6	6
TEST1	reg8, CL	0	7	7	–	–	7	7	–	–
	mem8, CL	0	EA+11+T	EA+11+T	–	–	EA+12+T	EA+12+T	–	–
	reg16, CL	0	–	–	7	7	–	–	7	7
	mem16, CL	1	–	–	EA+13+2•T	EA+13+2•T	–	–	EA+12+T	EA+12+T
	reg8, imm3	0	6	6	–	–	6	6	–	–
	mem8, imm3	0	EA+8+T	EA+8+T	–	–	EA+9+T	EA+9+T	–	–
	reg16, imm4	0	–	–	6	6	–	–	6	6
	mem16, imm4	1	–	–	EA+10+2•T	EA+10+2•T	–	–	EA+9+T	EA+9+T
TRANS	src-table	1	10+T	10+T	–	–	11+T	11+T	–	–
	None	1	10+T	10+T	–	–	11+T	11+T	–	–
TRANSB	None	1	10+T	10+T	–	–	11+T	11+T	–	–
TSKSW	reg16	0	–	–	20	20	–	–	20	20
XCH	reg, reg'	0	3	3	3	3	3	3	3	3
	mem, reg	2	EA+10+2•T	EA+8+2•T	EA+14+4•T	EA+10+4•T	EA+12+2•T	EA+9+T	EA+12+2•T	EA+9+T
	reg, mem	2	EA+10+2•T	EA+8+2•T	EA+14+4•T	EA+10+4•T	EA+12+2•T	EA+9+T	EA+12+2•T	EA+9+T
	AW, reg16	0	–	–	4	4	–	–	4	4
	reg16, AW	0	–	–	4	4	–	–	4	4

Table 2-8. Number of Instruction Execution Clocks (14/14)

Mnemonic	Operand	Number of word transfers	V25, V25+				V35, V35+			
			Byte processing		Word processing		Byte processing		Word processing	
			Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled	Internal RAM access enabled	Internal RAM access disabled
XOR	reg, reg'	0	2	2	2	2	2	2	2	2
	mem, reg	2	EA+8+2•T	EA+6+T	EA+12+4•T	EA+8+2•T	EA+10+2•T	EA+7+T	EA+10+2•T	EA+7+T
	reg, mem	1	EA+6+T	EA+6+T	EA+8+2•T	EA+8+2•T	EA+7+T	EA+7+T	EA+7+T	EA+7+T
	reg, imm	0	5	5	6	6	5	5	6	6
	mem, imm	2	EA+9+2•T	EA+7+2•T	EA+14+4•T	EA+10+4•T	EA+11+2•T	EA+9+2•T	EA+12+2•T	EA+8+2•T
	acc, imm	0	5	5	6	6	5	5	6	6

[MEMO]

CHAPTER 3 ADDITIONAL INSTRUCTIONS V20/V30

The instruction set of the V25/V35 family is upwardly compatible with the instruction set of the V20/V30. This chapter describes the additional instructions of the V20/V30 family.

(1) Conditional branch instruction

- **BTCLR** Special function register bit test instruction
 When this instruction is executed and if a bit of a specified special function register is 1, that bit is reset to 0, and execution branches to a short-label described as the operand. If the bit is 0, the instruction next to this instruction is executed. The contents of the PSW are not affected.

(Format)

Mnemonic	Operand		
	Special function register address	Special function register bit	Branch destination
BTCLR	sfr	imm3	short-label

(2) Interrupt instructions

- **RETRBI** Register bank restore instruction
 This instruction is used to restore from an interrupt processing routine using the register bank switching function. It cannot be used to restore from a vector interrupt.

(Format)

Mnemonic	Operand
RETRBI	None

- **FINT** Instruction that informs interrupt controller of end of interrupt processing
 Execute this instruction before the instruction that restores execution from interrupts except NMI, INT, and software interrupt. Do not use this instruction to restore from NMI, INT, or software interrupt.

(Format)

Mnemonic	Operand
FINT	None

(3) CPU control instruction

- STOP Sets STOP status

(Format)

Mnemonic	Operand
STOP	None

(4) Register bank switching instructions

- BRKCS Register bank switching instruction
 When this instruction is executed, the register bank specified by the lower 3 bits of the 16-bit register described as the operand is selected. Execution branches to an address indicated by the PS and vector PC stored in advance to the newly selected register bank. To restore from the new register bank, use the RETRBI instruction.

(Format)

Mnemonic	Operand
BRKCS	reg16

- TSKSW Register bank switching instruction
 Like the BRKCS instruction, this instruction switches the register bank and branches execution to the address obtained from PS stored in advance to the new register bank and PC save area.

(Format)

Mnemonic	Operand
TSKSW	reg16

(5) Data transfer instruction

- MOVSPA SS, SP transfer instruction
 This instruction transfers the values of SS and SP of the old register bank to SS and SP of the newly selected register bank.

(Format)

Mnemonic	Operand
MOVSPA	None

- **MOVSPB** SS, SP transfer instruction

Transfers the values of SS and SP of the current register bank to SS and SP of the newly selected register bank indicated by the lower 3 bits of the 16-bit register described as the operand.

(Format)

Mnemonic	Operand
MOVSPB	reg16

In addition to the above instructions, keep in mind the following points when executing the following instructions of the V25/V35 family:

- **I/O instructions (IN, OUT) and primitive I/O instructions (INM, OUTM)**

These instructions are not executed but an interrupt occurs if the $\overline{\text{IBRK}}$ flag of the PSW is reset to 0. Set the $\overline{\text{IBRK}}$ flag to 1 before executing these instructions.

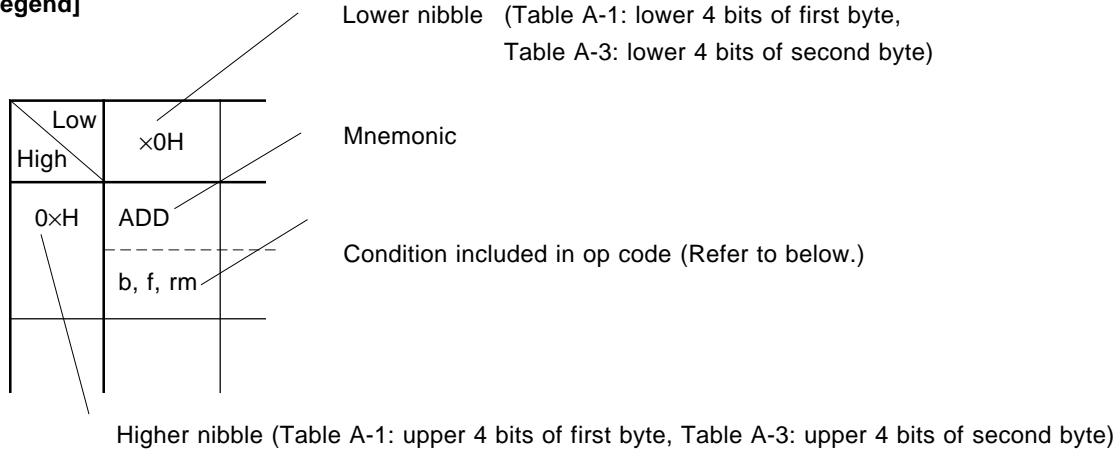
- **FPO1, FPO2**

These instructions are not executed but an interrupt occurs.

[MEMO]

APPENDIX A INSTRUCTION MAP

[Legend]



[Condition included in op code]

- b : performs byte operation
- d : uses direct addressing
- f : accompanies read operation of register in CPU
- i : uses immediate data
- ia : uses immediate data and writes to accumulator
- id : uses indirect addressing
- l : accompanies control between segments
- m : uses memory data
- reg8 : uses 8-bit register
- rm : has effective address field in second byte
- s : uses sign-extended 16-bit immediate data
- sr : uses segment register
- t : write operation to register in CPU
- v : specifies port number indirectly
- w : performs word operation

For conditions other than those shown above, refer to **Table 2-4 Legend of Instruction Formats and Operation Descriptions.**

Table A-1. Instruction Map

Low High	×0H	×1H	×2H	×3H	×4H	×5H	×6H	×7H	×8H	×9H	×AH	×BH	×CH	×DH	×EH	×FH
	0 × H	ADD b,f,rm	ADD w,f,rm	ADD b,t,rm	ADD w,t,rm	ADD b,ia	ADD w,ia	PUSH DS1	POP DS1	OR b,f,rm	OR w,f,rm	OR b,t,rm	OR w,t,rm	OR b,ia	OR w,ia	PUSH PS
1 × H	ADDC b,f,rm	ADDC w,f,rm	ADDC b,t,rm	ADDC w,t,rm	ADDC b,ia	ADDC w,ia	PUSH SS	POP SS	SUBC b,f,rm	SUBC w,f,rm	SUBC b,t,rm	SUBC w,f,rm	SUBC b,ia	SUBC w,ia	PUSH DS0	POP DS0
2 × H	AND b,f,rm	AND w,f,rm	AND b,t,rm	AND w,t,rm	AND b,ia	AND w,ia	DS1 : ADJ4A	SUB b,f,rm	SUB w,f,rm	SUB b,t,rm	SUB w,t,rm	SUB b,ia	SUB w,ia	PS: ADJ4S		
3 × H	XOR b,f,rm	XOR w,f,rm	XOR b,t,rm	XOR w,t,rm	XOR b,ia	XOR w,ia	SS : ADJBA	CMP b,f,rm	CMP w,f,rm	CMP b,t,rm	CMP w,t,rm	CMP b,ia	CMP w,ia	DS0 : ADJBS		
4 × H	INC AW	INC CW	INC DW	INC BW	INC SP	INC BP	INC IX	INC IY	DEC AW	DEC CW	DEC DW	DEC BW	DEC SP	DEC BP	DEC IX	DEC IY
5 × H	PUSH AW	PUSH CW	PUSH DW	PUSH BW	PUSH SP	PUSH BP	PUSH IX	PUSH IY	POP AW	POP CW	POP DW	POP BW	POP SP	POP BP	POP IX	POP IY
6 × H	PUSH R	POP R	CHKIND	Undefined	REPNC	REPC	FPO2 0	FPO2 1	PUSH w,i	MUL w,i	PUSH s,i	MUL s,i	INM b	INM w	OUTM b	OUTM w
7 × H	BV	BNV	BC BL	BNC BNL	BE BZ	BNE BNZ	BNH	BH	BN	BP	BPE	BPO	BLT	BGE	BLE	BGT
8 × H	Imm b,rm	Imm w,rm	Imm b,s,rm	Imm w,s,rm	TEST b,rm	TEST w,rm	XCH b,rm	XCH w,rm	MOV b,f,rm	MOV w,f,rm	MOV b,t,rm	MOV w,t,rm	MOV sr,f,rm	LDEA	MOV sr,t,rm	POP rm
9 × H	XCH AW	XCH CW	XCH DW	XCH BW	XCH SP	XCH BP	XCH IX	XCH IY	CVTBW	CVVTWL	CALL l,d	POLL	PUSH PSW	POP PSW	MOV PSW,AH	MOV AH,PSW
A × H	MOV AL,m	MOV AW,m	MOV m,AL	MOV m,AW	MOV b	MOV w	MOV b	MOV w	MOV b,ia	MOV w,ia	MOV b	MOV w	MOV b	MOV w	MOV b	MOV w
B × H	MOV AL,i	MOV CL,i	MOV DL,i	MOV BL,i	MOV AH,i	MOV CH,i	MOV DH,i	MOV BH,i	MOV AW,i	MOV CW,i	MOV DW,i	MOV BW,i	MOV SP,i	MOV BP,i	MOV IX,i	MOV IY,i
C × H	Shift b,i	Shift w,i	RET (SP)	RET	MOV DS1	MOV DS0	MOV b,i,rm	MOV w,i,rm	PREPARE	DISPOSE	RET l,(SP)	RET l	BRK 3	BRK i	BRKV	RETI
D × H	Shift b	Shift w	Shift b,v	Shift w,v	CVTBD	CVTDB	Undefined	TRANS TRANSB	FPO1 0	FPO1 1	FPO1 2	FPO1 3	FPO1 4	FPO1 5	FPO1 6	FPO1 7
E × H	DBNZE	DBNZE	DBNZ	BCWZ	IN b	IN w	OUT b	OUT w	CALL d	BR d	BR l,d	BR si,d	IN b,v	IN w,v	OUT b,v	OUT w,v
F × H	BUSLOCK	Undefined	REPNE REPNZ	REP REPE REPZ	HALT	NOT1	Group1	Group1	CLR1	SET1	DI	EI	CLR1	SET1	Group2	Group2
						CY	b	w	CY	CY			DIR	DIR	b	w


Caution  : Instructions of Group 1, Group 2, Imm, and Shift are determined by bits 3-5 of second byte of op code (refer to Table A-2).
 Instructions of Group 3 are determined by second byte of op code (refer to Table A-3).

Table A-2. Group1, Group2, Imm, Shift Code Table

Note	000	001	010	011	100	101	110	111
Imm	ADD	OR	ADDC	SUBC	AND	SUB	XOR	CMP
Shift	ROL	ROR	ROLC	RORC	SHL	SHR	Undefined	SHRA
Group1	TEST	Undefined	NOT	NEG	MULU	MUL	DIVU	DIV
	rm		rm	rm	rm	rm	rm	rm
Group2	INC	DEC	CALL	CALL	BR	BR	PUSH	Undefined
	rm	rm	id	l, id	id	l, id	rm	

Note Bits 5 to 3 of second byte

Table A-3. Group3 Code Table

Low High	x0H	x1H	x2H	x3H	x4H	x5H	x6H	x7H	x8H	x9H	xAH	xBH	xCH	xDH	xEH	xFH
0 × H																
1 × H	TEST1 b	TEST1 w	CLR1 b	CLR1 w	SET1 b	SET1 w	NOT1 b	NOT1 w	TEST1 i,b	TEST1 i,w	CLR1 i,b	CLR1 i,w	SET1 i,b	SET1 i,w	NOT1 i,b	NOT1 i,w
2 × H	ADD4S		SUB4S			MOVSPA	CMP4S		ROL4		ROR4				BRKCS	
3 × H		INS reg8		EXT reg8						INS i		EXT i				
9 × H		RETRBI	FINT		TSKSW	MOVSPB							BTCLR		STOP	

Remark Blank indicates undefined code.

[MEMO]

APPENDIX B CORRESPONDENCE OF MNEMONICS TO μ PD8086 AND 8088

The instruction set of the V25/V35 Family is upwardly compatible with the μ PD8086 and 8088 at the object code level.

Table B-1 shows the correspondence of mnemonics between the V25/V35 Family and μ PD8086 and 8088.

Table B-1. Mnemonic Correspondence with μ PD8086 and 8088

μ PD8086, 8088	V25/V35 Family	μ PD8086, 8088	V25/V35 Family	μ PD8086, 8088	V25/V35 Family	μ PD8086, 8088	V25/V35 Family
AAA	ADJBA	JBE	BNH	LOOPNE	DBNZNE	STD	SET1 DIR
AAD	CVTDB	JC	BC/BL	LOOPNZ	DBNZNE	STI	EI
AAM	CVTBD	JCXZ	BCWZ	LOOPZ	DBNZE	STOS	STM/STMB/
AAS	ADJBS	JE	BE/BZ	MOV	MOV		STMW
ADC	ADDC	JG	BGT	MOVS	MOVBK	SUB	SUB
ADD	ADD	JGE	BGE	MOVSB	MOVBKB	TEST	TEST
AND	AND	JL	BLT	MOVSW	MOVBKW	WAIT	POLL
CALL	CALL	JLE	BLE	MUL	MULU	XCHG	XCH
CBW	CVTBW	JMP	BR	NEG	NEG	XLAT	TRANS
CLC	CLR1 CY	JNA	BNH	NOP	NOP	XLATB	TRANSB
CLD	CLR1 DIR	JNAE	BC/BL	NOT	NOT	XOR	XOR
CLI	DI	JNB	BNC/BNL	OR	OR	-	ADD4S
CMC	NOT1 CY	JNBE	BH	OUT	OUT	-	BRKCS
CMP	CMP	JNC	BNC/BNL	POP	POP	-	BTCLR
CMPS	CMPBK/ CMPBKB/ CMPBKW	JNE	BNE/BNZ	POPF	POP PSW	-	CHKIND
		JNG	BLE	PUSH	PUSH	-	CMP4S
		JNGE	BLT	PUSHF	PUSH PSW	-	DISPOSE
CS:	PS:	JNL	BGE	RCL	ROL	-	EXT
CWD	CVTWL	JNLE	BGT	RCR	RORC	-	FINT
DAA	ADJ4A	JNO	BNV	REP	REP	-	FPO2
DAS	ADJ4S	JNP	BPO	REPE	REPE	-	INM
DEC	DEC	JNS	BP	REPNE	REPNE	-	INS
CIV	DIVU	JNZ	BNE/BNZ	REPZ	REPZ	-	MOVSPA
DS:	DS0:	JO	BV	REPZ	REPZ	-	MOVSPB
ES:	DS1:	JP	BPE	RET	RET	-	OUTM
ESC	FPO1	JPE	BPE	ROL	ROL	-	PREPARE
HLT	HALT	JPO	BPO	ROR	ROR	-	REPC
IDIV	DIV	JS	BN	SAHF	MOV PSW,AH	-	REPNC
IMUL	MUL	JZ	BE/BZ	SAL	SHL	-	RETRBI
IN	IN	LAHF	MOV AH,PSW	SAR	SHRA	-	ROL4
INC	INC	LDS	MOV DS0,	SBB	SUBC	-	ROR4
INT	BRK	LEA	LDEA	SCAS	CMPM/	-	STOP
INT 3	BRK 3	LES	MOV DS1,		CMPMB/	-	SUB4S
INTO	BRKV	LOCK	BUSLOCK		CMPMW	-	TEST1
IRET	RETI	LODS	LDM/LDMB/	SHL	SHL	-	TSKSW
JA	BH		LDMW	SHR	SHR		
JAE	BNC/BNL	LOOP	DBNZ	SS:	SS:		
JB	BC/BL	LOOPE	DBNZE	STC	SET1 CY		

Remark - : no corresponding instruction

APPENDIX C NUMBER OF PROGRAM EXECUTION CLOCKS

Strictly speaking, the number of execution clocks of the V25/V35 family varies, as described below, depending on the processing status of the pipeline. It is extremely difficult to predict the operation of the CPU pipeline. Desktop calculation of the number of execution clocks of the program is accordingly difficult. Actually measure the program execution time by using an in-circuit emulator.

This appendix describes the CPU pipeline of the V25/V35 family.

C.1 Synchronous Pipeline

The CPU pipeline of the V25/V35 family is of the synchronous type in which each unit is synchronized with the others for operation. When one pipeline processing for the units that are simultaneously activated has been completed, the next pipeline processing is activated at once.

Each instruction of the V25/V35 family is executed by using a combination of pipeline stages each dissipating 2 to 4 clocks. Therefore, the number of instruction execution clocks of the execution unit differs depending on the processing status of the other units.

For example, even if one pipeline stage of the execution unit ends in 2 clocks, if the processing time of the bus control unit which is activated at the same time is 3 clocks, there is a wait time of 1 clock before the execution unit proceeds to the next pipeline stage.

In this case, the number of execution clocks is 1 clock more than that indicated in **Table 2-8 Number of Instruction Execution Clocks**.

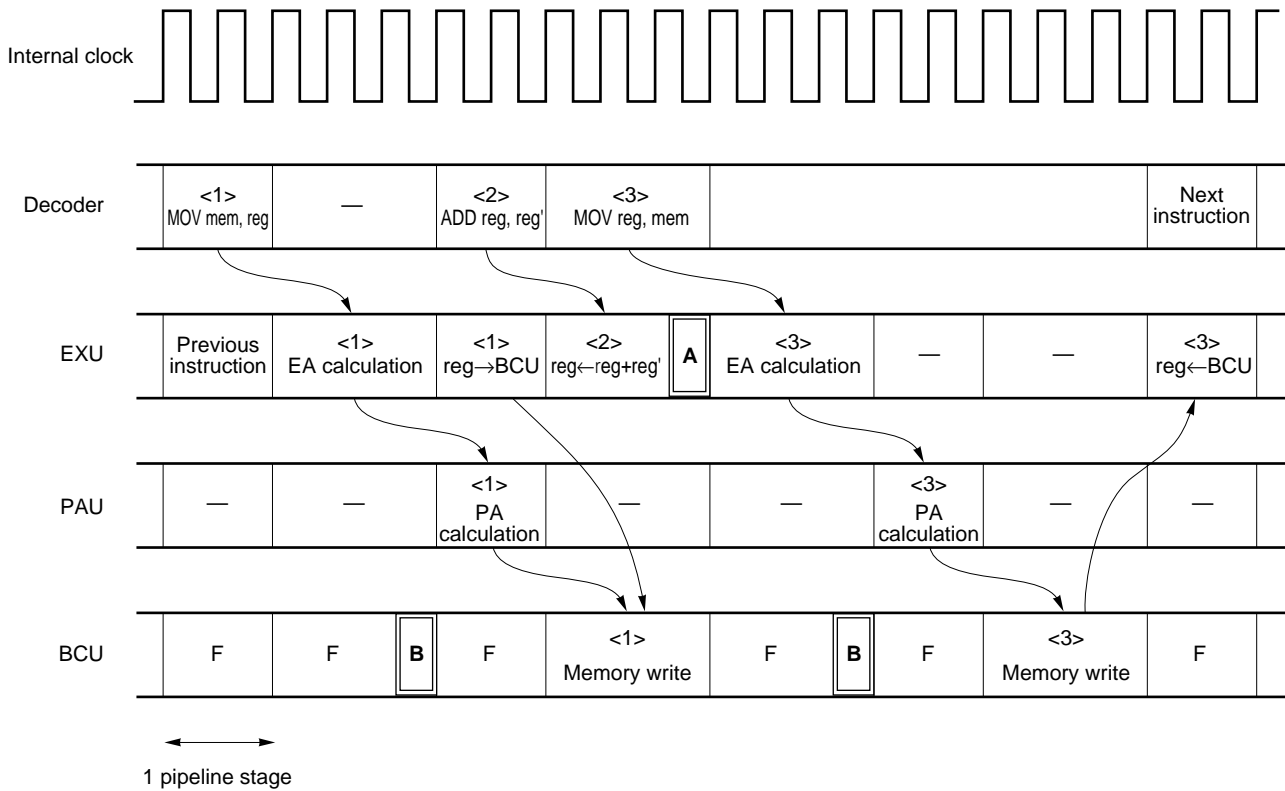
Example: V25

The movement of the pipeline when the following instructions are executed is described.

```
{  
MOV mem, reg ... <1>  
ADD reg, reg' ... <2>  
MOV reg, mem ... <3>  
}
```

In this example, the following conditions are assumed:

- Prefetch cycle: 2 clocks (0 wait)
- Memory read/write: 3 clocks (1 wait)
- Internal RAM access disabled
- When the prefetch queue always has a vacancy of 1 byte or more



- A** : BCU waits for completion of pipeline stage (this affects the number of instruction execution clocks of the EXU).
- B** : EXU waits for completion of pipeline stage (this does not affect the number of instruction execution clocks of the EXU).

Remark EXU : execution unit
 PAU : physical address unit
 BCU : bus control unit
 F : prefetch

C.2 Forced Prefetch Cycle

The V25/V35 family is provided with a 6-byte prefetch queue. Usually, an op code is prefetched if there is a vacancy of 1 byte or more in the queue.

If the instruction code in the queue is 2 bytes or less, and if no op code is prefetched, the next pipeline stage aborts instruction execution, and a prefetch cycle is forcibly activated until an op code of 3 bytes or more is stored in the prefetch queue.

If the addresses of a branch or CALL instruction are not contiguous, the contents of the prefetch queue are cleared, 2 bytes of the op code at the new location are fetched, and instruction execution is started.

For details on the CPU function, refer to the **User's Manual – Hardware** of each product.

APPENDIX D PROGRAM DEVELOPMENT WITH 86 C COMPILER AND ASSEMBLER

This appendix describes how to develop a program for the V25/V35 family by using an 86 C compiler and assembler manufactured by another maker.

D.1 C Language

(1) V25/V35 family original instructions

Because the V25/V35 family is upwardly compatible with the V20/V30, 86 C compilers of other makers can be used.

To use the eight original instructions of the V25/V35 family, call functions created with an assembler. If the C compiler to be used can execute in-line expansion, the original instructions of the V25/V35 family can be described in the C source.

(2) Special function registers

The special function registers (SFRs) are mapped on memory and can be manipulated through reference or assignment by using pointer variables.

```
Example: struct IDB *sfradr;
           sfradr = setidb (0xfe);      /* setidb function sets value to idb register and returns
                                         sfr location address */
           sfradr-> port0 = 0x10;
```

(3) Register bank manipulation

Because the register bank is mapped on memory, it can be manipulated through reference and assignment by using pointer variables.

```
Example: struct REGBNK *bnk;
           long tmp;
           bnk = (struct REGBNK *) BNK_ADR;
           tmp = (long) int_handler;    /* int_handler is a function */
           bnk. vec_pc = (int) (tmp & 0x0000ffff);
           bnk. ps     = (int) ((tmp & 0xffff0000)>>16);
```

D.2 Assembler

(1) V25/V35 original instructions

The V25/V35 original instructions are described by using the macro function.

```
Example:          BTCLR NO_P0 7 dummy
                  .....
dummy:  nop
```

(2) Special function register

The special function registers (SFRs) are manipulated like in C language by declaring structures.

```
Example:  MOV al, sfr. p0
```

D.3 Example of Include File/Macro File

Examples of describing a C language header file and assembler macro file are shown on the following pages. When the list shown on the following pages is created, include the C language file when the C language is used. When the assembler is used, include the assembler file.

Cautions 1. The C language header file is for the V25/V35 on the list shown on the following pages, and the assembler header file is for the V25+/V35+.

To develop a program for the V25+/V35+ in C language, modify the C language header file for the V25/V35 to that for the V25+/V35+.

To develop a program for the V25/V35 in assembler, modify the assembler file for the V25+/V35+ to that for the V25/V35.

Modify the members of the special function register (SFR) structure.

2. The list shown on the following pages has not been strictly tested. Therefore, thoroughly evaluate the list with the compiler to be used. Especially, a compiler that performs optimization may interpret assignment of a value to an SFR (especially if the value is not referenced) as a redundant instruction, and may not generate codes.

/*

THIS STRUCTURE IS DEFINED TO THE SPECIAL FACULTY REGISTER OF V25/V35.

DATE 08 JULY88

Copyright (C) NEC Corporation 1988

*/

```
struct SFR
{
char    port0;
char    portm0;
char    portmc0;
char    dummy1[5];
char    port1;
char    portm1;
char    portmc1;
char    dummy2[5];
char    port2;
char    portm2;
char    portmc2;
char    dummy3[37];
char    portT;
char    dummy4[2];
char    portmT;
char    dummy5[4];
char    intm;
char    dummy6[3];
char    ems0;
char    ems1;
char    ems2;
char    dummy7[5];
char    exic0;
char    exic1;
char    exic2;
char    dummy8[17];
char    rxb0;
char    dummy9;
char    txb0;
char    dummy10[2];
char    srms0;
char    stms0;
char    dummy11;
char    scm0;
char    scc0;
char    brg0;
char    sce0;
char    seic0;
char    sric0;
char    stic0;
```

```
char    dummy12;
char    rxb1;
char    dummy13;
char    txb1;
char    dummy14[2];
char    srms1;
char    stms1;
char    dummy15;
char    scm1;
char    scc1;
char    brg1;
char    sce1;
char    seic1;
char    sric1;
char    stic1;
char    dummy16;
int     tm0;
int     md0;
int     dummy17[2];
int     tm1;
int     md1;
INT     dummy18[2];
char    tmc0;
char    tmc1;
char    dummy19[2];
char    tmms0;
char    tmms1;
char    tmms2;
char    dummy20[5];
char    tmic0;
char    tmic1;
char    tmic2;
char    dummy21;
char    dmac0;
char    dmam0;
char    dmac1;
char    dmam1;
char    dummy22[8];
char    dic0;
char    dic1;
char    dummy23[50];
char    stbc;
char    rfm;
char    dummy24[6];
int     wtc;
char    flag;
char    prc;
char    tbic;
```



```
char    dummy25[15];
char    ispr;
char    dummy26[2];
char    idb;
} *sfr;

/*
THIS DEFINE NAME IS SPECIAL FACULTY REGISTER NAME.

DATE    22 AUG 88
Copyright (C) NEC Corporation 1988

*/
```

```
#define    P0                sfr->port0
#define    PM0               sfr->portm0
#define    PMC0              sfr->portmc0
#define    P1                sfr->port1
#define    PM1               sfr->portm1
#define    PMC1              sfr->portmc1
#define    P2                sfr->port2
#define    PM2               sfr->portm2
#define    PMC2              sfr->portmc2
#define    PT                sfr->portT
#define    PMT               sfr->portmT
#define    INTM              sfr->intm
#define    EMS0              sfr->ems0
#define    EMS1              sfr->ems1
#define    EXIC0             sfr->exic0
#define    EXIC1             sfr->exic1
#define    EXIC2             sfr->exic2
#define    RXB0              sfr->rxb0
#define    TXB0              sfr->txb0
#define    SRMS0             sfr->srms0
#define    STMS0             sfr->stms0
#define    SCM0              sfr->scm0
#define    SCC0              sfr->sc0
#define    BRG0              sfr->brg0
#define    SCE0              sfr->sce0
#define    SEIC0             sfr->seic0
#define    SRIC0             sfr->sr0
#define    STIC0             sfr->stic0
#define    RXB1              sfr->rxb1
#define    TXB1              sfr->txb1
#define    SRMS1             sfr->srms1
#define    STMS1             sfr->stms1
#define    SCM1              sfr->scm1
#define    SCC1              sfr->sc1
#define    BRG1              sfr->brg1
#define    SCE1              sfr->sce1
```

```

#define SEIC1          sfr->seic1
#define SRIC1          sfr->srlic1
#define STIC1          sfr->stic1
#define TM0            sfr->tm0
#define MD0            sfr->md0
#define TM1            sfr->tm1
#define MD1            sfr->md1
#define TMC0           sfr->tmc0
#define TMC1           sfr->tmc1
#define TMMS0          sfr->tmms0
#define TMMS1          sfr->tmms1
#define TMMS2          sfr->tmms2
#define TMIC0          sfr->tmic0
#define TMIC1          sfr->tmic1
#define TMIC2          sfr->tmic2
#define DMAC0          sfr->dmac0
#define DMAM0          sfr->dmam0
#define DMAC1          sfr->dmac1
#define DMAM1          sfr->dmam1
#define DIC0           sfr->dic0
#define DIC1           sfr->dic1
#define STBC           sfr->stbc
#define RFM            sfr->rfm
#define WTC            sfr->wtc
#define FLAG           sfr->flag
#define PRC            sfr->prc
#define TBIC           sfr->tbic
#define ISPR           sfr->ispr
#define IDB            sfr->idb

```

/**/

/*

THIS DEFINE NAME ARE FUNCTIONS TO GIVE ACCESS
TO SPECIAL FACULTY REGISTER

DATE 22 AUG 88

Copyright (C) NEC Corporation 1988

*/

```

#define c_dis0()      (SEIC0 |=0x40)
#define c_dis1()      (SEIC1 |=0x40)
#define c_ena0()      (SEIC0 &=0xbf)
#define c_ena1()      (SEIC1 &=0xbf)
#define c_rdis0()     (SRIC0 |=0x40)
#define c_rdis1()     (SRIC1 |=0x40)
#define c_rena0()     (SRIC0      &=0xbf)
#define c_rena1()     (SRIC1      &=0xbf)
#define c_read0()     (RXB0)
#define c_read1()     (RXB1)
#define c_tdis0()     (STIC0 |=0x40)
#define c_tdis1()     (STIC1 |=0x40)
#define c_tena0()     (STIC0 &=0xbf)
#define c_tena1()     (STIC1 &=0xbf)
#define c_trns0(data) (TXB0 = data)

```

```

#define      c_trns1(data)      (TXB1 = data)
#define      c_tstrt0()        (SCM0 |=0x40)
#define      c_tstrt1()        (SCM1 |=0x40)
#define      c_tstop0()        (SCM0 &=0xbf)
#define      c_tstop1()        (SCM1 &=0xbf)
#define      d_disa0()         (DIC0 |=0x40)
#define      d_disa1()         (DIC1 |=0x40)
#define      d_ena0()          (DIC0 &=0xbf)
#define      d_ena1()          (DIC1 &=0xbf)
#define      p_read0()         (P0)
#define      p_read1()         (P1)
#define      p_read2()         (P2)
#define      P_readt0()        (PT)
#define      p_write0(data)     (P0 =data)
#define      p_write1(data)     (P1 =data)
#define      t_disa0()         (TMIC0 |=0x40)
#define      t_disa1()         (TMIC1 |=0x40)
#define      t_ena0()          (TMIC0 &=0x40)
#define      t_ena1()          (TMIC1 &=0x40)
#define      t_start0()        (TMC0 |=0x80)
#define      t_start1()        (TMC1 |=0x80)
#define      t_stop0()         (TMC0 &=0x7f)
#define      t_stop1()         (TMC1 &=0x7f)
#define      d_hold0()         (DMAM0 =(DMAM0 | 0x08) & 0xfc)
#define      d_hold1()         (DMAM1 =(DMAM1 | 0x08) & 0xfc)
#define      d_start0()        (DMAM0 =DMAM0 | 0xc)
#define      d_start1()        (DMAM1 =DMAM1 | 0xc)
/*
*/
#define      ON                  1
#define      OFF                 0

```

```

struct      REGBNK {
int         reserve;
int         vec_pc;
int         save_psw;
int         save_pc;
int         ds0;
int         ss;
int         ps;
int         ds1;
int         iy;
int         ix;
int         bp;
int         sp;
int         bw;
int         dw;
int         cw;
int         aw;
};

```

```

*****
;
; *           Sample for V25 programing           *
; *                                           for MASM V5.1 *
; *                                           (C) NEC Corp. 1990 *
;
*****

```

```

*****
;
; SFR MACRO(for uPD70320)
;
*****

```

```

SFR_320  struc
        P0      db      ?      ;      BYTE      1      xxF00H
        PM0     db      ?      ;      BYTE      1      xxF01H
        PMC0    db      ?      ;      BYTE      1      xxF02H
        DMY03   db      5      dup(?)

        P1      db      ?      ;      BYTE      1      xxF08H
        PM1     db      ?      ;      BYTE      1      xxF09H
        PMC1    db      ?      ;      BYTE      1      xxF0AH

        DMY0B   db      5      dup(?)

        P2      db      ?      ;      BYTE      1      xxF10H
        PM2     db      ?      ;      BYTE      1      xxF11H
        PMC2    db      ?      ;      BYTE      1      xxF12H

        DMY13   db      37     dup(?)

        PT      db      ?      ;      BYTE      1      xxF38H
        DMY39   db      ?      dup(?)
        PMT     db      ?      ;      BYTE      1      xxF3BH

        DMY3C   db      4      dup(?)

        INTM    db      ?      ;      BYTE      1      xxF40H
        DMY41   db      3      dup(?) ;      BYTE 1      xxF41H

        EMS0    db      ?      ;      BYTE      1      xxF44H
        EMS1    db      ?      ;      BYTE      1      xxF45H
        EMS2    db      ?      ;      BYTE      1      xxF46H

        DMY47   db      5      dup(?) ;      BYTE 1      xxF47H

        EXIC0   db      ?      ;      BYTE      1      xxF4CH
        EXIC1   db      ?      ;      BYTE      1      xxF4DH
        EXIC2   db      ?      ;      BYTE      1      xxF4EH
        DMY4F   db      17     dup(?) ;      BYTE 1      xxF4FH

```

```

RXB0    db    ?    ;    BYTE    1    xxF60H
DMY61   db    ?    ;    BYTE    1    xxF61H
TXB0    db    ?    ;    BYTE    1    xxF62H
DMY63   db    2    dup(?) ;    BYTE    1    xxF63H
SRMS0   db    ?    ;    BYTE    1    xxF65H
STMS0   db    ?    ;    BYTE    1    xxF66H

DMY67   db    ?    ;    BYTE    1    xxF67H

SCM0    db    ?    ;    BYTE    1    xxF68H
SCC0    db    ?    ;    BYTE    1    xxF69H
BRG0    db    ?    ;    BYTE    1    xxF6AH
SCE0    db    ?    ;*   BYTE    1    xxF6BH
SEIC0   db    ?    ;    BYTE    1    xxF6CH
SRIC0   db    ?    ;    BYTE    1    xxF6DH
STIC0   db    ?    ;    BYTE    1    xxF6EH
DMY6F   db    ?    ;    BYTE    1    xxF6FH

RXB1    db    ?    ;    BYTE    1    xxF70H
DMY71   db    ?    ;    BYTE    1    xxF71H
TXB1    db    ?    ;    BYTE    1    xxF72H
DMY73   db    ?    ;    BYTE    1    xxF73H
DMY74   db    ?    ;    BYTE    1    xxF74H
SRMS1   db    ?    ;    BYTE    1    xxF75H
STMS1   db    ?    ;    BYTE    1    xxF76H
DMY77   db    ?    ;    BYTE    1    xxF77H
SCM1    db    ?    ;    BYTE    1    xxF78H
SCC1    db    ?    ;    BYTE    1    xxF79H
BRG1    db    ?    ;    BYTE    1    xxF7AH
SCE1    db    ?    ;*   BYTE    1    xxF7BH
SEIC1   db    ?    ;    BYTE    1    xxF7CH
SRIC1   db    ?    ;    BYTE    1    xxF7DH
STIC1   db    ?    ;    BYTE    1    xxF7EH
DMY7F   db    ?    ;    BYTE    1    xxF7FH

TM0     dw    ?    ;    WORD    1    xxF80H
MD0     dw    ?    ;    WORD    1    xxF82H
DMY84   db    4    dup(?) ;    BYTE    1    xxF84H
TM1     dw    ?    ;    WORD    1    xxF88H
MD1     dw    ?    ;    WORD    1    xxF8AH
DMY8C   db    4    dup(?) ;    BYTE    1    xxF8CH
TMC0    db    ?    ;    BYTE    1    xxF90H
TMC1    db    ?    ;    BYTE    1    xxF91H
DMY92   db    2    dup(?) ;    BYTE    1    xxF92H
TMMS0   db    ?    ;    BYTE    1    xxF94H
TMMS1   db    ?    ;    BYTE    1    xxF95H
TMMS2   db    ?    ;    BYTE    1    xxF96H
DMY97   db    5    dup(?) ;    BYTE    1    xxF97H

```

```

TMIC0 db ? ; BYTE 1 XXF9CH
TMIC1 db ? ; BYTE 1 xxF9DH
TMIC2 db ? ; BYTE 1 xxF9EH
DMY9F db ? ; BYTE 1 xxF9FH

DMAC0 db ? ; BYTE 1 xxFA0H
DMAM0 db ? ; BYTE 1 xxFA1H
DMAC1 db ? ; BYTE 1 xxFA2H
DMAM1 db ? ; BYTE 1 xxFA3H
DMYA4 db 8 dup(?) ; BYTE 1 xxFA4H
DIC0 db ? ; BYTE 1 xxFACH
DIC1 db ? ; BYTE 1 xxFADH
DMYAE db 50 dup(?) ; BYTE 1 xxFAEH

STBC db ? ; BYTE 1 xxFE0H
RFM db ? ; BYTE 1 xxFE1H
DMYE2 db 6 dup(?) ; BYTE 1 xxFE2H
WTC db ? ; BYTE 1 xxFE8H
DMYE9 db ? ; BYTE 1 xxFE9H
FLAG db ? ; BYTE 1 xxFEAH
PRC db ? ; BYTE 1 xxFEBH
TBIC db ? ; BYTE 1 xxFECH
DMYED db 15 dup(?) ; BYTE 1 xxFEDH
ISPR db ? ;* BYTE 1 xxFFCH
DMYFD db 2 dup(?) ; BYTE 1 xxFFDH
IDB db ? ; BYTE 1 xxFFFH

```

SFR_320 ends

```

;
; *****
; * SFR Number for BTCLR instruction
; *****
;

```

```

NO_P0 EQU 00h
NO_PM0 EQU 01h
NO_PMC0 EQU 02h

NO_P1 EQU 08h
NO_PM1 EQU 09h
NO_PMC1 EQU 0ah

NO_P2 EQU 010h
NO_PM2 EQU 011h
NO_PMC2 EQU 012h

NO_PT EQU 038h
NO_PMT EQU 03bh

```

NO_INTM	EQU	040h
NO_EMS0	EQU	044h
NO_EMS1	EQU	045h
NO_EMS2	EQU	046h
NO_EXIC0	EQU	04ch
NO_EXIC1	EQU	04dh
NO_EXIC2	EQU	04eh
NO_RXB0	EQU	060h
NO_TXB0	EQU	062h
NO_SRMS0	EQU	065h
NO_STMS0	EQU	066h
NO_SCM0	EQU	068h
NO_SCC0	EQU	069h
NO_BRG0	EQU	06ah
NO_SCE0	EQU	06bh
NO_SEIC0	EQU	06ch
NO_SRIC0	EQU	06dh
NO_STIC0	EQU	06eh
NO_RXB1	EQU	070h
NO_TXB1	EQU	072h
NO_SRMS1	EQU	075h
NO_STMS1	EQU	076h
NO_SCM1	EQU	078h
NO_SCC1	EQU	079h
NO_BRG1	EQU	07ah
NO_SCE1	EQU	07bh
NO_SEIC1	EQU	07ch
NO_SRIC1	EQU	07dh
NO_STIC1	EQU	07eh
NO_TM0	EQU	080h
NO_MD0	EQU	082h
NO_TM1	EQU	088h
NO_MD1	EQU	08ah
NO_TMC0	EQU	090h
NO_TMC1	EQU	091h
NO_TMMS0	EQU	094h
NO_TMMS1	EQU	095h
NO_TMMS2	EQU	096h
NO_TMIC0	EQU	09ch
NO_TMIC1	EQU	09dh
NO_TMIC2	EQU	09eh
NO_DMAC0	EQU	0a0h
NO_DMAM0	EQU	0a1h
NO_DMAC1	EQU	0a2h

```

NO_DMAM1      EQU    0a3h
NO_DIC0       EQU    0ach
NO_DICI       EQU    0adh

NO_STBC       EQU    0e0h
NO_RFM        EQU    0e1h
NO_WTC        EQU    0e8h
NO_FLAG       EQU    0eah
NO_PRC        EQU    0ebh
NO_TBIC       EQU    0ech
NO_ISPR       EQU    0fch
NO_IDB        EQU    0ffh

```

```

;
;
;

```

```

; *****
; V25/V35 Extend instructions
; *****
;

```

```

BTCLR    macro sfr,imm3,br_label
          db      0fh,9ch
          db      sfr,imm3,offset br_label - $ -1
          endm

RETRBI   macro
          db      0fh,91h
          endm

FINT     macro
          db      0fh,92h
          endm

STOP     macro
          db      0fh,9eh
          endm

BRKCS    macro    reg16
ifidni   <reg16>,<ax>      ;; if reg16 is ax or AX
          db      0fh,2dh,0c0h
endif
ifidni   <reg16>,<bx>      ;; if reg16 is bx or BX
          db      0fh,2dh,0c3h
endif
ifidni   <reg16>,<cx>      ;; if reg16 is cx or CX
          db      0fh,2dh,0c1h
endif
ifidni   <reg16>,<dx>      ;; if reg16 is dx or DX
          db      0fh,2dh,0c2h

```



```
endif
    endm

TSKSW macro    reg16
ifidni    <reg16>,<ax>        ;; if reg16 is ax or AX
db        0fh,94h,0f8h

endif
ifidni    <reg16>,<bx>        ;; if reg16 is bx or BX
db        0fh,94h,0ffh

endif
ifidni    <reg16>,<cx>        ;; if reg16 is cx or CX
db        0fh,94h,0f9h

endif
ifidni    <reg16>,<dx>        ;; if reg16 is dx or DX
db        0fh,94h,0fah

endif
    endm

MOVSPA macro
db        0fh,25h
    endm

MOVSPB macro    reg16
ifidni    <reg16>,<ax>        ;; if reg16 is ax or AX
db        0fh,95h,0f8h

endif
ifidni    <reg16>,<bx>        ;; if reg16 is bx or BX
db        0fh,95h,0ffh

endif
ifidni    <reg16>,<cx>        ;; if reg16 is cx or CX
db        0fh,95h,0f9h

endif
ifidni    <reg16>,<dx>        ;; if reg16 is dx or DX
db        0fh,95h,0fah

endif
    endm
```

[MEMO]

APPENDIX E INSTRUCTION INDEX (mnemonic: by function)

[Data transfer]		[Primitive I/O]	
LDEA	114	INM	110
MOV	117	OUTM	140
MOVSPA	123		
MOVSPB	124	[Addition/subtraction]	
TRANS	195	ADD	28
TRANSB	195	ADDC	32
XCH	197	SUB	183
		SUBC	189
[Repeat prefix]		[BCD operation]	
REP	149	ADD4S	30
REPC	151	CMP4S	75
REPE	149	ROL4	163
REPNC	153	ROR4	169
REPNE	155	SUB4S	187
REPZ	155		
REPZ	149	[Increment/decrement]	
		DEC	88
[Primitive block transfer]		INC	109
CMPBK	77		
CMPBKB	77	[Multiplication/division]	
CMPBKW	77	DIV	91
CMPM	79	DIVU	94
CMPMB	79	MUL	125
CMPMW	79	MULU	128
LDM	115		
LDMB	115	[BCD adjustment]	
LDMW	115	ADJ4A	34
MOVBK	121	ADJ4S	35
MOVBKB	121	ADJBA	36
MOVBKW	121	ADJBS	37
STM	182		
STMB	182	[Data conversion]	
STMW	182	CVTBD	81
		CVTBW	82
[Bit field manipulation]		CVTDB	83
EXT	99	CVTWL	84
INS	112		
		[Compare]	
[I/O]		CMP	73
IN	107		
OUT	138	[Complement operation]	
		NEG	130
		NOT	132

[Logical operation]		BNE	50
AND	38	BNH	51
OR	136	BNL	49
TEST	191	BNV	52
XOR	199	BNZ	50
[Bit manipulation]		BP	53
CLR1	70	BPE	54
NOT1	133	BPO	55
SET1	173	BTCLR	62
TEST1	193	BV	64
[Shift]		BZ	42
SHL	176	DBNZ	85
SHR	178	DBNZE	86
SHRA	180	DBNZNE	87
[Rotate]		[Interrupt]	
ROL	161	BRK	58
ROLC	165	BRKV	61
ROR	167	CHKIND	68
RORC	171	FINT	101
[Subroutine control]		RETI	159
CALL	65	RETRBI	160
RET	157	[CPU control]	
[Stack manipulation]		BUSLOCK	63
DISPOSE	90	DI	89
POP	143	EI	98
PREPARE	145	FPO1	102
PUSH	157	FPO2	104
[Branch]		HALT	106
BR	56	NOP	131
[Conditional branch]		POLL	142
BC	40	STOP	184
BCWZ	41	[Segment override prefix]	
BE	42	DS0:	96
BGE	43	DS1:	96
BGT	44	PS:	96
BH	45	SS:	96
BL	40	[Register bank switching]	
BLE	46	BRKCS	60
BLT	47	TSKSW	96
BN	48		
BNC	49		

APPENDIX F INSTRUCTION INDEX (mnemonic: in alphabetical order)

[A]		CMP4S	75
ADD	28	CMPBK	77
ADD4S	30	CMPBKB	77
ADDC	32	CMPBKW	77
ADJ4A	34	CMPM	79
ADJ4S	35	CMPMB	79
ADJBA	36	CMPMW	79
ADJBS	37	CVTBD	81
AND	38	CVTBW	82
		CVTDB	83
		CVTWL	84
[B]		[D]	
BC	40	DBNZ	85
BCWZ	41	DBNZE	86
BE	42	DBNZNE	87
BGE	43	DEC	88
BGT	44	DI	89
BH	45	DISPOSE	90
BL	40	DIV	91
BLE	46	DIVU	94
BLT	47	DS0:	96
BN	48	DS1:	96
BNC	49	[E]	
BNE	50	EI	98
BNH	51	EXT	99
BNL	49	[F]	
BNV	52	FINT	101
BNZ	50	FPO1	102
BP	53	FPO2	104
BPE	54	[H]	
BPO	55	HALT	106
BR	56	[I]	
BRK	58	IN	107
BRKCS	60	INC	109
BRKV	61	INM	110
BTCLR	62	INS	112
BUSLOCK	63	[L]	
BV	64	LDEA	114
BZ	42		
[C]			
CALL	65		
CHKIND	68		
CLR1	70		
CMP	73		

LDM	115	ROR4	169
LDMB	115	RORC	171
LDMW	115		
[M]		[S]	
MOV	117	SET1	173
MOVBK	111	SHL	176
MOVBKB	111	SHR	178
MOVBKW	111	SHRA	180
MOVSPA	113	SS:	96
MOVSPB	124	STM	182
MUL	125	STMB	182
MULU	128	STMW	182
		STOP	184
[N]		SUB	185
NEG	130	SUB4S	187
NOP	131	SUBC	189
NOT	132		
NOT1	133	[T]	
		TEST	191
[O]		TEST1	193
OR	136	TRANS	195
OUT	138	TRANSB	195
OUTM	140	TSKSW	196
[P]		[X]	
POLL	142	XCH	197
POP	143	XOR	199
PREPARE	145		
PS:	96		
PUSH	147		
[R]			
REP	149		
REPC	151		
REPE	149		
REPNC	153		
REPNE	155		
REPNZ	155		
REPZ	149		
RET	157		
RETI	159		
RETRBI	160		
ROL	161		
ROL4	163		
ROLC	165		
ROR	167		

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

Hong Kong, Philippines, Oceania

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

Asian Nations except Philippines

NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Europe

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

Korea

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

Japan

NEC Corporation
Semiconductor Solution Engineering Division
Technical Information Support Dept.
Fax: 044-548-7900

South America

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

Taiwan

NEC Electronics Taiwan Ltd.
Fax: 02-719-5951

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	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Technical Accuracy	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Organization	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>