FUJITSU SEMICONDUCTOR
CONTROLLER MANUAL

CM26-00301-2E

F²MC-8FX

8-BIT MICROCONTROLLER

PROGRAMMING MANUAL

FUJITSU

F²MC-8FX

8-BIT MICROCONTROLLER

PROGRAMMING MANUAL

FUJITSU LIMITED

PREFACE

■ Purpose and Audience

The F^2MC-8FX is original 8-bit one-chip microcontrollers that support application specific IC (ASIC). It can be widely applied from household to industrial equipment starting with portable equipment.

This manual is intended for engineers who actually develop products using the F^2MC-8FX microcontrollers, especially for programmers who prepare programs using the assembly language for the F^2MC-8FX series assembler. It describes various instructions for the F^2MC-8FX.

Note: F^2MC is the abbreviation of FUJITSU Flexible Microcontroller.

■ Trademark

The company names and brand names herein are the trademarks or registered trademarks of their respective owners.

■ Organization of This Manual

This manual consists of the following six chapters:

CHAPTER 1 OUTLINE AND CONFIGURATION EXAMPLE OF F^2MC-8FX CPU

This chapter outlines the F^2MC-8FX CPU and explains its configuration by example.

CHAPTER 2 MEMORY SPACE

This chapter explains the F^2MC-8FX CPU memory space.

CHAPTER 3 REGISTERS

This chapter explains the F^2MC-8FX dedicated registers and general-purpose registers.

CHAPTER 4 INTERRUPT PROCESSING

This chapter explains the functions and operation of F^2MC-8FX interrupt processing.

CHAPTER 5 CPU SOFTWARE ARCHITECTURE

This chapter explains the instructions for the F^2MC-8FX CPU.

CHAPTER 6 DETAILED RULES FOR EXECUTION INSTRUCTIONS

This chapter explains each execution instruction, used in the assembler, in reference format.

APPENDIX

The appendix contains instruction and bus operation lists and an instruction map.

- The contents of this document are subject to change without notice.
  Customers are advised to consult with sales representatives before ordering.
- The information, such as descriptions of function and application circuit examples, in this document are presented solely for the purpose of reference to show examples of operations and uses of FUJITSU semiconductor device; FUJITSU does not warrant proper operation of the device with respect to use based on such information. When you develop equipment incorporating the device based on such information, you must assume any responsibility arising out of such use of the information. FUJITSU assumes no liability for any damages whatsoever arising out of the use of the information.
- Any information in this document, including descriptions of function and schematic diagrams, shall not be construed as license of the use or exercise of any intellectual property right, such as patent right or copyright, or any other right of FUJITSU or any third party or does FUJITSU warrant non-infringement of any third-party's intellectual property right or other right by using such information. FUJITSU assumes no liability for any infringement of the intellectual property rights or other rights of third parties which would result from the use of information contained herein.
- The products described in this document are designed, developed and manufactured as contemplated for general use, including without limitation, ordinary industrial use, general office use, personal use, and household use, but are not designed, developed and manufactured as contemplated (1) for use accompanying fatal risks or dangers that, unless extremely high safety is secured, could have a serious effect to the public, and could lead directly to death, personal injury, severe physical damage or other loss (i.e., nuclear reaction control in nuclear facility, aircraft flight control, air traffic control, mass transport control, medical life support system, missile launch control in weapon system), or (2) for use requiring extremely high reliability (i.e., submersible repeater and artificial satellite).
  Please note that FUJITSU will not be liable against you and/or any third party for any claims or damages arising in connection with above-mentioned uses of the products.
- Any semiconductor devices have an inherent chance of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions.
- Exportation/release of any products described in this document may require necessary procedures in accordance with the regulations of the Foreign Exchange and Foreign Trade Control Law of Japan and/or US export control laws.
- The company names and brand names herein are the trademarks or registered trademarks of their respective owners.

CONTENTS

Main changes in this edition

Page	Changes (For details, refer to main body.)
11	2.2.2 Program Area Table 2.2-2 CALLV Jump Address Table (" $FFC8_H$ " → " $FFC9_H$ ")
53	Execution example : ADDCW A (NZVC = "1010" → NZVC = "0000")
147	Execution example : MOVW A, PC (A = "F0 63" → A = "F0 62") (PC = "F0 63" → PC = "F0 62")
176	6.65 PUSHW (PUSH Word Data of Inherent Register to Stack Memory) (" Transfer the word value from the memory indicated by SP to dr. Then, subtract 2 fromthe value of SP. " → " Subtract 2 from the value of SP. Then, transfer the word value from the memory indicated by SP to dr. ")
	6.65 PUSHW (PUSH Word Data of Inherent Register to Stack Memory) ■ PUSHW (PUSH Word Data of Inherent Register to Stack Memory) ("((SP)) <-- (dr) (Word transfer) " → " (SP) ← (SP) - 2 (Word subtraction) ") (" (SP) <-- (SP) - 2 (Word subtraction) " → " ((SP)) ← (dr) (Word transfer) ")
226	A.2 Operation List ("((iX)+off) <-- d8 " → " ((IX)+off) ← d8 ")
232	Table A.2-4 Operation List (for Other Instructions) ("(SP) ← (SP)-2, ((SP)) ← (A) (A) ← ((SP)), (SP) ← (SP)+2 (SP) ← (SP)-2, ((SP)) ← (IX) (IX) ← ((SP)), (SP) ← (SP)+2 No operation (C) ← 0 (C) ← 1 (I) ← 0 (I) ← 1 ") is added.

The vertical lines marked in the left side of the page show the changes.

CHAPTER 1

OUTLINE AND CONFIGURATION EXAMPLE OF F^{2}MC-8FX CPU

This chapter outlines the F^{2}MC-8FX CPU and explains its configuration by example.

1.1 Outline of F2MC-8FX CPU

The F2MC-8FX CPU is a high-performance 8-bit CPU designed for the embedded control of various industrial and OA equipment.

■ Outline of F2MC-8FX CPU

The F2MC-8FX CPU is a high-performance 8-bit CPU designed for the control of various industrial and OA equipment. It is especially intended for applications requiring low voltages and low power consumption. This 8-bit CPU can perform 16-bit data operations and transfer and is suitable for applications requiring 16-bit control data. The F2MC-8FX CPU is upper compatibility CPU of the F2MC-8L CPU, and the instruction cycle number is shortened, the division instruction is strengthened, and a direct area is enhanced.

■ F2MC-8FX CPU Features

The F2MC-8FX CPU features are as follows:

- Minimum instruction execution time: 100 ns
- Memory: 64 Kbytes
- Instruction configuration suitable for controller

  Data type: bit, byte, word

  Addressing modes: 9 types

  High code efficiency

  16-bit data operation: Operations between accumulator (A) and temporary accumulator (T)

  Bit instruction: set, reset, check

  Multiplication/division instruction: 8 × 8 = 16 bits, 16/16 = 16 bits
- Interrupt priorities : 4 levels

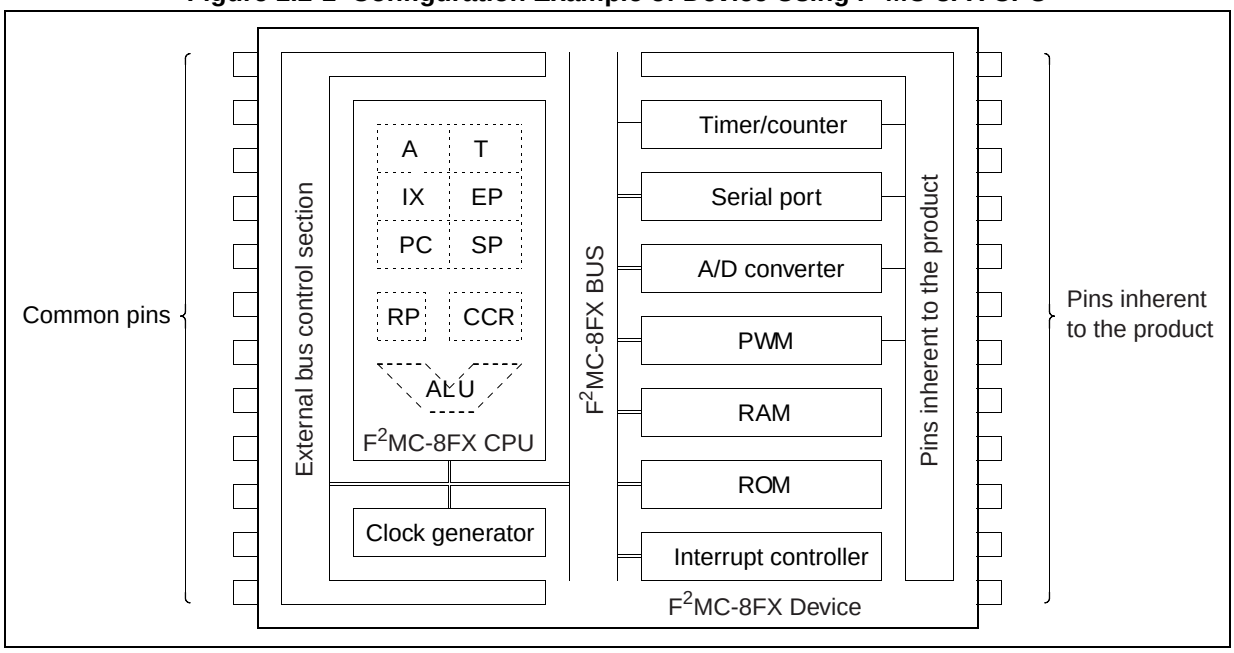
1.2 Configuration Example of Device Using F2MC-8FX CPU

The CPU, ROM, RAM and various resources for each F2MC-8FX device are designed in modules. The change in memory size and replacement of resources facilitate manufacturing of products for various applications.

■ Configuration Example of Device Using F2MC-8FX CPU

Figure 1.2-1 shows a configuration example of a device using the F2MC-8FX CPU.

Figure 1.2-1 Configuration Example of Device Using F2MC-8FX CPU



CHAPTER 2

MEMORY SPACE

This chapter explains the F^{2}MC-8FX CPU memory space.

2.1 CPU Memory Space

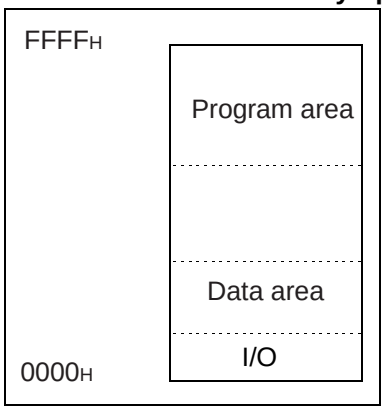
All of the data, program, and I/O areas managed by the F^{2}MC-8FX CPU are assigned to the 64 Kbyte memory space of the F^{2}MC-8FX CPU. The CPU can access each resource by indicating its address on the 16-bit address bus.

■ CPU Memory Space

Figure 2.1-1 shows the address configuration of the F^{2}MC-8FX memory space.

The I/O area is located close to the least significant address, and the data area is arranged right above it. The data area can be divided into the register bank, stack and direct areas for each application. In contrast to the I/O area, the program area is located close to the most significant address. The reset, interrupt reset vector and vector call instruction tables are arranged in the highest part.

Figure 2.1-1 F^{2}MC-8FX Memory Space



2.2 Memory Space and Addressing

In addressing by the F^{2}MC-8FX CPU, the applicable addressing mode related to memory access may change according to the address.
Therefore, the use of the proper addressing mode increases the code efficiency of instructions.

■ Memory Space and Addressing

The F^{2}MC-8FX CPU has the following addressing modes related to memory access. ([ ] indicates one byte):

- Direct addressing: Specify the lower 8 bits of the address using the operand. The accesses of operand address 00_H to $7F_H$ are always 0000_H to $007F_H$. The accesses of operand address 80_H to FF_H are mapped to 0080_H to $047F_H$ by setting of direct bank pointer (DP).

  [Structure] [← OP code →] [← lower 8 bits →] ([← if operand available →]

- Extended addressing: Specify all 16 bits using the operand.

  [Structure] [← OP code →] [← upper 8 bits →] [← lower 8 bits →]

- Bit direct addressing: Specify the lower 8 bits of the address using the operand. The accesses of operand address 00_H to $7F_H$ are always 0000_H to $007F_H$. The accesses of operand address 80_H to FF_H are mapped to 0080_H to $047F_H$ by setting of direct bank pointer (DP). The bit positions are included in the OP code.

  [Structure] [← OP code: bit →] [← lower 8 bits →]

- Indexed addressing: Add the 8 bits of the operand to the index register (IX) together with the sign and use the result as the address.

  [Structure] [← OP code →] [← 8 offset bits →] ([← if operand available →])

- Pointer addressing: Use the contents of the extra pointer (EP) directly as the address.

  [Structure] [← OP code →]

- General-purpose register addressing: Specify the general-purpose registers. The register numbers are included in the OP code.

  [Structure] [← OP code: register →]

- Immediate addressing: Use one byte following the OP code as data.

  [Structure] [← OP code →] [← Immediate data →]

- Vector addressing: Read the data from a table corresponding to the table number. The table numbers are included in the OP code.

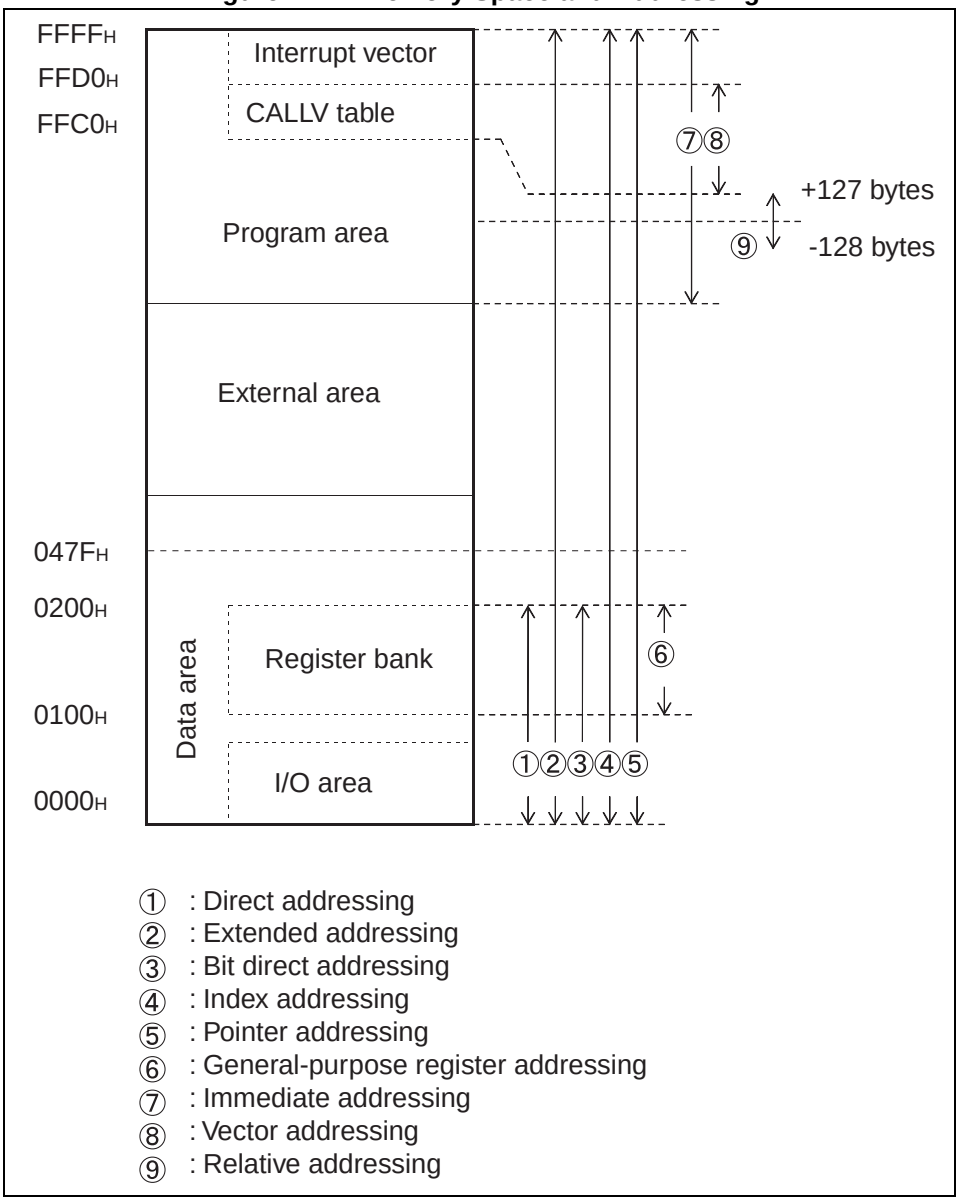
  [Structure] [← OP code: table →]

- Relative addressing: Calculate the address relatively to the contents of the current PC. This addressing mode is used during the execution of the relative jump and bit check instructions.

  [Structure] [← OP code: table →] [← 8 bit relative value →]

Figure 2.2-1 shows the memory space accessible by each addressing mode.

Figure 2.2-1 Memory Space and Addressing



2.2.1 Data Area

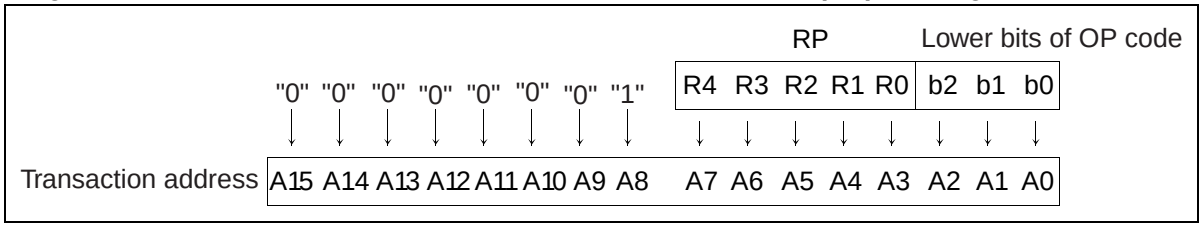
The F^{2}MC-8FX CPU data area can be divided into the following three for each purpose:

- **General-purpose register bank area**
- **Stack area**
- **Direct area**

■ General-Purpose Register Bank Area

The general-purpose register bank area in the F^{2}MC-8FX CPU is assigned to 0100_H to $01FF_H$. The general-purpose register numbers are converted to the actual addresses according to the conversion rule shown in Figure 2.2-2 by using the register bank pointer (RP) and the lower 3 bits of the OP code.

Figure 2.2-2 Conversion Rule for Actual Addresses of General-purpose Register Bank Area



■ Stack Area

The stack area in the F^{2}MC-8FX CPU is used as the saving area for return addresses and dedicated registers when the subroutine call instruction is executed and when an interrupt occurs. Before pushing data into the stack area, decrease the contents of the 16-bit stack pointer (SP) by 2 and then write the data to be saved to the address indicated by the SP. To pop data off the stack area, return data from the address indicated by the SP and then increase the contents of the SP by 2. This shows that the most recently pushed data in the stack is stored at the address indicated by the SP. Figure 2.2-3 and Figure 2.2-4 give examples of saving data in the stack area and returning data from it.

Figure 2.2-3 Example of Saving Data in Stack Area

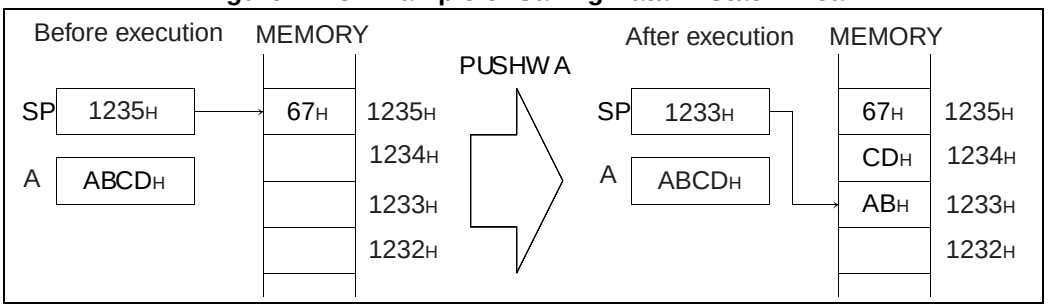
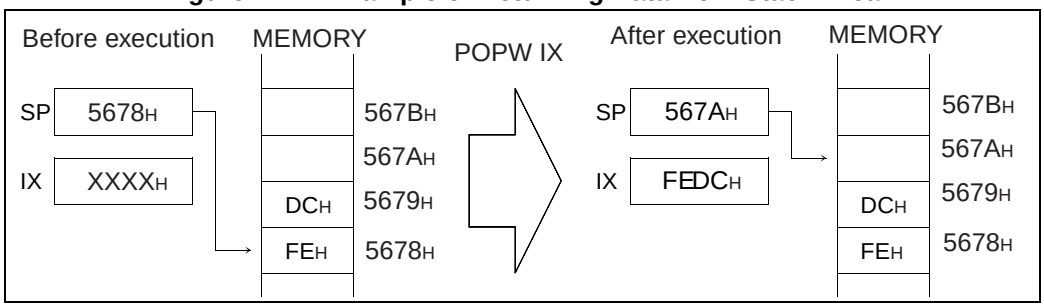


Figure 2.2-4 Example of Returning Data from Stack Area



■ Direct Area

The direct area in the F^{2}MC-8FX CPU is located at the lower side of the memory space or the 1152 bytes from 0000_H to $047F_H$ and is mainly accessed by direct addressing and bit direct addressing. The area that can be used at a time by direct addressing and bit direct addressing is 256 bytes. 128 bytes of 0000_H to $007F_H$ can be used at any time as a direct area. 0080_H to $047F_H$ is a direct bank of 128 bytes × 8 and can use one direct bank as a direct area by setting the direct bank pointer (DP). Conversion from the operand address of direct addressing and bit direct addressing to the real address is done by the conversion rule shown in Table 2.2-1 by using DP.

Access to it is obtained by the 2-byte instruction.

The I/O control registers and part of RAM that are frequently accessed are arranged in this direct area.

Table 2.2-1 Conversion Rule for Actual Address of Direct Addressing and Bit Direct Addressing

Operand address	Direct bank pointer (DP)	Actual address
00_H to $7F_H$		0000_H to $007F_H$
80_H to FF_H	000	0080_H to $00FF_H$
	001	0100_H to $017F_H$
	010	0180_H to $01FF_H$
	011	0200_H to $027F_H$
	100	0280_H to $02FF_H$
	101	0300_H to $037F_H$
	110	0380_H to $03FF_H$
	111	0400_H to $047F_H$

2.2.2 Program Area

The program area in the F^{2}MC-8FX CPU includes the following two:

- **Vector call instruction table**
- **Reset and interrupt vector table**

■ Vector Call Instruction Table

$FFC0_H$ to $FFCF_H$ of the memory space is used as the vector call instruction table. The vector call instruction for the F^{2}MC-8FX CPU provides access to this area according to the vector numbers included in the OP code and makes a subroutine call using the data written there as the jump address. Table 2.2-2 indicates the correspondence of the vector numbers with the jump address table.

Table 2.2-2 CALLV Jump Address Table

CALLV #k	Jump address table	
	Upper address	Lower address
#0	$FFC0_H$	$FFC1_H$
#1	$FFC2_H$	$FFC3_H$
#2	$FFC4_H$	$FFC5_H$
#3	$FFC6_H$	$FFC7_H$
#4	$FFC8_H$	$FFC9_H$
#5	$FFCA_H$	$FFCB_H$
#6	$FFCC_H$	$FFCD_H$
#7	$FFCE_H$	$FFCF_H$

■ Reset and Interrupt Vector Table

$FFCC_H$ to $FFFF_H$ of the memory space is used as the table indicating the starting address of an interrupt or reset Table 2.2-3 indicates the correspondence between the interrupt numbers or resets and the reference table.

Table 2.2-3 Reset and Interrupt Vector Table

Interrupt No.	Table address	
	Upper data	Lower data
Reset	$FFFE_H$	$FFFF_H$
	$FFFC_H$	$FFFD_H$
#0	$FFFA_H$	$FFFB_H$
#1	$FFF8_H$	$FFF9_H$
#2	$FFF6_H$	$FFF7_H$
#3	$FFF4_H$	$FFF5_H$
#4	$FFF2_H$	$FFF3_H$
#5	$FFF0_H$	$FFF1_H$
#6	$FFFE_H$	$FFFF_H$
#7	$FFEC_H$	$FFFD_H$
#8	$FFEA_H$	$FFFB_H$
#9	$FFE8_H$	$FFF9_H$
#10	$FFE6_H$	$FFE7_H$

Interrupt No.	Table address	
	Upper data	Lower data
#11	$FFE4_H$	$FFE5_H$
#12	$FFE2_H$	$FFE3_H$
#13	$FFE0_H$	$FFE1_H$
#14	$FFDE_H$	$FFDF_H$
#15	$FFDC_H$	$FFDD_H$
#16	$FFDA_H$	$FFDB_H$
#17	$FFD8_H$	$FFD9_H$
#18	$FFD6_H$	$FFD7_H$
#19	$FFD4_H$	$FFD5_H$
#20	$FFD2_H$	$FFD3_H$
#21	$FFD0_H$	$FFD1_H$
#22	$FFCE_H$	$FFCF_H$
#23	$FFCC_H$	$FFCD_H$

$FFFC_H$: Reserved

$FFFD_H$: Mode

Note: The actual number varies according to the product.

Use the interrupt number #22 and #23 exclusively for vector call instruction, CALLV #6 and CALLV #7

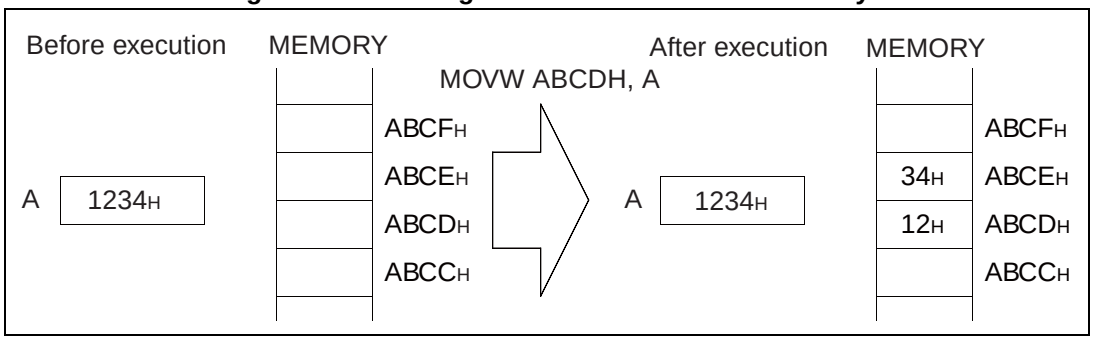
2.2.3 Arrangement of 16-bit Data in Memory Space

The F^{2}MC-8FX CPU can perform 16-bit data transfer and arithmetic operation though it is an 8-bit CPU. Arrangement of 16-bit data in the memory space is shown below.

■ Arrangement of 16-bit Data in Memory Space

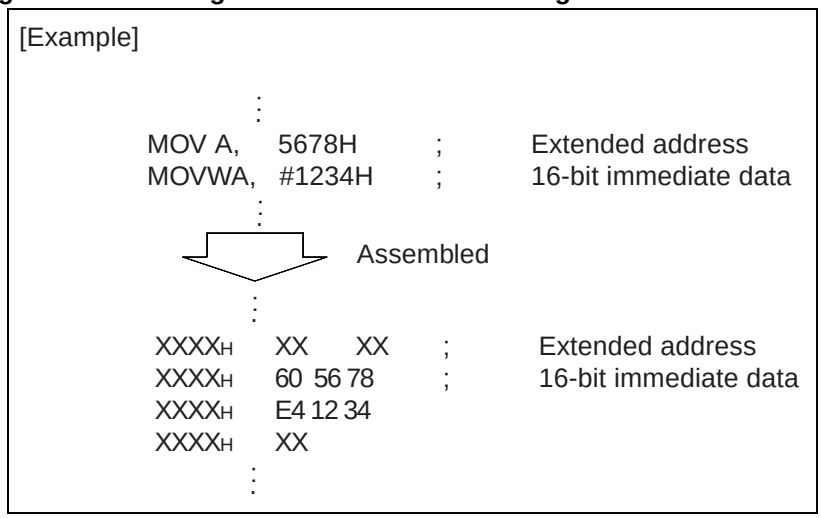
As shown in Figure 2.2-5, the F^{2}MC-8FX CPU treats 16-bit data in the memory as upper data if it is written at the first location having a lower address and as lower data if it is written at the next location after that.

Figure 2.2-5 Arrangement of 16-bit Data in Memory



As when 16 bits are specified by the operand during the execution of an instruction, bytes are assumed to be upper and lower in the order of their proximity to the OP code. This applies when the operand indicates the memory address and 16-bit immediate data as shown in Figure 2.2-6.

Figure 2.2-6 Arrangement of 16-bit Data during Instruction Execution



The same may also apply to data saved in the stack by interrupts.

CHAPTER 3
REGISTERS

This chapter explains the F^{2}MC-8FX dedicated registers and general-purpose registers.

3.1 F²MC-8FX Registers

In the F²MC-8FX series, there are two types of registers: dedicated registers in the CPU, and general-purpose registers in memory.

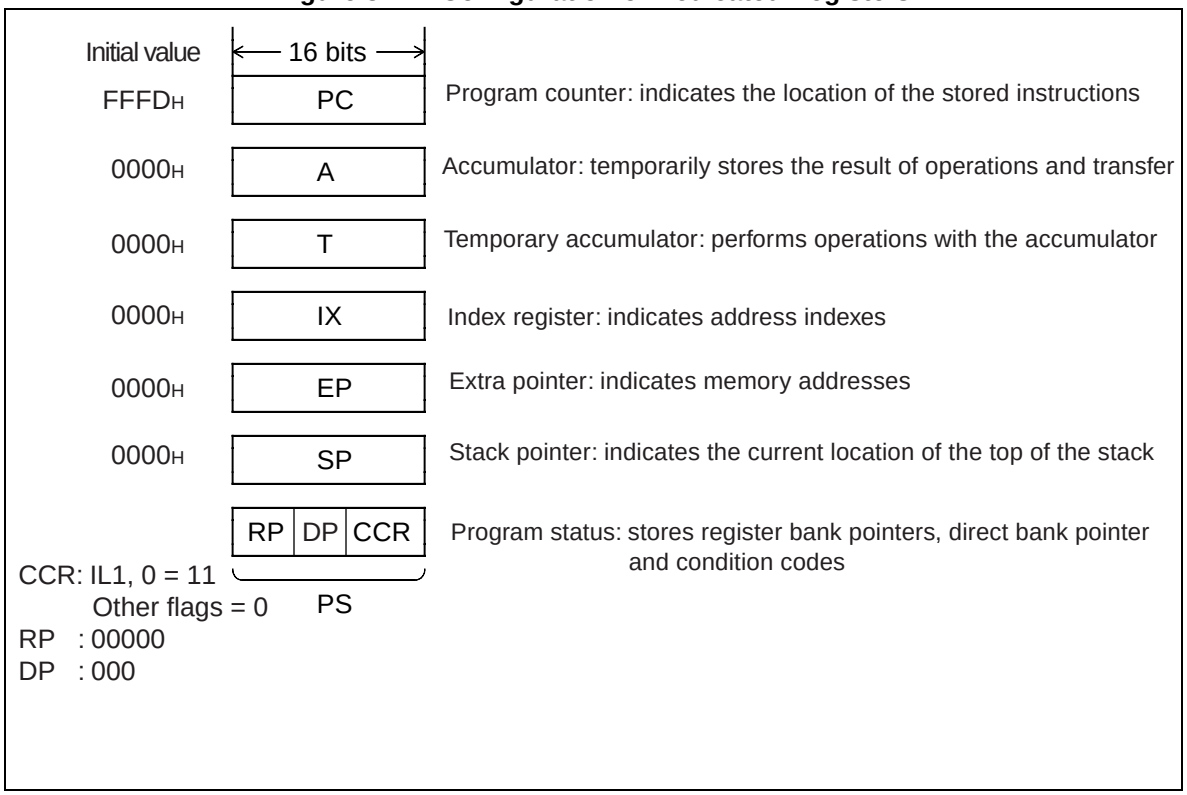
■ F²MC-8FX Dedicated Registers

The dedicated register exists in the CPU as a dedicated hardware resource whose application is restricted to the CPU architecture.

The dedicated register is composed of seven types of 16-bit registers. Some of these registers can be operated with only the lower 8 bits.

Figure 3.1-1 shows the configuration of seven dedicated registers.

Figure 3.1-1 Configuration of Dedicated Registers



■ F²MC-8FX General-Purpose Registers

The general-purpose register is as follows:

- Register bank: 8-bit length: stores data

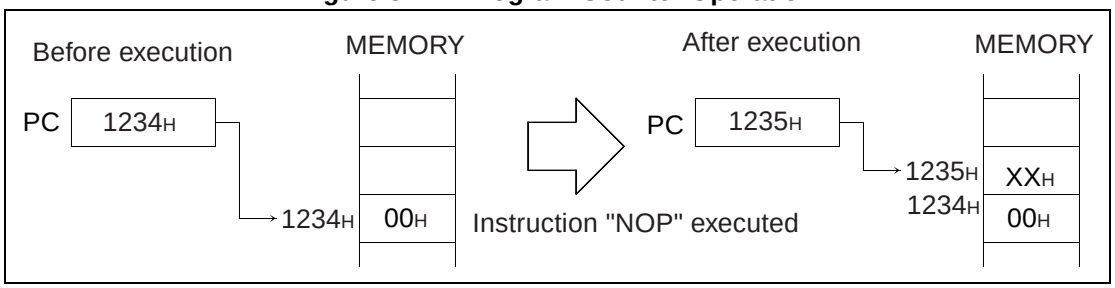
3.2 Program Counter (PC) and Stack Pointer (SP)

The program counter (PC) and stack pointer (SP) are application-specific registers existing in the CPU.
The program counter (PC) indicates the address of the location at which the instruction currently being executed is stored.
The stack pointer (SP) holds the addresses of the data location to be referenced by the interrupt and stack push/pop instructions. The value of the current stack pointer (SP) indicates the address at which the last data pushed onto the stack is stored.

■ Program Counter (PC)

Figure 3.2-1 shows the operation of the program counter (PC).

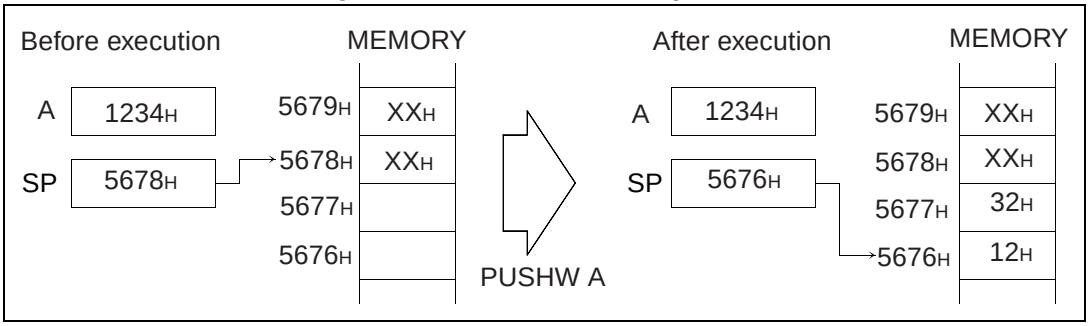
Figure 3.2-1 Program Counter Operation



■ Stack Pointer (SP)

Figure 3.2-2 shows the operation of the stack pointer (SP).

Figure 3.2-2 Stack Pointer Operation



3.3 Accumulator (A) and Temporary Accumulator (T)

The accumulator (A) and temporary accumulator (T) are application-specific registers existing in the CPU.
The accumulator (A) is used as the area where the results of operations are temporarily stored.
The temporary accumulator (T) is used as the area where the old data is temporarily saved for data transfer to the accumulator (A) or the operand for operations.

■ Accumulator (A)

For 16-bit operation all 16 bits are used as shown in Figure 3.3-1. For 8-bit operation only the lower 8 bits are used as shown in Figure 3.3-2.

Figure 3.3-1 Accumulator (A) Operation (16-bit Operation)

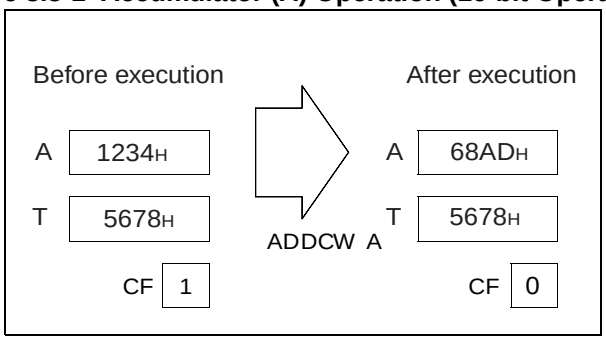
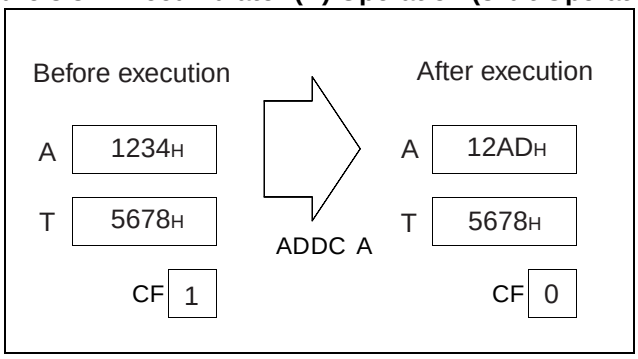


Figure 3.3-2 Accumulator (A) Operation (8-bit Operation)



■ Temporary Accumulator (T)

When 16-bit data is transferred to the accumulator (A), all the old 16-bit data in the accumulator is transferred to the temporary accumulator (T) as shown in Figure 3.3-3. When 8-bit data is transferred to the accumulator, old 8-bit data stored in the lower 8 bits of the accumulator is transferred to the lower 8 bits of the temporary accumulator as shown in Figure 3.3-4. Although all 16-bits are used as the operand for 16-bit operations as shown in Figure 3.3-5, only the lower 8 bits are used for 8-bit operations as shown in Figure 3.3-6.

Figure 3.3-3 Data Transfer between Accumulator (A) and Temporary Accumulator (T) (16-bit Transfer)

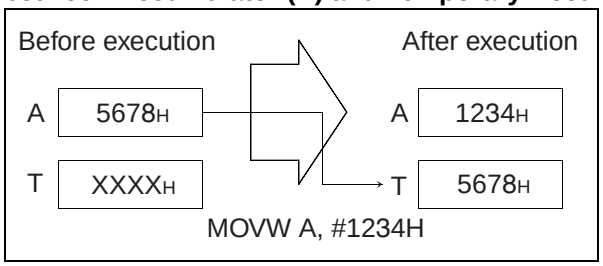


Figure 3.3-4 Data Transfer between Accumulator (A) and Temporary Accumulator (T) (8-bit Transfer)

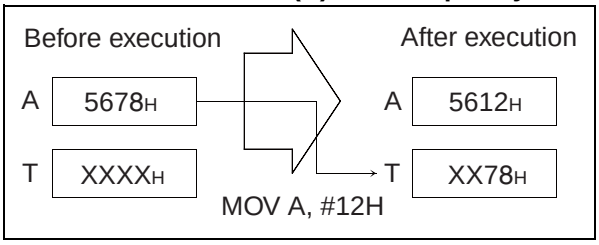


Figure 3.3-5 Operations between Accumulator (A) and Temporary Accumulator (T) (16-bit Operations)

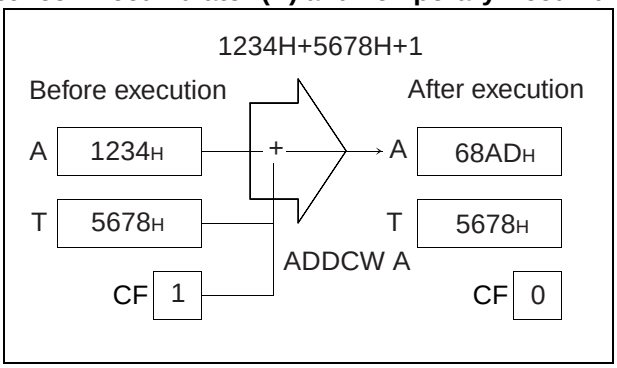
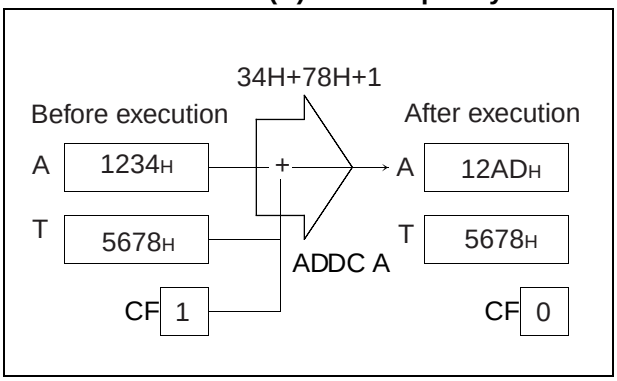


Figure 3.3-6 Operations between Accumulator (A) and Temporary Accumulator (T) (8-bit Operations)



3.3.1 How To Use The Temporary Accumulator (T)

The F^{2}MC-8FX CPU has a special-purpose register called a temporary accumulator. This section described the operation of this register.

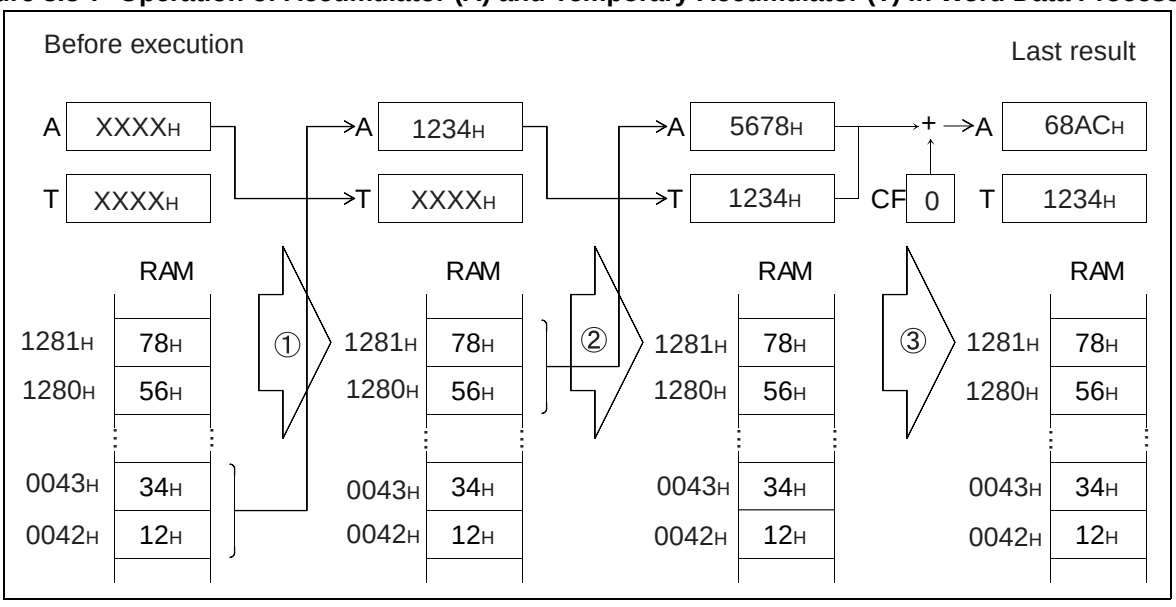
■ How to Use the Temporary Accumulator (T)

The F^{2}MC-8FX CPU has various binary operation instructions, some data transfer instructions and the temporary accumulator (T) for 16-bit data operation. Although there is no instruction for direct data transfer to the temporary accumulator, the value of the original accumulator is transferred to the temporary accumulator before executing the instruction for data transfer to the accumulator. Therefore, to perform operations between the accumulator and temporary accumulator, execute operations after carrying out the instruction for data transfer to the accumulator twice. Since data is not automatically transferred by all instructions to the temporary accumulator, see the columns of TL and TH in the instruction list for details of actual data transfer instructions. An example of addition with carry of 16-bit data stored at addresses 1280_H and 0042_H is shown below.

```
MOVW A, 0042H   - ①
MOVW A, 1280H   - ②
ADDCW A         - ③
```

Figure 3.3-7 shows the operation for the accumulator and temporary accumulator when the above example is executed.

Figure 3.3-7 Operation of Accumulator (A) and Temporary Accumulator (T) in Word Data Processing



3.3.2 Byte Data Transfer and Operation of Accumulator (A) and Temporary Accumulator (T)

When data transfer to the accumulator (A) is performed byte-by-byte, the transfer data is stored in the AL. Automatic data transfer to the temporary accumulator (T) is also performed byte-by-byte and only the contents of the original AL are stored in the TL. Neither the upper 8 bits of the accumulator nor the temporary accumulator are affected by the transfer. Only the lower 8 bits are used for byte operation between the accumulator and temporary accumulator. None of the upper 8 bits of the accumulator or temporary accumulator are affected by the operation.

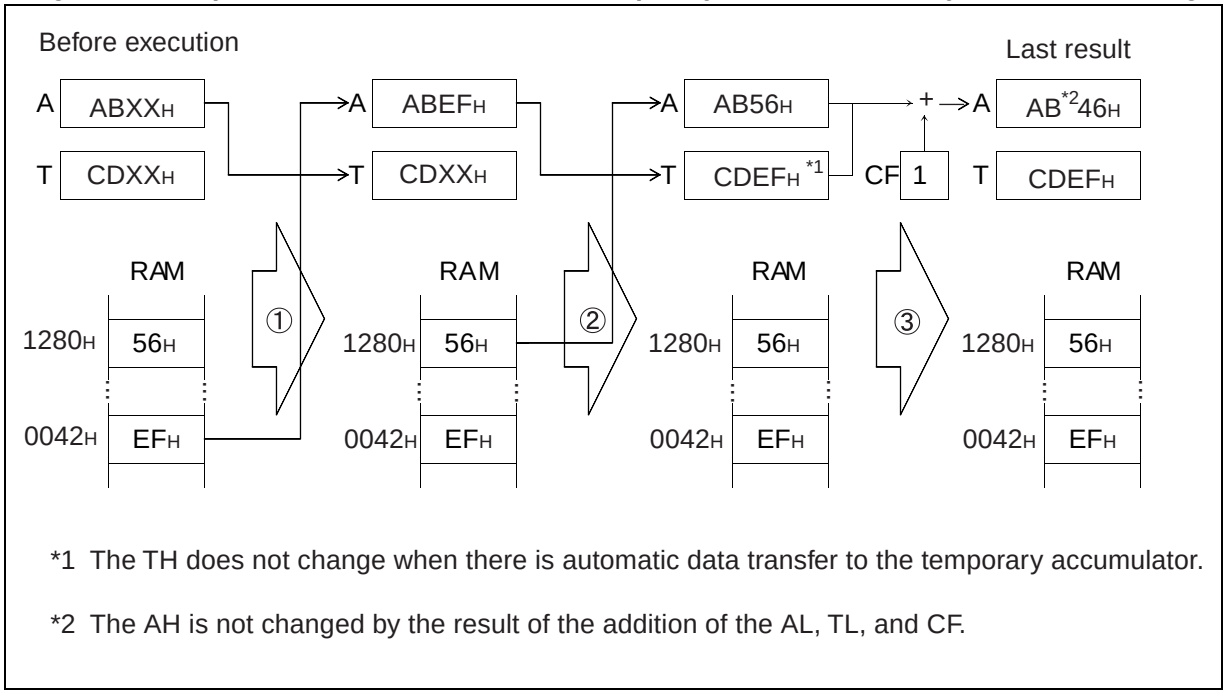
■ Example of Operation of Accumulator (A) and Temporary Accumulator (T) in Byte Data Processing

An example of addition with carry of 8-bit data stored at addresses 1280_H and 0042_H is shown below.

```
MOV A,  0042H   -   ①
MOV A,  1280H   -   ②
ADDC A          -   ③
```

Figure 3.3-8 shows the operation of the accumulator and temporary accumulator when the above example is executed.

Figure 3.3-8 Operation of Accumulator and Temporary Accumulator in Byte Data Processing



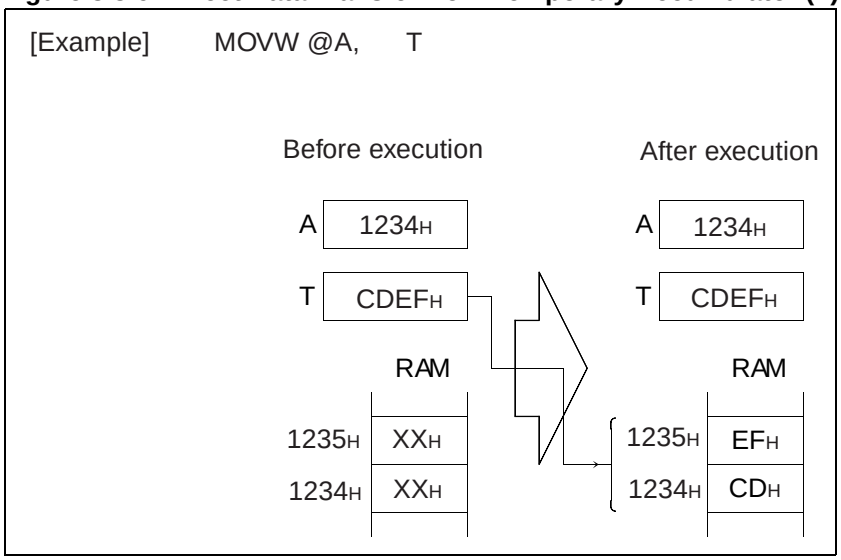
*1 The TH does not change when there is automatic data transfer to the temporary accumulator.

*2 The AH is not changed by the result of the addition of the AL, TL, and CF.

■ Direct Data Transfer from Temporary Accumulator (T)

The temporary accumulator (T) is basically temporary storage for the accumulator (A). Therefore, data from the temporary accumulator cannot be transferred directly to memory. However, as an exception, using the accumulator as a pointer enabling saving of the contents of the temporary accumulator in memory. An example of this case is shown below.

Figure 3.3-9 Direct Data Transfer from Temporary Accumulator (T)



3.4 Program Status (PS)

The program status (PS) is a 16-bit application-specific register existing in the CPU. In upper byte of program status (PS), the upper 5-bit is the register bank pointer (RP) and lower 3-bit is the direct bank pointer (DP). The lower byte of program status (PS) is the condition code register (CCR). The upper byte of program status (PS), i.e. RP and DP, is mapped to address 0078_H. So it is possible to make read and write accesses to them by an access to address 0078_H.

■ Structure of Program Status (PS)

Figure 3.4-1 shows the structure of the program status.

The register bank pointer (RP) indicates the address of the register bank currently in use. The relationship between the contents of the register bank pointer and actual addresses is as shown in Figure 3.4-2.

DP shows the memory area (direct bank) used for direct addressing and bit direct addressing. Conversion from the operand address of direct addressing and bit direct addressing to the real address follows the conversion rule shown in Table 3.4-1 by using DP.

The condition code register (CCR) has bits for indicating the result of operations and the content of transfer data and bits for controlling the operation of the CPU in the event of an interrupt.

Figure 3.4-1 Structure of Program Status (PS)

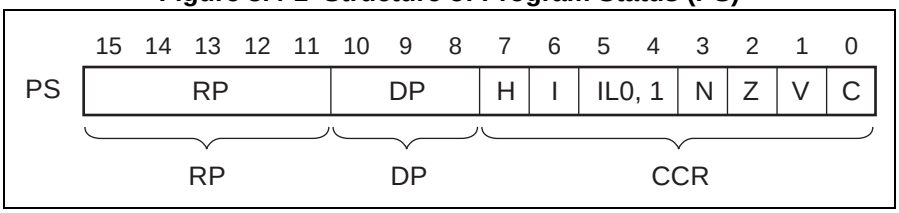


Figure 3.4-2 Conversion Rule for Actual Address of General-purpose Register Area

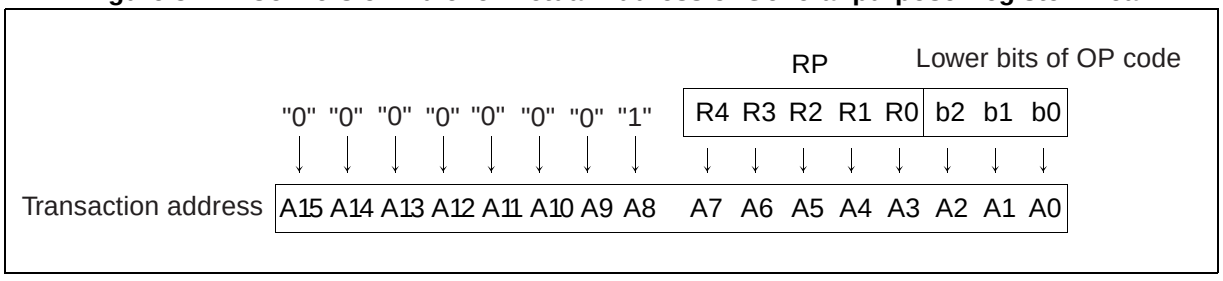


Table 3.4-1 Conversion Rule for Actual Address of Direct Addressing and Bit Direct Addressing

Operand address	Direct bank pointer (DP)	Actual address
00_H to $7F_H$		0000_H to $007F_H$
80_H to FF_H	000	0080_H to $00FF_H$
	001	0100_H to $017F_H$
	010	0180_H to $01FF_H$
	011	0200_H to $027F_H$
	100	0280_H to $02FF_H$
	101	0300_H to $037F_H$
	110	0380_H to $03FF_H$
	111	0400_H to $047F_H$

■ Program Status (PS) Flags

The program status flags are explained below.

- H flag

  This flag is 1 if a carry from bit 3 to bit 4 or a borrow from bit 4 to bit 3 is generated as the result of an operation, and it is 0 in other cases. Because it is used for decimal compensation instructions, it cannot be guaranteed if it is used for applications other than addition or subtraction.

- I flag

  An interrupt is enabled when this flag is 1 and is disabled when it is 0. It is set to 0 at reset which results in the interrupt disabled state.

- IL1, IL0

  These bits indicate the level of the currently-enabled interrupt. The interrupt is processed only when an interrupt request with a value less than that indicated by these bits is issued.

IL1	IL0	Interrupt level	High and low
0	0	0	Highest
0	1	1	↕
1	0	2	
1	1	3	Lowest

- N flag

  This flag is 1 when the most significant bit is 1 and is 0 when it is 0 as the result of an operation.

- Z flag

  This flag is 1 when the most significant bit is 0 and is 0 in other cases as the result of an operation.

- V flag

This flag is 1 when a two's complement overflow occurs and is 0 when one does not as the result of an operation.

- C flag

  This flag is 1 when a carry or a borrow, from bit 7 in byte mode and from bit 15 in word mode, is generated as the result of an operation but 0 in other cases. The shifted-out value is provided by the shift instruction.

■ Access to Register Bank Pointer and Direct Bank Pointer

The upper byte of program status (PS), i.e. register bank pointer (RP) and direct bank pointer (DP), is mapped to address 0078_H. So it is possible to make read and write accesses to them by an access to address 0078_H, besides using instructions that have access to PS (MOVW A, PS or MOVW PS, A).

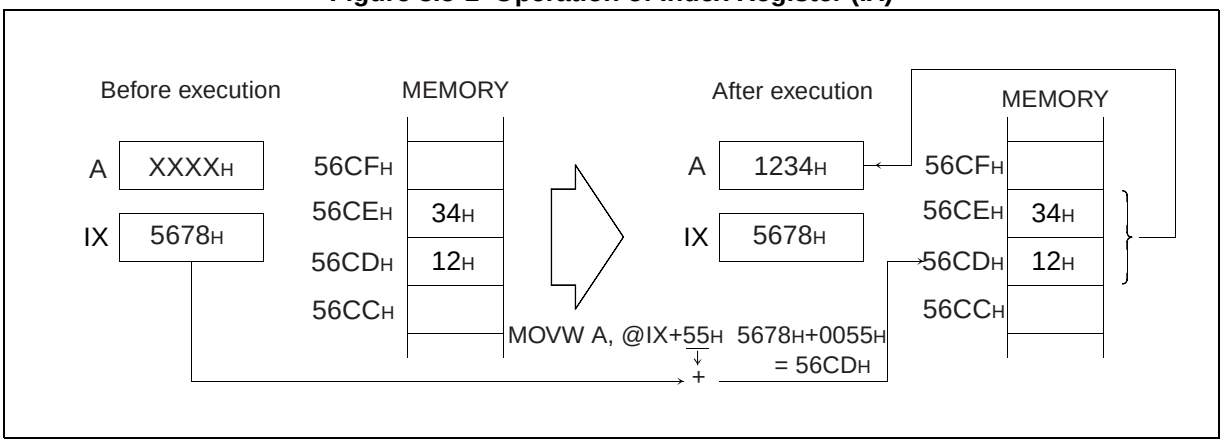
3.5 Index Register (IX) and Extra Pointer (EP)

The index register (IX) and extra pointer (EP) are 16-bit application-specific registers existing in the CPU.
The index register (IX) adds an 8-bit offset value with its sign to generate the address stored by the operand.
The extra pointer (EP) indicates the address stored by the operand.

■ Index Register (IX)

Figure 3.5-1 indicates the operation of the index register.

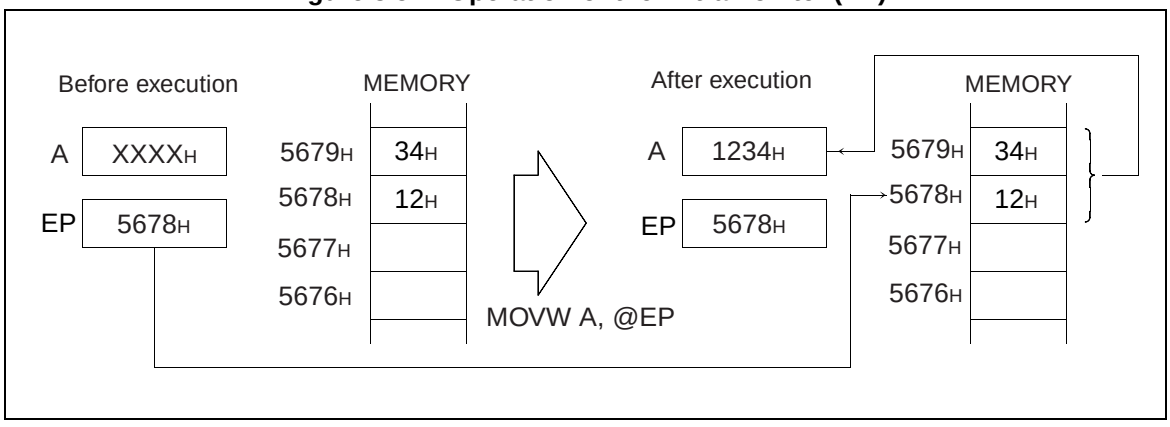
Figure 3.5-1 Operation of Index Register (IX)



■ Extra Pointer (EP)

Figure 3.5-2 shows the operation of the extra pointer.

Figure 3.5-2 Operation of the Extra Pointer (EP)



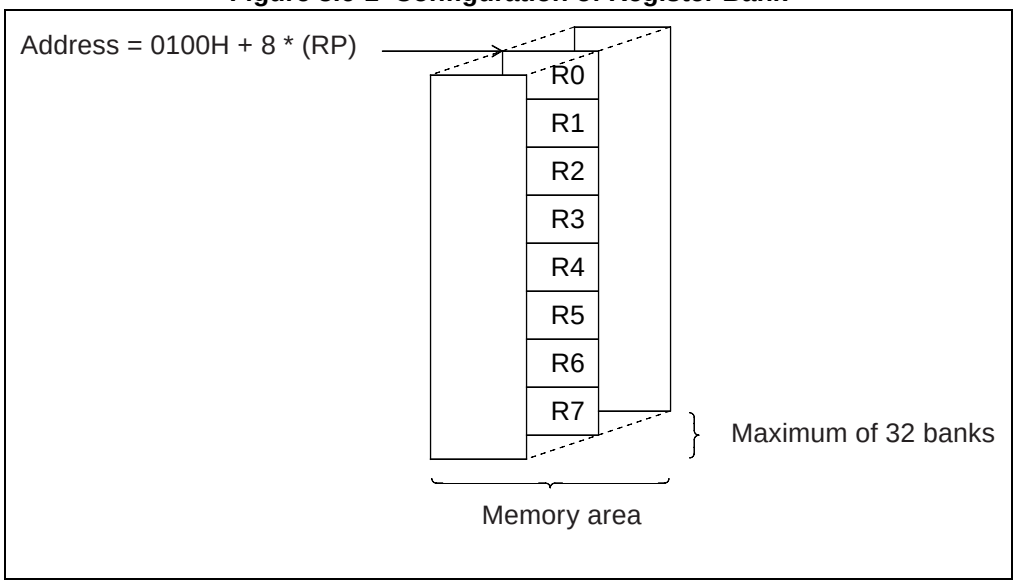
3.6 Register Banks

The register bank register is an 8-bit general-purpose register existing in memory. There are eight registers per bank of which there can be 32 altogether. The current bank is indicated by the register bank pointer (RP).

■ Register Bank Register

Figure 3.6-1 shows the configuration of the register bank.

Figure 3.6-1 Configuration of Register Bank



3.7 Direct Banks

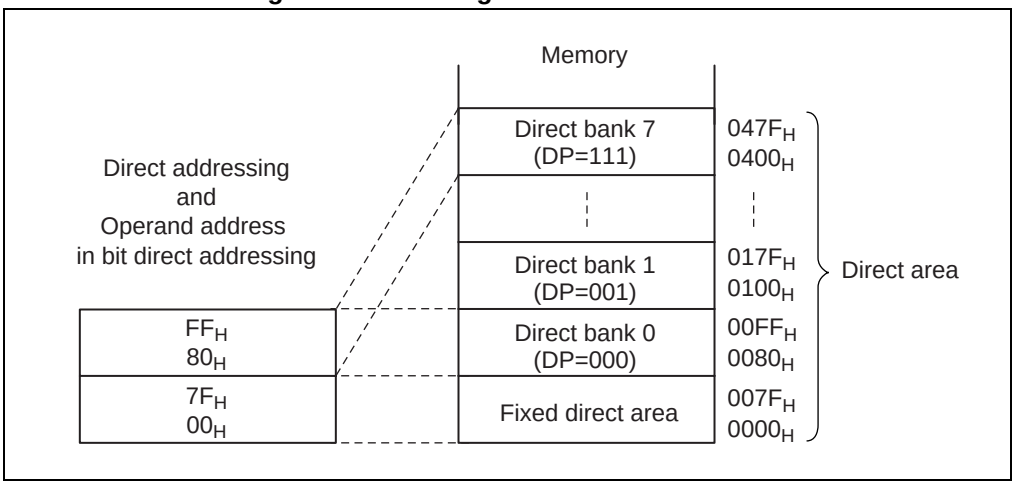
The direct bank is in 0080_H to $047F_H$ of direct area, and composed of 128 bytes $\times$ 8 banks. The access that uses direct addressing and bit direct addressing in operand address 80_H to FF_H can be extended to 8 direct banks according to the value of the direct bank pointer (DP). The current bank is indicated by the direct bank pointer (DP).

■ Direct Bank

Figure 3.7-1 shows the configuration of a direct bank.

The access that uses direct addressing and bit direct addressing in operand address 80_H to FF_H can be extended to 8 direct banks according to the value of the direct bank pointer (DP). The access that uses direct addressing and bit direct addressing in operand address 00_H to $7F_H$ is not affected by the value of the direct bank pointer (DP). This access is directed to fixed direct area 0000_H to $007F_H$.

Figure 3.7-1 Configuration of Direct Bank



CHAPTER 4

INTERRUPT PROCESSING

This chapter explains the functions and operation of F^{2}MC-8FX interrupt processing.

4.1 Outline of Interrupt Operation

F²MC-8FX series interrupts have the following features:

- **Four interrupt priority levels**
- **All maskable features**
- **Vector jump feature by which the program jumps to address mentioned in the interrupt vector.**

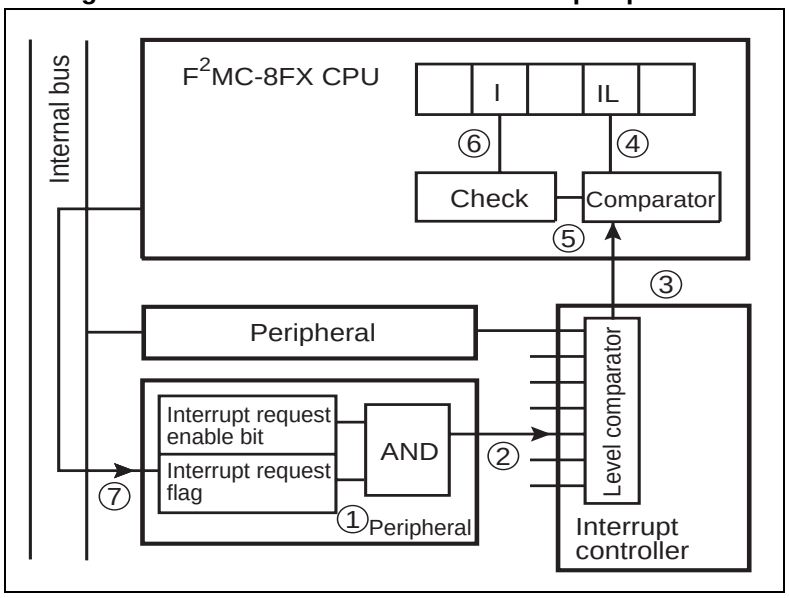
■ Outline of Interrupt Operation

In the F²MC-8FX series, interrupts are transferred and processed according to the following procedure:

1. An interrupt source occurs in resources.
2. Refer to interrupt enable bits in resources. If an interrupt is enabled, interrupt requests are issued from resources to the interrupt controller.
3. As soon as an interrupt request is received, the interrupt controller decides the priorities of the interrupt requested and then transfers the interrupt level corresponding to the interrupts applicable to the CPU.
4. The CPU compares the interrupt levels requested by the interrupt controller with the IL bit in the program status register.
5. In the comparison, the CPU checks the contents of the I flag in the same program status register only if the priority is higher than the current interrupt processing level.
6. In the check in 5., the CPU sets the contents of the IL bit to the requested level only if the I flag is enabled for interrupts, processes interrupts as soon as the instruction currently being executed is completed and then transfers control to the interrupt processing routine.
7. The CPU clears the interrupt source caused in 1. using software in the user's interrupt processing routine to terminate the processing of interrupts.

Figure 4.1-1 shows the flow diagram of F^{2}MC-8FX interrupt operation.

Figure 4.1-1 Outline of F^{2}MC-8FX Interrupt Operation



4.2 Interrupt Enable/Disable and Interrupt Priority Functions

In the F^{2}MC-8FX series, interrupt requests are transferred to the CPU using the three types of enable/disable functions listed below.

- **Request enable check by interrupt enable flags in resources**
- **Checking the level using the interrupt level determination function**
- **Interrupt start check by the I flag in the CPU**

Interrupts generated in resources are transferred to the CPU with the priority levels determined by the interrupt priority function.

■ Interrupt Enable/Disable Functions

- Request enable check by interrupt enable flags in resources

  This is a function to enable/disable a request at the interrupt source. If interrupt enable flags in resources are enabled, interrupt request signals are sent from resources to the interrupt controller. This function is used for controlling the presence or absence of an interrupt, resource-by-resource. It is very useful because when software is described for each resource operation, interrupts in another resource do not need to be checked for whether they are enabled or disabled.

- Checking the level using the interrupt level determination function

  This function determines the interrupt level. The interrupt levels corresponding to interrupts generated in resources are compared with the IL bit in the CPU. If the value is less than the IL bit, a decision is made to issue an interrupt request. This function is able to assign priorities if there are two or more interrupts.

- Interrupt start check by the I flag in the CPU

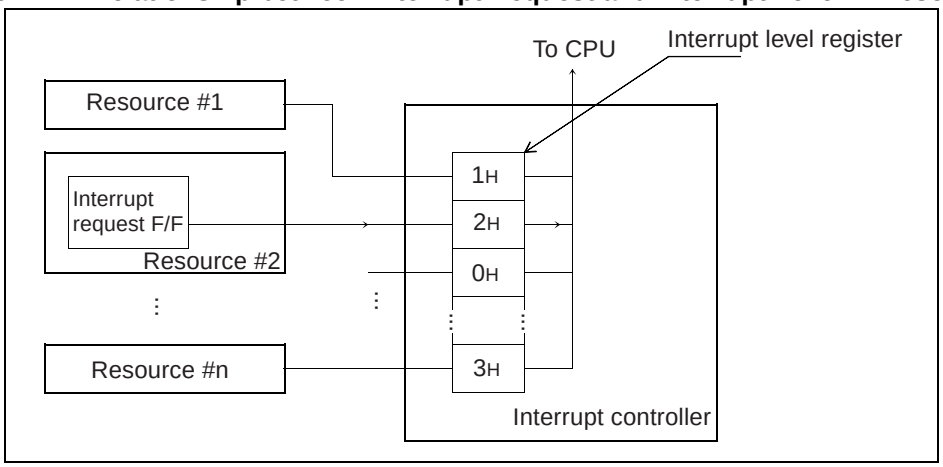
  The I flag enables or disables the entire interrupt. If an interrupt request is issued and the I flag in the CPU is set to interrupt enable, the CPU temporarily suspends the flow of instruction execution to process interrupts. This function is able to temporarily disable the entire interrupt.

■ Interrupt Requests in Resources

As shown in Figure 4.2-1, interrupts generated in resources are converted by the corresponding interrupt level registers in the interrupt controller into the values set by software and then transferred to the CPU.

The interrupt level is defined as high if its numerical value is lower, and low if it is higher.

Figure 4.2-1 Relationship between Interrupt Request and Interrupt Level in Resources



4.3 Creating an Interrupt Processing Program

In the F^{2}MC-8FX series, basically, interrupt requests from resources are issued by hardware and cleared by software.

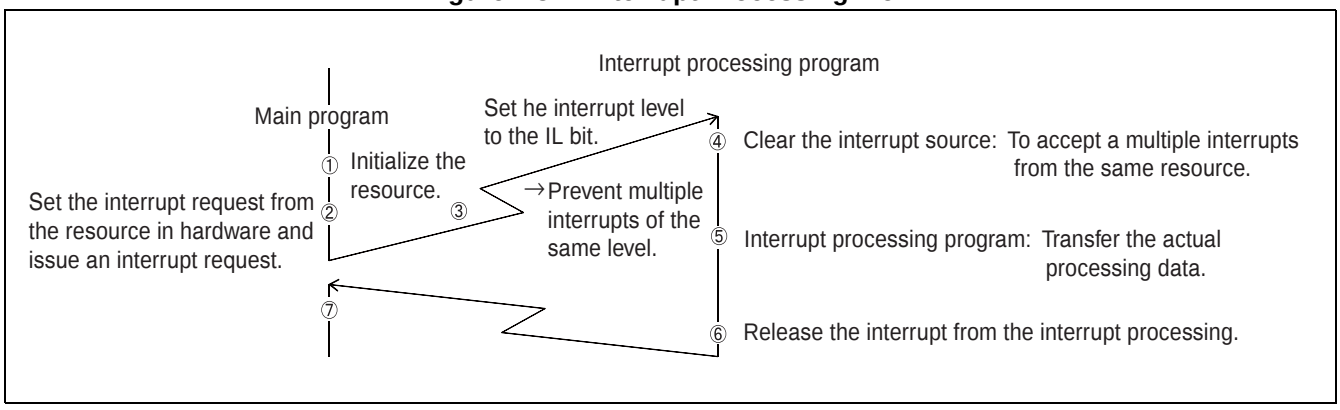
■ Creating an Interrupt Processing Program

The interrupt processing control flow is as follows:

1. Initialize resources before operation.
2. Wait until an interrupt occurs.
3. In the event of an interrupt, if the interrupt can be accepted, perform interrupt processing to branch to the interrupt processing routine.
4. First, set software so as to clear the interrupt source at the beginning of the interrupt processing routine. This is done so that the resource causing an interrupt can regenerate the interrupt during the interrupt processing program.
5. Next, perform interrupt processing to transfer the necessary data.
6. Use the interrupt release instruction to release the interrupt from interrupt processing.
7. Then, continue to execute the main program until an interrupt recurs. The typical interrupt processing flow is shown in Figure 4.3-1.

The numbers in the figure correspond to the numbers above.

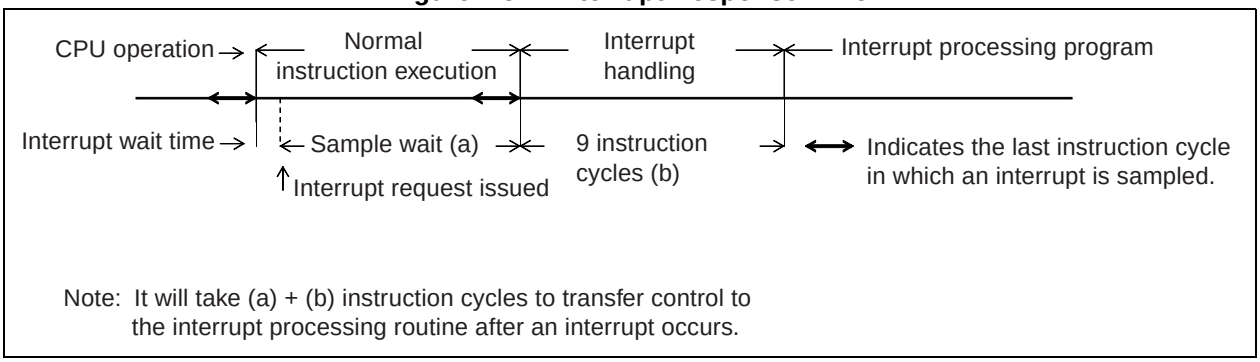
Figure 4.3-1 Interrupt Processing Flow



The time to transfer control to the interrupt processing routine after the occurrence of an interrupt 3 in Figure 4.3-1) is 9 instruction cycles. An interrupt can only be processed in the last cycle of each instruction. The time shown in Figure 4.3-2 is required to transfer control to the interrupt processing routine after an interrupt occurs.

The longest cycle (17 + 9 = 26 instruction cycles) is required when an interrupt request is issued immediately after starting the execution of the DIVU instruction.

Figure 4.3-2 Interrupt Response Time



Note: It will take (a) + (b) instruction cycles to transfer control to the interrupt processing routine after an interrupt occurs.

4.4 Multiple Interrupt

The F²MC-8FX CPU can have a maximum of four levels as maskable interrupts. These can be used to assign priorities to interrupts from resources.

■ Multiple Interrupt

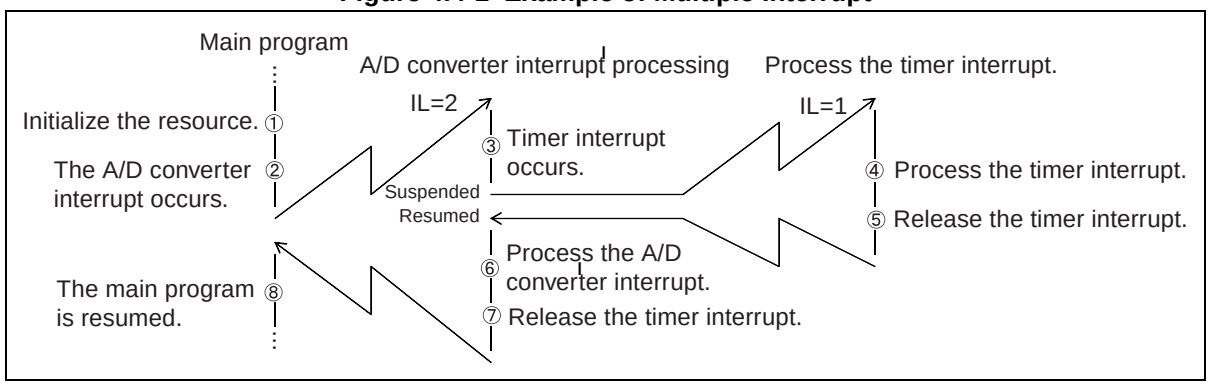
A specific example is given below.

- When giving priority over the A/D converter to the timer interrupt

```
START   MOV   ADIL,   #2      Set the interrupt level of the A/D converter to 2.
        MOV   TMIL,   #1      Set the interrupt level of the timer to 1. ADIL and
                              TMIL are IL bits in the interrupt controller.
        CALL  STAD            Start the A/D converter.
        CALL  STTM            Start the timer.
        .
        .
        .
```

When the above program is started, interrupts are generated from the A/D converter and timer after an elapsed time. In this case, when the timer interrupt occurs while processing the A/D converter interrupt, it will be processed through the sequence shown in Figure 4.4-1.

Figure 4.4-1 Example of Multiple Interrupt



When starting processing of an A/D converter interrupt, the IL bit in the PS register of the CPU is automatically the same as the value of request (2 here). Therefore, when a level 1 or 0 interrupt request is issued during the processing of an A/D converter interrupt, the processing proceeds without disabling the A/D converter interrupt request. When temporarily disabling interrupts lower in priority than this interrupt during A/D converter interrupt processing, disable the I flag in the PS register of the CPU for the interrupts or set the IL bit to 0.

When control is returned to the interrupted routine by the release instruction after completion of each interrupt processing routine, the PS register is set to the value saved in the stack. Consequently, the IL bit takes on the value before interruption.

For actual coding, refer to the Hardware Manual for each device to check the addresses of the interrupt controller and each resource and the interrupts to be supported.

4.5 Reset Operation

In the F^{2}MC-8FX series, when a reset occurs, the flag of program status is 0 and the IL bit is set to 11. When cleared, the reset operation is executed from the starting address written to set vectors ($FFFE_H$, $FFFF_H$).

■ Reset Operation

A reset affects:

- Accumulator, temporary accumulator: Initializes to 0000_H
- Stack pointer: Initializes to 0000_H
- Extra pointer, index register: Initializes to 0000_H
- Program status: Sets flag to 0, sets IL bit to 11, sets RP bit to 00000 and Initializes DP bit to 000
- Program counter: Reset vector values
- RAM (including general-purpose registers): Keeps value before reset
- Resources: Basically stop
- Others: Refer to the manual for each product for the condition of each pin

Refer to the manual for each product for details of the value and operation of each register for special reset conditions.

CHAPTER 5

CPU SOFTWARE ARCHITECTURE

This chapter explains the instructions for the F^{2}MC-8FX CPU.

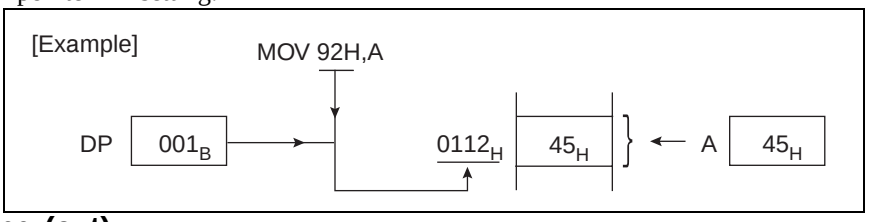
5.1 Types of Addressing Modes

The F²MC-8FX CPU has the following ten addressing modes:

- **Direct addressing (dir)**
- **Extended addressing (ext)**
- **Bit direct addressing (dir:b)**
- **Indexed addressing (@IX+off)**
- **Pointer addressing (@EP)**
- **General-purpose register addressing (Ri)**
- **Immediate addressing (#imm)**
- **Vector addressing (#k)**
- **Relative addressing (rel)**
- **Inherent addressing**

■ Direct Addressing (dir)

This addressing mode, indicated as "dir" in the instruction list, is used to access the direct area from 0000_H to $047F_H$. In this addressing, when the operand address is 00_H to $7F_H$, it accesses 0000_H to $007F_H$. Moreover, when the operand address is 80_H to FF_H, the access is good to 0080_H to $047F_H$ at the mapping by direct bank pointer DP setting.

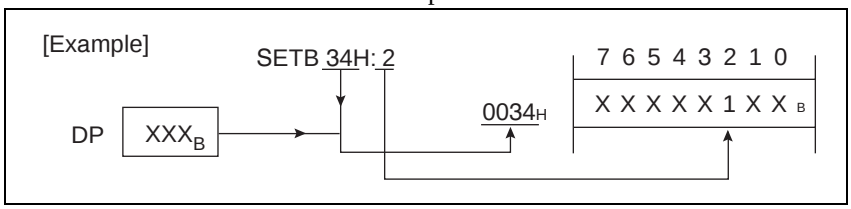


■ Extended Addressing (ext)

This addressing mode, indicated as "ext" in the instruction list, is used to access the entire 64-Kbyte area. In this addressing mode, the upper byte is specified by the first operand and the lower byte by the second operand.

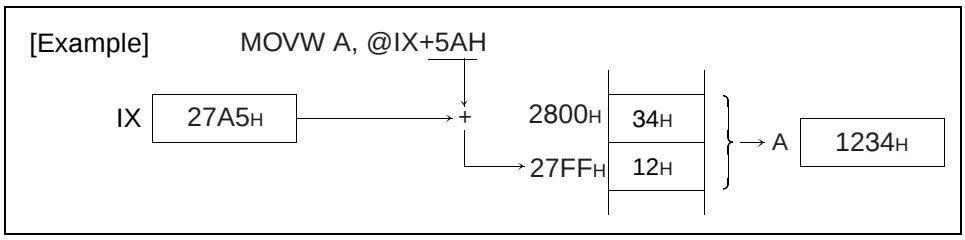
■ Bit Direct Addressing (dir:b)

This addressing mode, indicated as "dir:b" in the instruction list, is used for bit-by-bit access of the direct area from 0000_H to $047F_H$. In this addressing, when the operand address is 00_H to $7F_H$, it accesses 0000_H to $007F_H$. Moreover, when the operand address is 80_H to FF_H, the access is good to 0080_H to $047F_H$ at the mapping by direct bank pointer DP setting. The position of the bit in the specified address is specified by the value for the instruction code of three subordinate position bits.



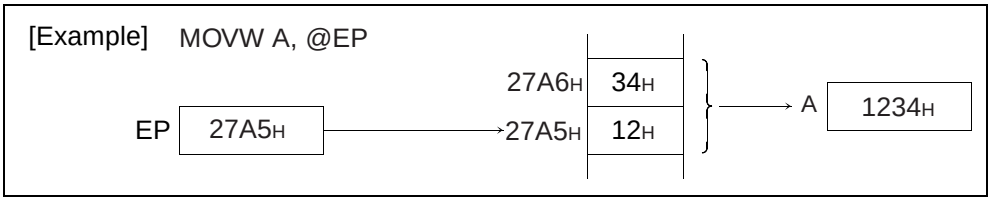
■ Index Addressing (@IX+off)

This addressing mode, indicated as "@IX+off" in the instruction list, is used to access the entire 64-Kbyte area. In this addressing mode, the contents of the first operand are sign-extended and then added to the index register (IX). The result is used as the address.



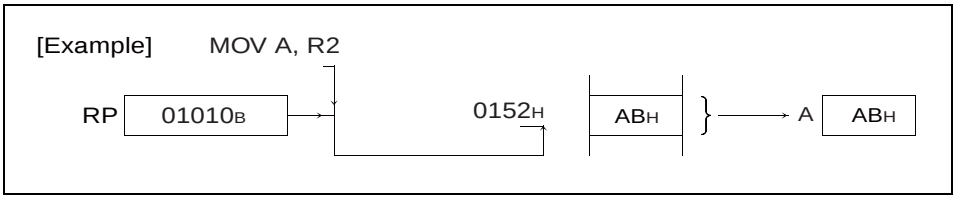
■ Pointer Addressing (@EP)

This addressing mode, indicated as "@EP" in the instruction list, is used to access the entire 64-Kbyte area. In this addressing mode, the contents of the extra pointer (EP) are used as the address.



■ General-Purpose Register Addressing (Ri)

This addressing mode, indicated as "Ri" in the instruction list, is used to access the register bank area. In this addressing mode, one upper byte of the address is set to 01 and one lower byte is created from the contents of the register bank pointer (RP) and the 3 lower bits of the instruction to access this address.



■ Immediate Addressing (#imm)

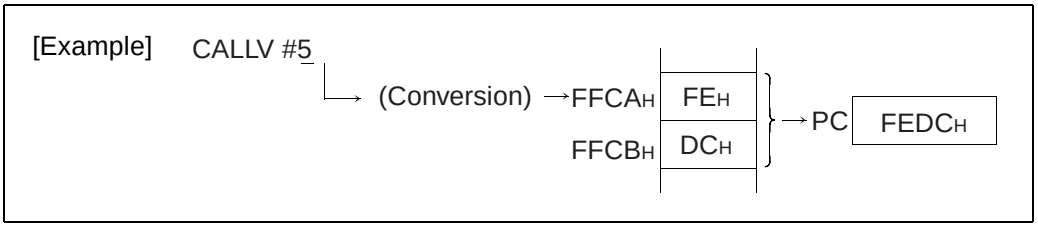
This addressing mode, indicated as "#imm" in the instruction list, is used for acquiring the immediate data. In this addressing mode, the operand is used directly as the immediate data. The byte or word is specified by the instruction code.

■ Vector Addressing (#k)

This addressing mode, indicated as "#k" in the instruction list, is used for branching to the subroutine address registered in the table. In this addressing mode, the information about #k is contained in the instruction code and the table addresses listed in Table 5.1-1 are created.

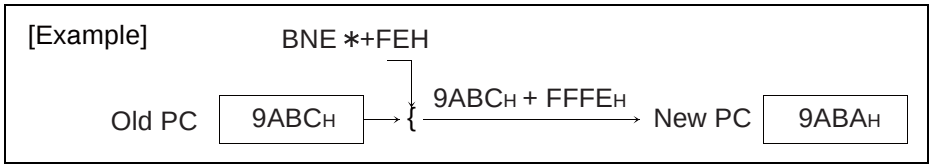
Table 5.1-1 Jump Address Table

#k	Address table (upper jump address: lower jump address)
0	$FFC0_H$:$FFC1_H$
1	$FFC2_H$:$FFC3_H$
2	$FFC4_H$:$FFC5_H$
3	$FFC6_H$:$FFC7_H$
4	$FFC8_H$:$FFC9_H$
5	$FFCA_H$:$FFCB_H$
6	$FFCC_H$:$FFCD_H$
7	$FFCE_H$:$FFCF_H$



■ Relative Addressing (rel)

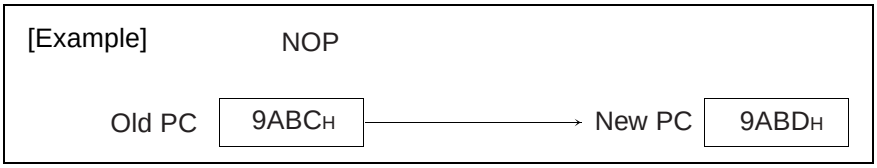
This addressing mode, indicated as "rel" in the instruction list, is used for branching to the 128-byte area across the program counter (PC). In this addressing mode, the contents of the operand are added with their sign, to the program counter. The result is stored in the program counter.



In this example, the program jumps to the address where the instruction code BNE is stored, resulting in an infinite loop.

■ Inherent Addressing

This addressing mode, which has no operand in the instruction list, is used for operations to be determined by the instruction code. In this addressing mode, the operation varies for every instruction.



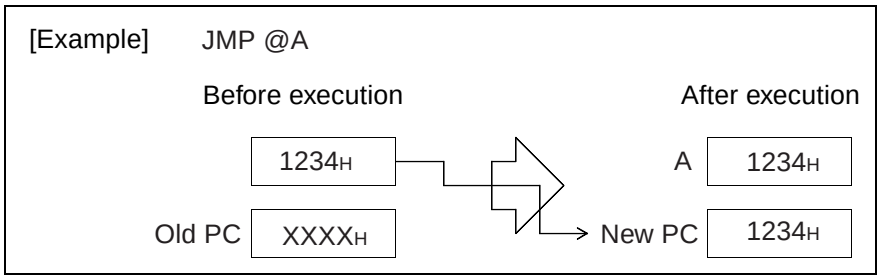
5.2 Special Instructions

In the F^{2}MC-8FX series, the following six special instructions are available:

- **JMP @A**
- **MOVW A, PC**
- **MULU A**
- **DIVU A**
- **XCHW A, PC**
- **CALLV #k**

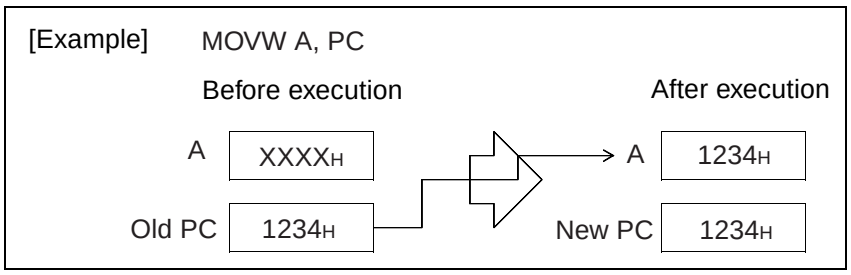
■ JMP @A

This instruction is used for branching to an address where the contents of the accumulator (A) are used. The contents of one of the N jump addresses arranged in table form is selected and transferred to the accumulator. Executing this instruction enables the N-branch processing.



■ MOVW A, PC

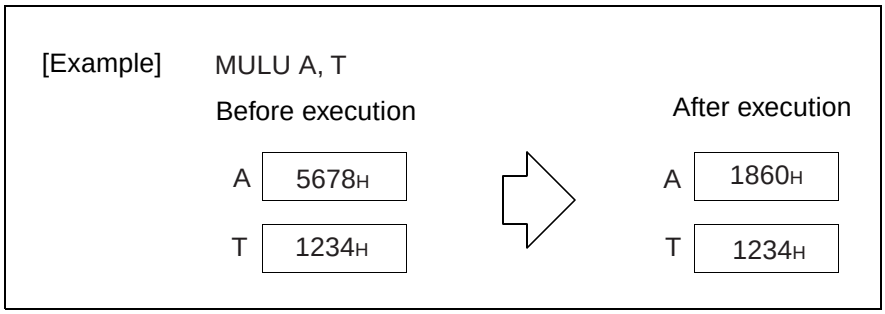
This instruction is used for performing the opposite operation to JMP @A. In other words, it stores, the contents of the program counter (PC) in the accumulator (A). When this instruction is executed in the main routine and a specific subroutine is to be called, make sure that the contents of the accumulator are the specified value in the subroutine, that is the branch is from the expected section, enabling a decision on crash.



When this instruction is executed, the contents of the accumulator are the same as those of the address where the code for the next instruction is stored and not the address where the code for this instruction is stored. The above example shows that the value 1234_H stored in the accumulator agrees with that of the address where the instruction code next to MOVW A, PC is stored.

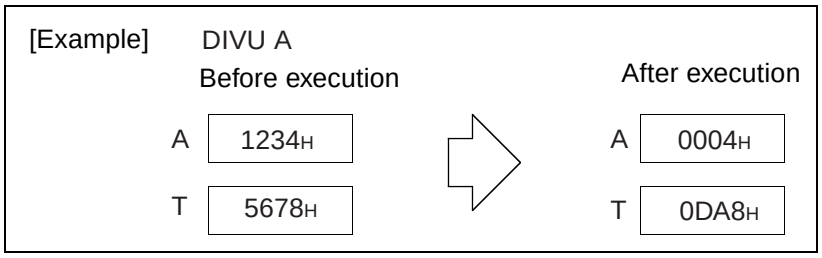
■ MULU A

This instruction is used for multiplying 8 bits of the AL by 8 bits of the TL without a sign and stores the 16-bit result in the accumulator (A). The contents of the temporary accumulator (T) do not change. In the operation, the original contents of the AH and TH are not used. Since the flag does not change, attention must be paid to the result of multiplication when branching accordingly.



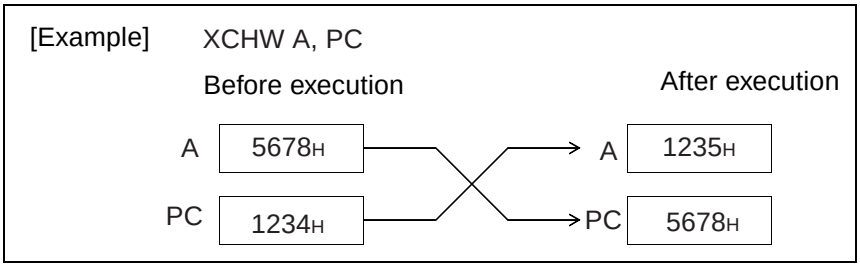
■ DIVU A

This instruction is used for dividing 16 bits of the temporary accumulator (T) by 16 bits of the A without a sign and stores the results as 16 bits in the A and the remainder as 16 bits in the T. When A is 0000_H, Z flag is 1 as 0 division. At this time, the operation result is not guaranteed.



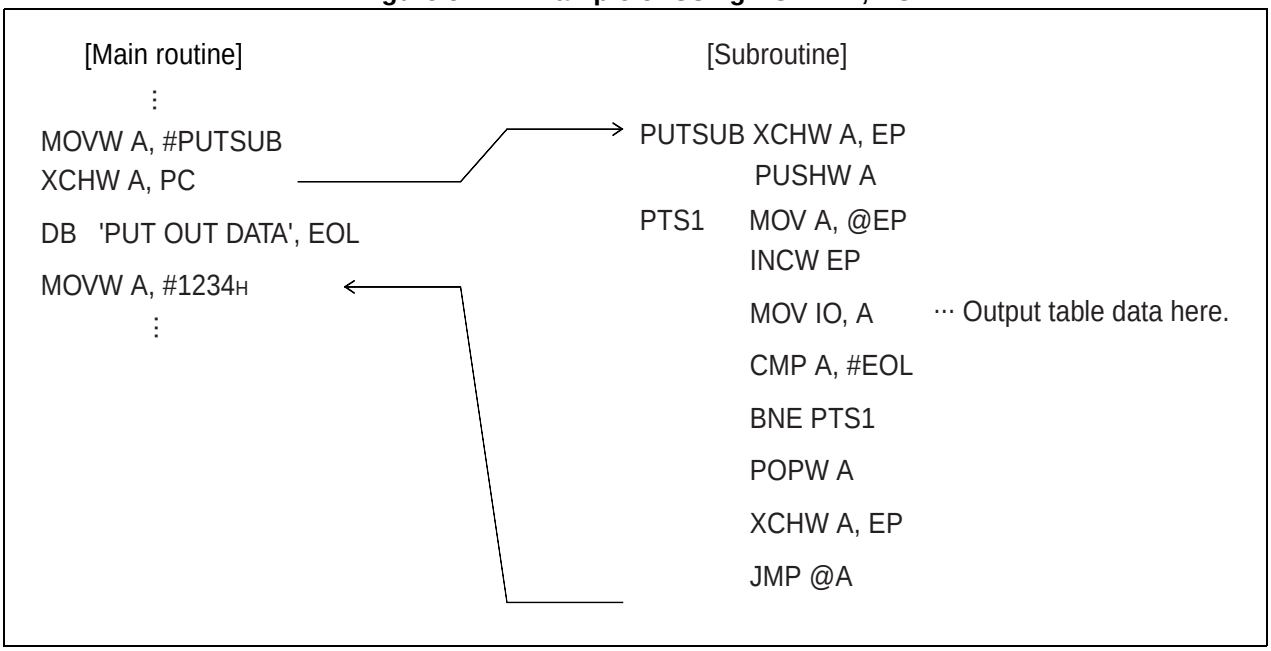
■ XCHW A, PC

This instruction is used for exchanging the contents of the accumulator (A) for those of the program counter (PC). As a result, the program branches to the address indicated by the contents of the original accumulator and the contents of the current accumulator become the value of the address next to the one where the instruction code XCHW A, PC is stored. This instruction is provided especially for specifying tables using the main routine and for subroutines to use them.



When this instruction is executed, the contents of the accumulator are the same as those of the address where the code for the next instruction is stored and not the address where the code for this instruction is stored. The above example shows that the value of the accumulator 1235_H agrees with that of the address where the instruction code next to XCHW A, PC is stored. Consequently, 1235_H not 1234_H is indicated.

Figure 5.2-1 Example of Using XCHW A, PC

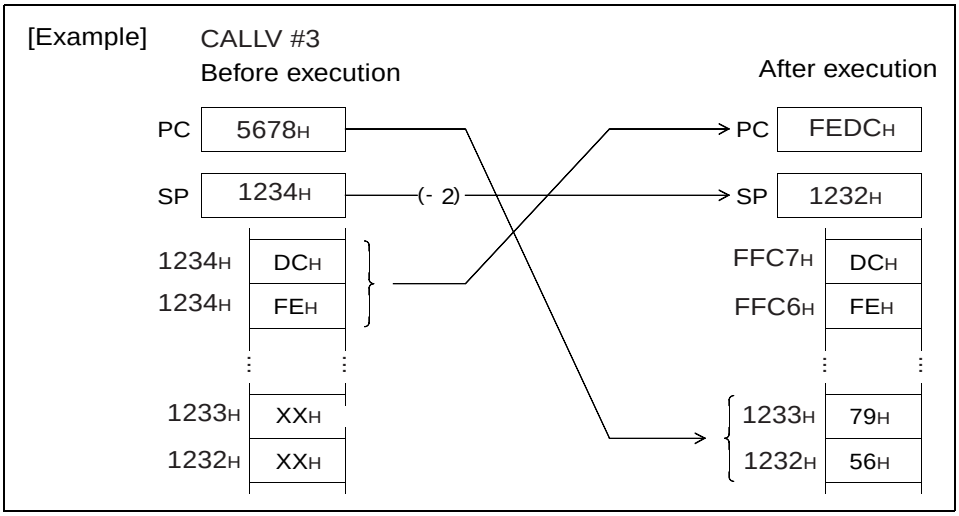


■ CALLV #k

This instruction is used for branching to a subroutine address registered in the table. In this addressing mode, the information about #k is included in the instruction code and the tale addresses listed in Table 5.2-1 are created. After saving the contents of the current program counter (PC) in the stack, the program branches to the address in the table. Because it is a 1-byte instruction, using it for frequently-used subroutines reduces the size of the entire program.

Table 5.2-1 Jump Address Table

#k	Address table (upper jump address : lower jump address)
0	$FFC0_H$:$FFC1_H$
1	$FFC2_H$:$FFC3_H$
2	$FFC4_H$:$FFC5_H$
3	$FFC6_H$:$FFC7_H$
4	$FFC8_H$:$FFC9_H$
5	$FFCA_H$:$FFCB_H$
6	$FFCC_H$:$FFCD_H$
7	$FFCE_H$:$FFCF_H$



CHAPTER 6

DETAILED RULES FOR EXECUTION INSTRUCTIONS

This chapter explains each execution instruction, used in the assembler, in reference format.
All execution insurrections are described in alphabetical order.

For information about the outline of each item and the meaning of symbols (abbreviations) explained for each execution instruction, see "CHAPTER 5 CPU SOFTWARE ARCHITECTURE".

6.1 ADDC (ADD Byte Data of Accumulator and Temporary Accumulator with Carry to Accumulator)

Add the byte data of TL to that of AL, add a carry to the LSB and then return the results to AL. The contents of AH are not changed.

■ ADDC (ADD Byte Data of Accumulator and Temporary Accumulator with Carry to Accumulator)

Operation

(AL) ← (AL) + (TL) + (C) (Byte addition with carry)

Assembler format

ADDC A

Condition code (CCR)

N	Z	V	C
+	+	+	+

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB of AL is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

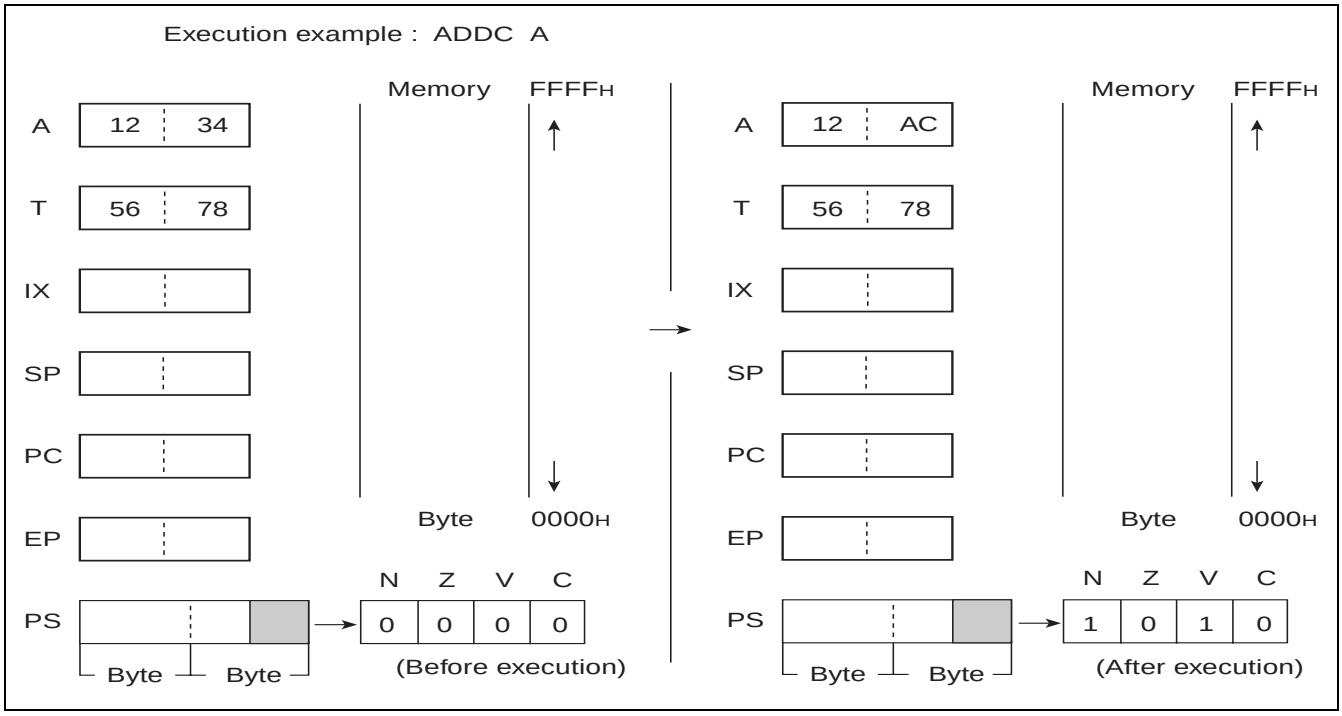
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

C: Set to 1 if a carry occurs as the result of operation and set to 0 in other cases.

Number of execution cycle: 1

Byte count: 1

OP code: 22



6.2 ADDC (ADD Byte Data of Accumulator and Memory with Carry to Accumulator)

Add the byte data of EA memory (memory expressed in each type of addressing) to that of AL, add a carry to the LSB and then return the results to AL. The contents of AH are not changed.

■ ADDC (ADD Byte Data of Accumulator and Memory with Carry to Accumulator)

Operation

(AL) ← (AL) + (EA) + (C) (Byte addition with carry)

Assembler format

ADDC A, EA

Condition code (CCR)

N	Z	V	C
+	+	+	+

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB of AL is 1 as the result of operation and set to 0 in other cases.

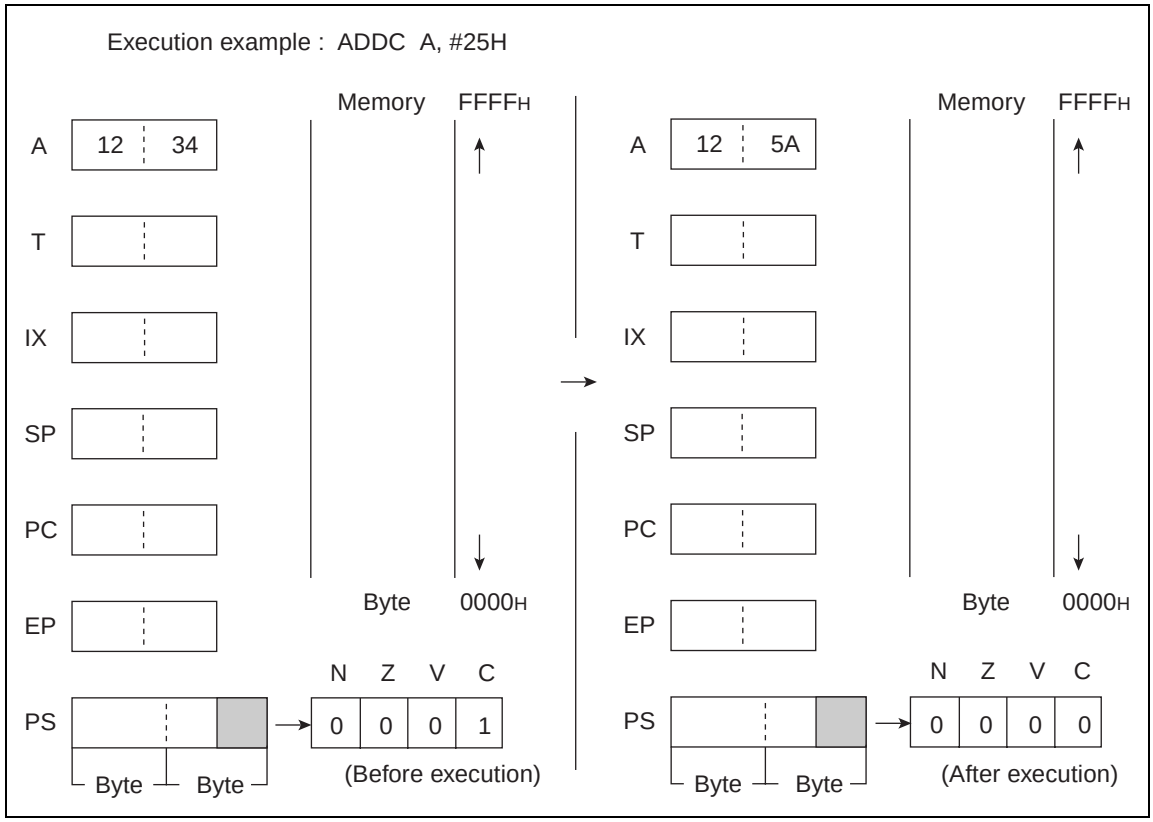
Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

C: Set to 1 if a carry occurs as the result of operation and set to 0 in other cases.

Table 6.2-1 Number of Execution Cycles / Byte Count / OP Code

EA	#d8	dir	@IX+off	@EP	Ri
Number of execution cycles	2	3	3	2	2
Byte count	2	2	2	1	1
OP code	24	25	26	27	28 to 2F



6.3 ADDCW (ADD Word Data of Accumulator and Temporary Accumulator with Carry to Accumulator)

Add the word data of T to that of A, add a carry to the LSB and then return the results to A.

■ ADDCW (ADD Word Data of Accumulator and Temporary Accumulator with Carry to Accumulator)

Operation

(A) ← (A) + (T) + (C) (Word addition with carry)

Assembler format

ADDCW A

Condition code (CCR)

N	Z	V	C
+	+	+	+

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB of A is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 0000_H and set to 0 in other cases.

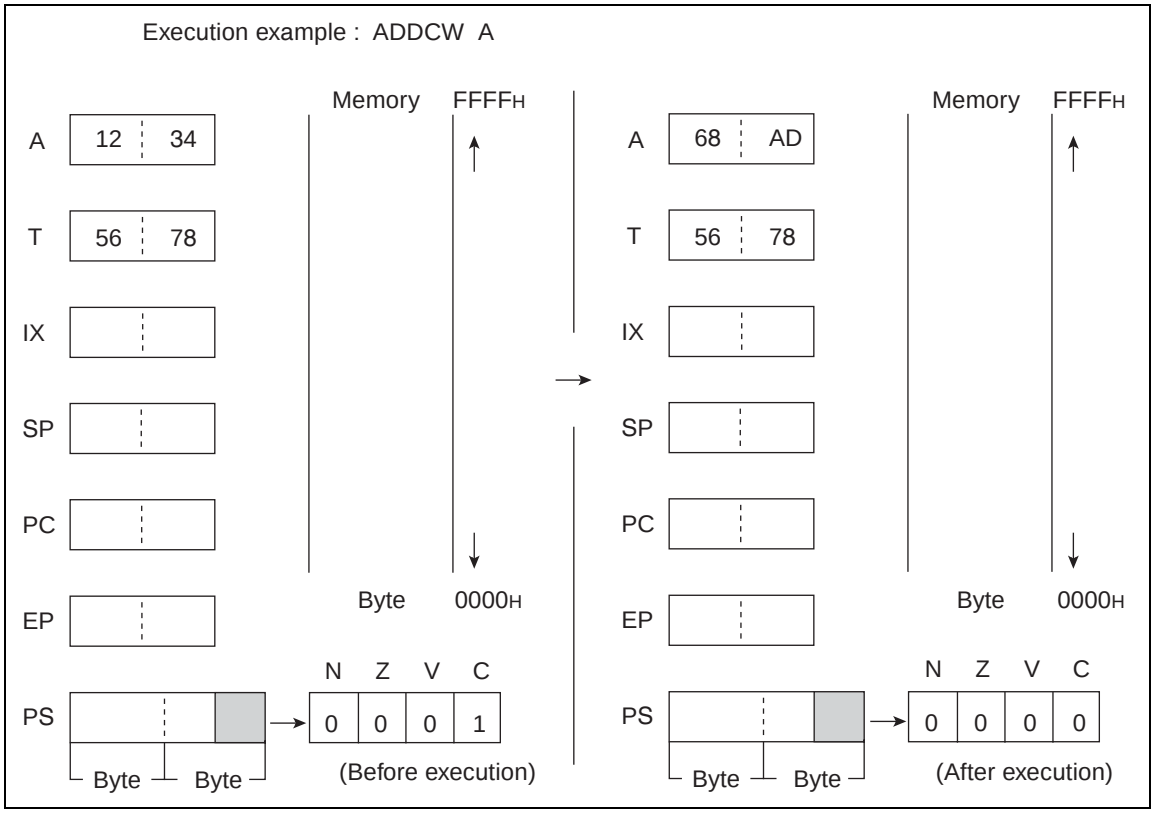
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

C: Set to 1 if a carry occurs as the result of operation and set to 0 in other cases.

Number of execution cycle: 1

Byte count: 1

OP code: 23



6.4 AND (AND Byte Data of Accumulator and Temporary Accumulator to Accumulator)

Carry out the logical AND on the byte data of AL and TL for every bit and return the result to AL. The byte data of AH is not changed.

■ AND (AND Byte Data of Accumulator and Temporary Accumulator to Accumulator)

Operation

(AL) ← (AL) ^ (TL) (Byte AND)

Assembler format

AND A

Condition code (CCR)

N	Z	V	C
+	+	R	-

+: Changed by executing instruction

-: Not changed

R: Set to 0 by executing instruction

N: Set to 1 if the MSB of AL is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

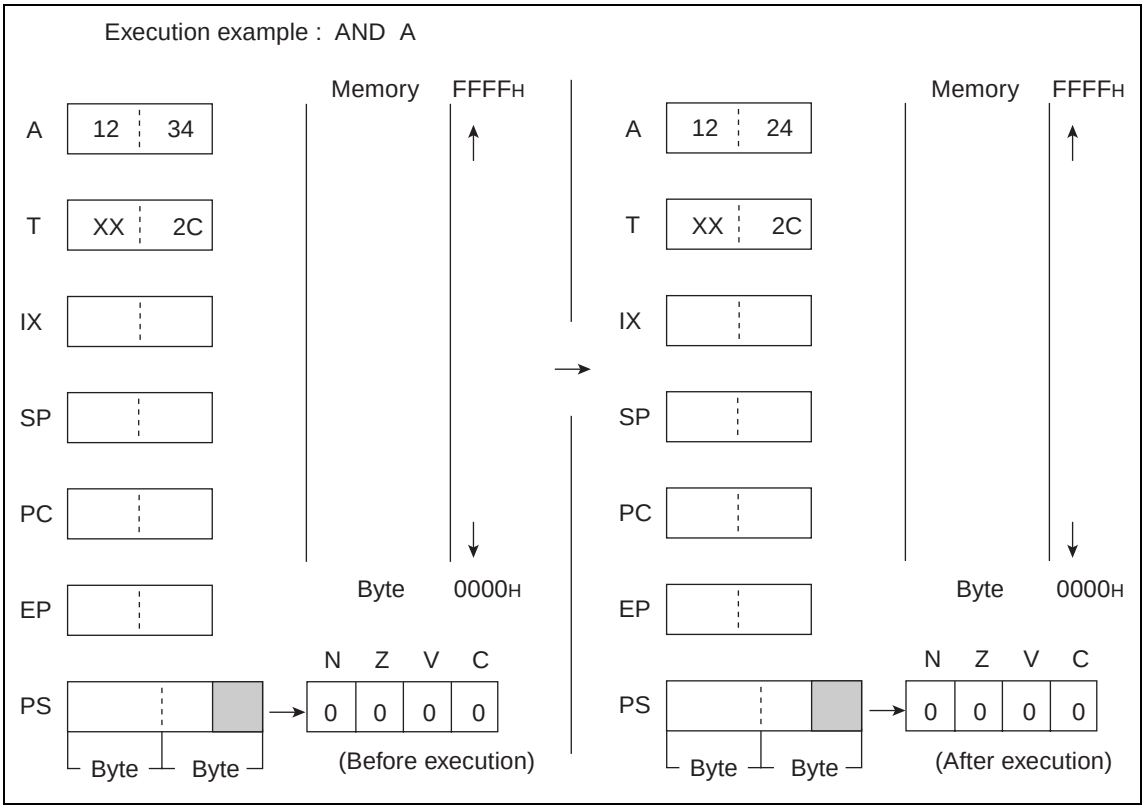
V: Always set to 0

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: 62



6.5 AND (AND Byte Data of Accumulator and Memory to Accumulator)

Carry out the logical AND on the byte data of AL and EA memory (memory expressed in each type of addressing) for every bit and return the result to AL. The byte data of AH is not changed.

■ AND (AND Byte Data of Accumulator and Memory to Accumulator)

Operation

(AL) ← (AL) ^ (EA) (Byte AND)

Assembler format

AND A, EA

Condition code (CCR)

N	Z	V	C
+	+	R	-

+: Changed by executing instruction

-: Not changed

R: Set to 0 by executing instruction

N: Set to 1 if the MSB of AL is 1 as the result of operation and set to 0 in other cases.

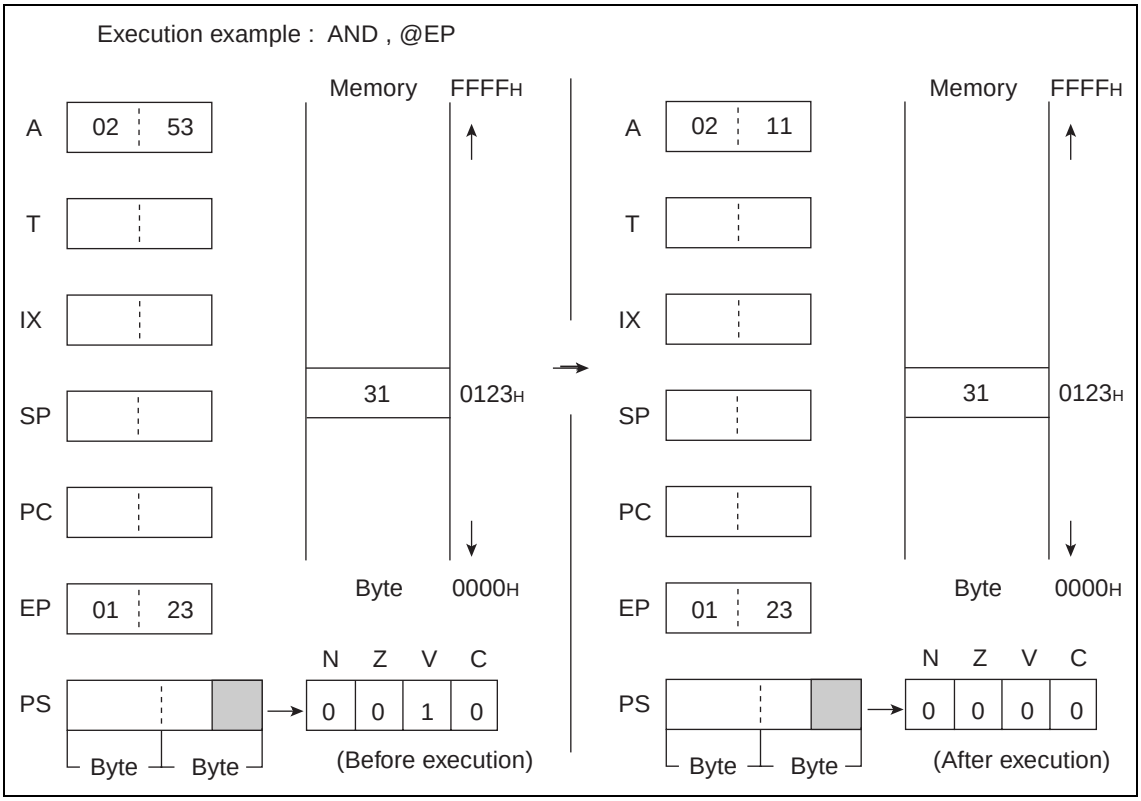
Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

V: Always set to 0

C: Not changed

Table 6.5-1 Number of Execution Cycles / Byte Count / OP Code

EA	#d8	dir	@IX+off	@EP	Ri
Number of execution cycles	2	3	3	2	2
Byte count	2	2	2	1	1
OP code	64	65	66	67	68 to 6F



6.6 ANDW (AND Word Data of Accumulator and Temporary Accumulator to Accumulator)

Carry out the logical AND on the word data of A and T for every bit and return the results to A.

■ ANDW (AND Word Data of Accumulator and Temporary Accumulator to Accumulator)

Operation

(A) ← (A) ^ (T) (Word AND)

Assembler format

ANDW A

Condition code (CCR)

N	Z	V	C
+	+	R	-

+: Changed by executing instruction

-: Not changed

R: Set to 0 by executing instruction

N: Set to 1 if the MSB of A is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 0000_H and set to 0 in other cases.

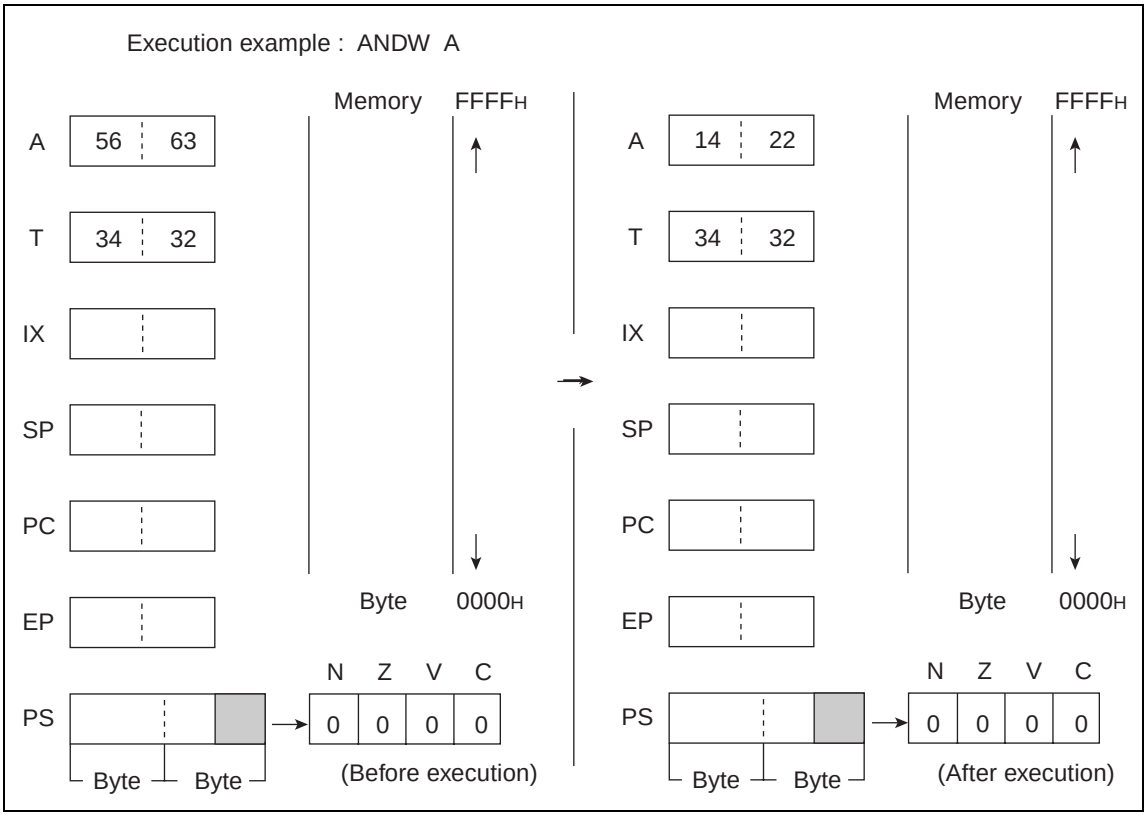
V: Always set to 0

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: 63



6.7 BBC (Branch if Bit is Clear)

Branch when the value of bit b in dir memory is 0. Branch address corresponds to the value of addition between the PC value (word value) of the next instruction and the value with rel code-extended (word value).

■ BBC (Branch if Bit is Clear)

Operation

(bit)b = 0: (PC) ← (PC) + 3 + rel (Word addition)

(bit)b = 1: (PC) ← (PC) + 3 (Word addition)

Assembler format

BBC dir:b, rel

Condition code (CCR)

N	Z	V	C
-	+	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Set to 1 when the value of dir:b is 0 and set to 0 when it is 1.

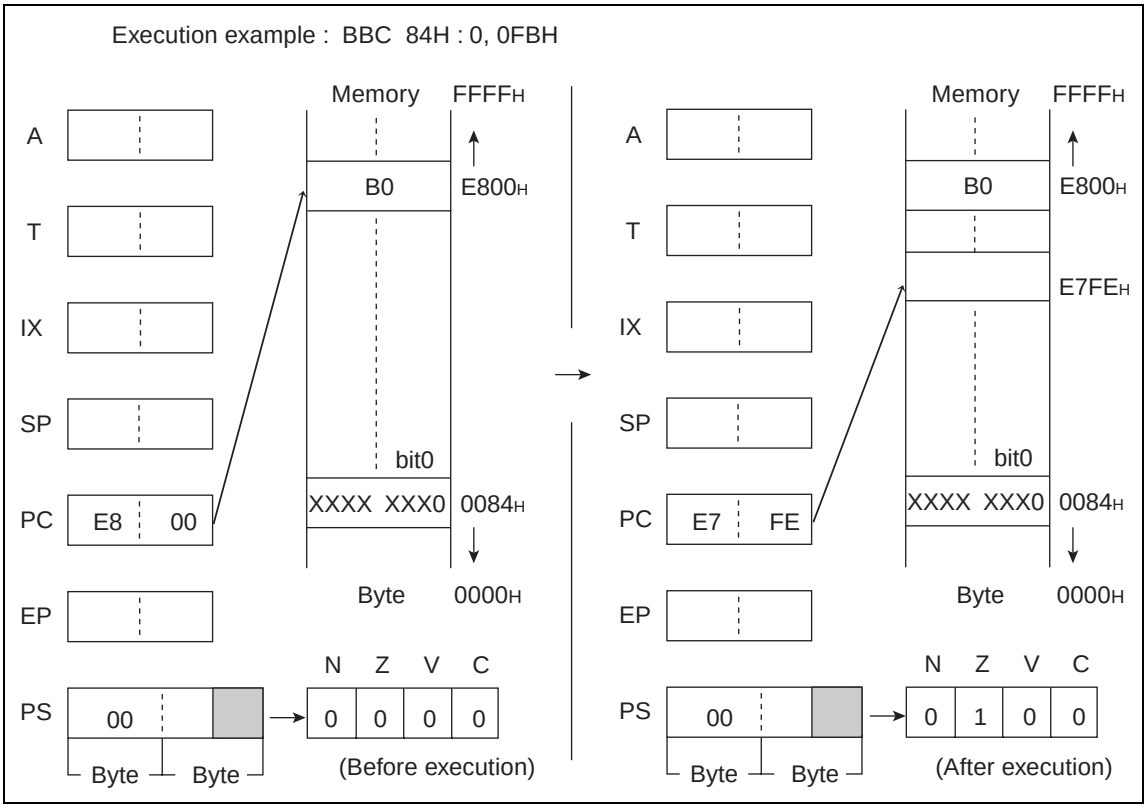
V: Not changed

C: Not changed

Number of execution cycles: 5

Byte count: 3

OP code: B0 to B7



6.8 BBS (Branch if Bit is Set)

Branch when the value of bit b in dir memory is 1. Branch address corresponds to the value of addition between the PC value (word value) of the next instruction and the value with rel code-extended (word value).

■ BBS (Branch if Bit is Set)

Operation

(bit)b = 0: (PC) ← (PC) + 3 (Word addition)

(bit)b = 1: (PC) ← (PC) + 3 + rel (Word addition)

Assembler format

BBS dir:b, rel

Condition code (CCR)

N	Z	V	C
-	+	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Set to 1 when the value of dir:b is 0 and set to 0 when it is 1.

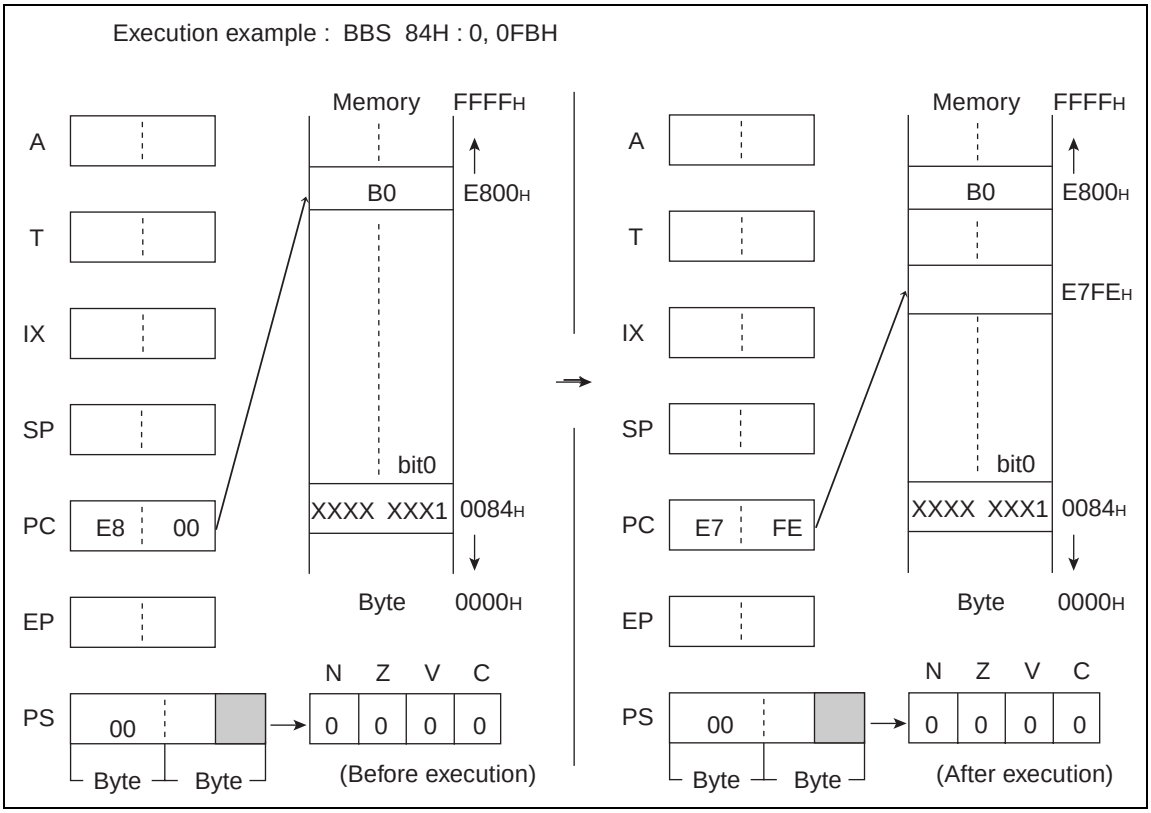
V: Not changed

C: Not changed

Number of execution cycles: 5

Byte count: 3

OP code: B8 to BF



6.9 BC (Branch relative if C=1)/BLO (Branch if LOwer)

Execute the next instruction if the C-flag is 0 and the branch if it is 1. Branch address corresponds to the value of addition between the PC value (word value) of the next instruction and the value with rel code-extended (word value).

■ BC (Branch relative if C=1)/BLO (Branch if LOwer)

Operation

(C) = 0: (PC) ← (PC) + 2 (Word addition)

(C) = 1: (PC) ← (PC) + 2 + rel (Word addition)

Assembler format

BC rel/BLO rel

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

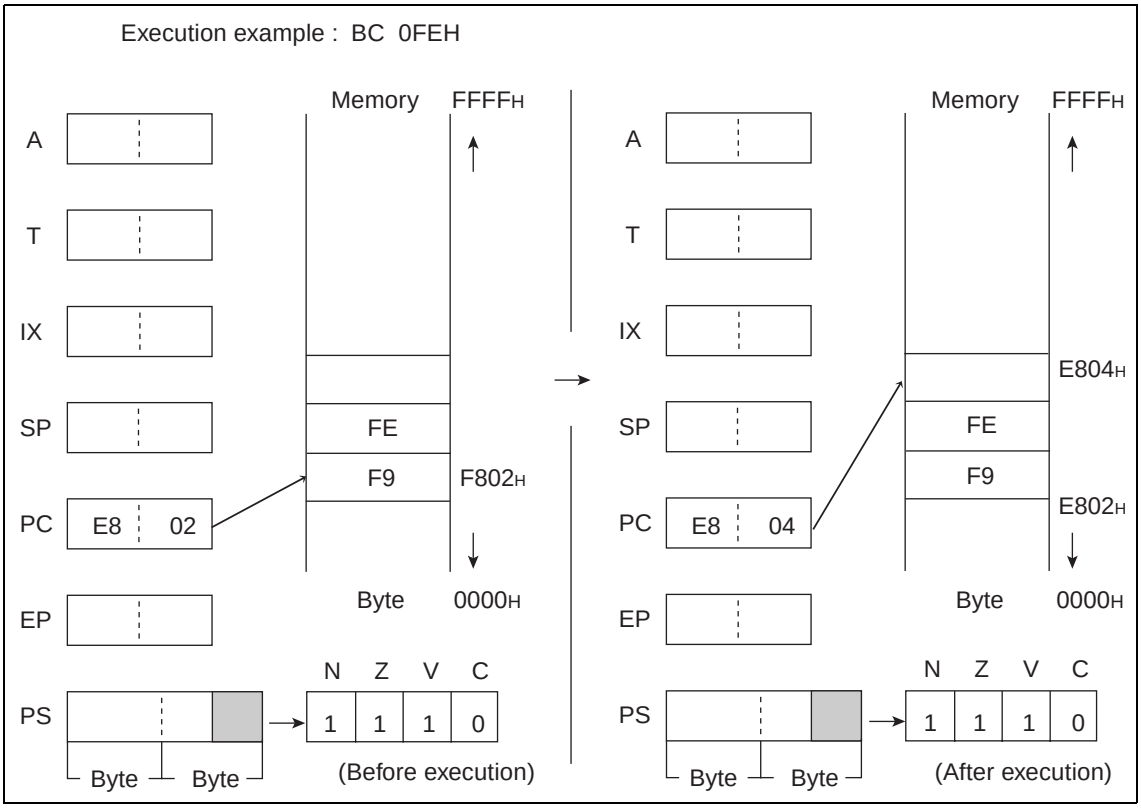
V: Not changed

C: Not changed

Number of execution cycles: 4 (at divergence)/ 2 (at non-divergence)

Byte count: 2

OP code: F9



6.10 BGE (Branch Great or Equal: relative if larger than or equal to Zero)

Execute the next instruction if the logical exclusive-OR for the V and N flags is 1 and the branch if it is 0. Branch address corresponds to the value of addition between the PC value (word value) of the next instruction and the value with rel code-extended (word value).

■ BGE (Branch Great or Equal: relative if larger than or equal to Zero)

Operation

(V) ∀ (N) = 1: (PC) ← (PC) + 2 (Word addition)

(V) ∀ (N) = 0: (PC) ← (PC) + 2 + rel (Word addition)

Assembler format

BGE rel

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

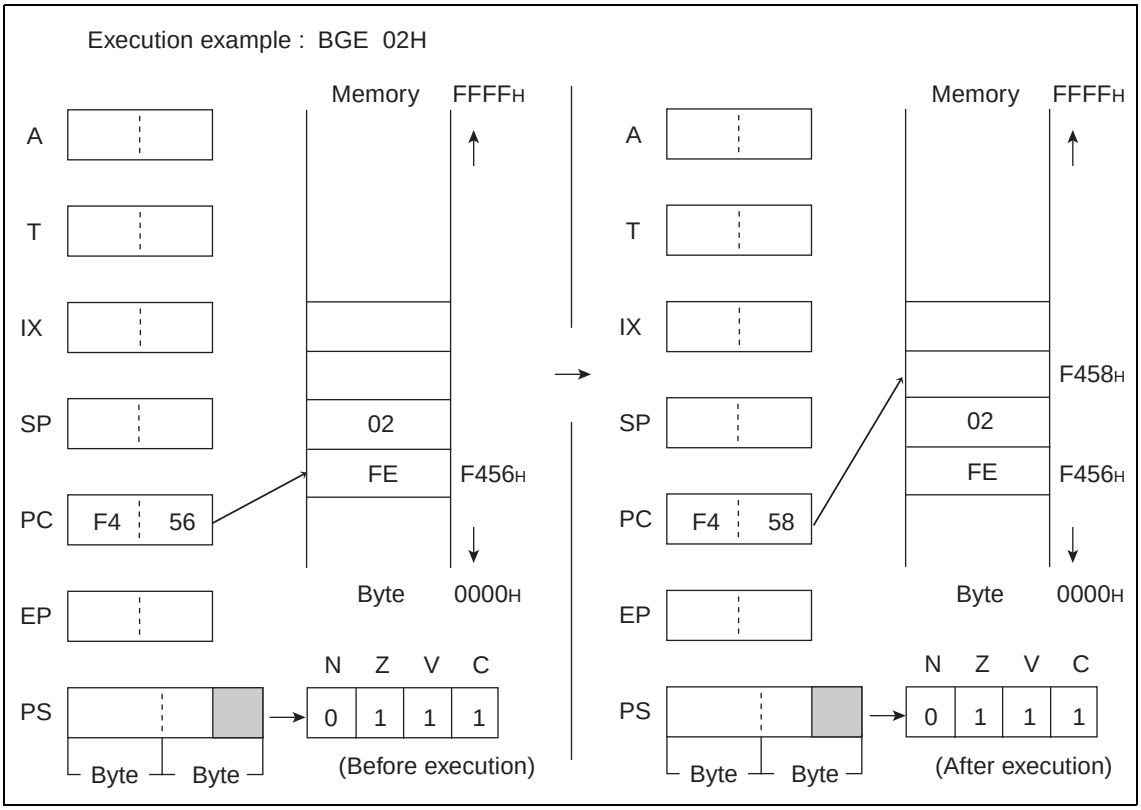
V: Not changed

C: Not changed

Number of execution cycles: 4 (at divergence)/ 2 (at non-divergence)

Byte count: 2

OP code: FE



6.11 BLT (Branch Less Than zero: relative if < Zero)

Execute the next instruction if the logical exclusive-OR for the V and N flags is 0 and the branch if it is 1. Branch address corresponds to the value of addition between the PC value (word value) of the next instruction and the value with rel code-extended (word value).

■ BLT (Branch Less Than zero: relative if < Zero)

Operation

(V) ∀ (N) = 0: (PC) ← (PC) + 2 (Word addition)

(V) ∀ (N) = 1: (PC) ← (PC) + 2 + rel (Word addition)

Assembler format

BLT rel

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

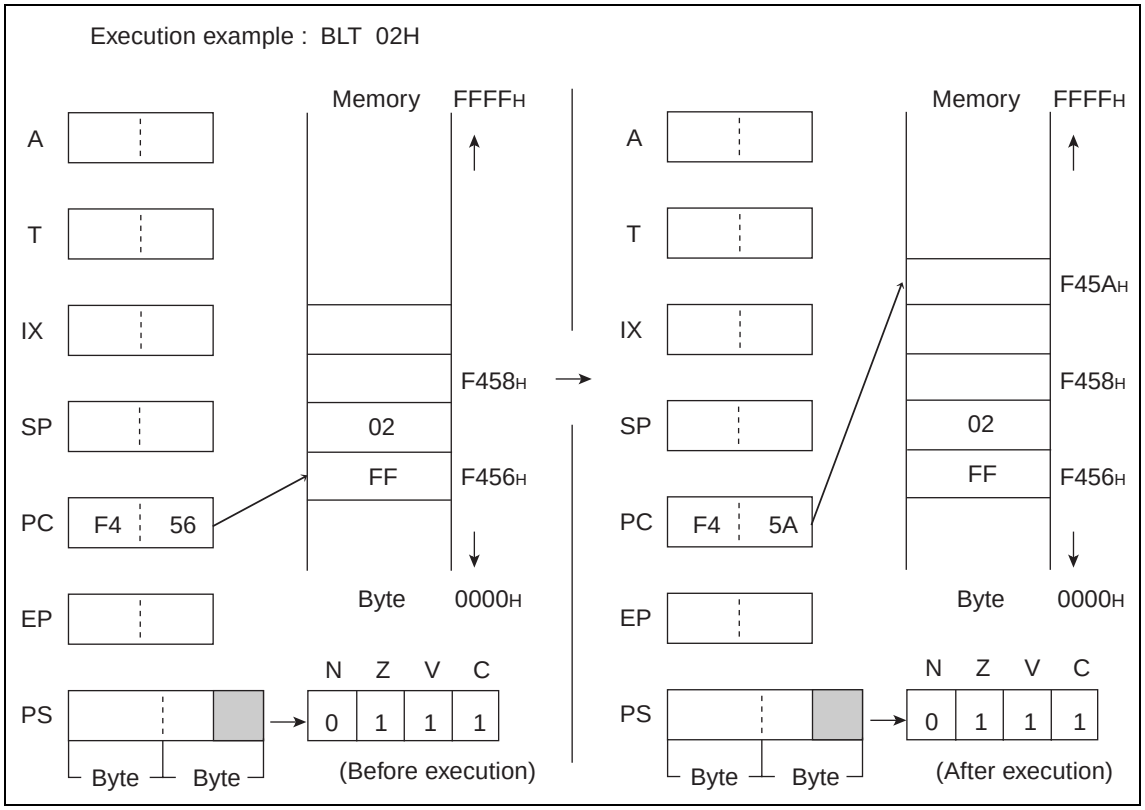
V: Not changed

C: Not changed

Number of execution cycles: 4 (at divergence)/ 2 (at non-divergence)

Byte count: 2

OP code: FF



6.12 BN (Branch relative if N = 1)

Execute the next instruction if the N-flag is 0 and the branch if it is 1. Branch address corresponds to the value of addition between the PC value (word value) of the next instruction and the value with rel code-extended (word value).

■ BN (Branch relative if N = 1)

Operation

N = 0: (PC) ← (PC) + 2 (Word addition)

N = 1: (PC) ← (PC) + 2 + rel (Word addition)

Assembler format

BN rel

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

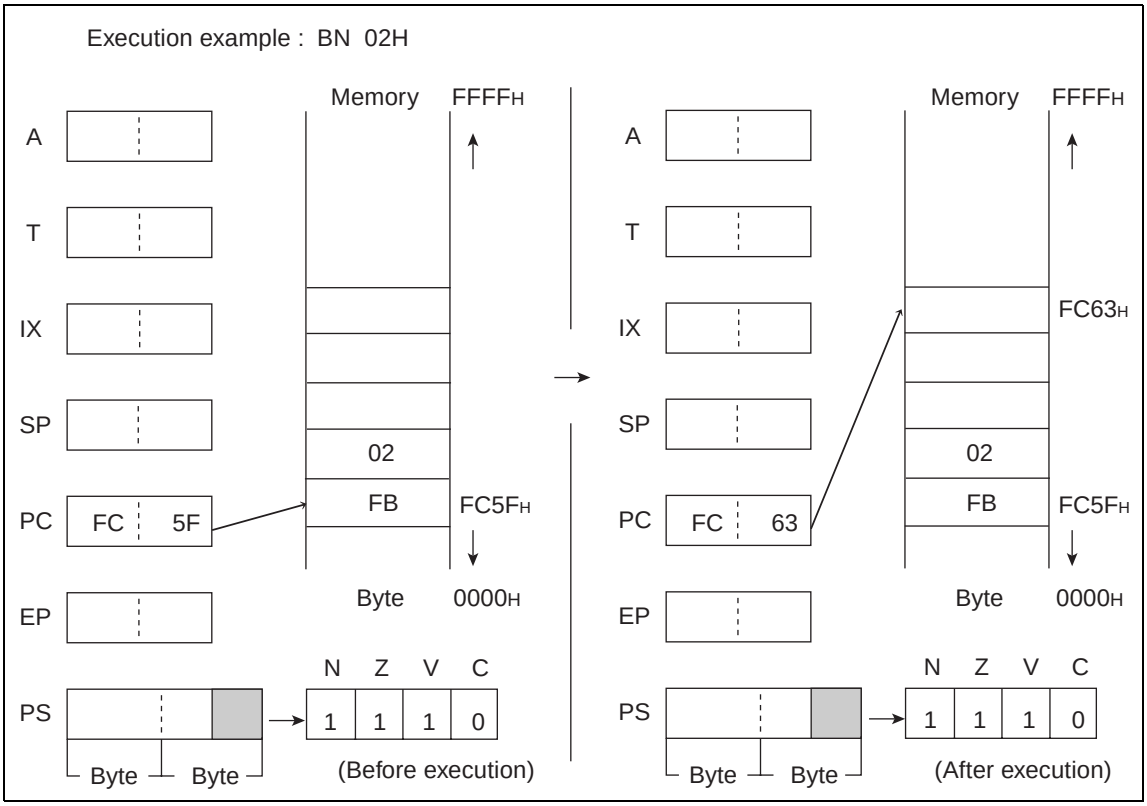
V: Not changed

C: Not changed

Number of execution cycles: 4 (at divergence)/ 2 (at non-divergence)

Byte count: 2

OP code: FB



6.13 BNZ (Branch relative if Z = 0)/BNE (Branch if Not Equal)

Execute the next instruction if the Z-flag is 1 and the branch if it is 0. Branch address corresponds to the value of addition between the PC value (word value) of the next instruction and the value with rel code-extended (word value).

■ BNZ (Branch relative if Z = 0)/BNE (Branch if Not Equal)

Operation

(Z) = 1: (PC) ← (PC) + 2 (Word addition)

(Z) = 0: (PC) ← (PC) + 2 + rel (Word addition)

Assembler format

BNZ rel/BNE rel

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

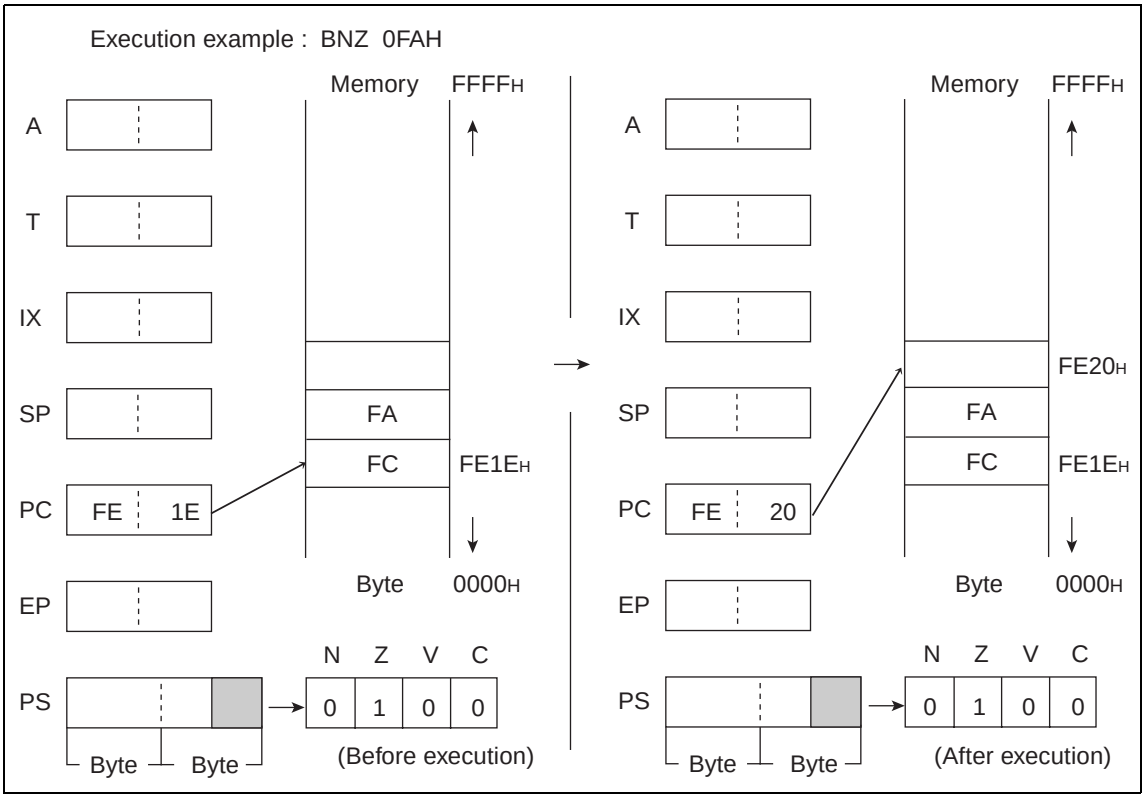
V: Not changed

C: Not changed

Number of execution cycles: 4 (at divergence)/ 2 (at non-divergence)

Byte count: 2

OP code: FC



6.14 BNC (Branch relative if C = 0)/BHS (Branch if Higher or Same)

Execute the next instruction if the C-flag is 1 and the branch if it is 0 . Branch address corresponds to the value of addition between the PC value (word value) of the next instruction and the value with rel code-extended (word value).

■ BNC (Branch relative if C = 0)/BHS (Branch if Higher or Same)

Operation

(C) = 1: (PC) ← (PC) + 2 (Word addition)

(C) = 0: (PC) ← (PC) + 2 + rel (Word addition)

Assembler format

BNC rel/BHS rel

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

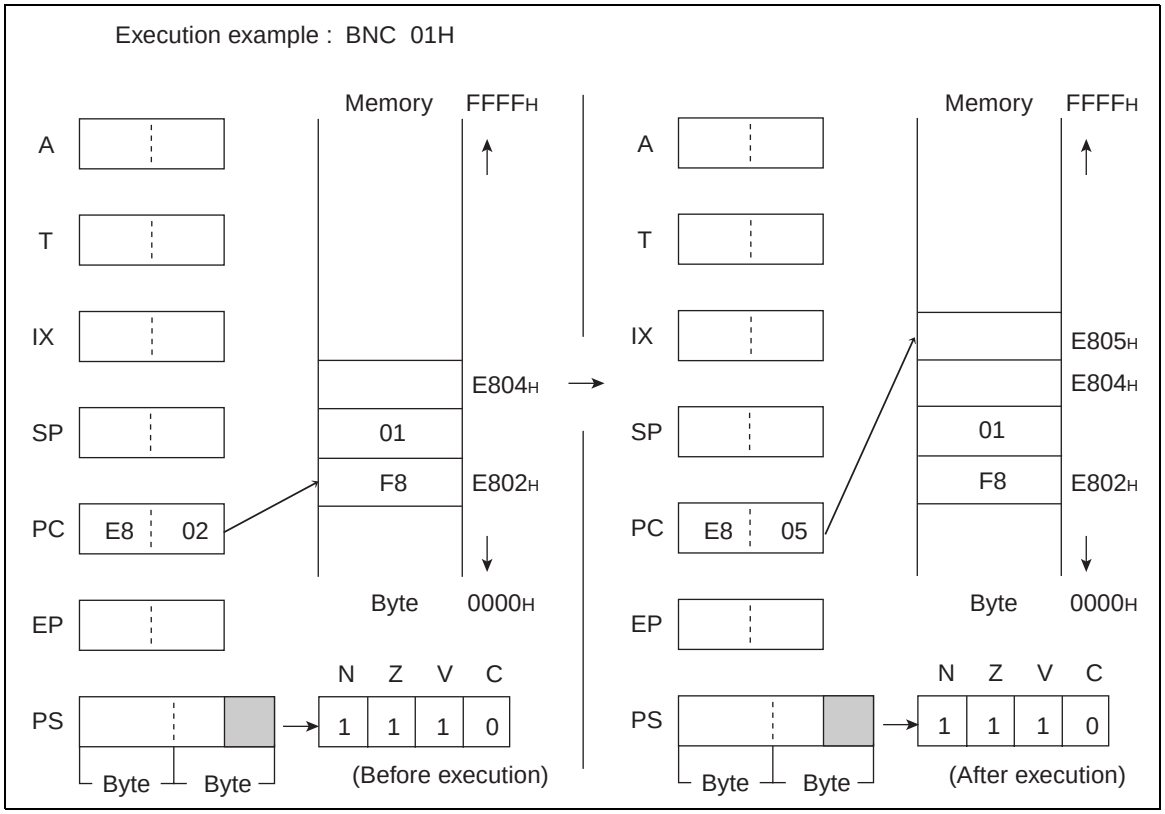
V: Not changed

C: Not changed

Number of execution cycles: 4 (at divergence)/ 2 (at non-divergence)

Byte count: 2

OP code: F8



6.15 BP (Branch relative if N = 0: PLUS)

Execute the next instruction if the N-flag is 1 and the branch if it is 0 . Branch address corresponds to the value of addition between the PC value (word value) of the next instruction and the value with rel code-extended (word value).

■ BP (Branch relative if N = 0: PLUS)

Operation

(N) = 1: (PC) ← (PC) + 2 (Word addition)

(N) = 1: (PC) ← (PC) + 2 + rel (Word addition)

Assembler format

BP rel

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

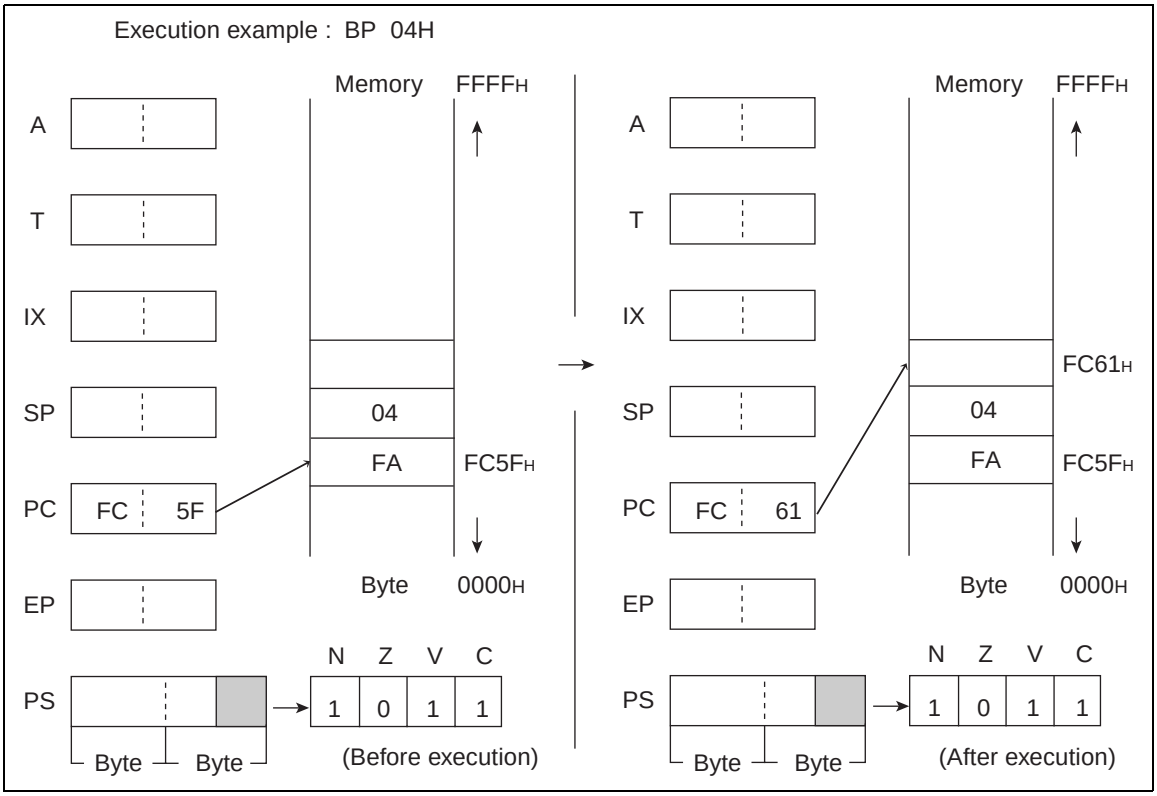
V: Not changed

C: Not changed

Number of execution cycles: 4 (at divergence)/ 2 (at non-divergence)

Byte count: 2

OP code: FA



6.16 BZ (Branch relative if Z = 1)/BEQ (Branch if Equal)

Execute the next instruction if the Z-flag is 0 and the branch if it is 1 . Branch address corresponds to the value of addition between the PC value (word value) of the next instruction and the value with rel code-extended (word value).

■ BZ (Branch relative if Z = 1)/BEQ (Branch if Equal)

Operation

(Z) = 0: (PC) ← (PC) + 2 (Word addition)

(Z) = 1: (PC) ← (PC) + 2 + rel (Word addition)

Assembler format

BZ rel/BEQ rel

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

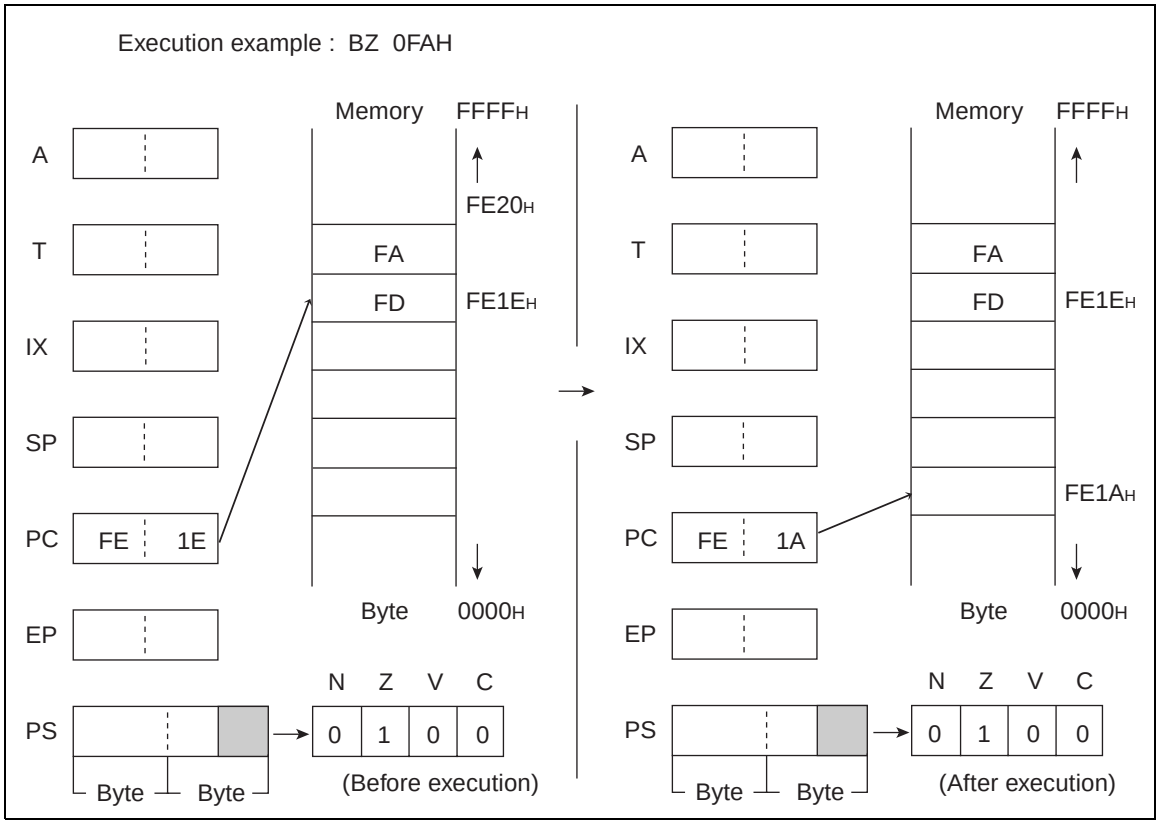
V: Not changed

C: Not changed

Number of execution cycles: 4 (at divergence)/ 2 (at non-divergence)

Byte count: 2

OP code: FD



6.17 CALL (CALL subroutine)

Branch to address of ext. Return to the instruction next to this one by using the RET instruction of the branch subroutine.

■ CALL (CALL subroutine)

Operation

(SP) ← (SP) - 2 (Word subtraction), ((SP)) ← (PC) (Word transfer)

(PC) ← ext

Assembler format

CALL ext

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

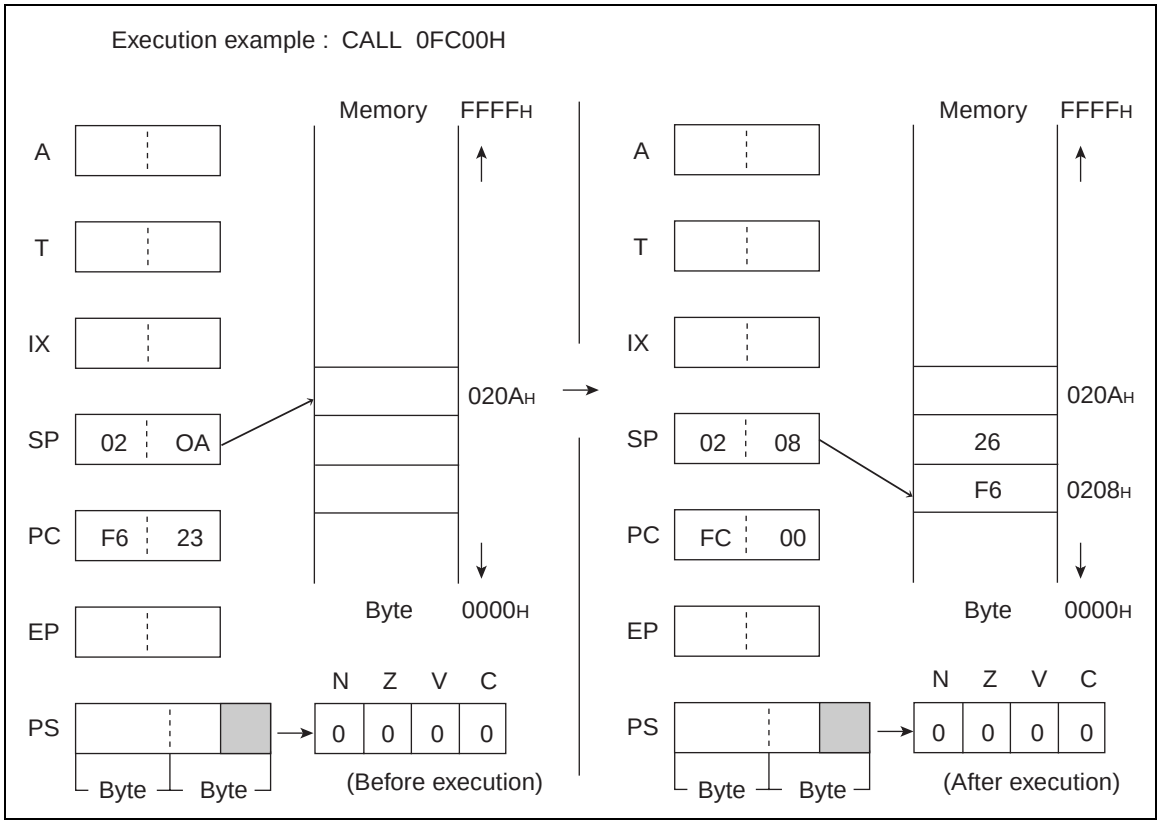
V: Not changed

C: Not changed

Number of execution cycles: 6

Byte count: 3

OP code: 31



6.18 CALLV (CALL Vectored subroutine)

Branch to the vector address (VA) of vct. Return to the instruction next to this one by using the RET instruction of the branch subroutine. The vector address (VA) indicated by VCT is shown on the next page.

■ CALLV (CALL Vectored subroutine)

Operation

(SP) ← (SP) - 2 (Word subtraction), ((SP)) ← (PC) (Word transfer)

(PC) ← (VA)

Assembler format

CALLV #vct

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

V: Not changed

C: Not changed

Number of execution cycles: 7

Byte count: 1

OP code: E8 to EF

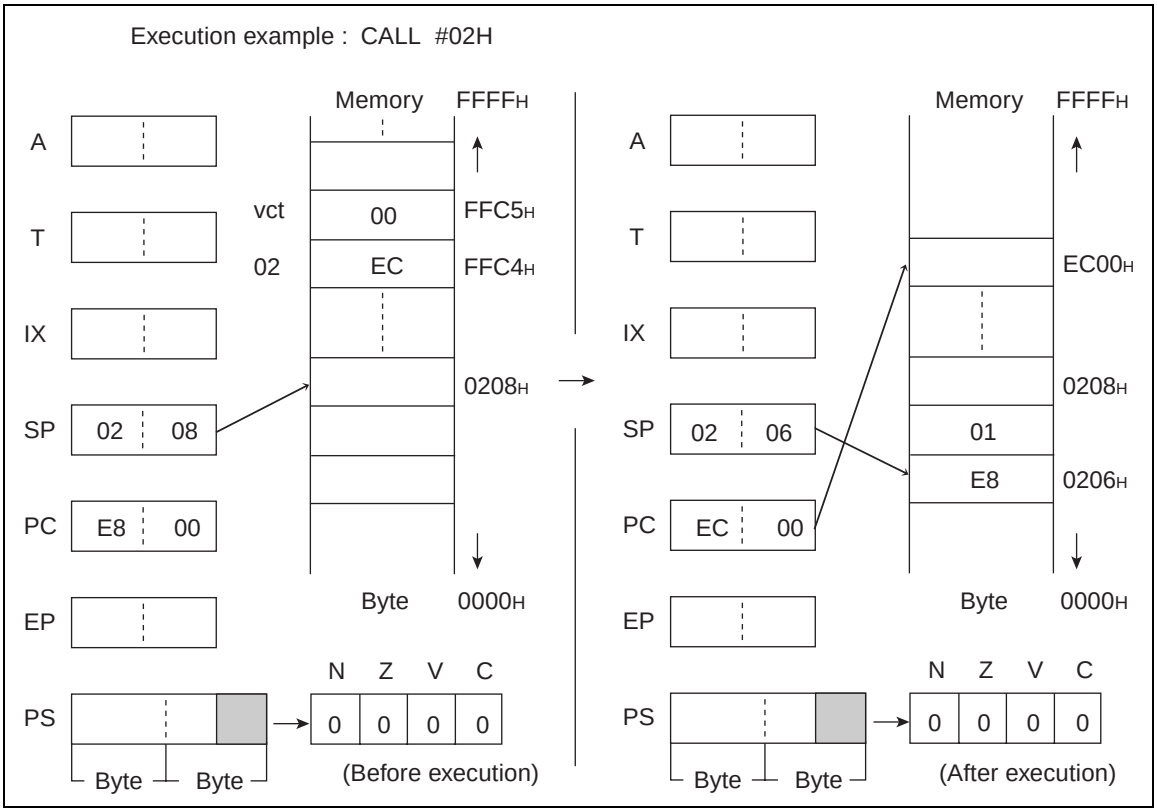


Table 6.18-1 Call Storage Address of Vector Call Instruction

Vector address (VA)		Instruction
Lower address	Upper address	
$FFCE_H$	$FFCF_H$	CALL#7
$FFCC_H$	$FFCD_H$	CALL#6
$FFCA_H$	$FFCB_H$	CALL#5
$FFC8_H$	$FFC9_H$	CALL#4
$FFC6_H$	$FFC7_H$	CALL#3
$FFC4_H$	$FFC5_H$	CALL#2
$FFC2_H$	$FFC3_H$	CALL#1
$FFC0_H$	$FFC1_H$	CALL#0

6.19 CLRB (Clear direct Memory Bit)

Set the contents of 1 bit (indicated by 3 lower bits (b) of mnemonic) of the direct area to 0.

■ CLRB (Clear direct Memory Bit)

Operation

(dir:b) ← 0

Assembler format

CLRB dir:b

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

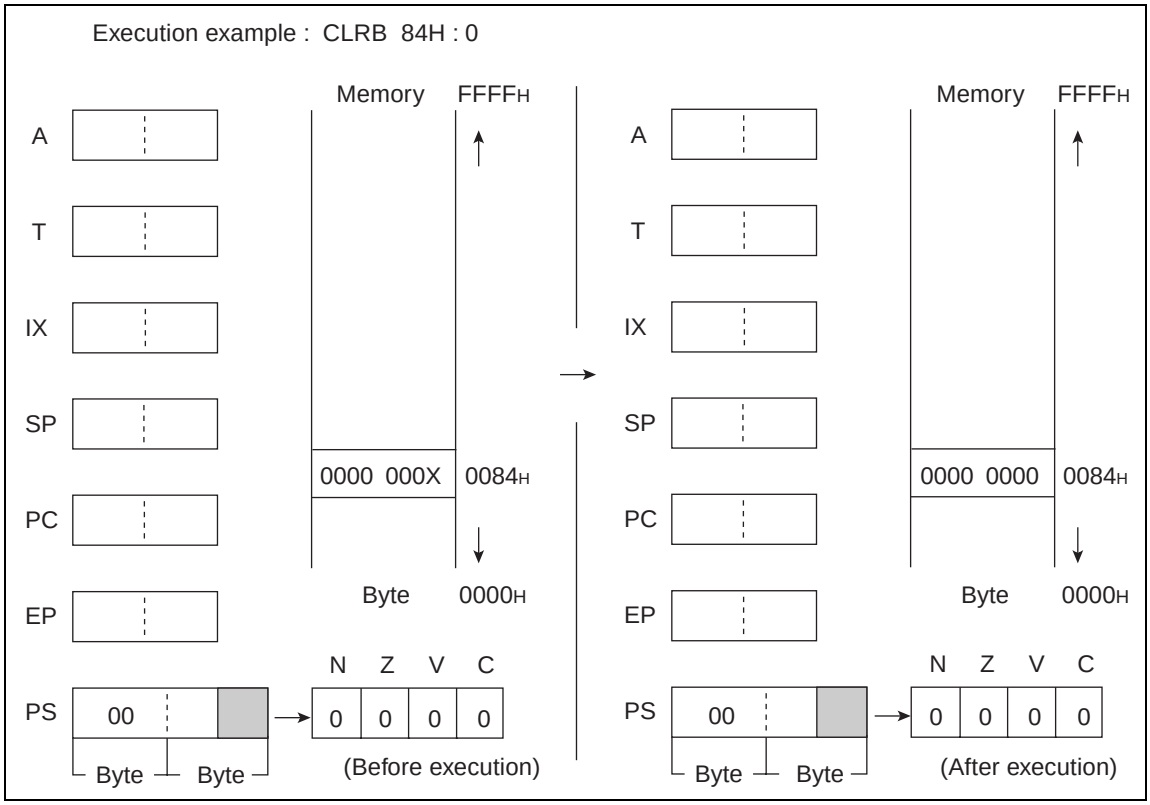
V: Not changed

C: Not changed

Number of execution cycles: 4

Byte count: 2

OP code: A0 to A7



6.20 CLRC (Clear Carry flag)

Set the C-flag to 0.

■ CLRC (Clear Carry flag)

Operation

(C) ← 0

Assembler format

CLRC

Condition code (CCR)

N	Z	V	C
-	-	-	R

+: Changed by executing instruction

-: Not changed

R: Set to 0 by executing instruction

N: Not changed

Z: Not changed

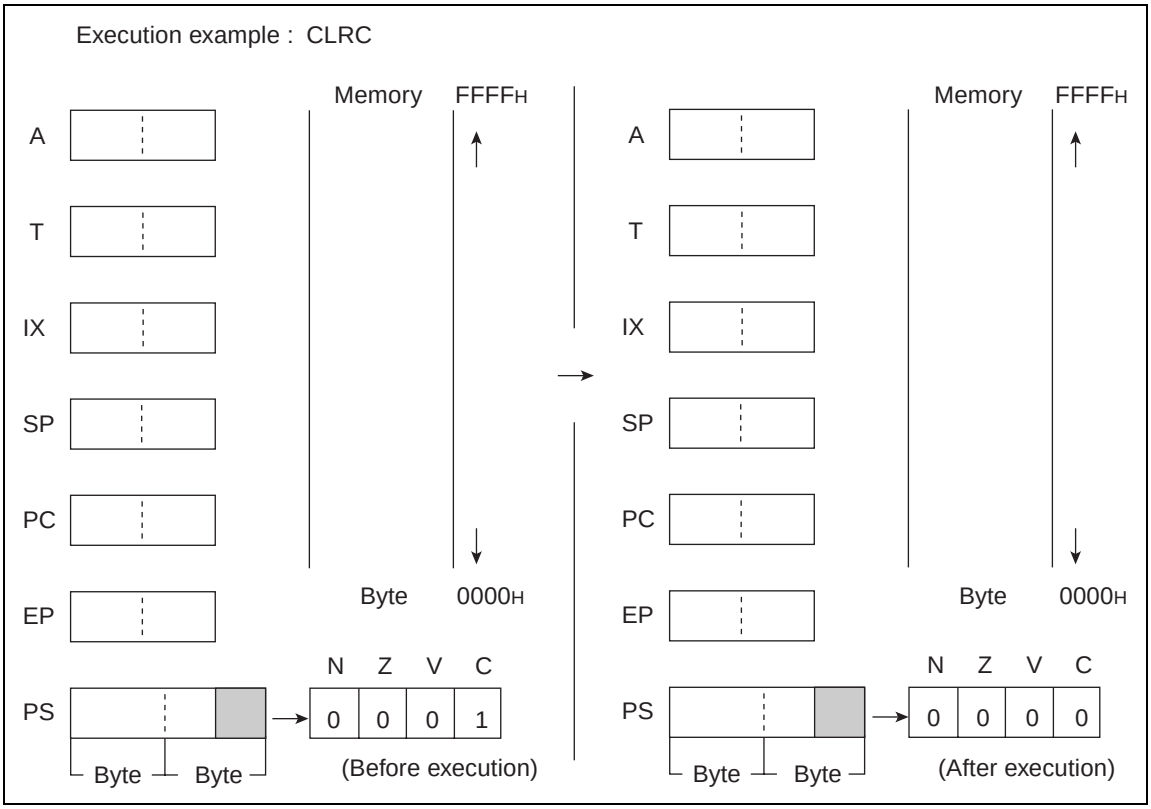
V: Not changed

C: Set to 0.

Number of execution cycle: 1

Byte count: 1

OP code: 81



6.21 CLRI (CLeaR Interrupt flag)

Set the I-flag to 0.

■ CLRI (CLeaR Interrupt flag)

Operation

(I) ← 0

Assembler format

CLRI

Condition code (CCR)

I	N	Z	V	C
R	-	-	-	-

+: Changed by executing instruction

-: Not changed

R: Set to 0 by executing instruction

I: Set to 0

N: Not changed

Z: Not changed

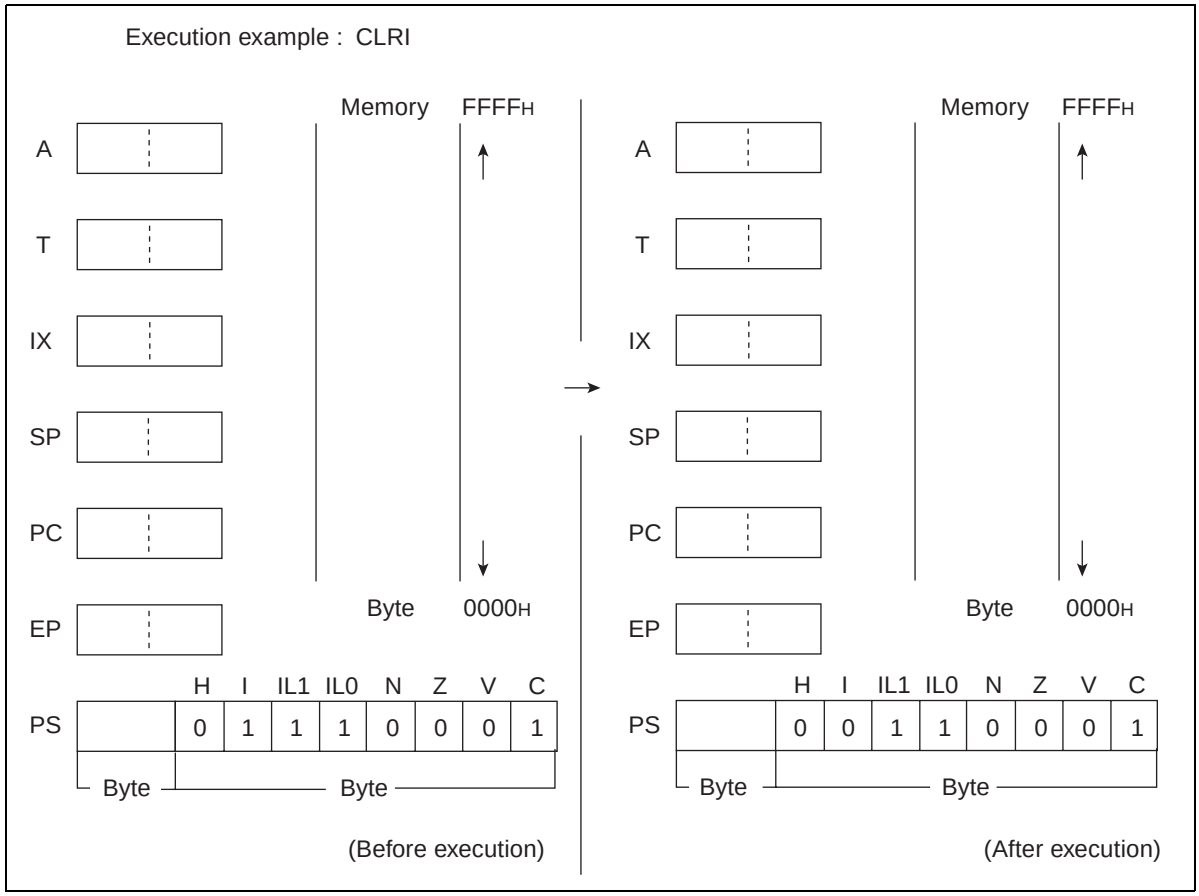
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: 80



6.22 CMP (CoMPare Byte Data of Accumulator and Temporary Accumulator)

Compare the byte data of AL with that of TL and set the results to CCR. AL and TL are not changed.

■ CMP (CoMPare Byte Data of Accumulator and Temporary Accumulator)

Operation

(TL) - (AL)

Assembler format

CMP A

Condition code (CCR)

N	Z	V	C
+	+	+	+

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

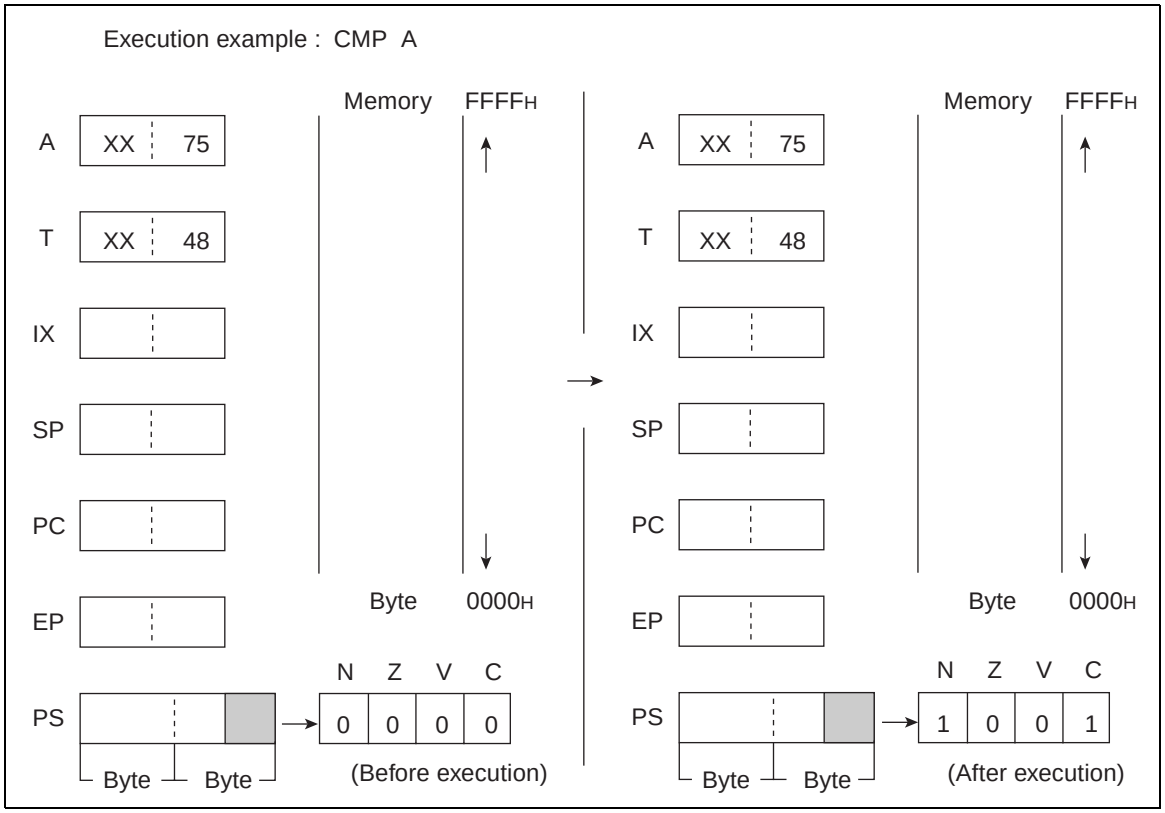
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

C: Set to 1 if a carry occurs as the result of operation and set to 0 in other cases.

Number of execution cycle: 1

Byte count: 1

OP code: 12



6.23 CMP (CoMPare Byte Data of Accumulator and Memory)

Compare the byte data of AL with that of the EA memory (memory expressed in each type of addressing) and set the results to CCR. AL and EA memory are not changed.

■ CMP (CoMPare Byte Data of Accumulator and Memory)

Operation

(AL) - (EA)

Assembler format

CMP A, EA

Condition code (CCR)

N	Z	V	C
+	+	+	+

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB is 1 as the result of operation and set to 0 in other cases.

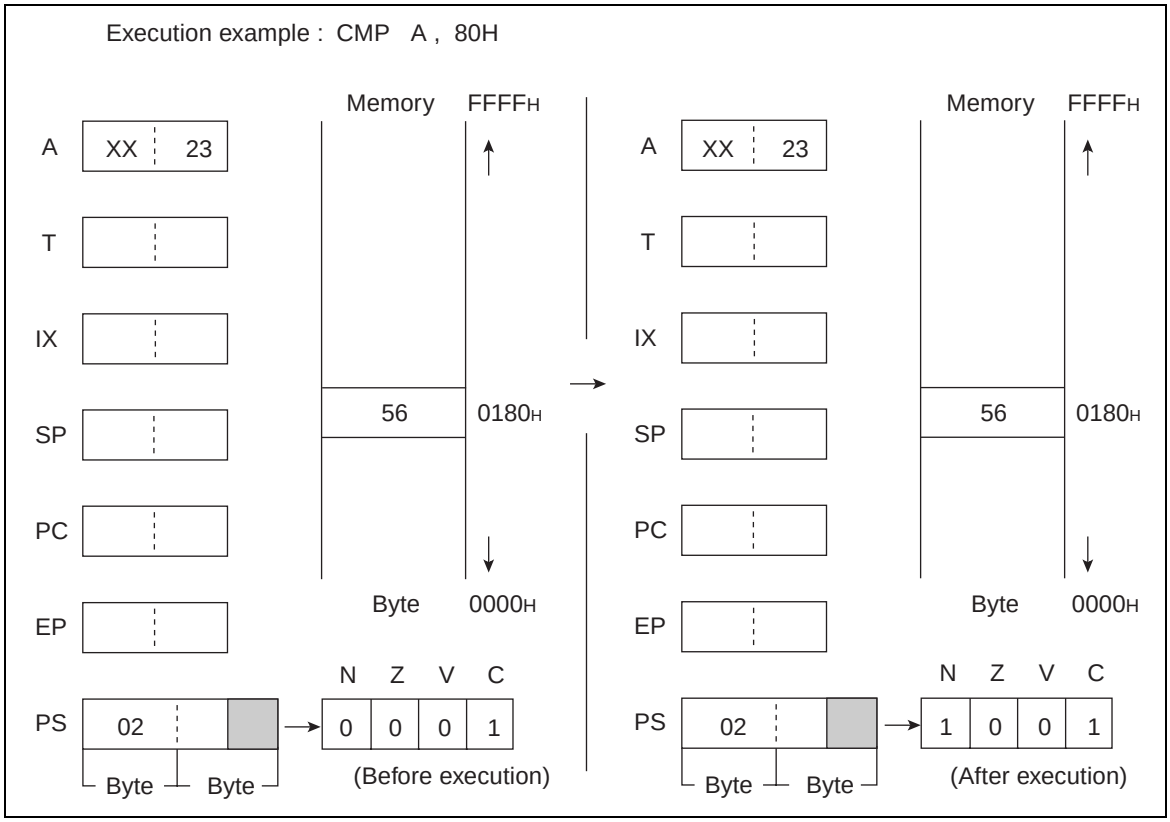
Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

C: Set to 1 if a carry occurs as the result of operation and set to 0 in other cases.

Table 6.23-1 Number of Execution Cycles / Byte Count / OP Code

EA	#d8	dir	@IX+off	@EP	Ri
Number of execution cycles	2	3	3	2	2
Byte count	2	2	2	1	1
OP code	14	15	16	17	18 to 1F



6.24 CMP (CoMPare Byte Data of Immediate Data and Memory)

Compare the byte data of EA memory (memory expressed in each type of addressing) with the immediate data and set the results to CCR. EA memory is not changed.

■ CMP (CoMPare Byte Data of Immediate Data and Memory)

Operation

(EA) - d8

Assembler format

CMP EA, #d8

Condition code (CCR)

N	Z	V	C
+	+	+	+

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB is 1 as the result of operation and set to 0 in other cases.

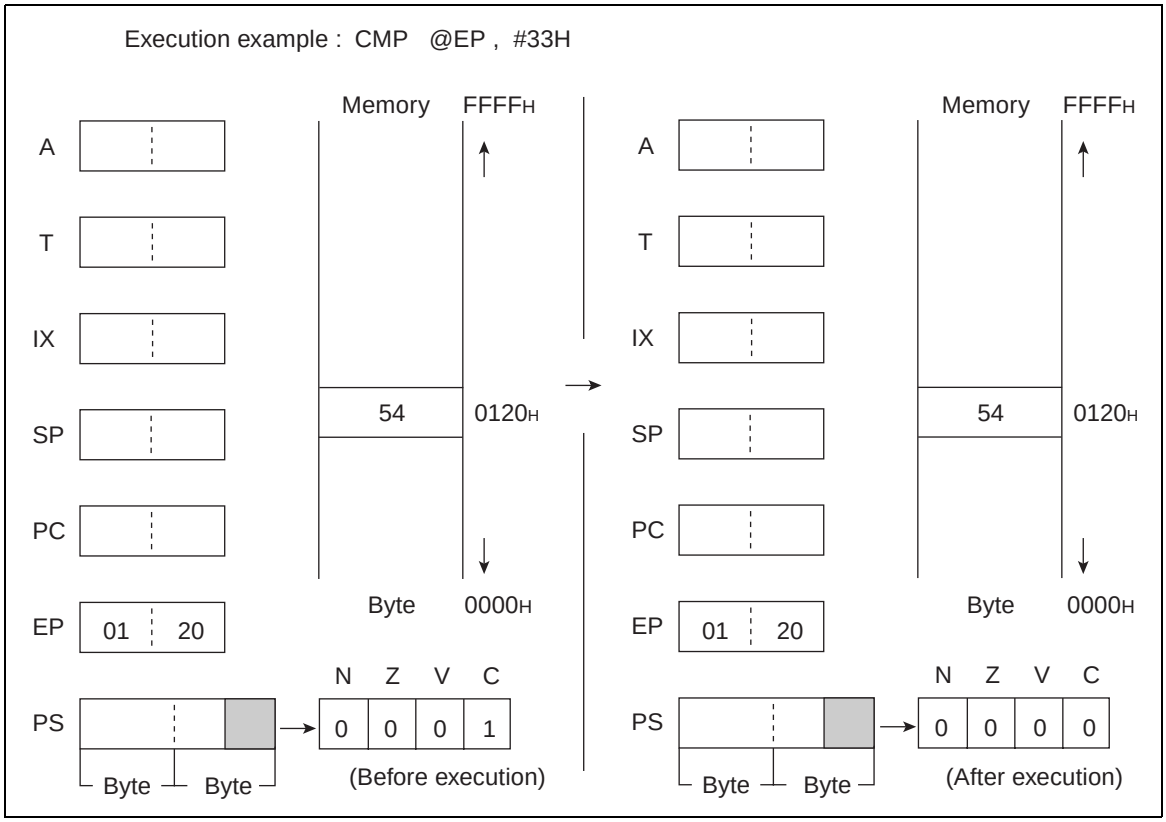
Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

C: Set to 1 if a carry occurs as the result of operation and set to 0 in other cases.

Table 6.24-1 Number of Execution Cycles / Byte Count / OP Code

EA	dir	@IX+off	@EP	Ri
Number of execution cycles	4	4	3	3
Byte count	3	3	2	2
OP code	95	96	97	98 to 9F



6.25 CMPW (CoMPare Word Data of Accumulator and Temporary Accumulator)

Compare the word data of A with that of T and set the results to CCR. A and T are not changed.

■ CMPW (CoMPare Word Data of Accumulator and Temporary Accumulator)

Operation

(T) - (A)

Assembler format

CMPW A

Condition code (CCR)

N	Z	V	C
+	+	+	+

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 0000_H and set to 0 in other cases.

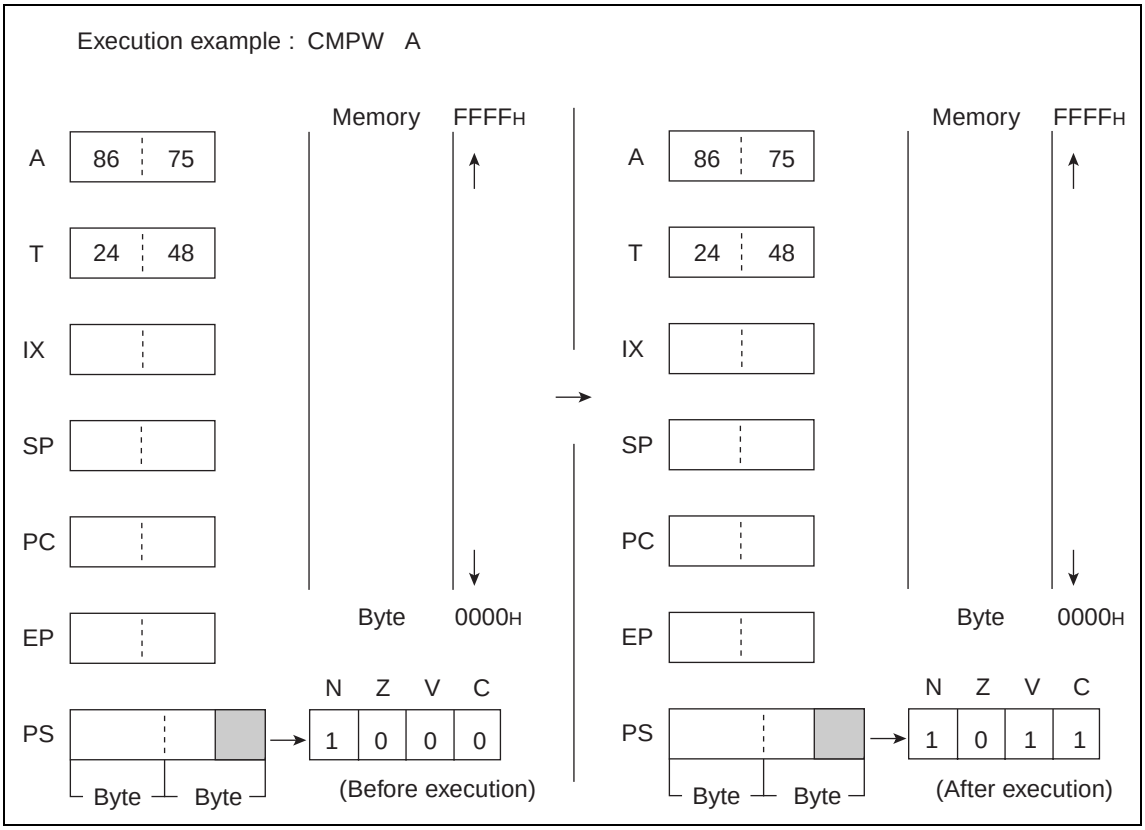
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

C: Set to 1 if a carry occurs as the result of operation and set to 0 in other cases.

Number of execution cycles: 2

Byte count: 1

OP code: 13



6.26 DAA (Decimal Adjust for Addition)

When adding the correction value to AL by the state in the carry before execution of instruction and half-carry, decimal operation is corrected.

■ DAA (Decimal Adjust for Addition)

Operation

(AL) ← (AL) + 6 or 60H or 66H

(Add a correction value shown in the next page to AL and the value of AL according to the state of the C or H-flag.)

Assembler format

DAA

Condition code (CCR)

N	Z	V	C
+	+	+	+

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

C: Change as indicated on the next page.

Number of execution cycle: 1

Byte count: 1

OP code: 84

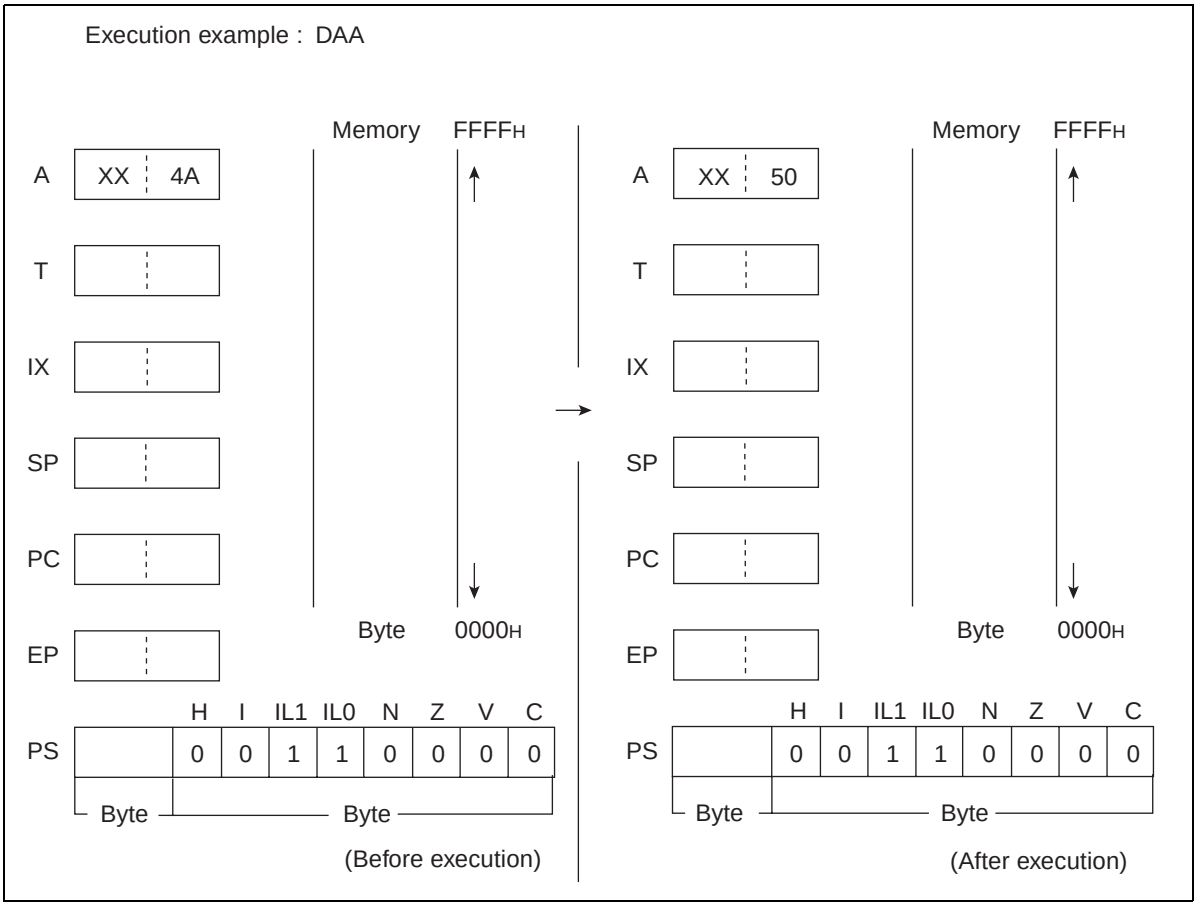


Table 6.26-1 Decimal Adjustment Table (DAA)

C-flag	AL (bit7 to bit4)	H-flag	AL (bit3 to bit0)	Correction value	C-flag after execution
0	0 to 9	0	0 to 9	00	0
0	0 to 8	0	A to F	06	0
0	0 to 9	1	0 to 3	06	0
0	A to F	0	0 to 9	60	1
0	9 to F	0	A to F	66	1
0	A to F	1	0 to 3	66	1
1	0 to 2	0	0 to 9	60	1
1	0 to 2	0	A to F	66	1
1	0 to 3	1	0 to 3	66	1

Table 6.26-2 Execution Example

Mnemonic	AL	C	H
MOV A, #75H	75	0	×
ADDC A, #25H	9A	0	0
DAA	00	1	0

6.27 DAS (Decimal Adjust for Subtraction)

Subtract the correction value from AL according to the state of the C or H-flag before executing instruction.

■ DAS (Decimal Adjust for Subtraction)

Operation

(AL) ← (AL) - 6 or 60H or 66H

(Subtract a correction value shown in the next page to AL and the value of AL according to the state of the C or H-flag.)

Assembler format

DAS

Condition code (CCR)

N	Z	V	C
+	+	+	+

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

C: Change as indicated on the next page.

Number of execution cycle: 1

Byte count: 1

OP code: 94

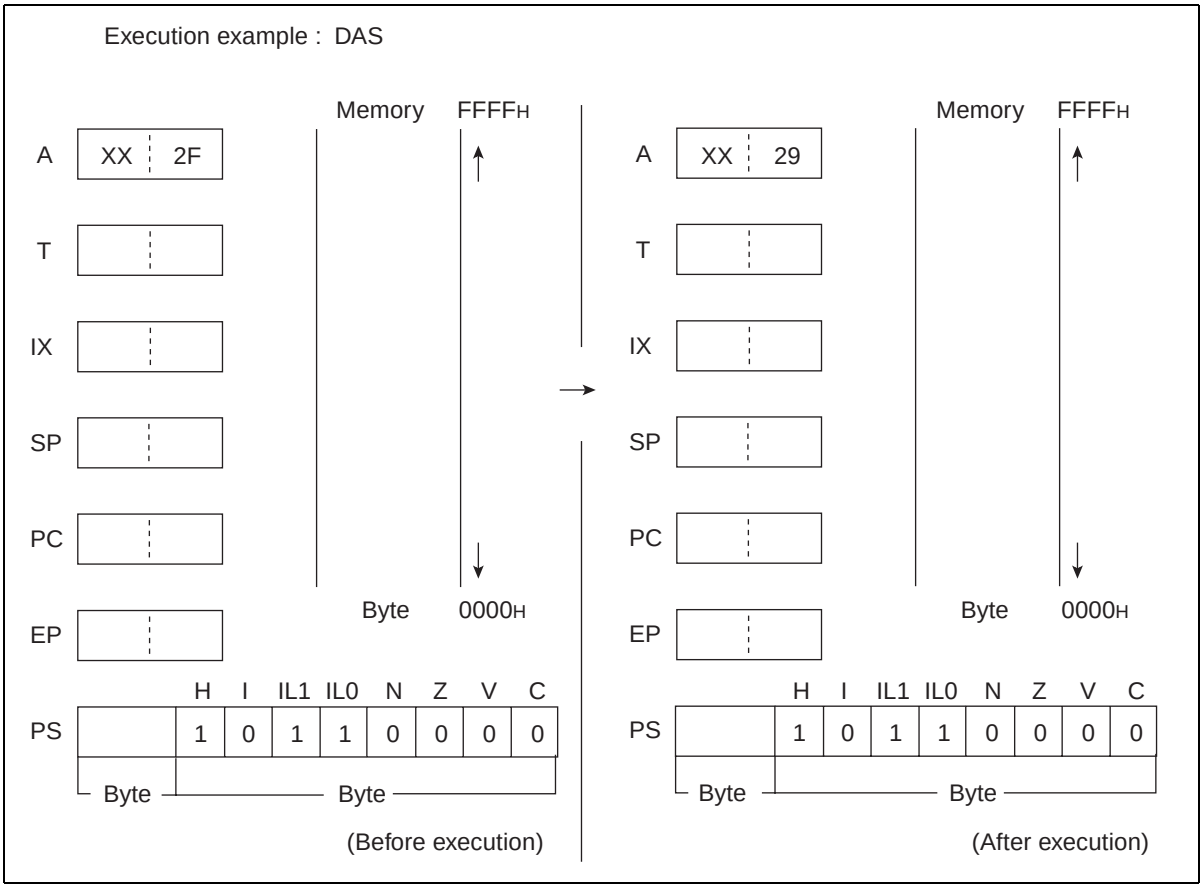


Table 6.27-1 Decimal Adjustment Table (DAS)

C-flag	H-flag	Correction value	C-flag after execution
0	0	00	0
1	1	66	1
0	1	06	0
1	0	60	1

Table 6.27-2 Execution Example

Mnemonic	AL	C	H
MOV A, #70H	70	×	×
SUBC A, #25H	4B	0	1
DAS	45	0	1

6.28 DEC (DECrement Byte Data of General-purpose Register)

Decrement byte data of Ri by one.

■ DEC (DECrement Byte Data of General-purpose Register)

Operation

(Ri) ← (Ri) - 1 (byte subtract)

Assembler format

DEC Ri

Condition code (CCR)

N	Z	V	C
+	+	+	-

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

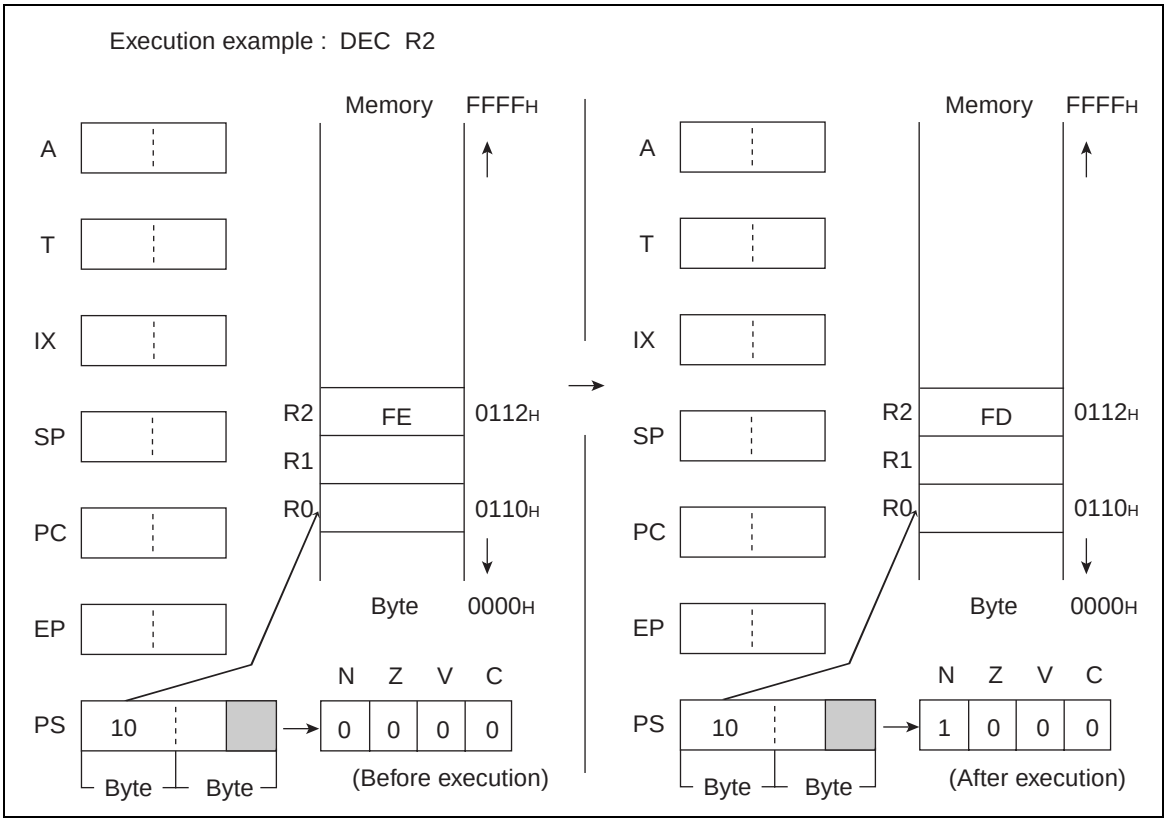
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

C: Not changed

Number of execution cycles: 3

Byte count: 1

OP code: D8 to DF



6.29 DECW (DECrement Word Data of Accumulator)

Decrement word data of A by one.

■ DECW (DECrement Word Data of Accumulator)

Operation

(A) ← (A) - 1 (Word subtraction)

Assembler format

DECW A

Condition code (CCR)

N	Z	V	C
+	+	-	-

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 0000_H and set to 0 in other cases.

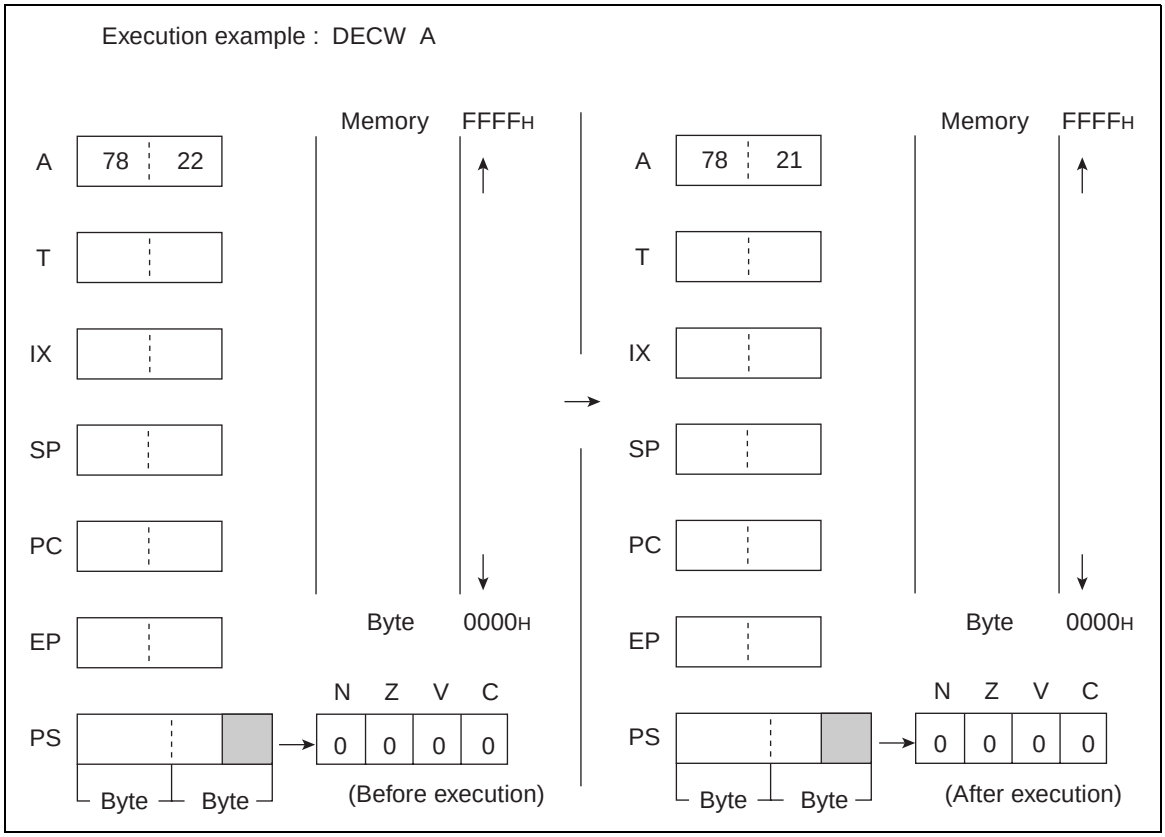
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: D0



6.30 DECW (DECrement Word Data of Extra Pointer)

Decrement word data of EP by one.

■ DECW (DECrement Word Data of Extra Pointer)

Operation

(EP) ← (EP) - 1 (Word subtraction)

Assembler format

DECW EP

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

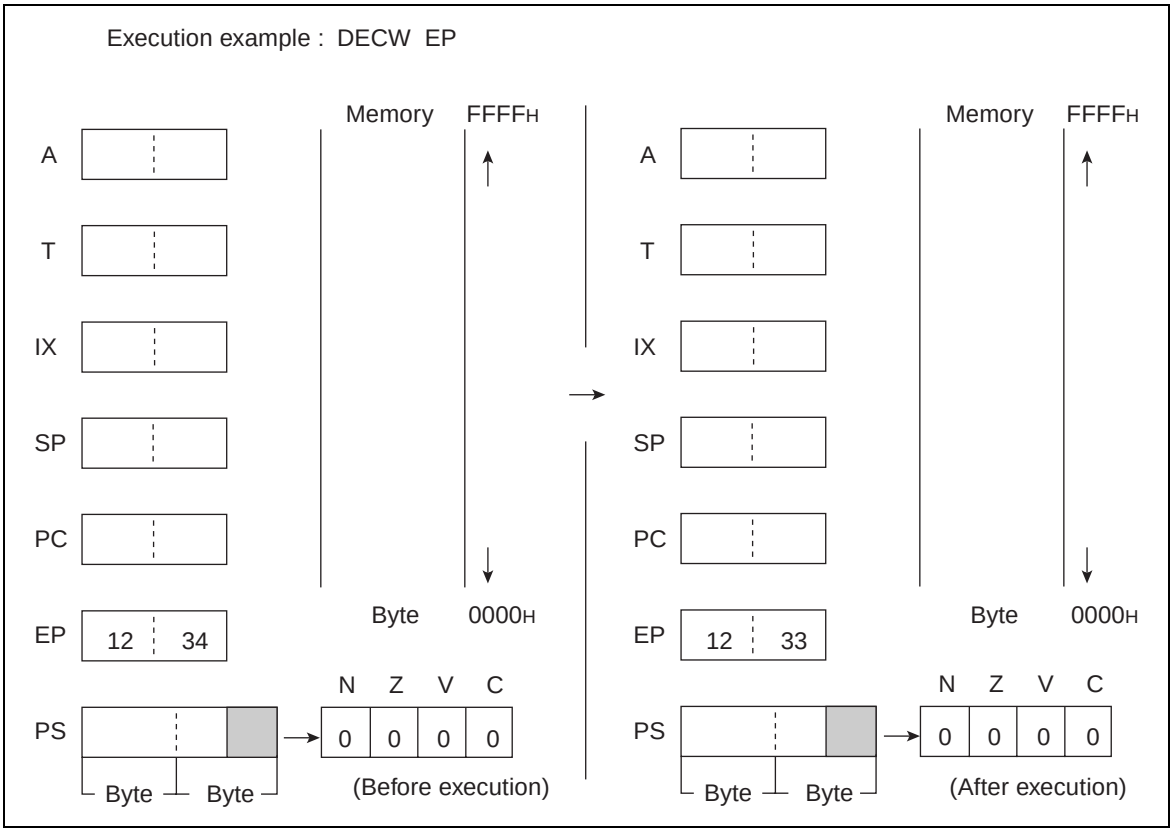
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: D3



6.31 DECW (DECrement Word Data of Index Pointer)

Decrement word data of IX by one.

■ DECW (DECrement Word Data of Index Pointer)

Operation

(IX) ← (IX) - 1 (Word subtraction)

Assembler format

DECW IX

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

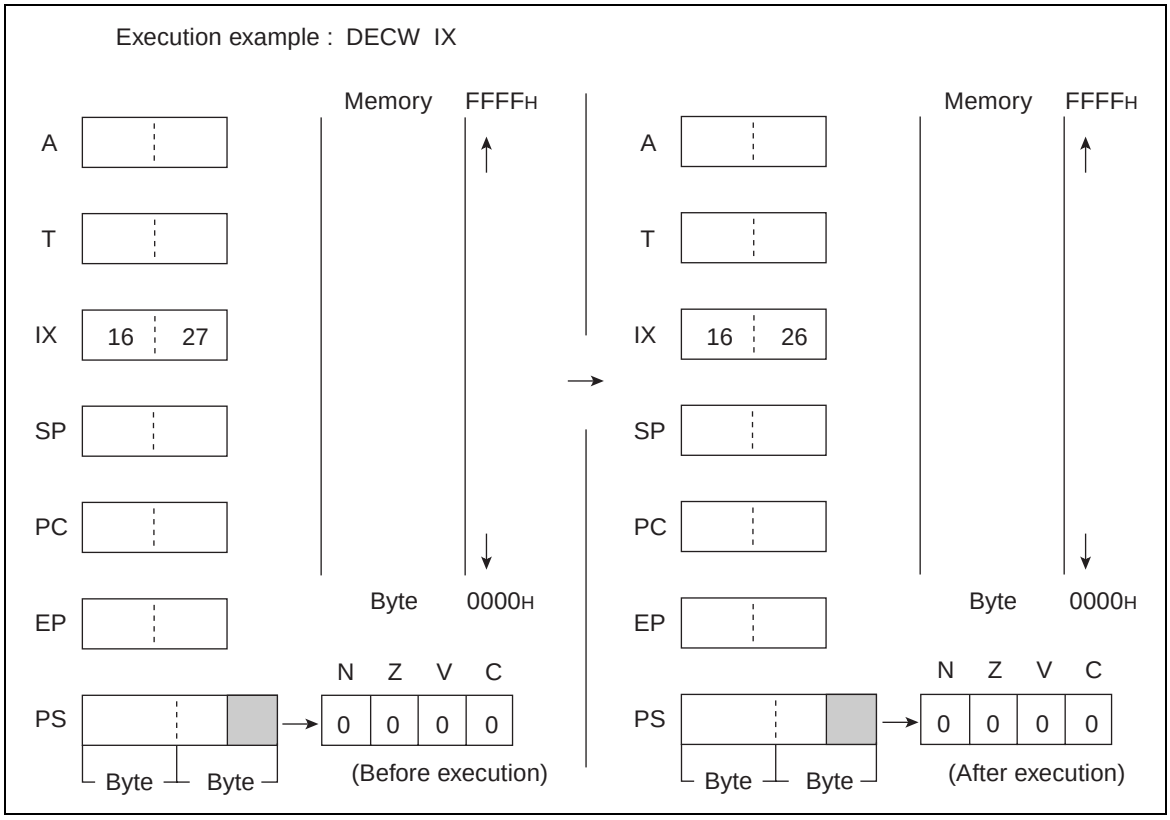
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: D2



6.32 DECW (DECrement Word Data of Stack Pointer)

Decrement word data of SP by one.

■ DECW (DECrement Word Data of Stack Pointer)

Operation

(SP) ← (SP) - 1 (Word subtraction)

Assembler format

DECW SP

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

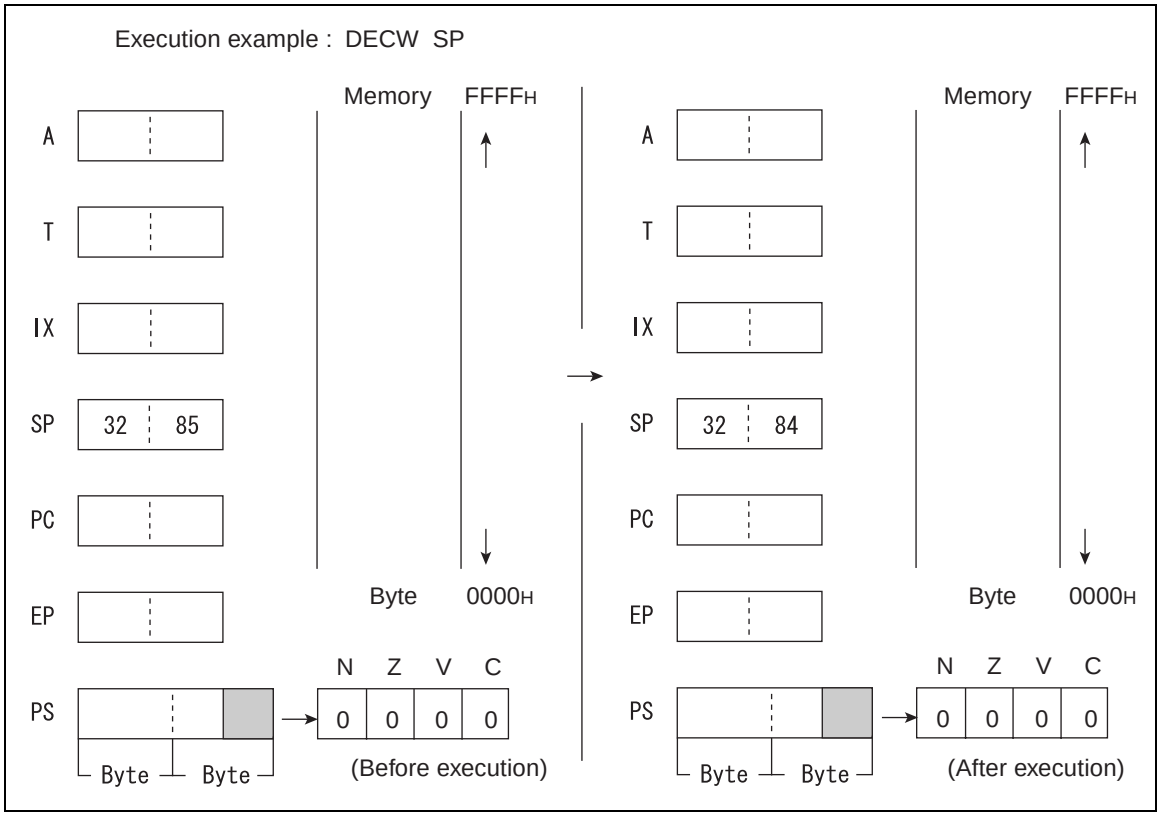
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: D1



6.33 DIVU (DIVide Unsigned)

Divide the word data of T by that of AL as an unsigned binary value. Return the quotient to A and the remainder to T.
When A is 0, the result is indefinite and Z flag is 1 to show 0 division.

■ DIVU (DIVide Unsigned)

Operation

Quotient (A) ← (T) / (A)

Remainder (T) ← (T) MOD (A)

Assembler format

DIVU A

Condition code (CCR)

N	Z	V	C
-	+	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Set to 1 if A before execution of instruction is 0000_H and set to 0 in other cases.

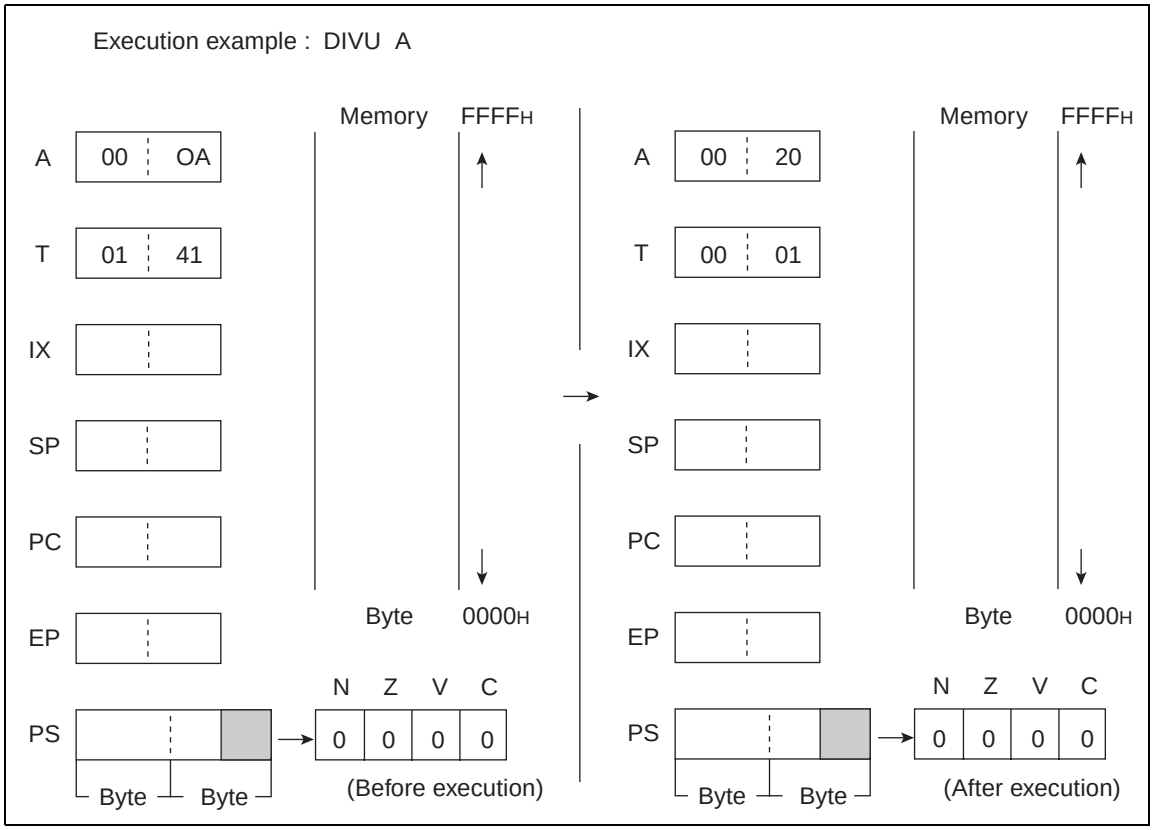
V: Not changed

C: Not changed

Number of execution cycles: 17

Byte count: 1

OP code: 11



6.34 INC (INCrement Byte Data of General-purpose Register)

Add 1 to byte data of Ri.

■ INC (INCrement Byte Data of General-purpose Register)

Operation

(Ri) ← (Ri) + 1 (Word addition)

Assembler format

INC Ri

Condition code (CCR)

N	Z	V	C
+	+	+	-

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

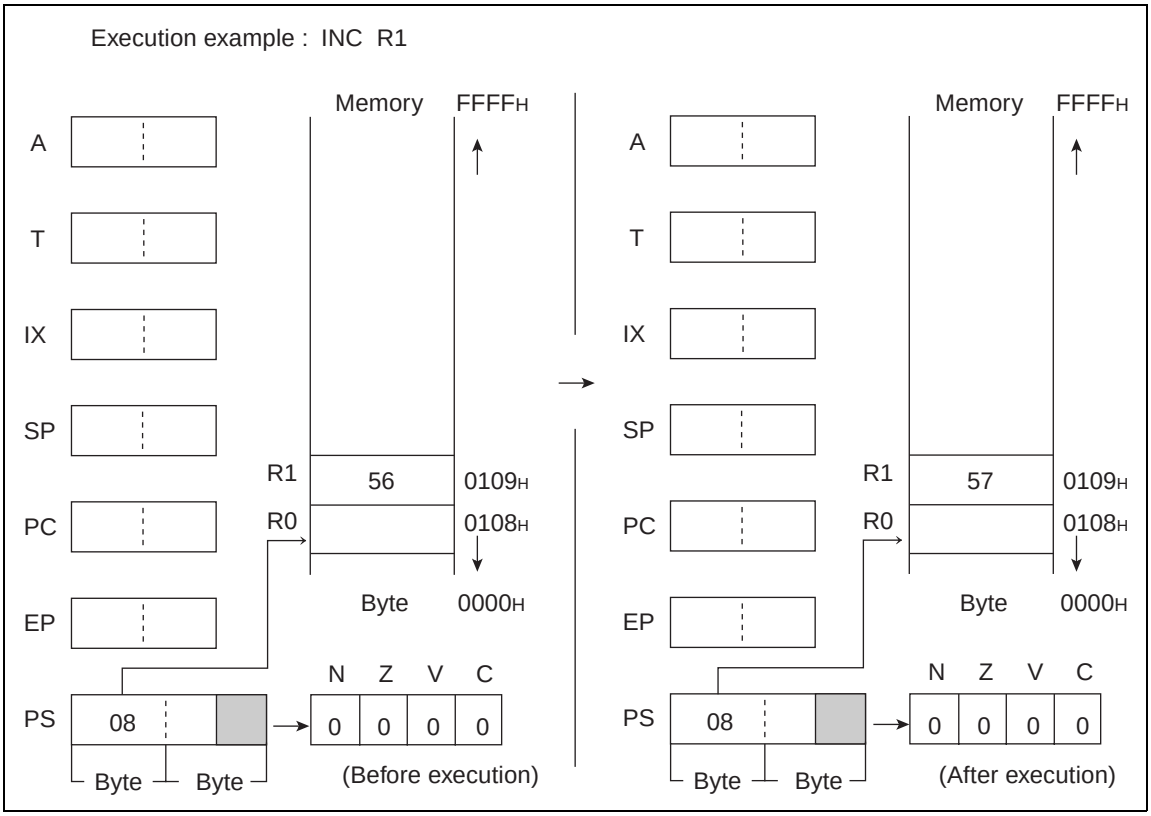
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

C: Not changed

Number of execution cycles: 3

Byte count: 1

OP code: C8 to CF



6.35 INCW (INCrement Word Data of Accumulator)

Add 1 to word data of A.

■ INCW (INCrement Word Data of Accumulator)

Operation

(A) ← (A) +1 (Word addition)

Assembler format

INCW A

Condition code (CCR)

N	Z	V	C
+	+	-	-

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB of A is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 0000_H and set to 0 in other cases.

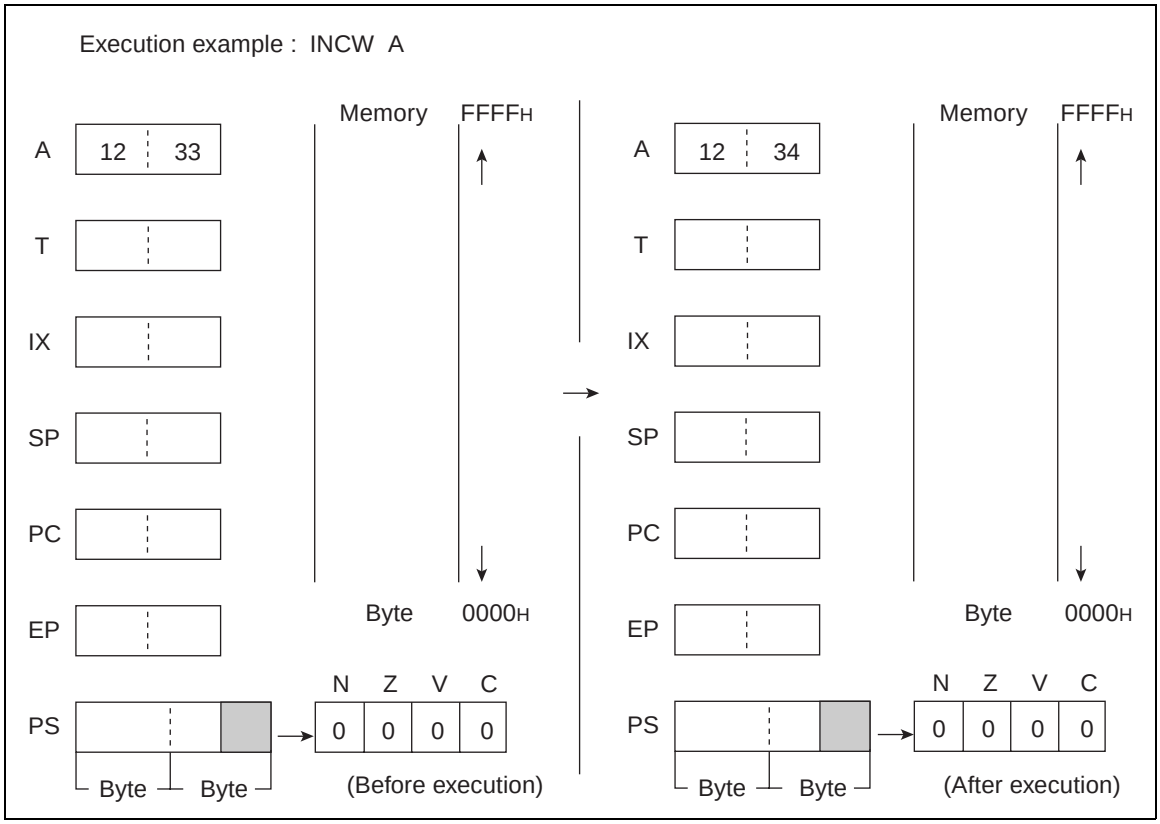
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: C0



6.36 INCW (INCrement Word Data of Extra Pointer)

Add 1 to word data of EP.

■ INCW (INCrement Word Data of Extra Pointer)

Operation

(EP) ← (EP) + 1 (Word addition)

Assembler format

INCW EP

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

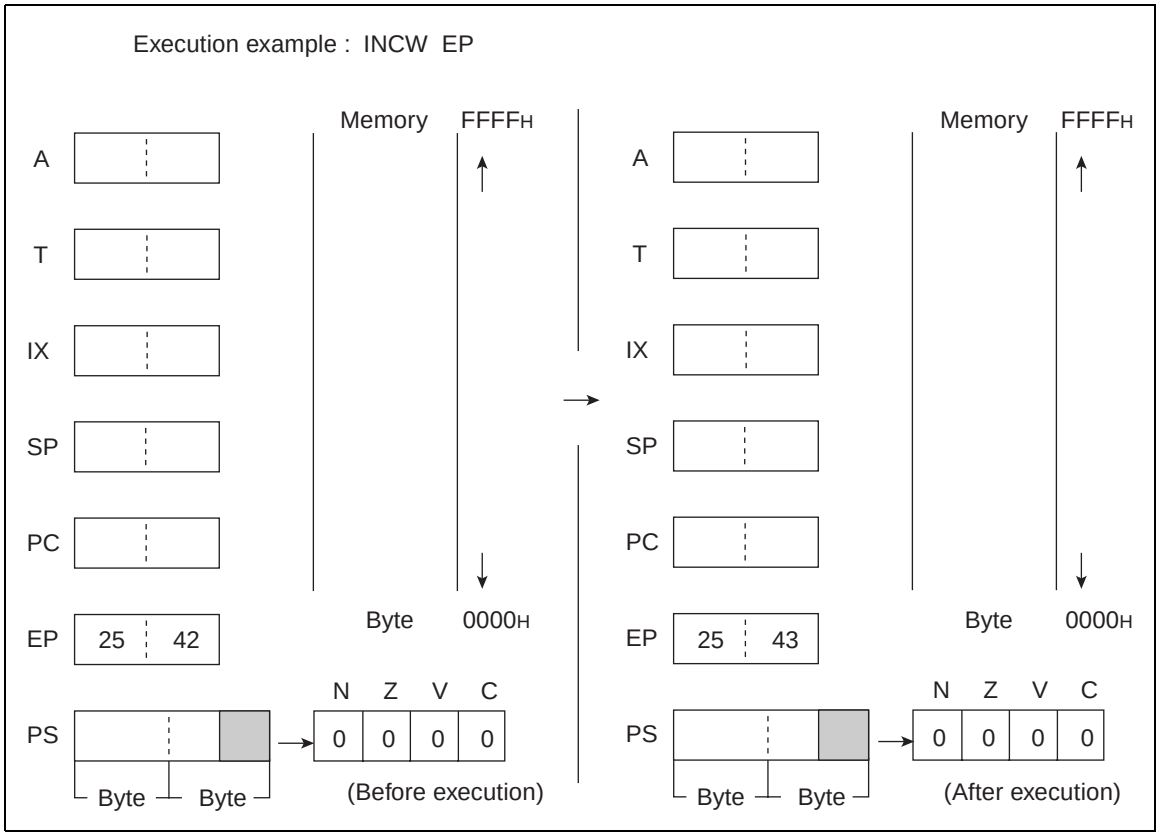
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: C3



6.37 INCW (INCrement Word Data of Index Register)

Add 1 to word data of IX.

■ INCW (INCrement Word Data of Index Register)

Operation

(IX) ← (IX) + 1 (Word addition)

Assembler format

INCW IX

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

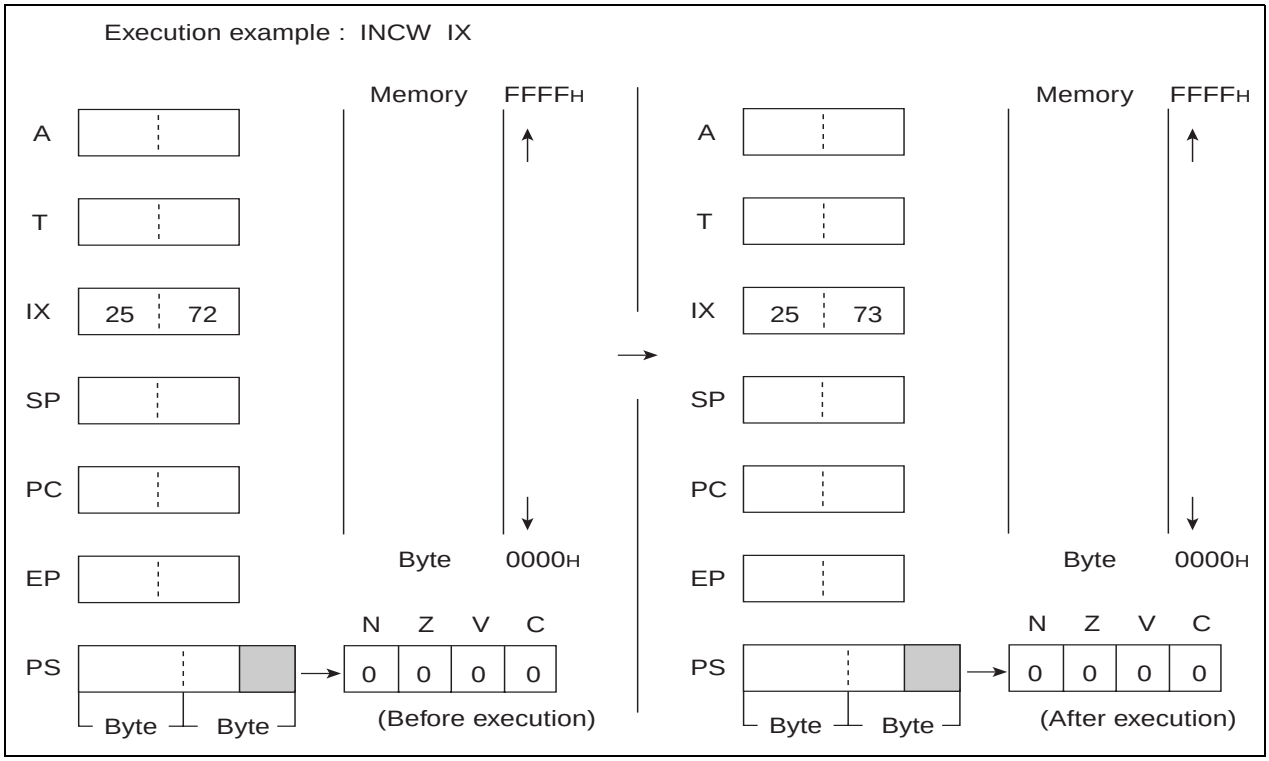
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: C2



6.38 INCW (INCrement Word Data of Stack Pointer)

Add 1 to word data of SP.

■ INCW (INCrement Word Data of Stack Pointer)

Operation

(SP) ← (SP) + 1 (Word addition)

Assembler format

INCW SP

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

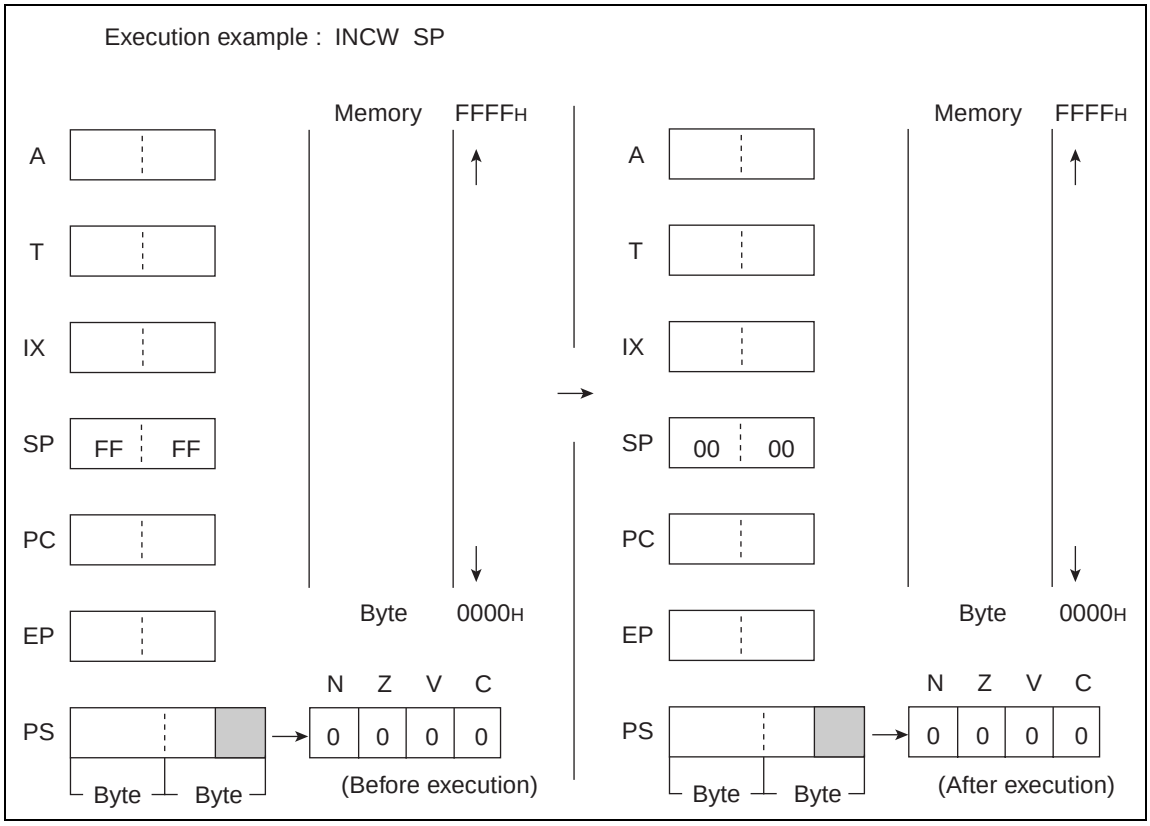
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: C1



6.39 JMP (JuMP to address pointed by Accumulator)

Transfer word data from A to PC.

■ JMP (JuMP to address pointed by Accumulator)

Operation

(PC) ← (A) (Word transfer)

Assembler format

JMP @A

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

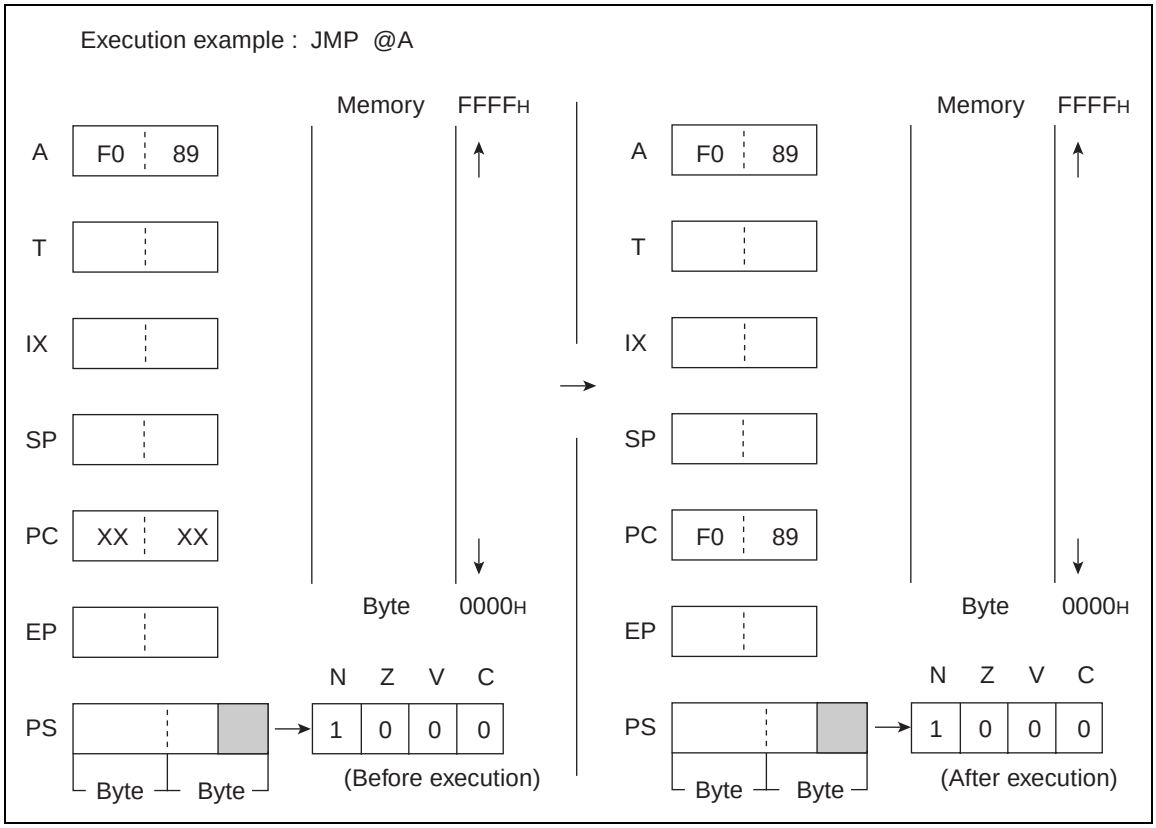
V: Not changed

C: Not changed

Number of execution cycles: 3

Byte count: 1

OP code: E0



6.40 JMP (JuMP to effective Address)

Branch to the PC value indicated by ext.

■ JMP (JuMP to effective Address)

Operation

(PC) ← ext (Word transfer)

Assembler format

JMP ext

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

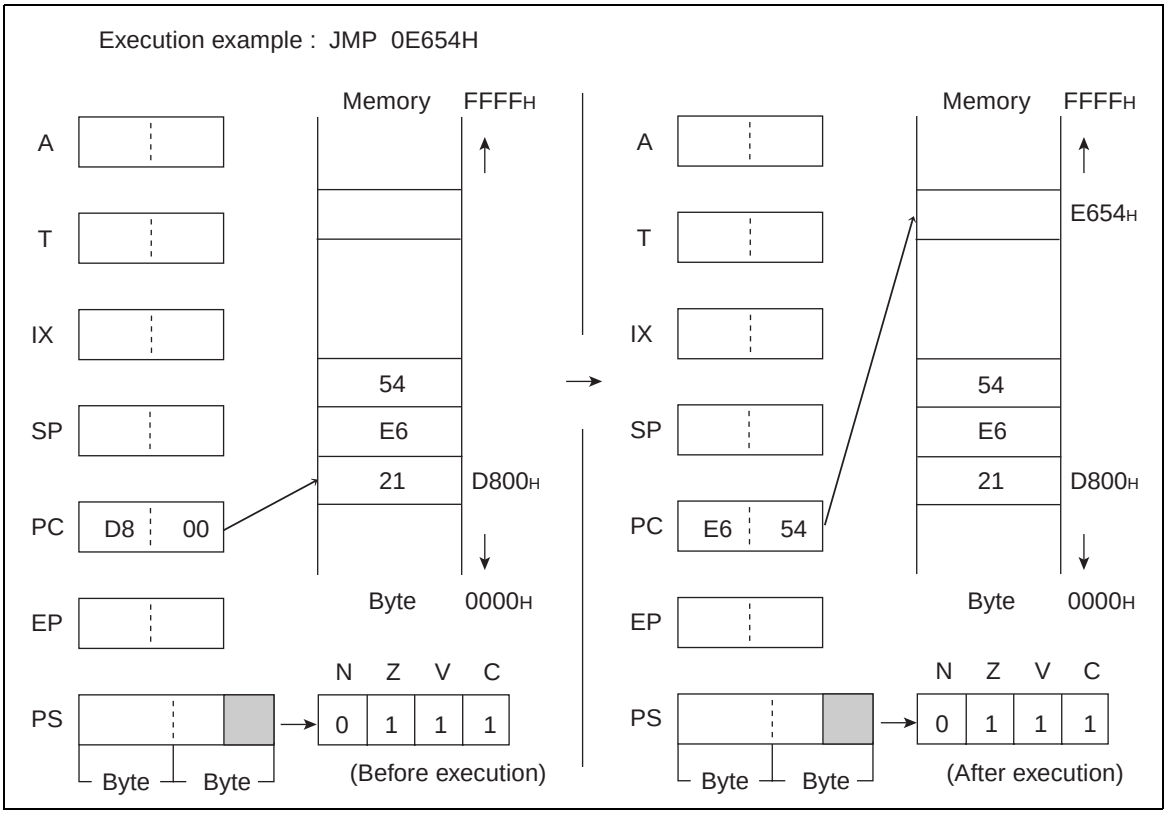
V: Not changed

C: Not changed

Number of execution cycles: 4

Byte count: 3

OP code: 21



6.41 MOV (MOVE Byte Data from Temporary Accumulator to Address Pointed by Accumulator)

Transfer byte data from T to memory indirectly addressed by A.

■ MOV (MOVE Byte Data from Temporary Accumulator to Address Pointed by Accumulator)

Operation

((A)) ← T (Word transfer)

Assembler format

MOV @A, T

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

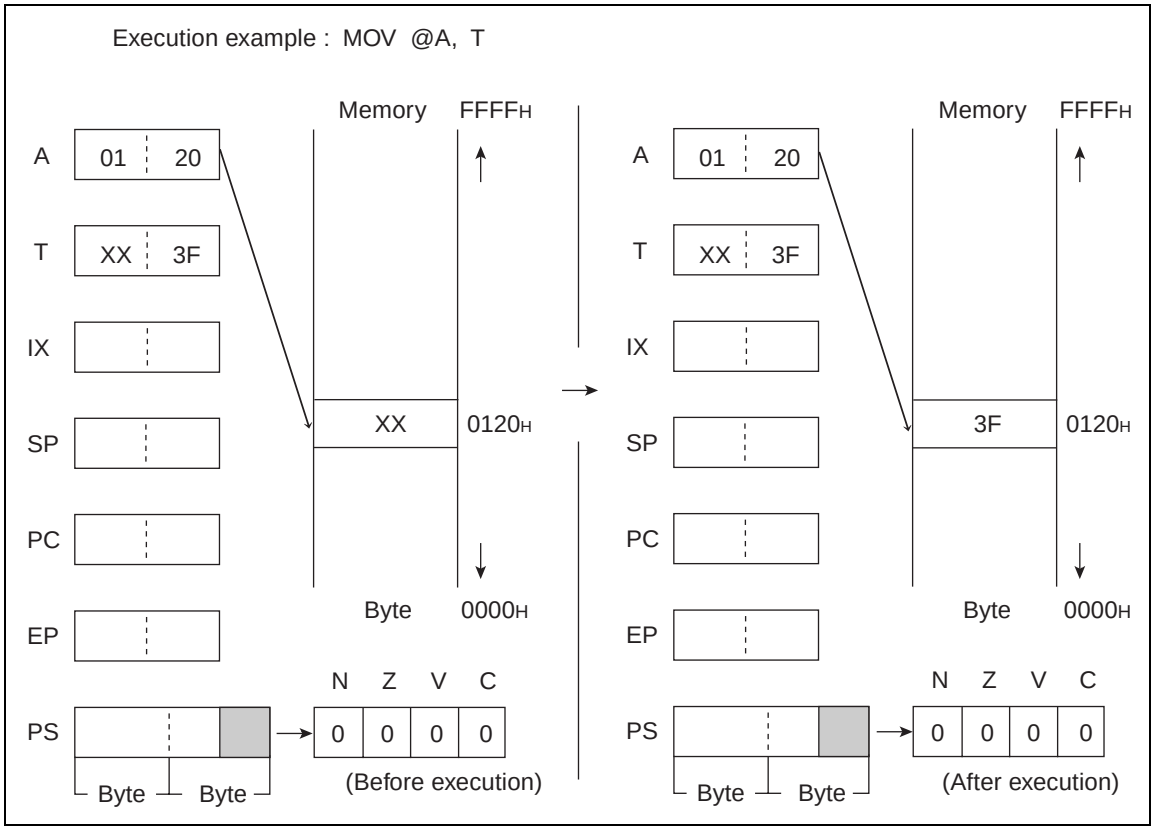
V: Not changed

C: Not changed

Number of execution cycles: 2

Byte count: 1

OP code: 82



6.42 MOV (MOVE Byte Data from Memory to Accumulator)

Transfer byte data from EA memory (memory expressed in each type of addressing) to A. Byte data in AL is transferred to TL. AH is not changed.

■ MOV (MOVE Byte Data from Memory to Accumulator)

Operation

(AL) ← (EA) (Byte transfer)

Assembler format

MOV A, EA

Condition code (CCR)

N	Z	V	C
+	+	-	-

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB of transferred data is 1 and set to 0 in other cases.

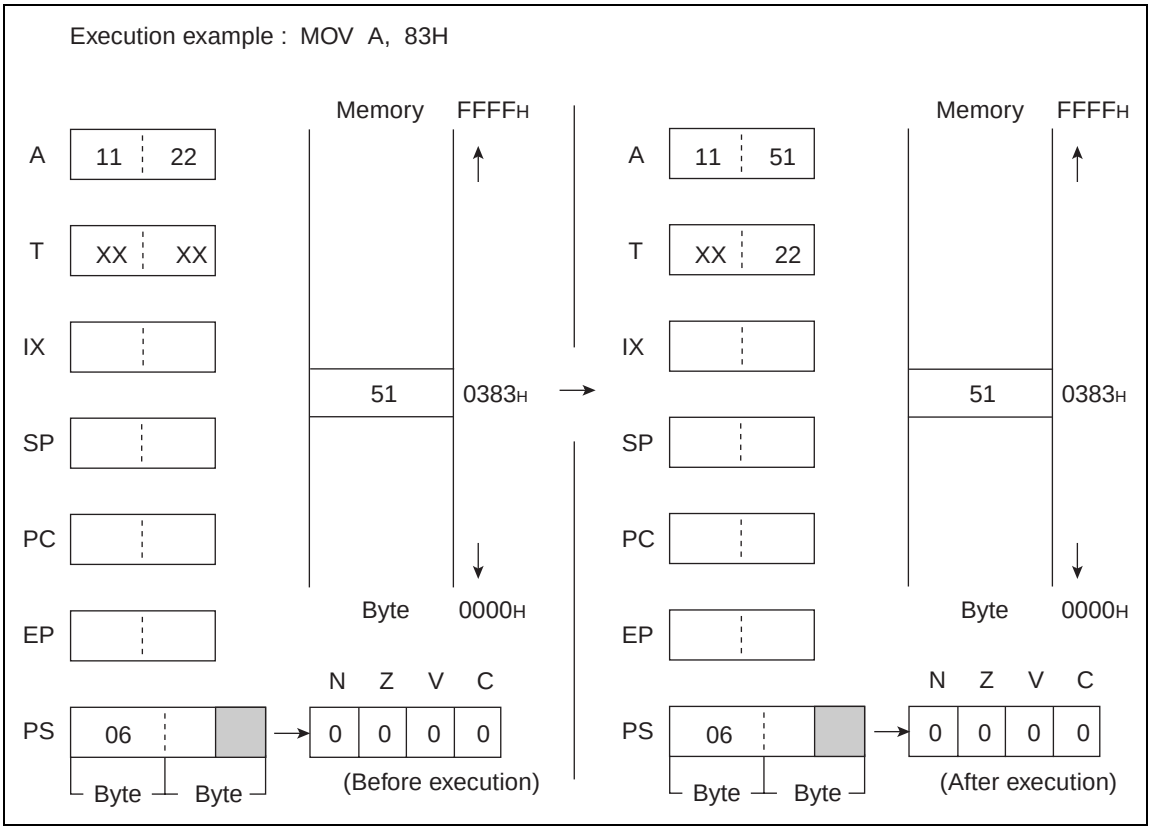
Z: Set to 1 if transferred data is 00_H and set to 0 in other cases.

V: Not changed

C: Not changed

Table 6.42-1 Number of Execution Cycles / Byte Count / OP Code

EA	#d8	dir	@IX+off	ext	@A	@EP	Ri
Number of execution cycles	2	3	3	4	2	2	2
Byte count	2	2	2	3	1	1	1
OP code	04	05	06	60	92	07	08 to 0F



6.43 MOV (MOVE Immediate Byte Data to Memory)

Transfer byte immediate data to EA memory (memory expressed in each type of addressing).

■ MOV (MOVE Immediate Byte Data to Memory)

Operation

(EA) ← d8 (Byte transfer)

Assembler format

MOV EA, #d8

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

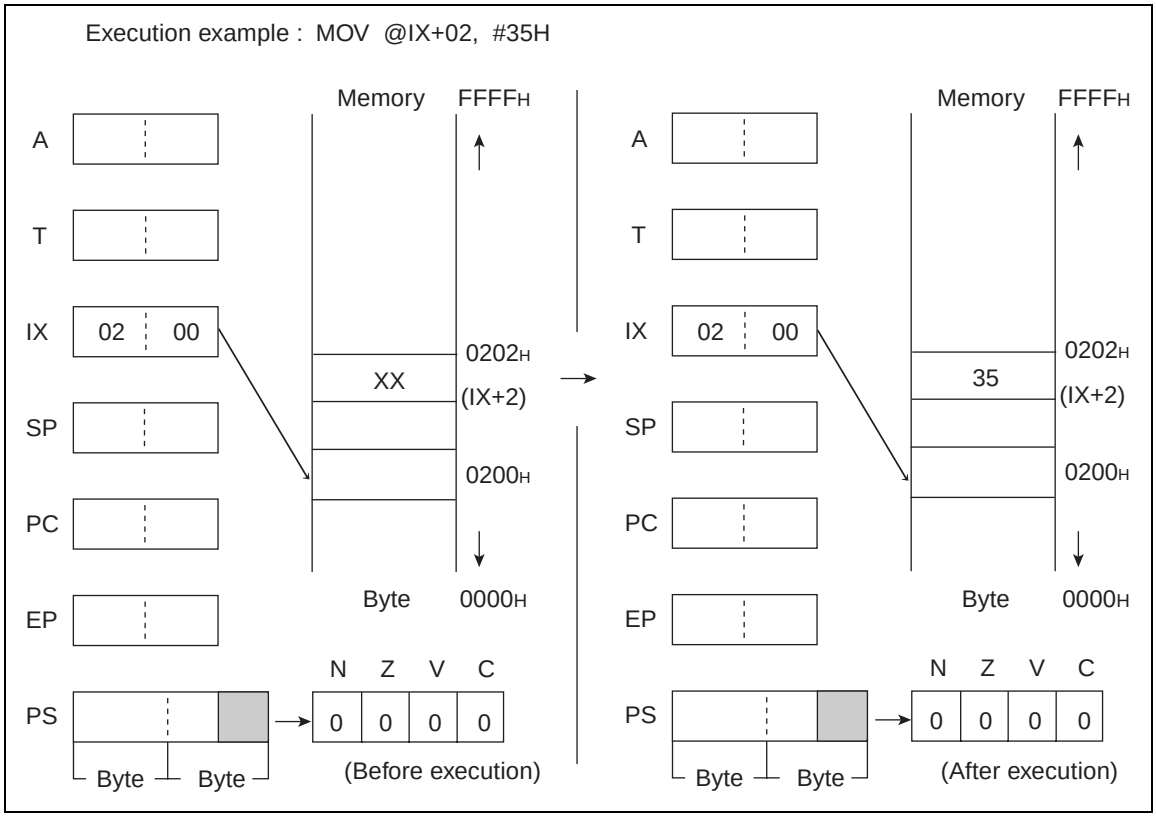
Z: Not changed

V: Not changed

C: Not changed

Table 6.43-1 Number of Execution Cycles / Byte Count / OP Code

EA	dir	@IX+off	@EP	Ri
Number of execution cycles	4	4	3	3
Byte count	3	3	2	2
OP code	85	86	87	88 to 8F



6.44 MOV (MOVE Byte Data from Accumulator to memory)

Transfer bytes (data from AL) immediate data to EA memory (memory expressed in each type of addressing).

■ MOV (MOVE Byte Data from Accumulator to memory)

Operation

(EA) ← (AL) (Byte transfer)

Assembler format

MOV EA, A

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

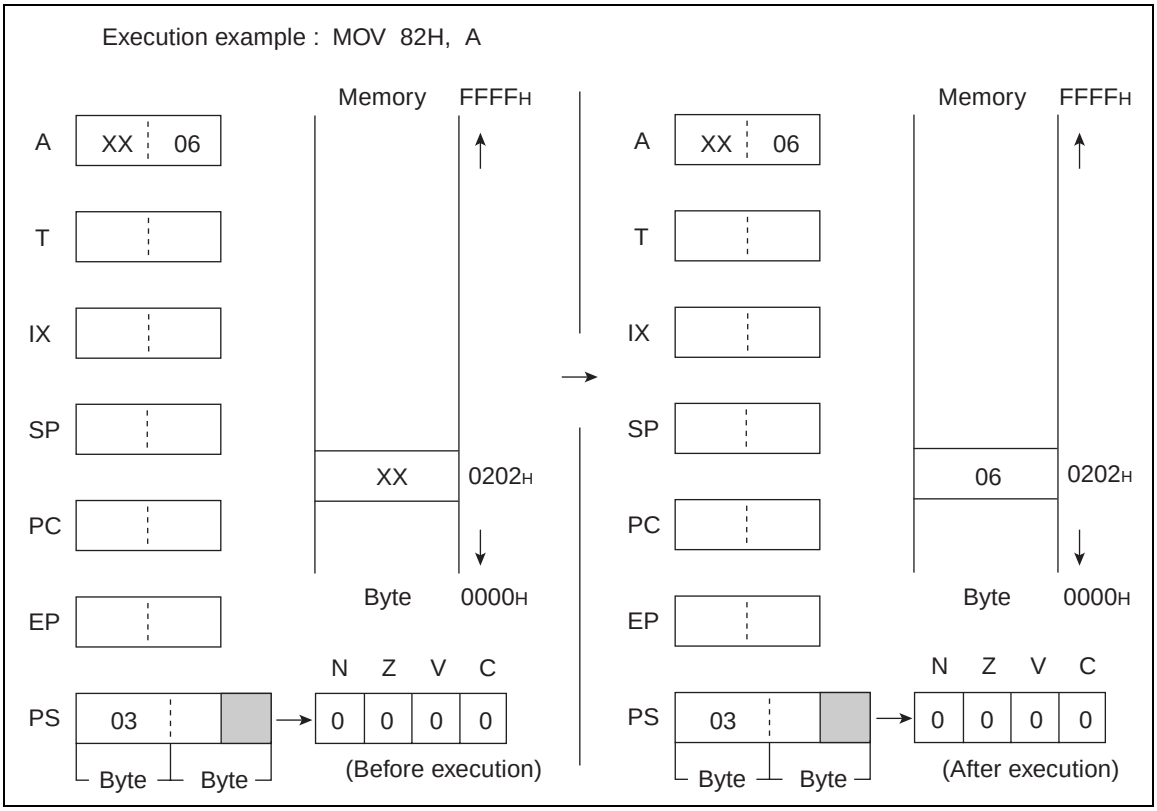
Z: Not changed

V: Not changed

C: Not changed

Table 6.44-1 Number of Execution Cycles / Byte Count / OP Code

EA	dir	@IX+off	ext	@EP	Ri
Number of execution cycles	3	3	4	2	2
Byte count	2	2	3	1	1
OP code	45	46	61	47	48 to 4F



6.45 MOVW (MOVE Word Data from Temporary Accumulator to Address Pointed by Accumulator)

Transfer word data from T to memory indirectly addressed by A.

■ MOVW (MOVE Word Data from Temporary Accumulator to Address Pointed by Accumulator)

Operation

((A)) ← (T) (Word transfer)

Assembler format

MOVW @A, T

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

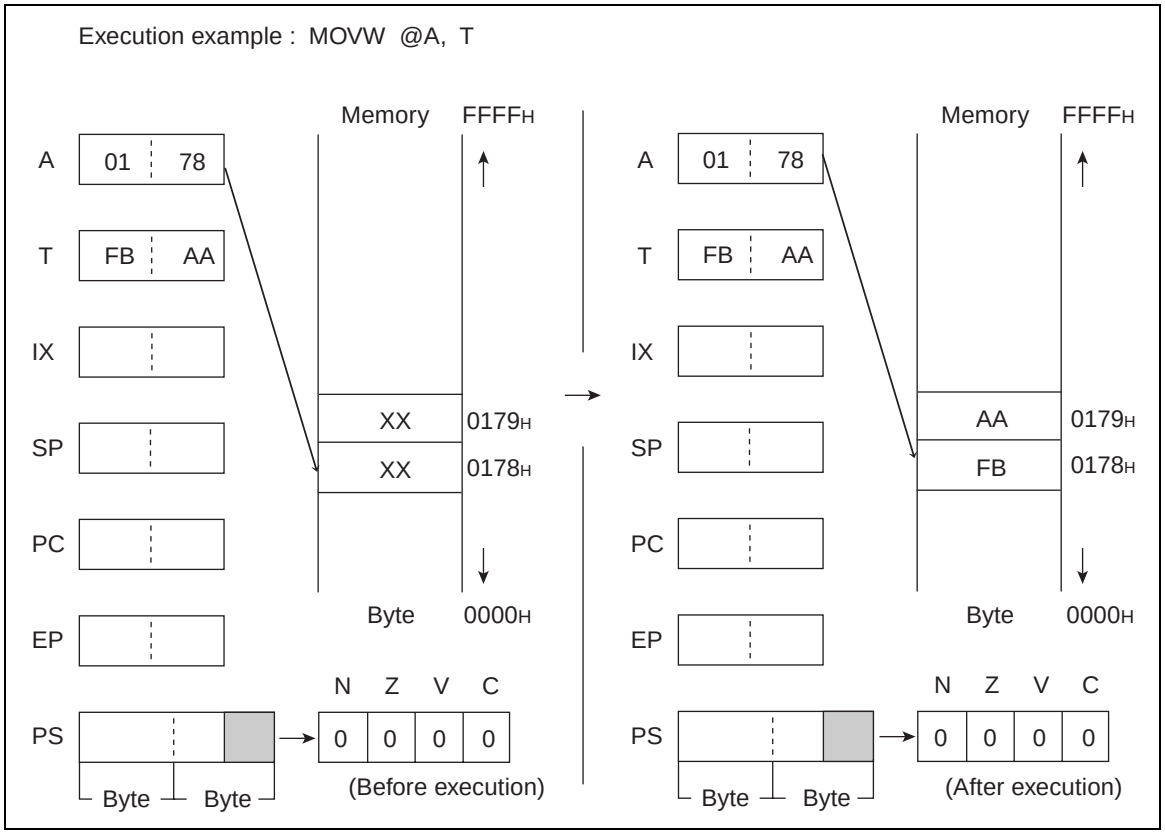
V: Not changed

C: Not changed

Number of execution cycles: 3

Byte count: 1

OP code: 83



6.46 MOVW (MOVE Word Data from Memory to Accumulator)

Transfer word data from EA and EA + 1 memories (EA is an address expressed in each type of addressing) to A. Word data in A is transferred to T.

■ MOVW (MOVE Word Data from Memory to Accumulator)

Operation

(A) ← (EA) (Word transfer)

Assembler format

MOVW A, EA

Condition code (CCR)

N	Z	V	C
+	+	-	-

+: Changed by executing instruction

-: Not changed

N: Set to 1 if MSB of transferred data is 1 and set to 0 in other cases.

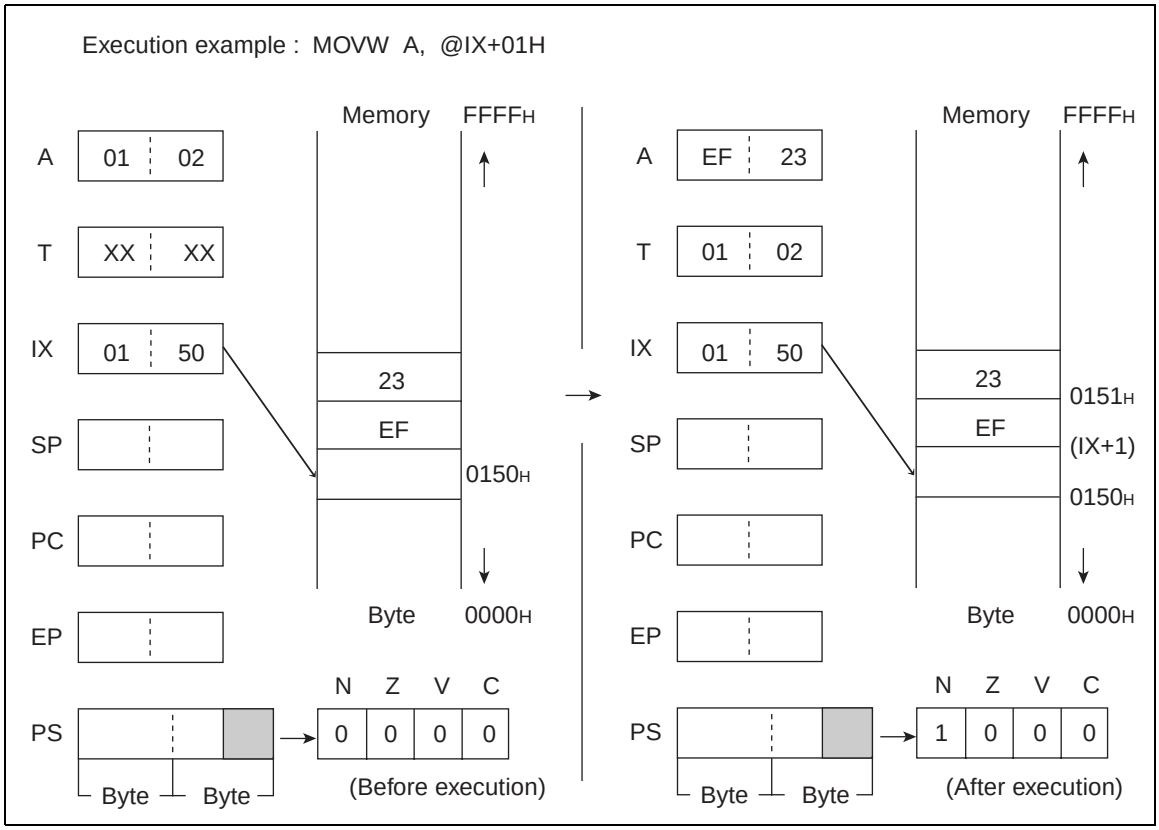
Z: Set to 1 if transferred data is 0000_H and set to 0 in other cases.

V: Not changed

C: Not changed

Table 6.46-1 Number of Execution Cycles / Byte Count / OP Code

EA	#d16	dir	@IX+off	ext	@A	@EP
Number of execution cycles	3	4	4	5	3	3
Byte count	3	2	2	3	1	1
OP code	E4	C5	C6	C4	93	C7



6.47 MOVW (MOVE Word Data from Extra Pointer to Accumulator)

Transfer word data from EP to A.

■ MOVW (MOVE Word Data from Extra Pointer to Accumulator)

Operation

(A) ← (EP) (Word transfer)

Assembler format

MOVW A, EP

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

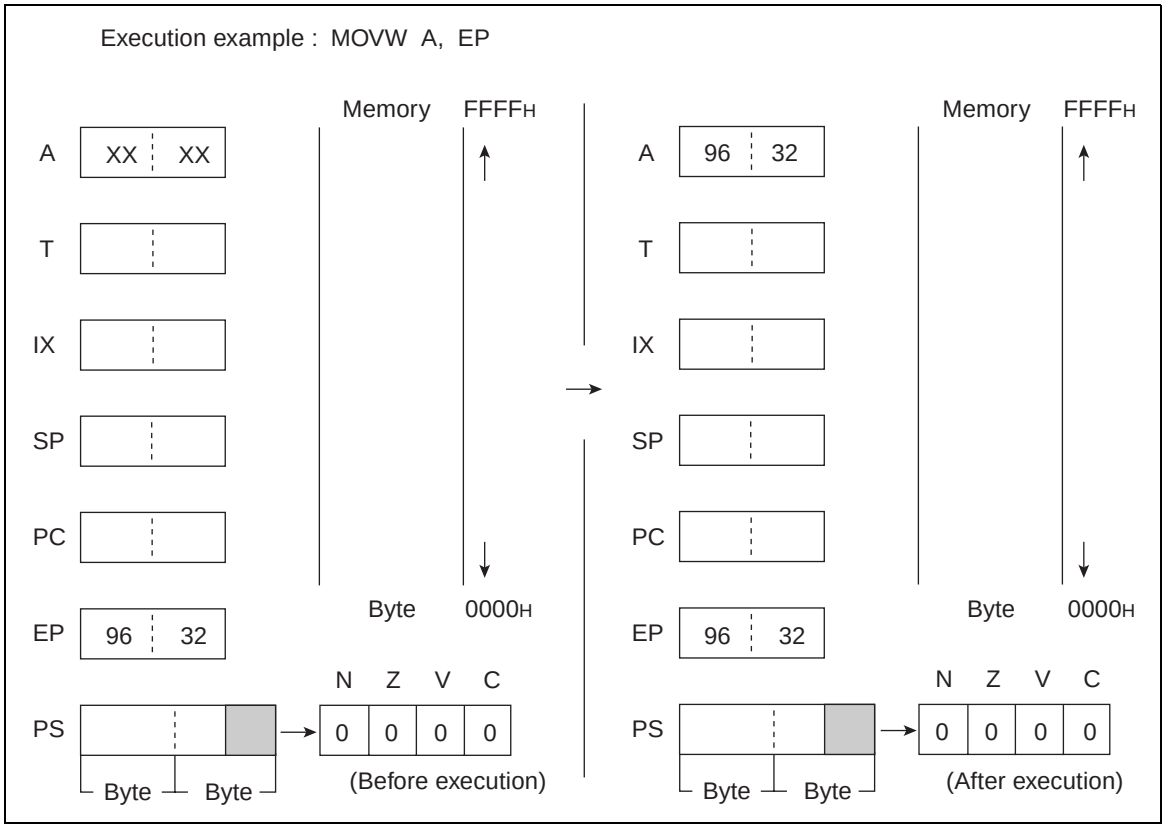
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: F3



6.48 MOVW (MOVE Word Data from Index Register to Accumulator)

Transfer word data from IX to A.

■ MOVW (MOVE Word Data from Index Register to Accumulator)

Operation

(A) ← (IX) (Word transfer)

Assembler format

MOVW A, IX

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

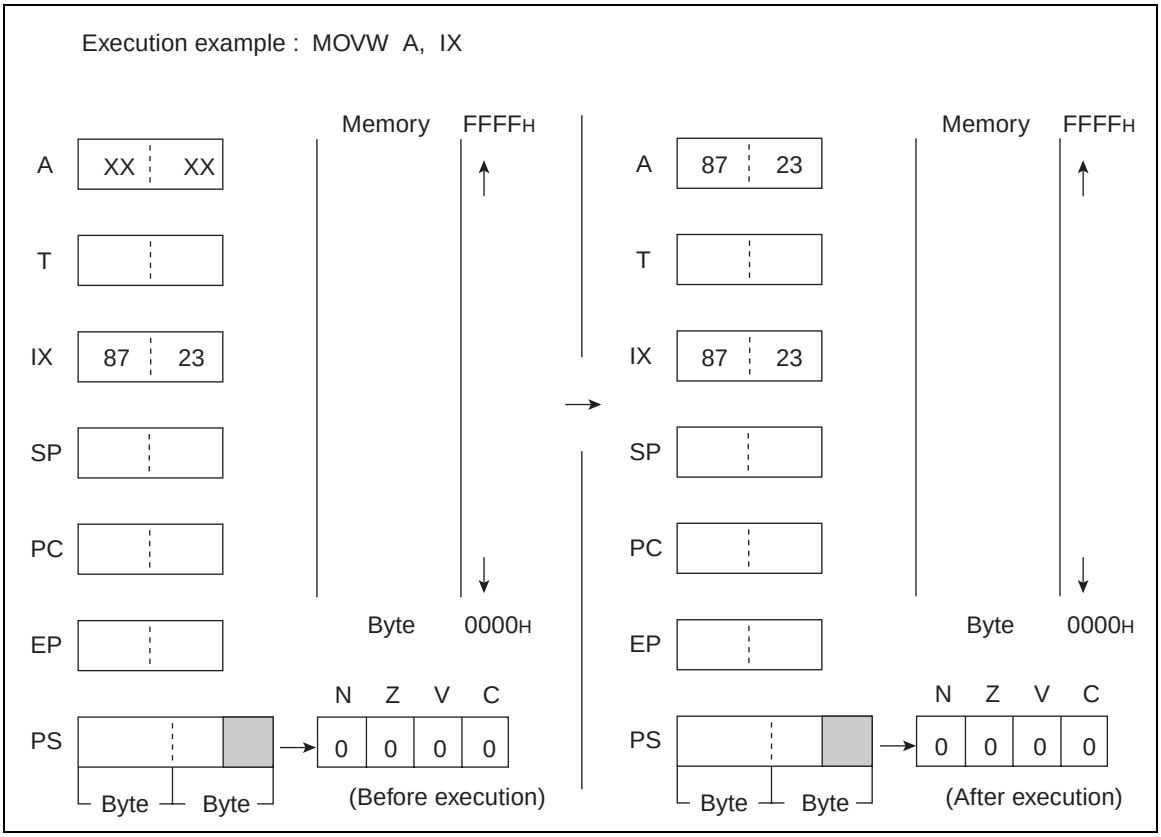
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: F2



6.49 MOVW (MOVE Word Data from Program Status Register to Accumulator)

Transfer word data from PS to A.

■ MOVW (MOVE Word Data from Program Status Register to Accumulator)

Operation

(A) ← (PS) (Word transfer)

Assembler format

MOVW A, PS

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

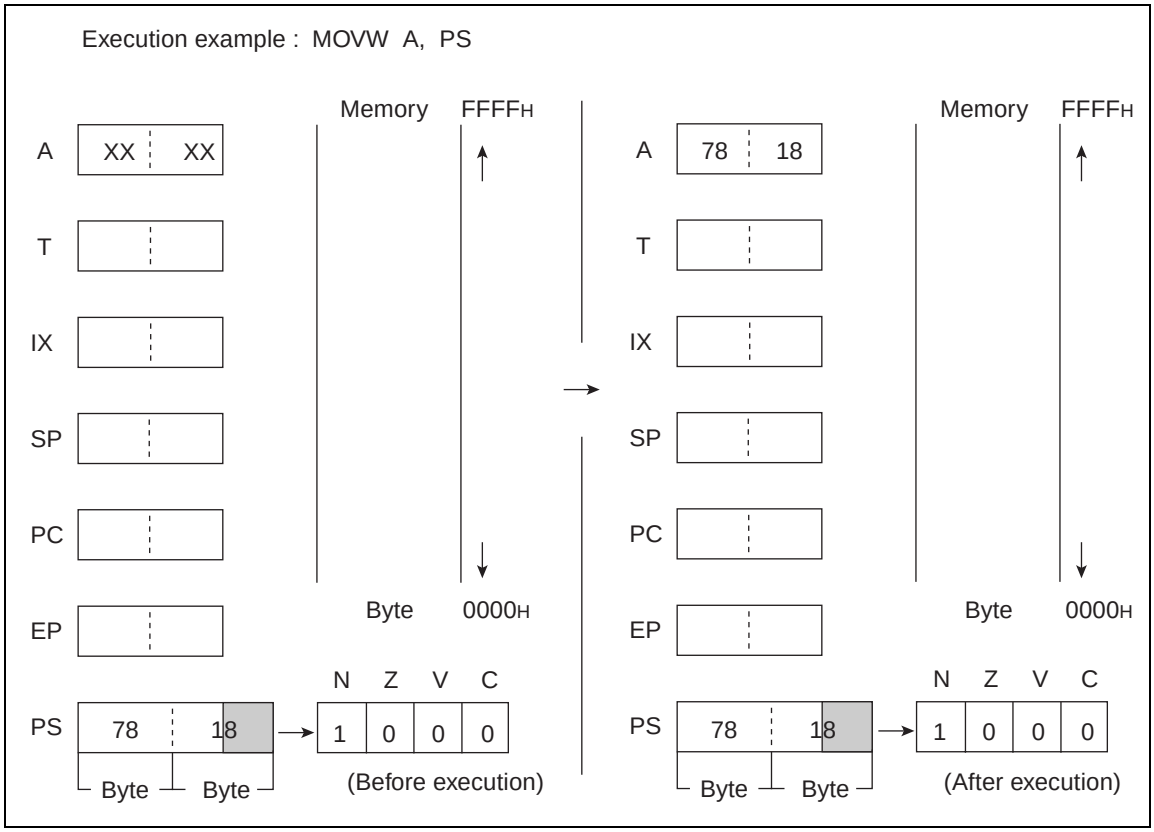
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: 70



6.50 MOVW (MOVE Word Data from Program Counter to Accumulator)

Transfer word data from PC to A.

■ MOVW (MOVE Word Data from Program Counter to Accumulator)

Operation

(A) ← (PC) (Word transfer)

Assembler format

MOVW A, PC

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

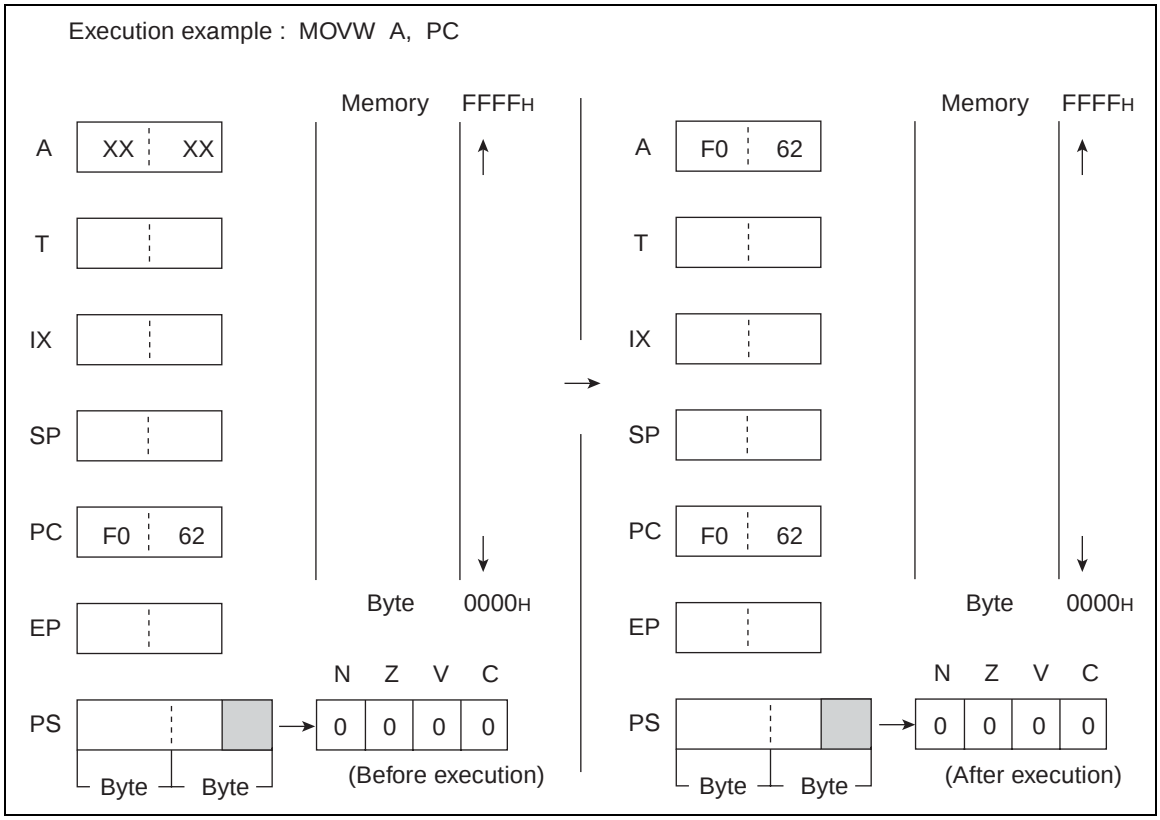
V: Not changed

C: Not changed

Number of execution cycles: 2

Byte count: 1

OP code: F0



6.51 MOVW (MOVE Word Data from Stack Pointer to Accumulator)

Transfer word data from SP to A.

■ MOVW (MOVE Word Data from Stack Pointer to Accumulator)

Operation

(A) ← (SP) (Word transfer)

Assembler format

MOVW A, SP

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

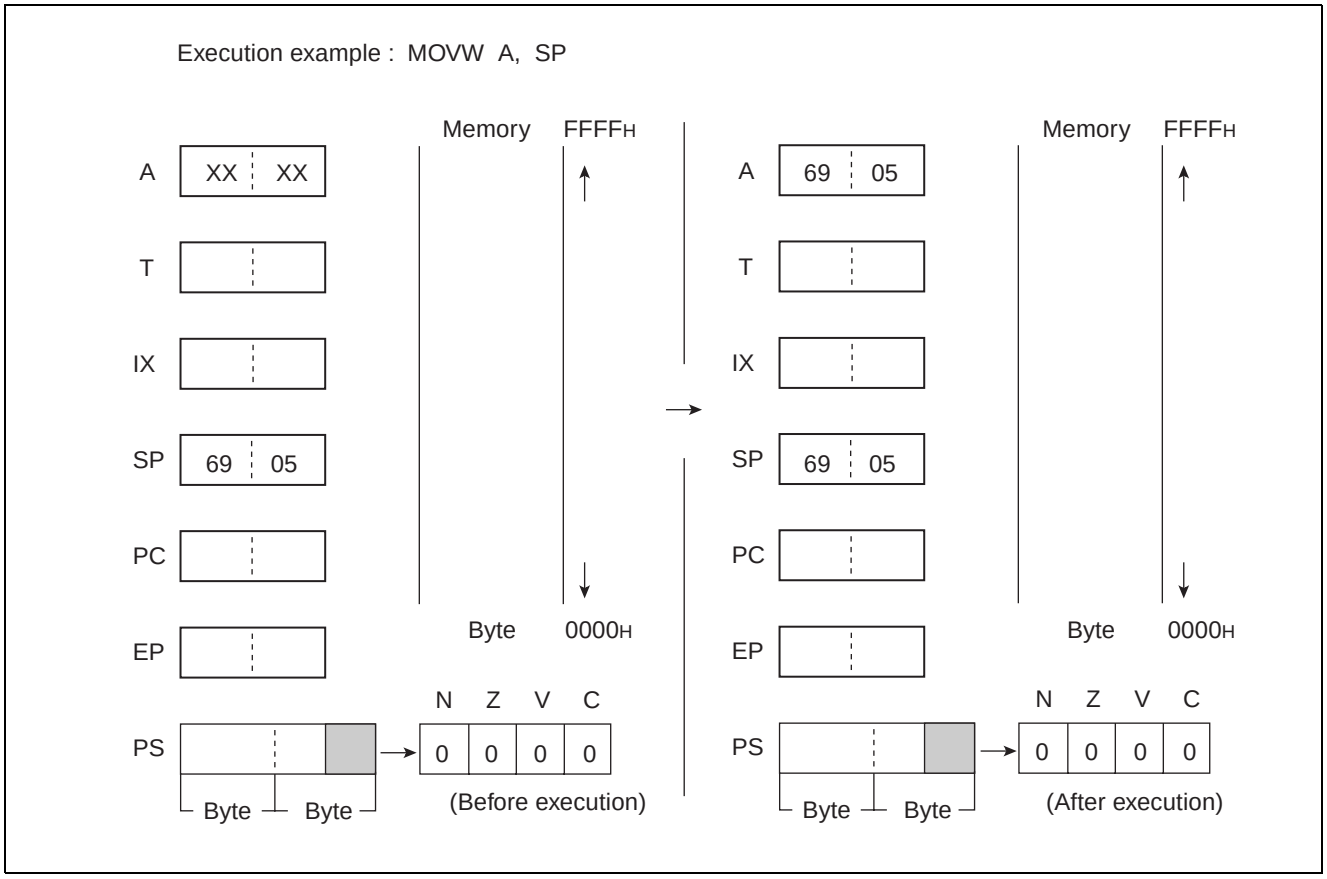
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: F1



6.52 MOVW (MOVE Word Data from Accumulator to Memory)

Transfer word data from A to EA and EA + 1 memories (memory expressed in each type of addressing).

■ MOVW (MOVE Word Data from Accumulator to Memory)

Operation

(EA) ← (A) (Word transfer)

Assembler format

MOVW EA, A

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

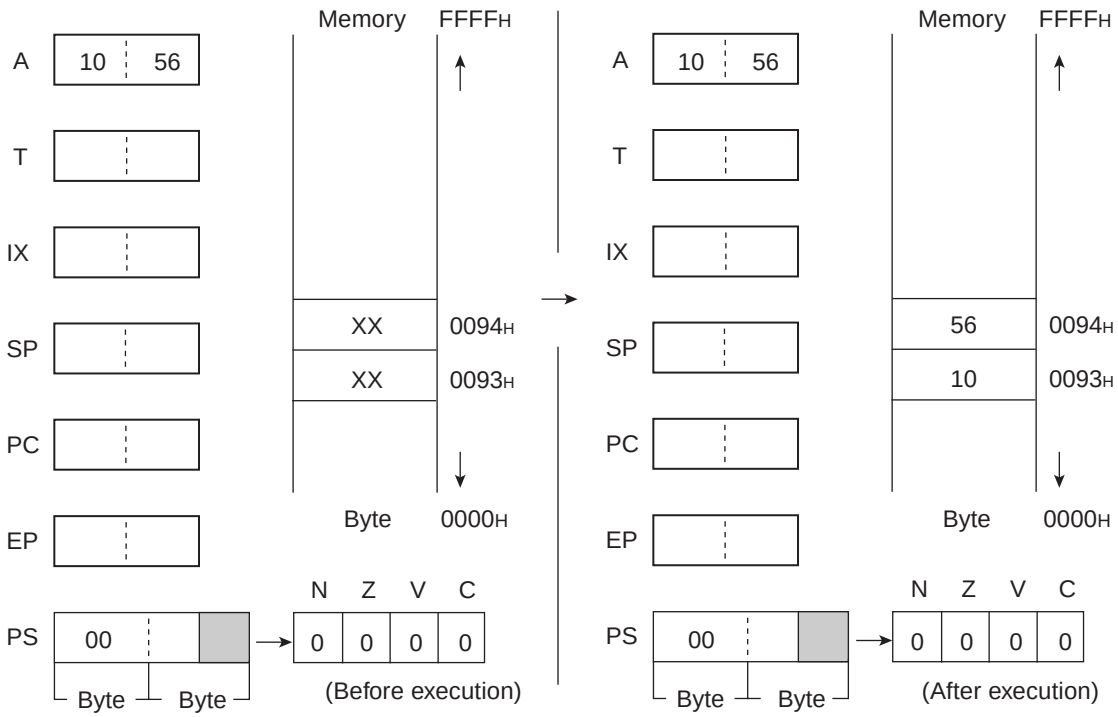
V: Not changed

C: Not changed

Table 6.52-1 Number of Execution Cycles / Byte Count / OP Code

EA	dir	@IX+off	ext	@EP
Number of execution cycles	4	4	5	3
Byte count	2	2	3	1
OP code	D5	D6	D4	D7

Execution example : MOVW 93H, A



6.53 MOVW (MOVE Word Data from Accumulator to Extra Pointer)

Transfer word data from A to EP.

■ MOVW (MOVE Word Data from Accumulator to Extra Pointer)

Operation

(EP) ← (A) (Word transfer)

Assembler format

MOVW EP, A

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

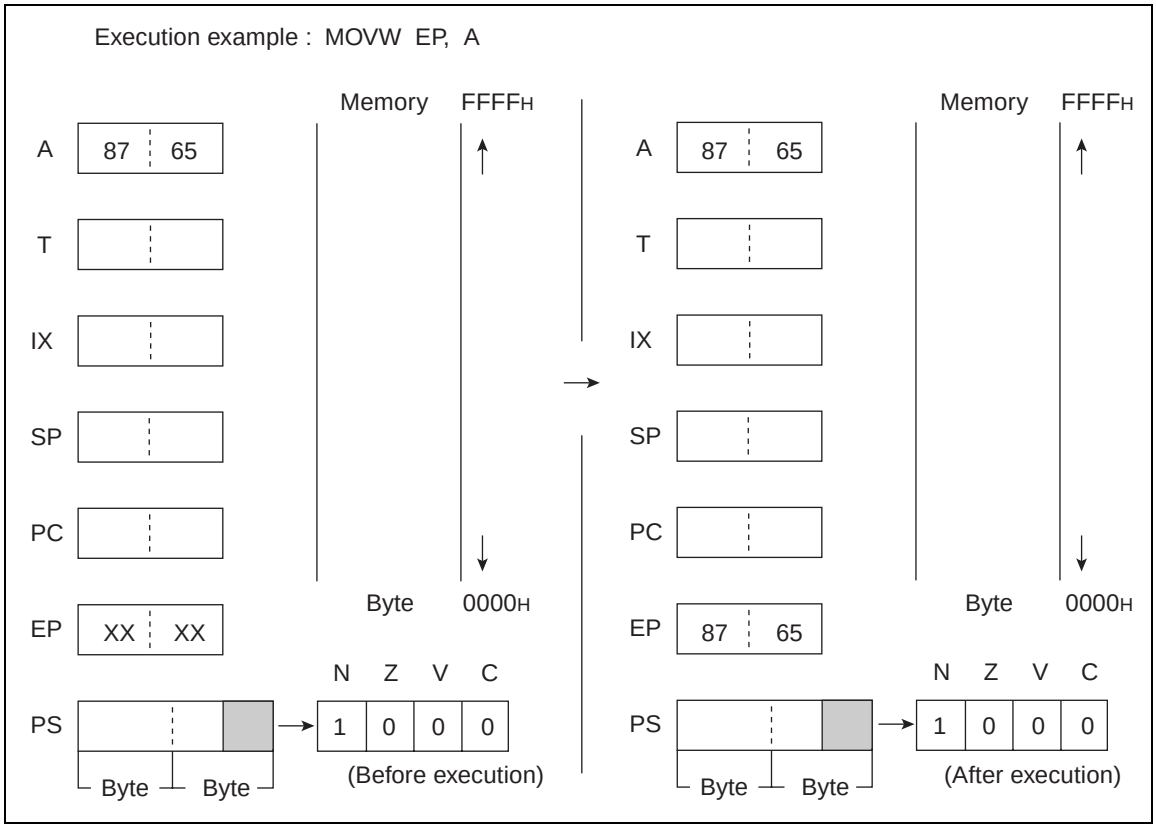
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: E3



6.54 MOVW (MOVE Immediate Word Data to Extra Pointer)

Transfer word immediate data to EP.

■ MOVW (MOVE Immediate Word Data to Extra Pointer)

Operation

(EP) ← d16 (Word transfer)

Assembler format

MOVW EP, #d16

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

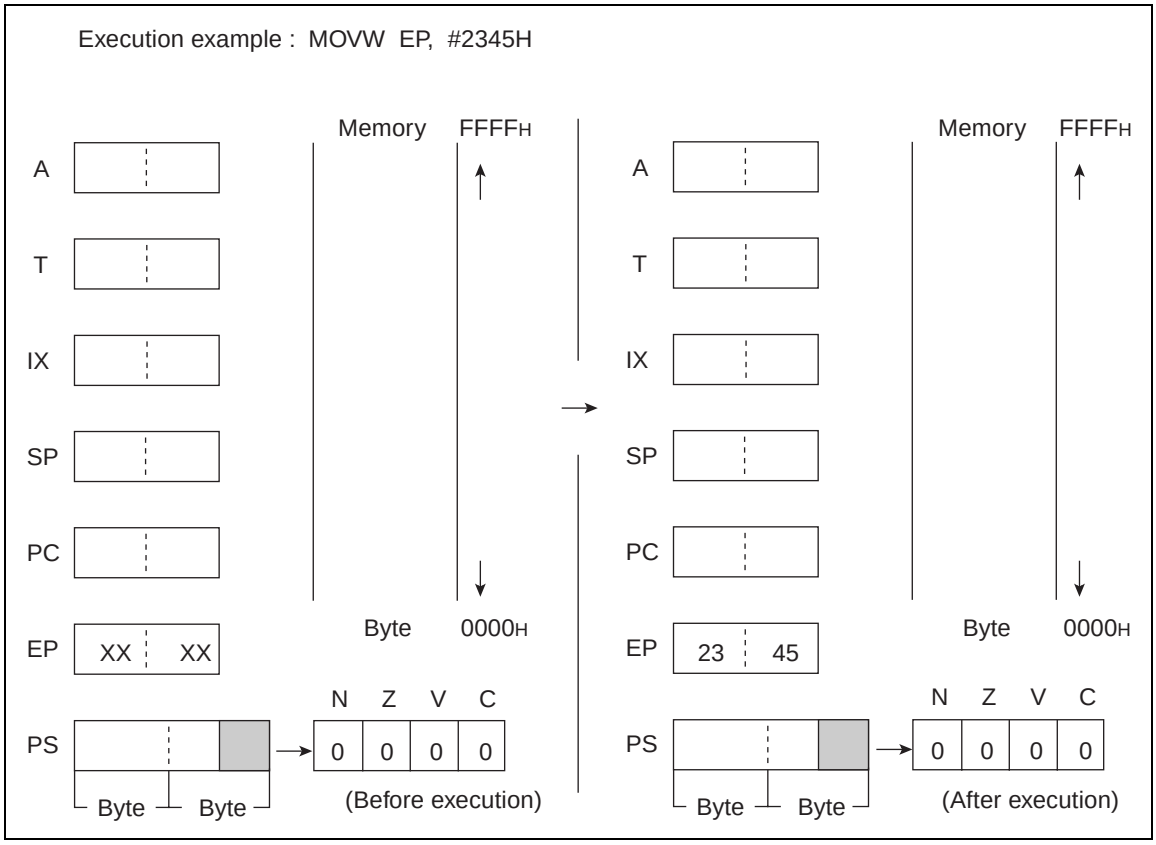
V: Not changed

C: Not changed

Number of execution cycles: 3

Byte count: 3

OP code: E7



6.55 MOVW (MOVE Word Data from Accumulator to Index Register)

Transfer word data from A to IX.

■ MOVW (MOVE Word Data from Accumulator to Index Register)

Operation

(IX) ← (A) (Word transfer)

Assembler format

MOVW IX, A

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

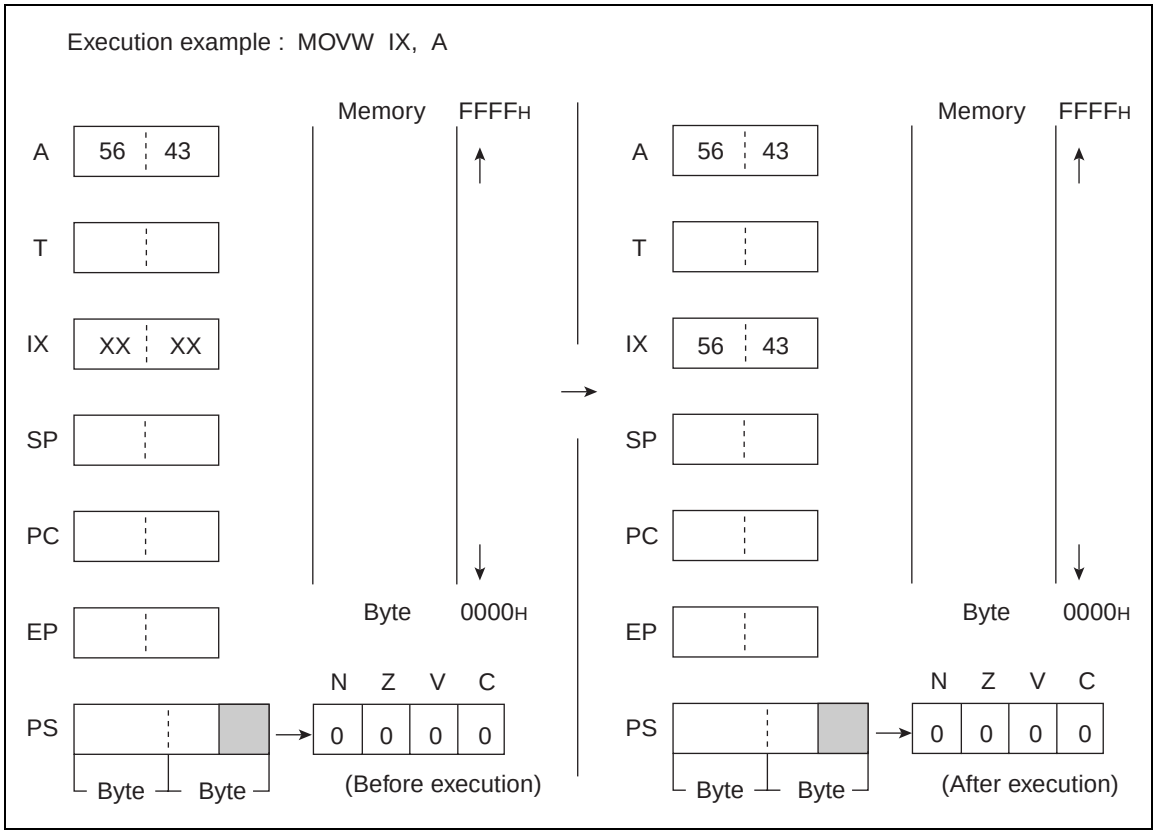
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: E2



6.56 MOVW (MOVE Immediate Word Data to Index Register)

Transfer word immediate data to IX.

■ MOVW (MOVE Immediate Word Data to Index Register)

Operation

(IX) ← d16 (Word transfer)

Assembler format

MOVW IX, #d16

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

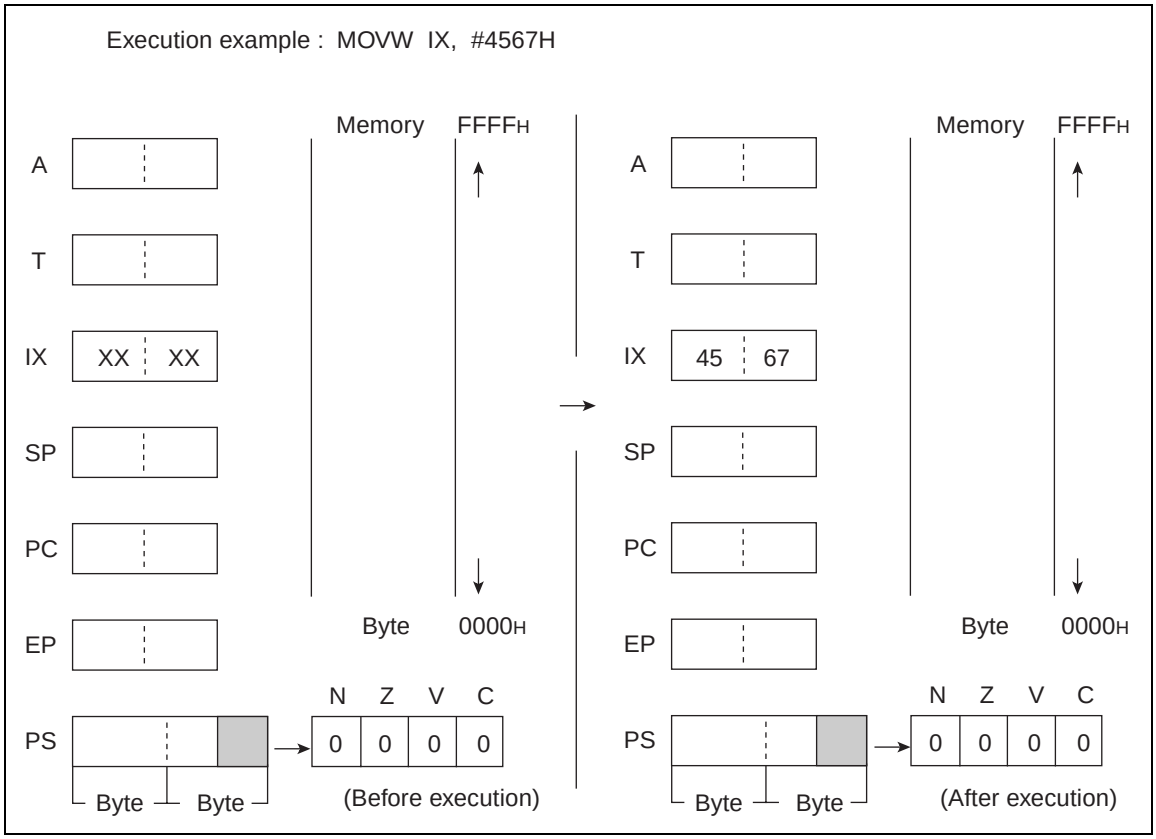
V: Not changed

C: Not changed

Number of execution cycles: 3

Byte count: 3

OP code: E6



6.57 MOVW (MOVE Word data from Accumulator to Program Status Register)

Transfer word data from A to PS.

■ MOVW (MOVE Word data from Accumulator to Program Status Register)

Operation

(PS) ← (A) (Word transfer)

Assembler format

MOVW PS, A

Condition code (CCR)

N	Z	V	C
+	+	+	+

+: Changed by executing instruction

-: Not changed

N: Become the value for lower bit 3 of A

Z: Become the value for lower bit 2 of A

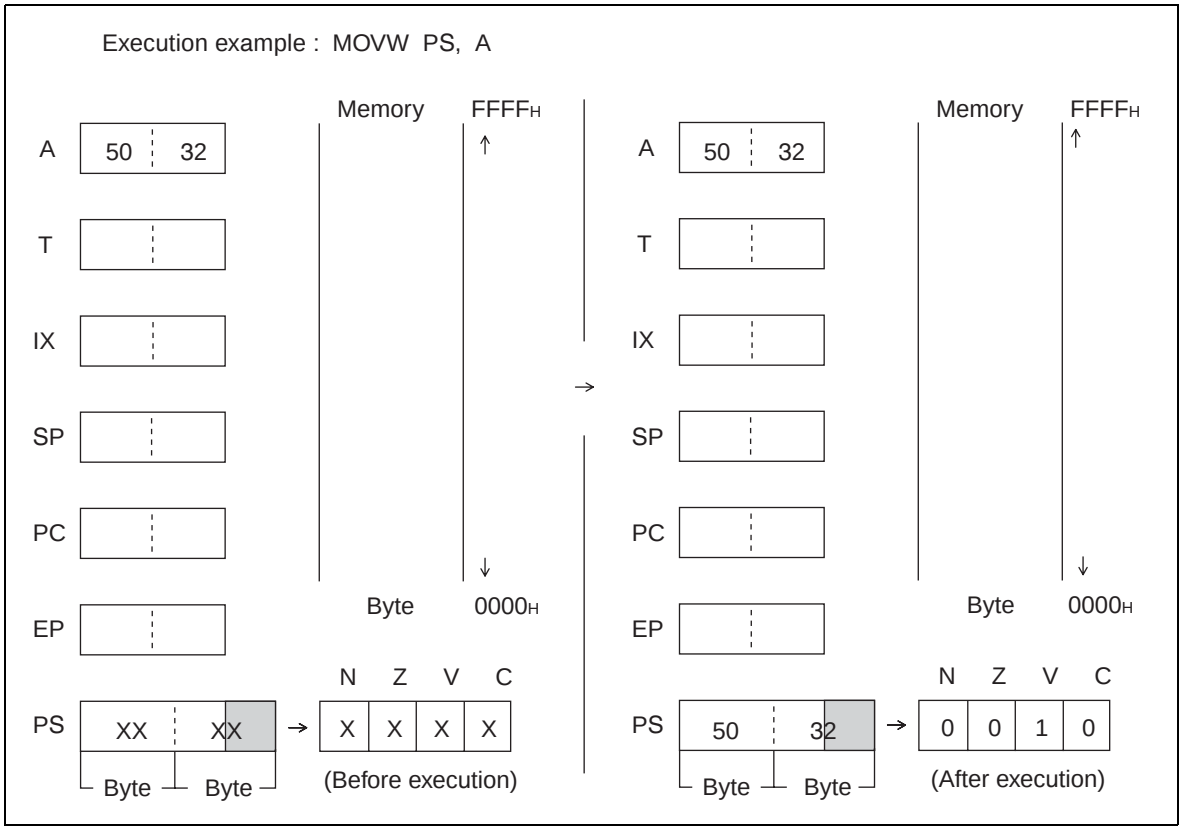
V: Become the value for lower bit 1 of A

C: Become the value for lower bit 0 of A

Number of execution cycle: 1

Byte count: 1

OP code: 71



6.58 MOVW (MOVE Immediate Word Data to Stack Pointer)

Transfer word immediate data to SP.

■ MOVW (MOVE Immediate Word Data to Stack Pointer)

Operation

(SP) ← d16 (Word transfer)

Assembler format

MOVW SP, #d16

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

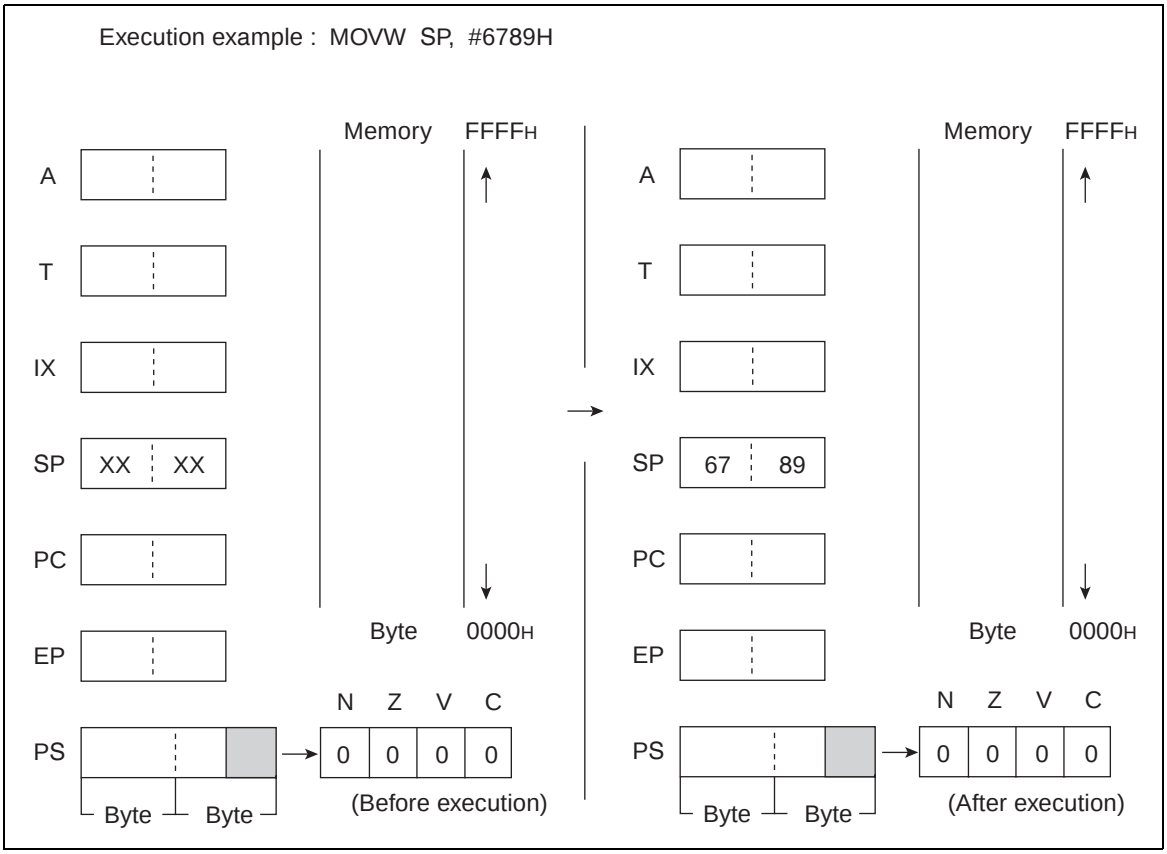
V: Not changed

C: Not changed

Number of execution cycles: 3

Byte count: 3

OP code: E5



6.59 MOVW (MOVE Word data from Accumulator to Stack Pointer)

Transfer word data from A to SP.

■ MOVW (MOVE Word data from Accumulator to Stack Pointer)

Operation

(SP) ← (A) (Word transfer)

Assembler format

MOVW SP, A

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

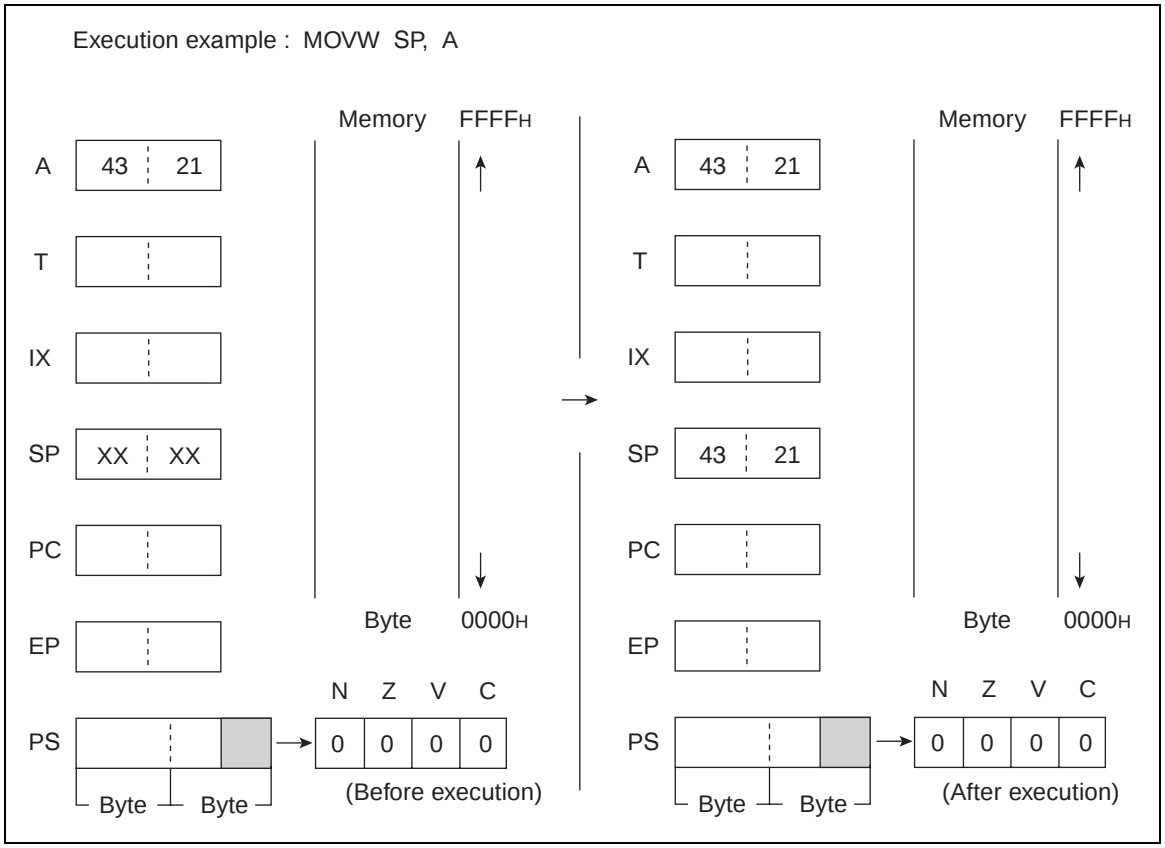
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: E1



6.60 MULU (MULtiply Unsigned)

Multiply the byte data of AL and TL as unsigned binary values. Return the results to the word data of A.

■ MULU (MULtiply Unsigned)

Operation

(A) ← (AL) * (TL)

Assembler format

MULU A

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

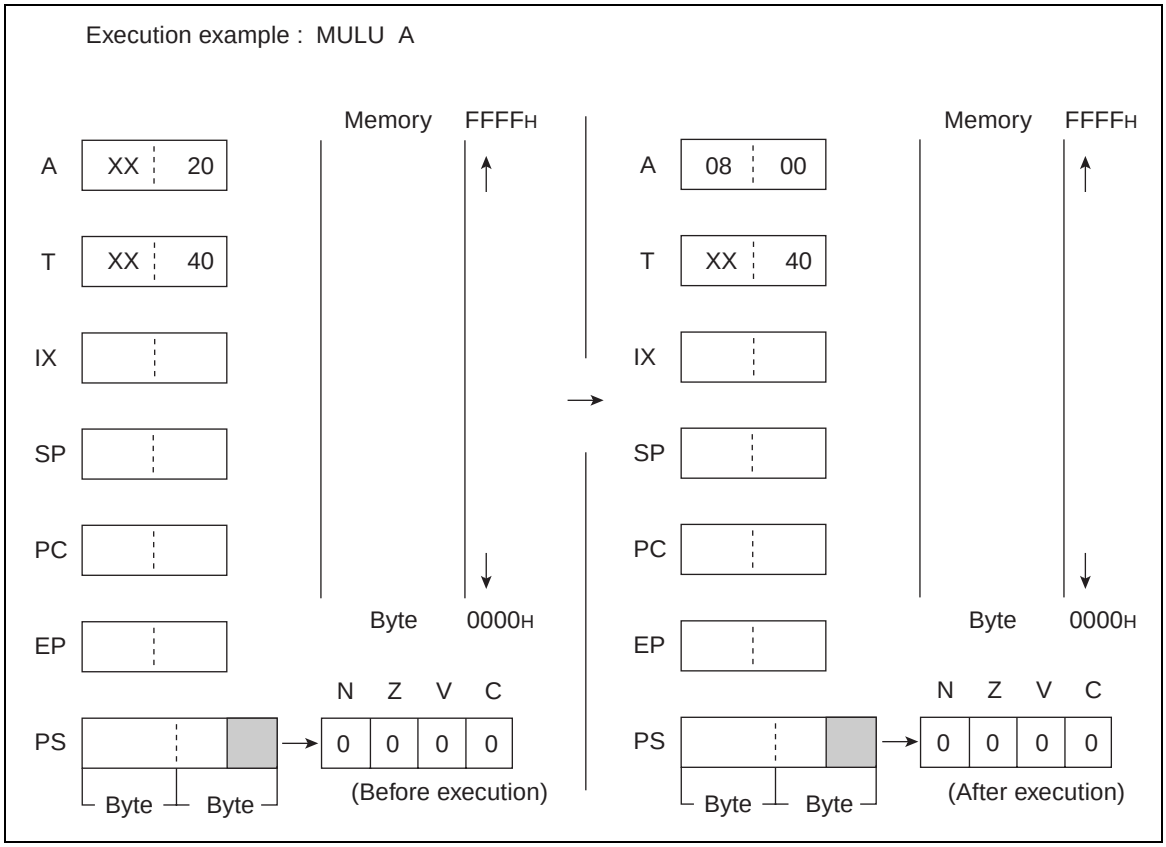
V: Not changed

C: Not changed

Number of execution cycles: 8

Byte count: 1

OP code: 01



6.61 NOP (NoOPeration)

No operation

■ NOP (NoOPeration)

Operation

————

Assembler format

NOP

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

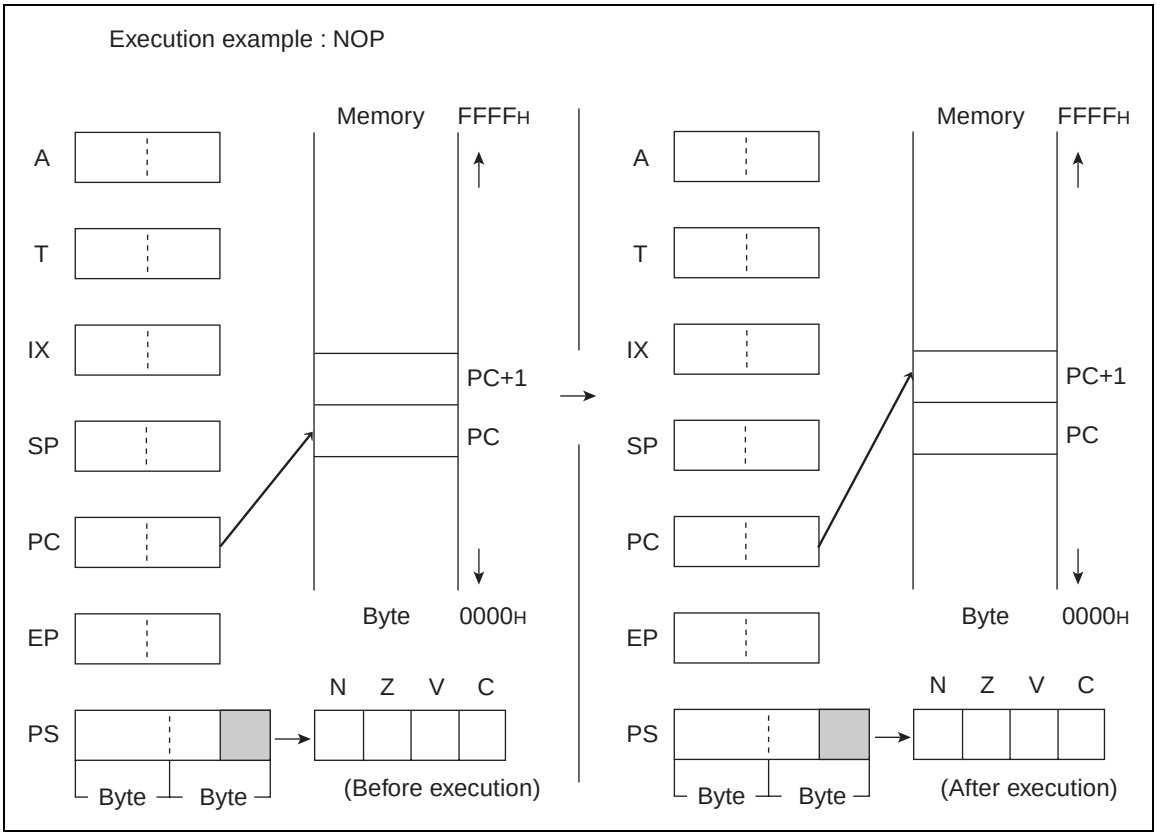
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: 00



6.62 OR (OR Byte Data of Accumulator and Temporary Accumulator to Accumulator)

Carry out the logical OR on byte data of AL and TL for every bit and return the results to AL. The contents of AH are not changed.

■ OR (OR Byte Data of Accumulator and Temporary Accumulator to Accumulator)

Operation

(AL) ← (AL) ∨ (TL) (byte logical OR)

Assembler format

OR A

Condition code (CCR)

N	Z	V	C
+	+	R	-

+: Changed by executing instruction

-: Not changed

R: Set to 0 by executing instruction

N: Set to 1 if the MSB of AL is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

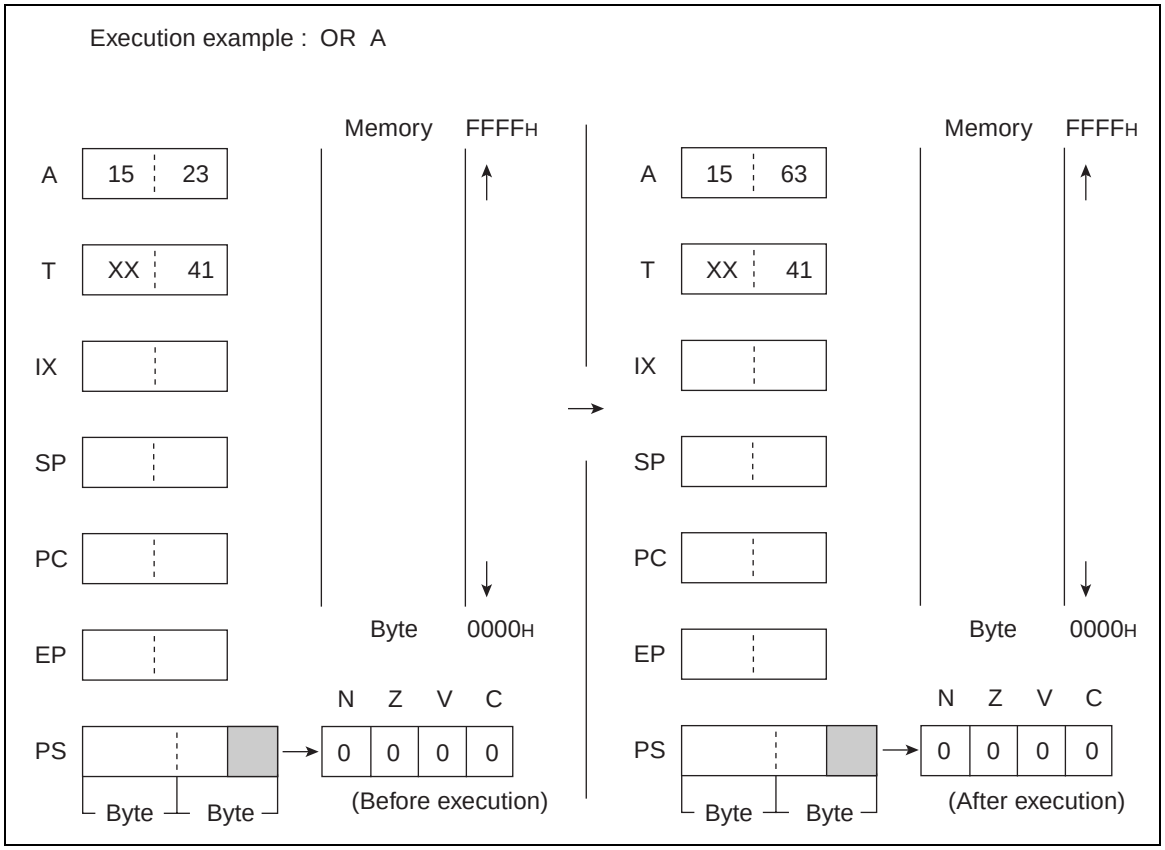
V: Always set to 0

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: 72



6.63 OR (OR Byte Data of Accumulator and Memory to Accumulator)

Carry out the logical OR on AL and EA memory (memory expressed in each type of addressing) for every bit and return the results to AL. The contents of AH are not changed.

■ OR (OR Byte Data of Accumulator and Memory to Accumulator)

Operation

(AL) ← (AL)∨ (EA) (byte logical OR)

Assembler format

OR A, EA

Condition code (CCR)

N	Z	V	C
+	+	R	-

+: Changed by executing instruction

-: Not changed

R: Set to 0 by executing instruction

N: Set to 1 if the MSB of AL is 1 as the result of operation and set to 0 in other cases.

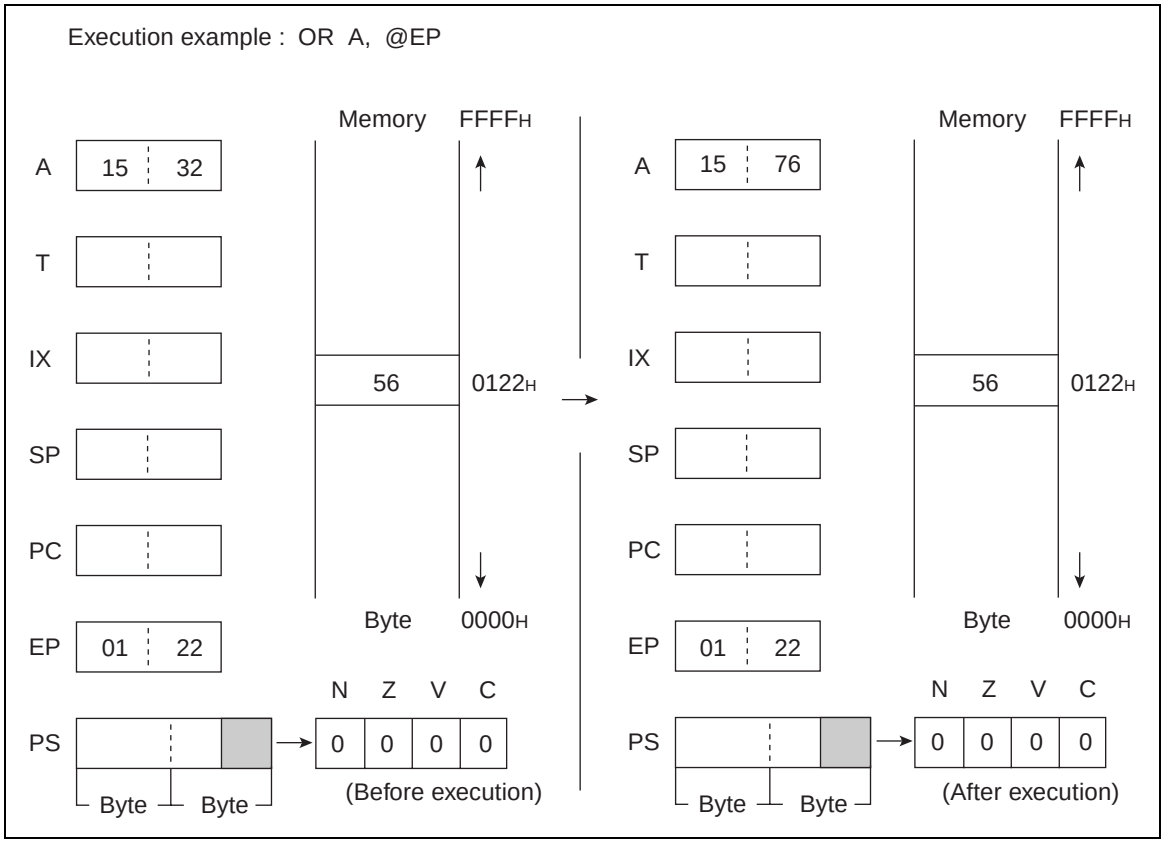
Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

V: Always set to 0

C: Not changed

Table 6.63-1 Number of Execution Cycles / Byte Count / OP Code

EA	#d8	dir	@IX+off	@EP	Ri
Number of execution cycles	2	3	3	2	2
Byte count	2	2	2	1	1
OP code	74	75	76	77	78 to 7F



6.64 ORW (OR Word Data of Accumulator and Temporary Accumulator to Accumulator)

Carry out the logical OR on the word data of A and T for every bit and return the results to A.

■ ORW (OR Word Data of Accumulator and Temporary Accumulator to Accumulator)

Operation

(A) ← (A) ∨ (T) (word logical OR)

Assembler format

ORW A

Condition code (CCR)

N	Z	V	C
+	+	R	-

+: Changed by executing instruction

-: Not changed

R: Set to 0 by executing instruction

N: Set to 1 if the MSB of A is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 0000_H and set to 0 in other cases.

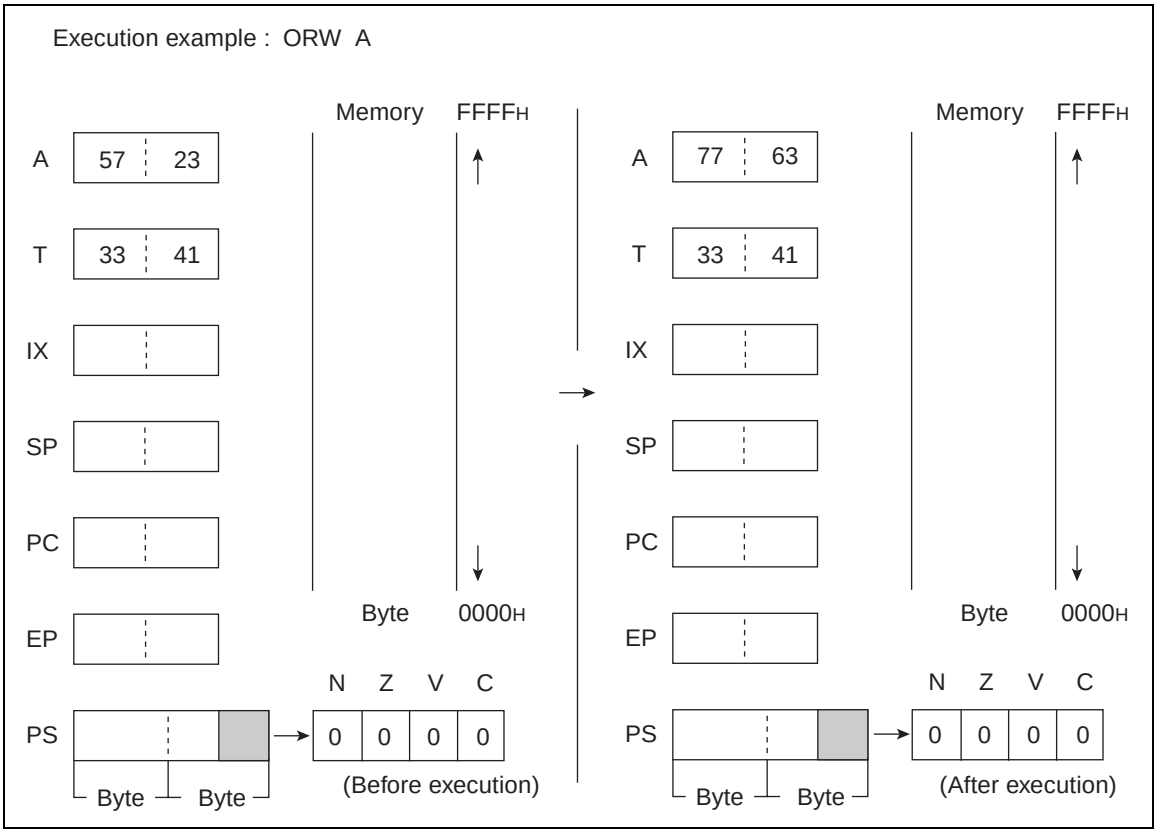
V: Always set to 0

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: 73



6.65 PUSHW (PUSH Word Data of Inherent Register to Stack Memory)

Subtract 2 from the value of SP. Then, transfer the word value from the memory indicated by SP to dr.

■ PUSHW (PUSH Word Data of Inherent Register to Stack Memory)

Operation

(SP) ← (SP) - 2 (Word subtraction)

((SP)) ← (dr) (Word transfer)

Assembler format

PUSHW dr

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

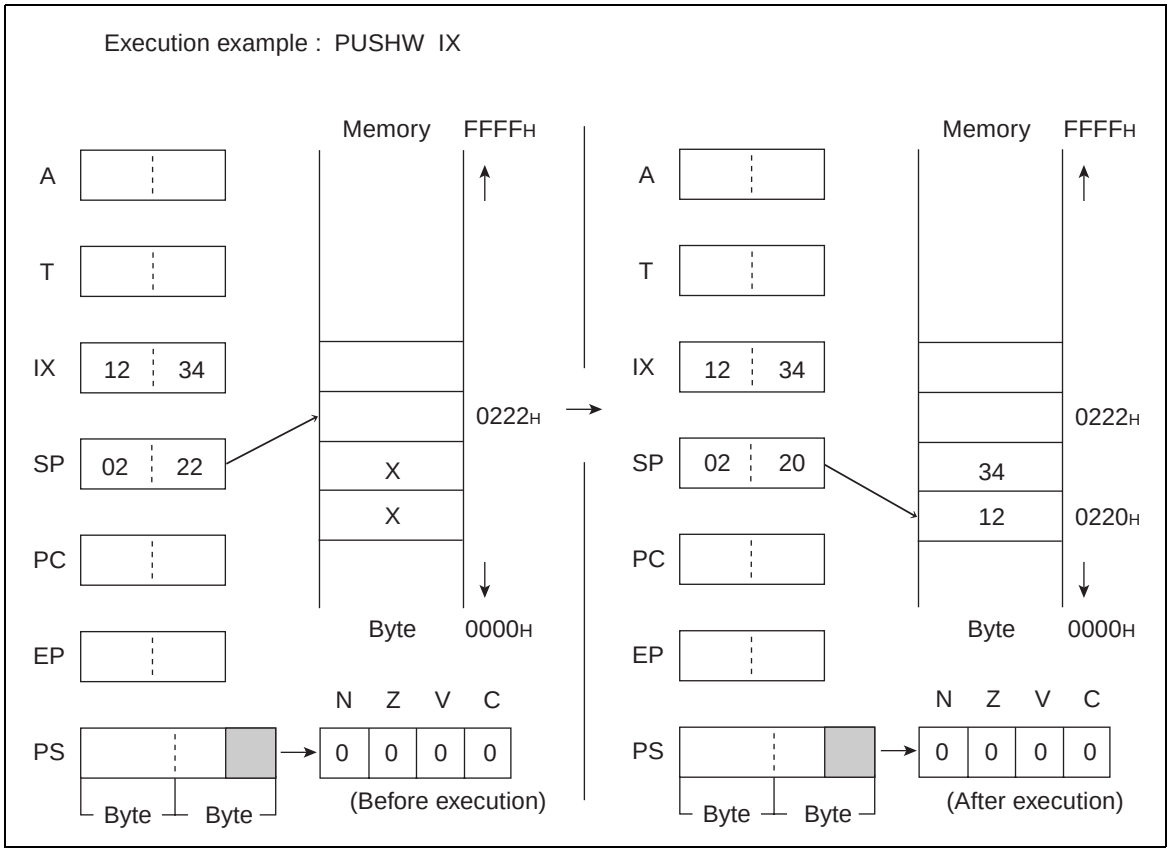
Z: Not changed

V: Not changed

C: Not changed

Table 6.65-1 Number of Execution Cycles / Byte Count / OP Code

DR	A	IX
Number of execution cycles	4	4
Byte count	1	1
OP code	40	41



6.66 POPW (POP Word Data of Intherent Register from Stack Memory)

Transfer the word value from the memory indicated by SP to dr. Then, add 2 to the value of SP.

■ POPW (POP Word Data of Intherent Register from Stack Memory)

Operation

(dr) ← ((SP)) (Word transfer)

(SP) ← (SP) + 2 (Word addition)

Assembler format

POPW dr

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

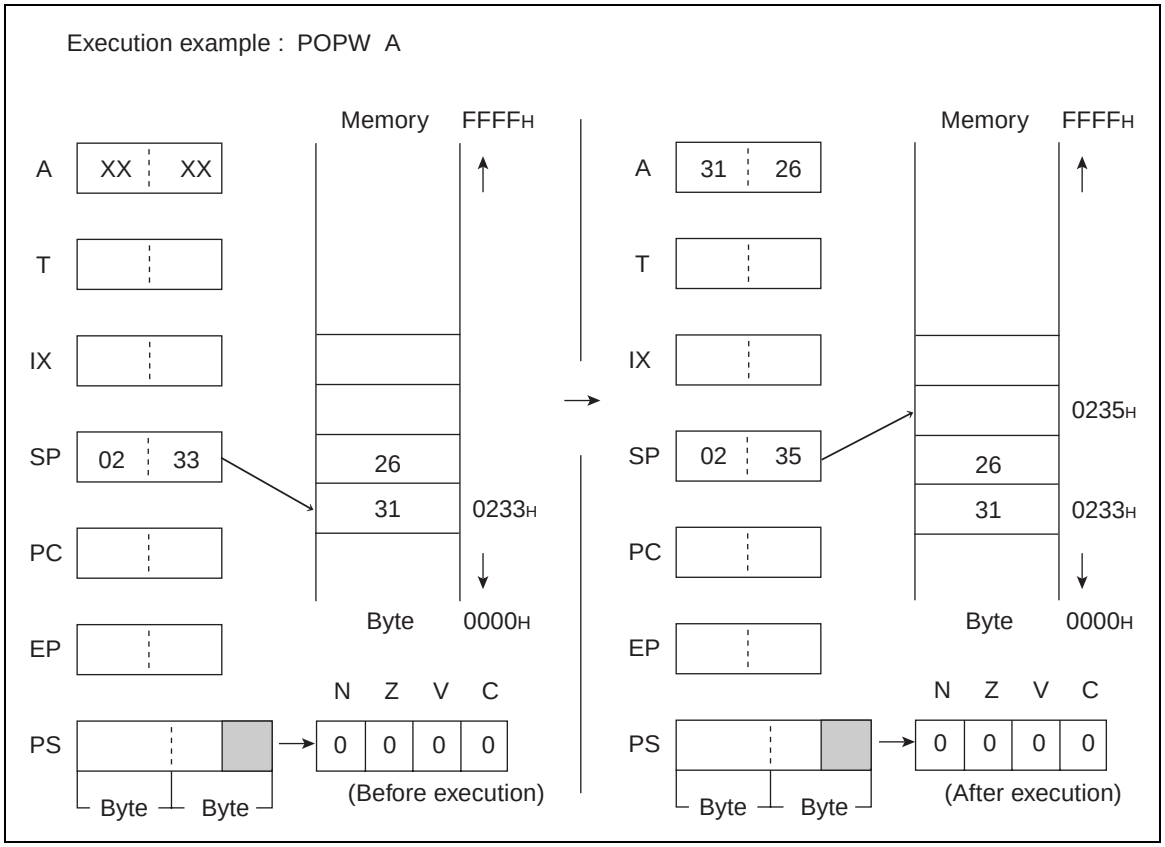
Z: Not changed

V: Not changed

C: Not changed

Table 6.66-1 Number of Execution Cycles / Byte Count / OP Code

DR	A	IX
Number of execution cycles	3	3
Byte count	1	1
OP code	50	51



6.67 RET (RETurn from subroutine)

Return the contents of PC saved in the stack. When this instruction is used in combination with the CALLV or CALL instruction, return to the next instruction to each of them.

■ RET (RETurn from subroutine)

Operation

(PC) ← ((SP)) (Word transfer)

(SP) ← (SP) + 2 (Word addition)

Assembler format

RET

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

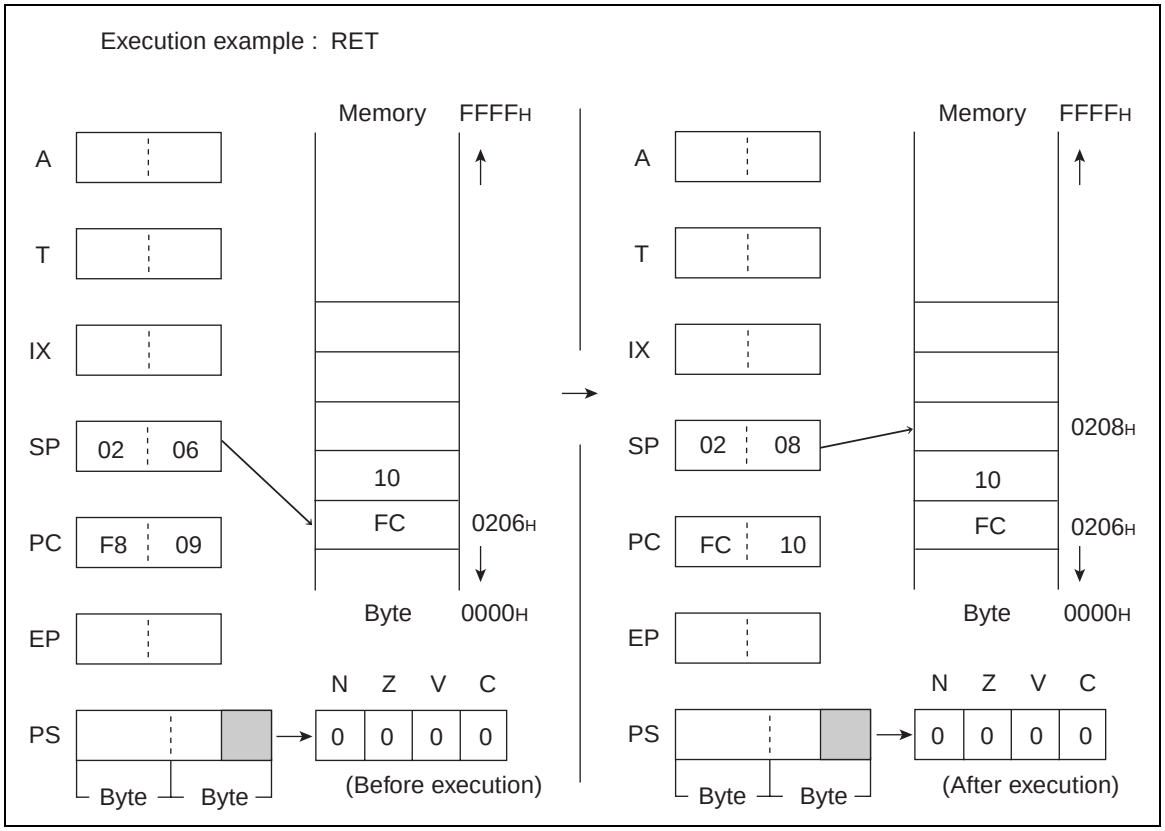
V: Not changed

C: Not changed

Number of execution cycles: 6

Byte count: 1

OP code: 20



6.68 RETI (RETurn from Interrupt)

Return the contents of PS and PC saved in the stack. Return PS and PC to the state before interrupt.

■ RETI (RETurn from Interrupt)

Operation

(PS) ← ((SP)), (PC) ← ((SP + 2)) (Word transfer)

(SP) ← (SP) + 4 (Word addition)

Assembler format

RETI

Condition code (CCR)

N	Z	V	C
+	+	+	+

+: Changed by executing instruction

-: Not changed

N: Become to the saved value of N.

Z: Become to the saved value of Z.

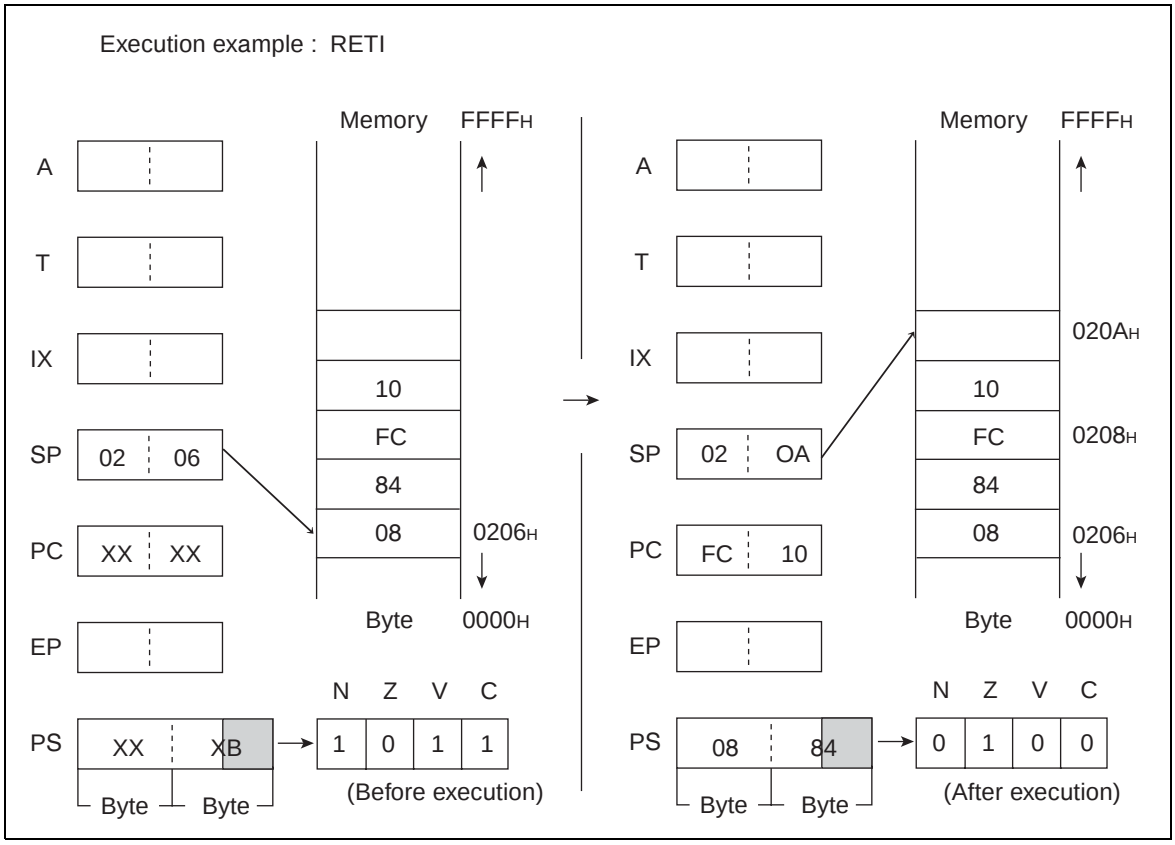
V: Become to the saved value of V.

C: Become to the saved value of C.

Number of execution cycles: 8

Byte count: 1

OP code: 30

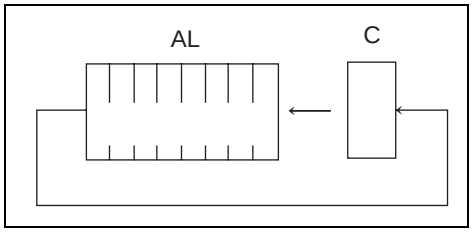


6.69 ROLC (Rotate Byte Data of Accumulator with Carry to Left)

Shift byte data of AL with a carry one bit to the left. The contents of AH are not changed.

■ ROLC (Rotate Byte Data of Accumulator with Carry to Left)

Operation



Assembler format

ROLC A

Condition code (CCR)

N	Z	V	C
+	+	-	+

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB is 1 as the result of the shift and set to 0 in other cases.

Z: Set to 1 if the result of the shift is 00_H and set to 0 in other cases.

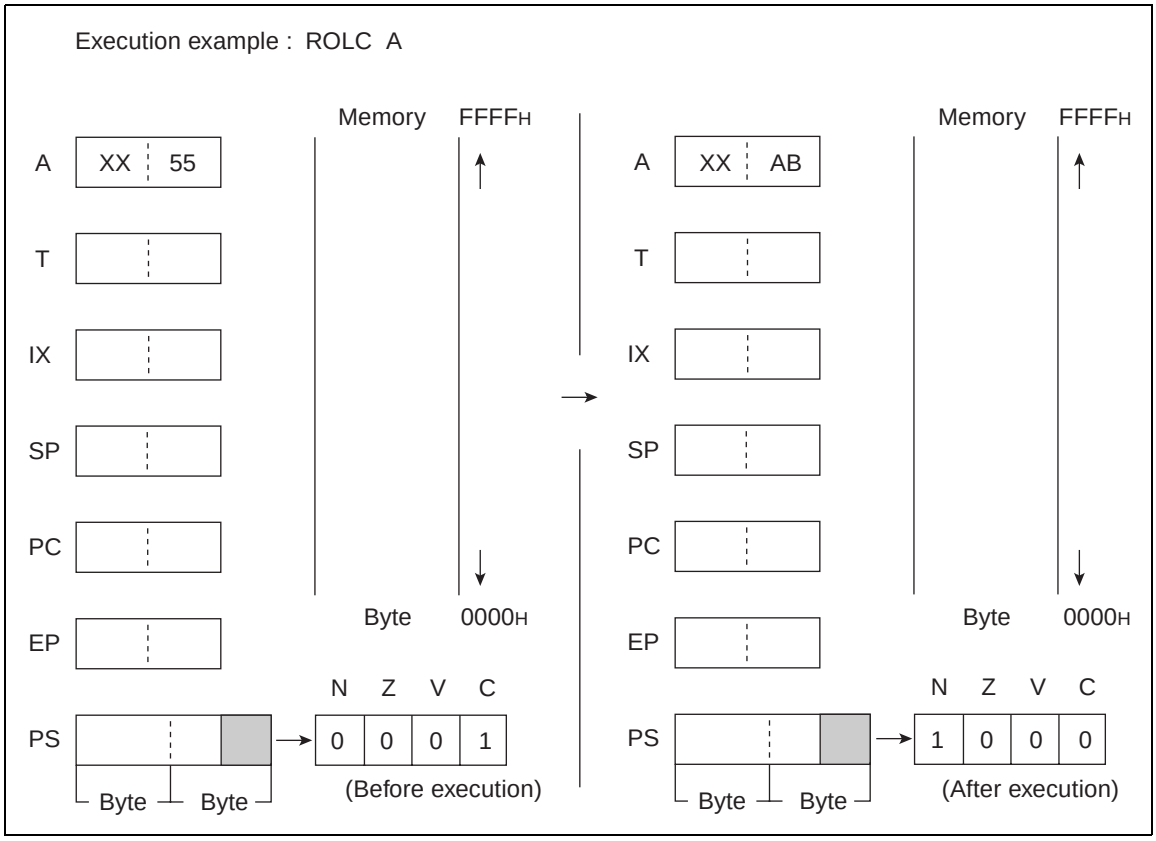
V: Not changed

C: Enter Bit 7 of A before shift.

Number of execution cycle: 1

Byte count: 1

OP code: 02

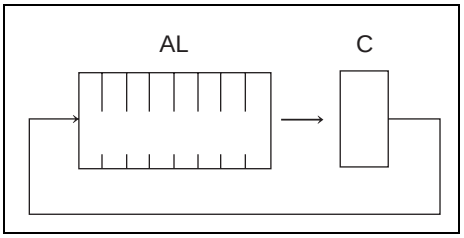


6.70 RORC (Rotate Byte Data of Accumulator with Carry to Right)

Shift byte data of AL with a carry bit to the right. The contents of AH are not changed.

■ RORC (Rotate Byte Data of Accumulator with Carry to Right)

Operation



Assembler format

RORC A

Condition code (CCR)

N	Z	V	C
+	+	-	+

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB is 1 as the result of the shift and set to 0 in other cases.

Z: Set to 1 if the result of the shift is 00_H and set to 0 in other cases.

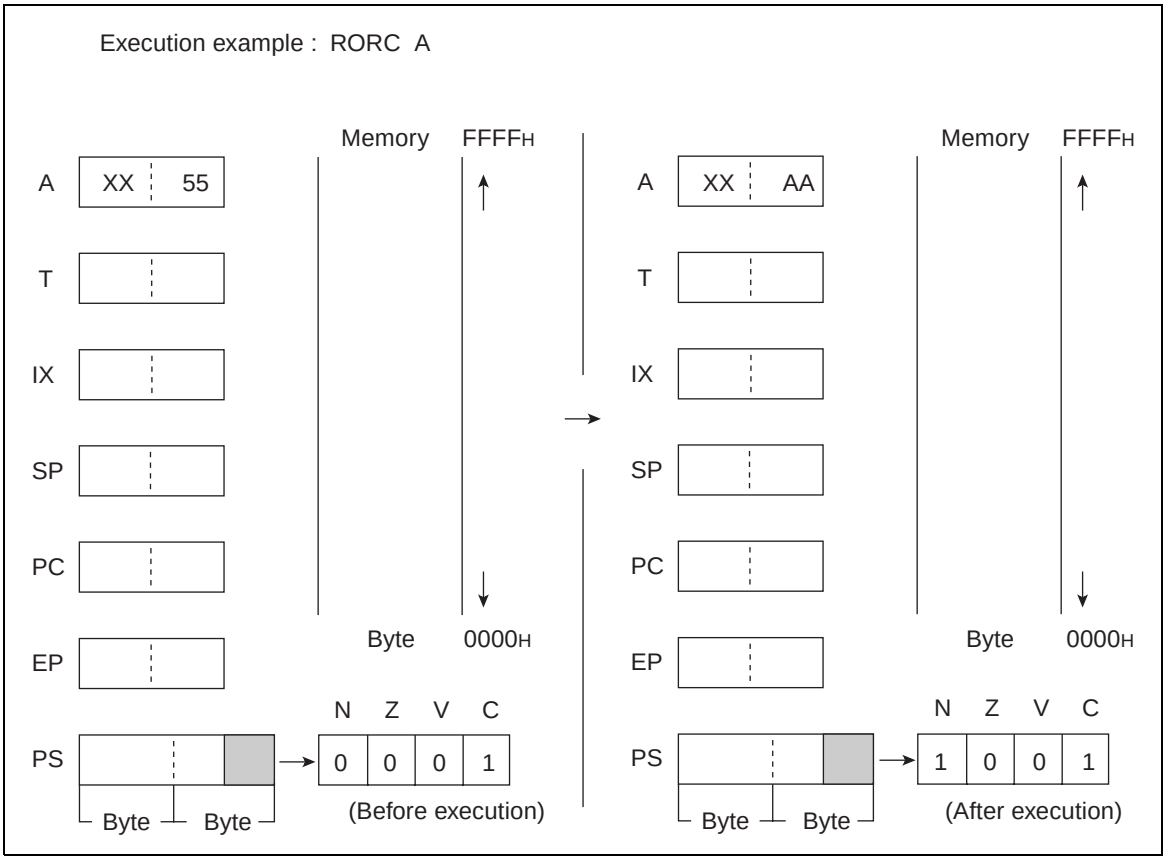
V: Not changed

C: LSB of A before entering shift

Number of execution cycle: 1

Byte count: 1

OP code: 03



6.71 SUBC (SUBtract Byte Data of Accumulator from Temporary Accumulator with Carry to Accumulator)

Subtract the byte data of AL from that of TL, subtract a carry and then return the result to AL. The contents of AH are not changed.

■ SUBC (SUBtract Byte Data of Accumulator from Temporary Accumulator with Carry to Accumulator)

Operation

(AL) ← (TL) - (AL) - C (Byte subtraction with carry)

Assembler format

SUBC A

Condition code (CCR)

N	Z	V	C
+	+	+	+

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB of AL is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

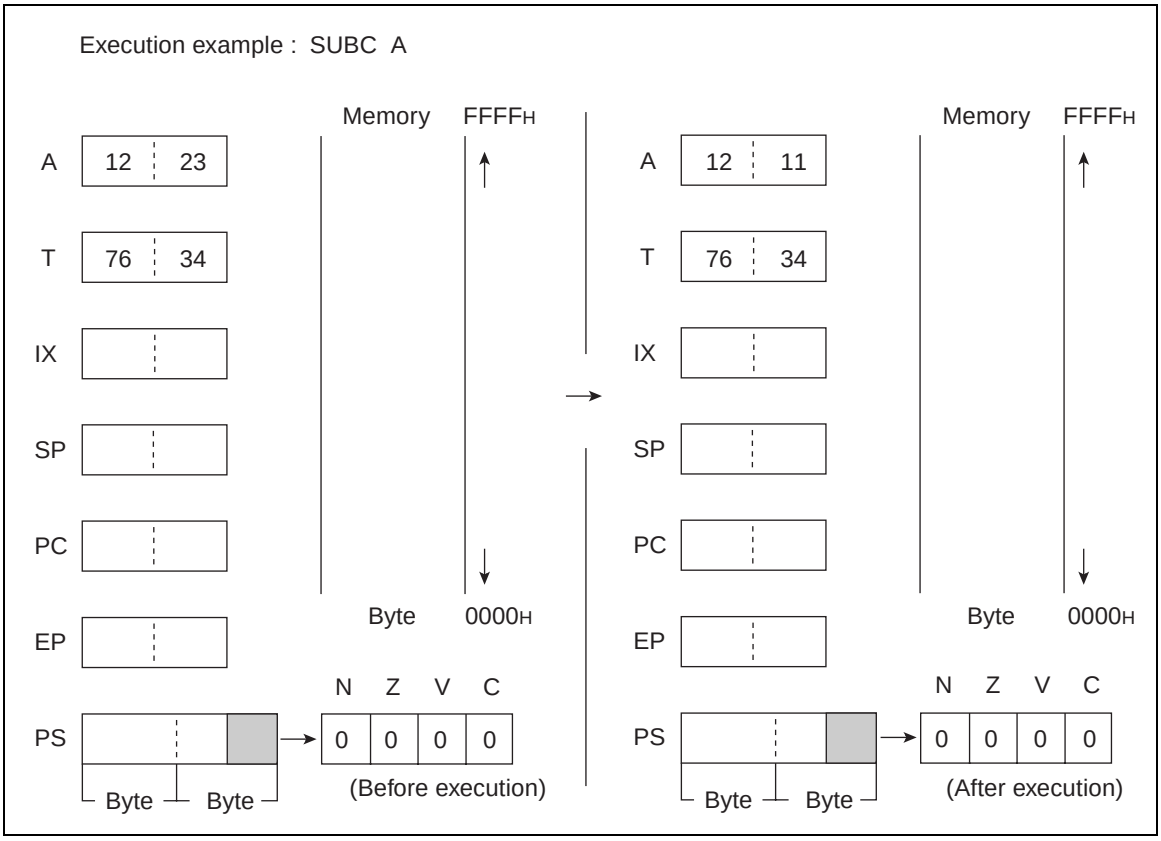
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

C: Set to 1 if a carry occurs as the result of operation and set to 0 in other cases.

Number of execution cycle: 1

Byte count: 1

OP code: 32



6.72 SUBC (SUBtract Byte Data of Memory from Accumulator with Carry to Accumulator)

Subtract the byte data of the EA memory (memory expressed in each type of addressing) from that of AL, subtract a carry and then return the results to AL. The contents of AH are not changed.

■ SUBC (SUBtract Byte Data of Memory from Accumulator with Carry to Accumulator)

Operation

(AL) ← (AL) - (EA) - C (Byte subtraction with carry)

Assembler format

SUBC A, EA

Condition code (CCR)

N	Z	V	C
+	+	+	+

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB of AL is 1 as the result of operation and set to 0 in other cases.

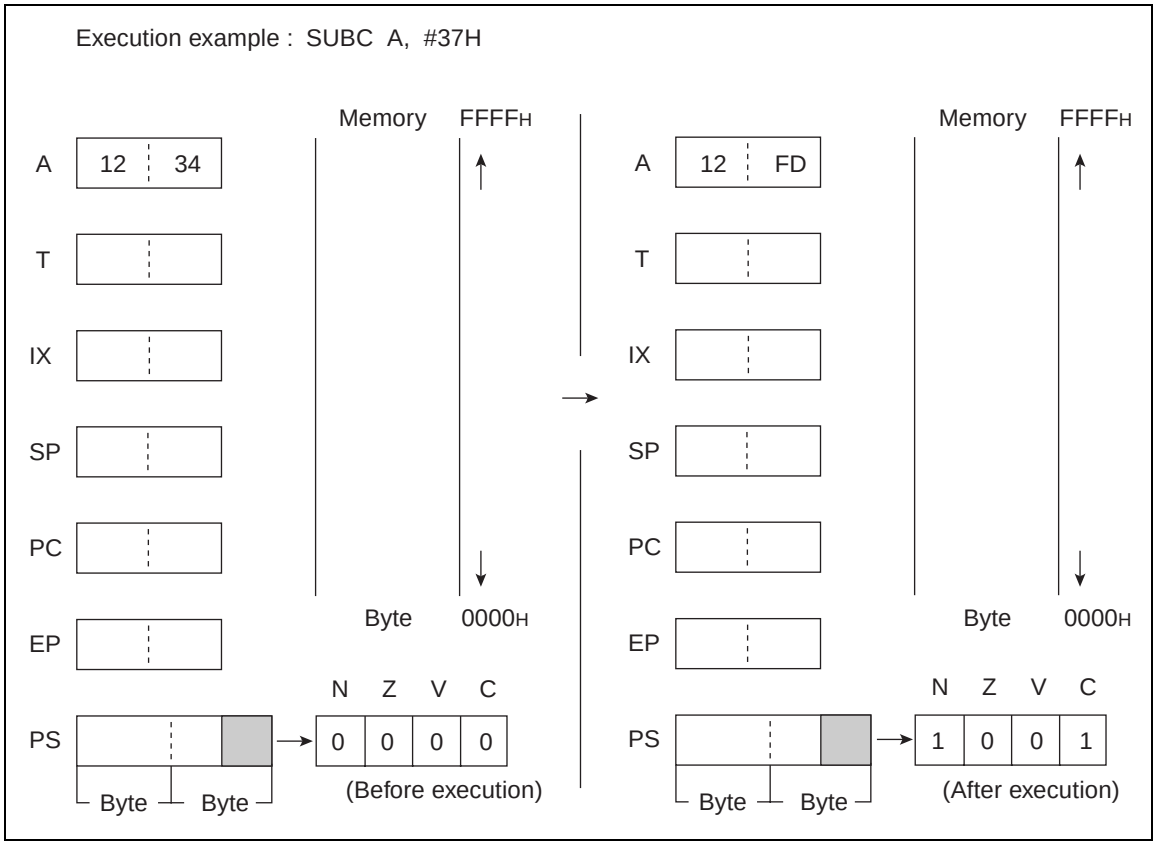
Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

C: Set to 1 if a carry occurs as the result of operation and set to 0 in other cases.

Table 6.72-1 Number of Execution Cycles / Byte Count / OP Code

EA	#d8	dir	@IX+off	@EP	Ri
Number of execution cycles	2	3	3	2	2
Byte count	2	2	2	1	1
OP code	34	35	36	37	38 to 3F



6.73 SUBCW (SUBtract Word Data of Accumulator from Temporary Accumulator with Carry to Accumulator)

Subtract the word data of A from that of T, subtract a carry and then return the result to A.

■ SUBCW (SUBtract Word Data of Accumulator from Temporary Accumulator with Carry to Accumulator)

Operation

(AL) ← (T) - (A) - C (Word subtraction with carry)

Assembler format

SUBCW A

Condition code (CCR)

N	Z	V	C
+	+	+	+

+: Changed by executing instruction

-: Not changed

N: Set to 1 if the MSB of A is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 0000_H and set to 0 in other cases.

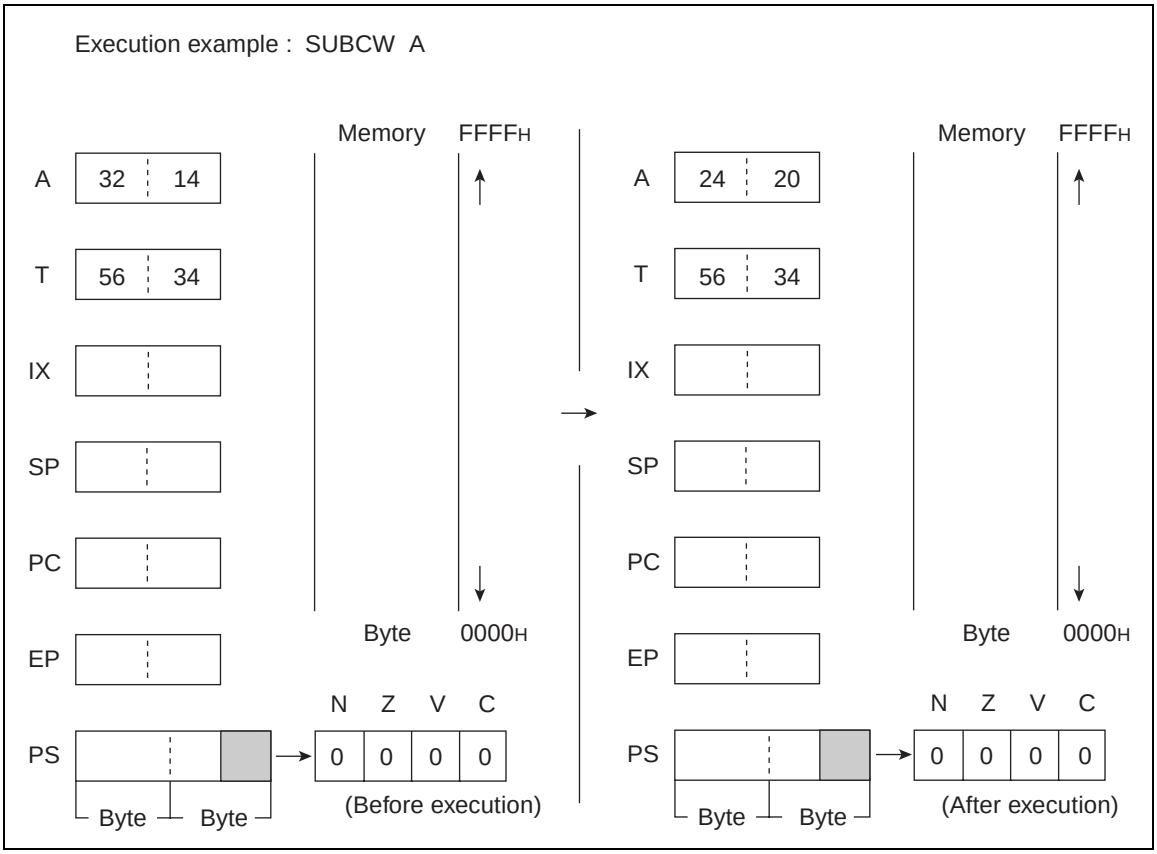
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

C: Set to 1 if a carry occurs as the result of operation and set to 0 in other cases.

Number of execution cycle: 1

Byte count: 1

OP code: 33



6.74 SETB (Set Direct Memory Bit)

Set the contents of 1 bit (indicated by 3 lower bits (b) of mnemonic) for the direct area to 1.

■ SETB (Set Direct Memory Bit)

Operation

(dir:b) ← 1

Assembler format

SETB dir:b

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

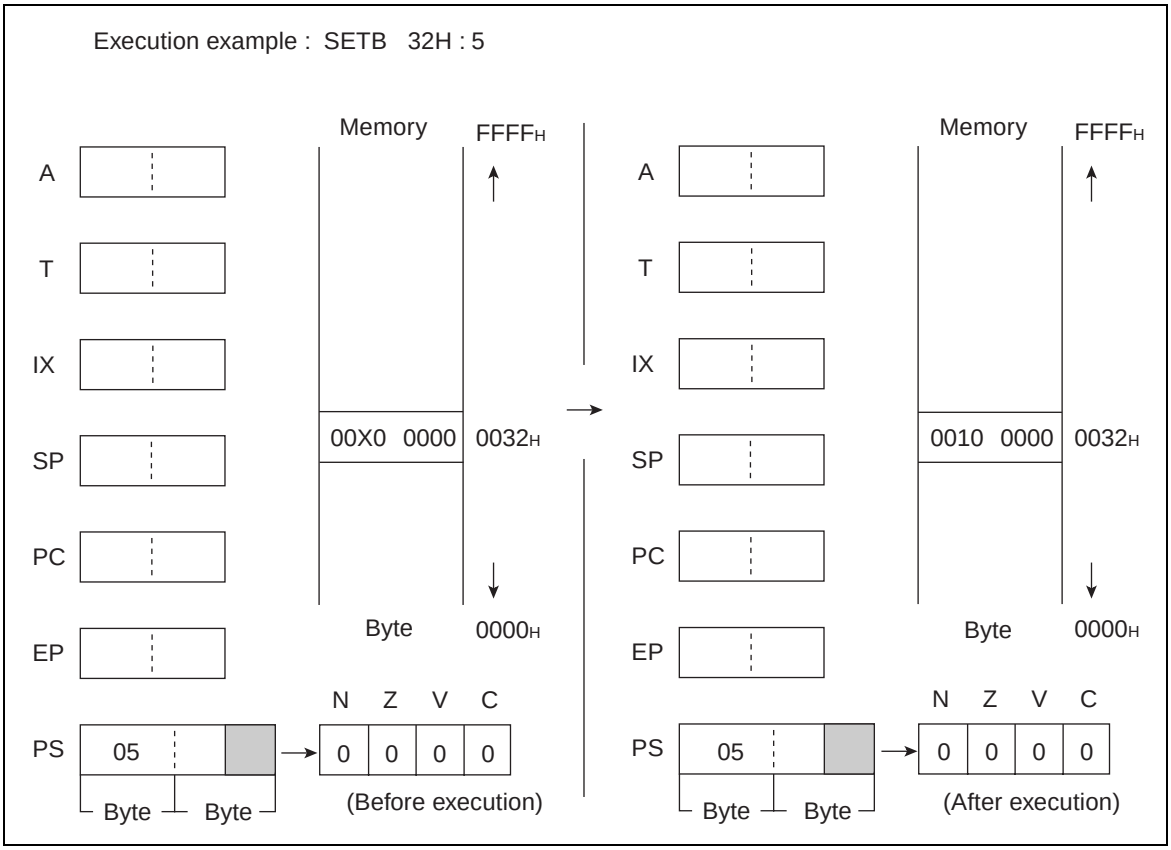
V: Not changed

C: Not changed

Number of execution cycles: 4

Byte count: 2

OP code: A8 to AF



6.75 SETC (SET Carry flag)

Set the C-flag to 1.

■ SETC (SET Carry flag)

Operation

(C) ← 1

Assembler format

SETC

Condition code (CCR)

N	Z	V	C
-	-	-	S

+: Changed by executing instruction

-: Not changed

S: Set to 1 by executing instruction

N: Not changed

Z: Not changed

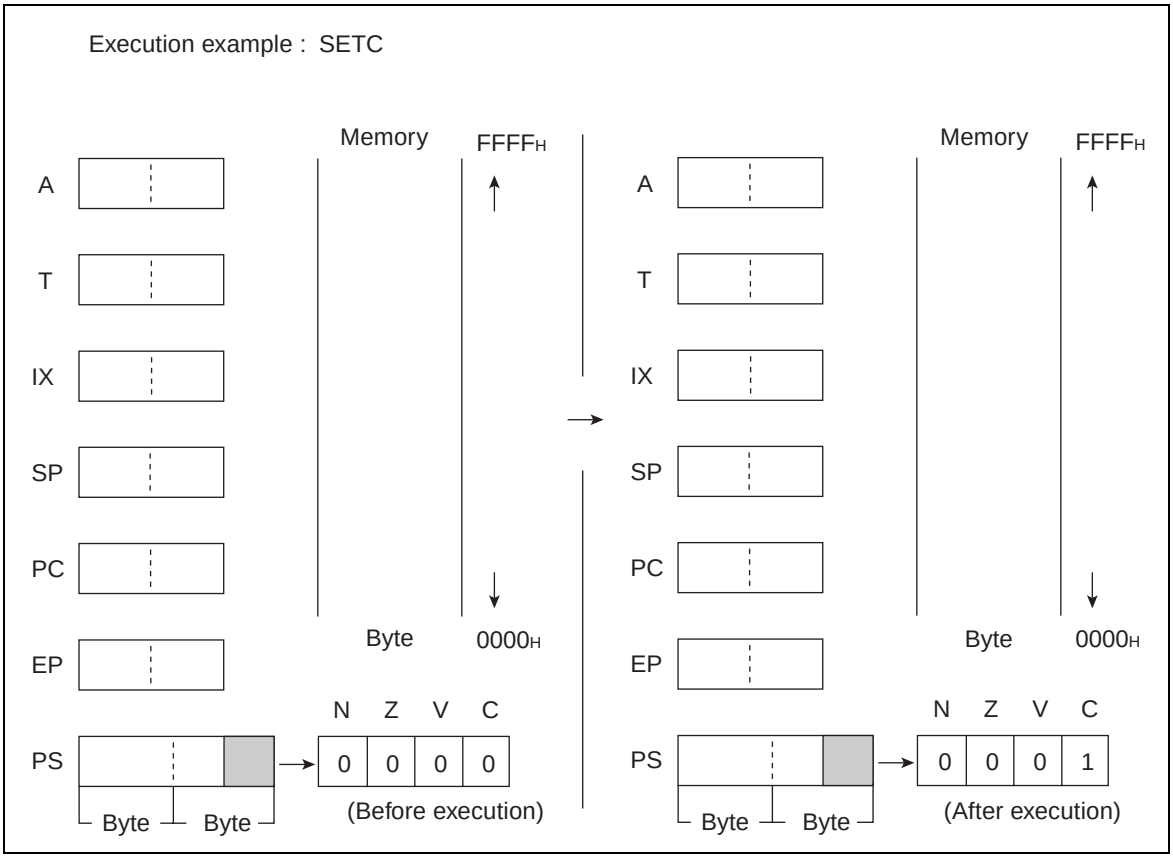
V: Not changed

C: Set to 1

Number of execution cycle: 1

Byte count: 1

OP code: 91



6.76 SETI (SET Interrupt flag)

Set the I-flag to 1 (enable an interrupt).

■ SETI (SET Interrupt flag)

Operation

(I) ← 1

Assembler format

SETI

Condition code (CCR)

I	N	Z	V	C
S	-	-	-	-

+: Changed by executing instruction

-: Not changed

S: Set to 1 by executing instruction

I: Set to 1

N: Not changed

Z: Not changed

V: Not changed

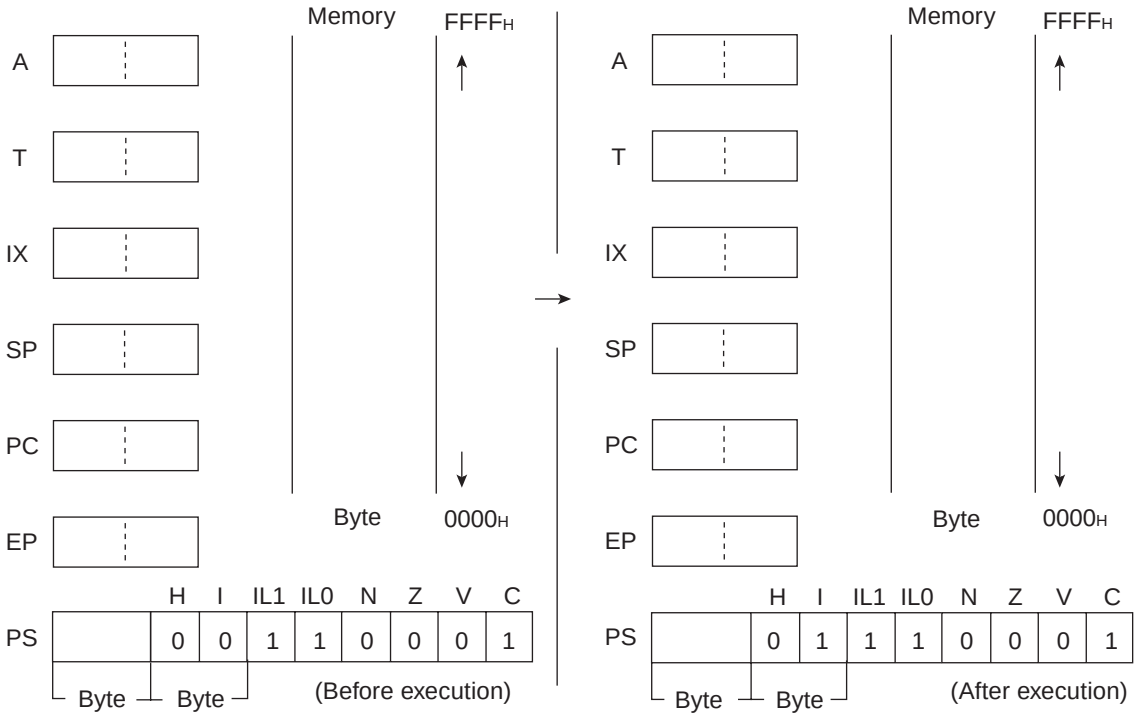
C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: 90

Execution example : SETI



6.77 SWAP (SWAP Byte Data Accumulator "H" and Accumulator "L")

Exchange the byte data of AH for that of AL.

■ SWAP (SWAP Byte Data Accumulator "H" and Accumulator "L")

Operation

(AH) ↔ (AL) (Byte data exchange)

Assembler format

SWAP

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

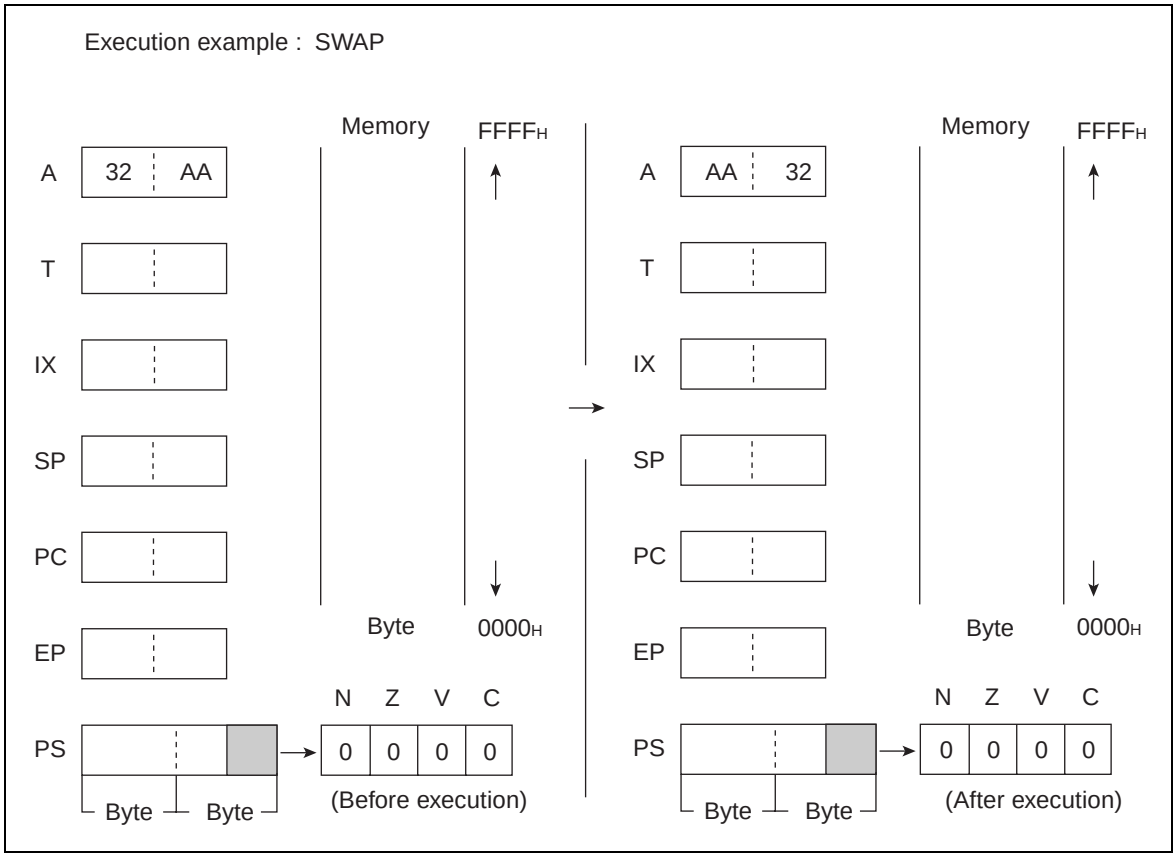
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: 10



6.78 XCH (eXCHange Byte Data Accumulator "L" and Temporary Accumulator "L")

Exchange the byte data of AL for that of TL.

■ XCH (eXCHange Byte Data Accumulator "L" and Temporary Accumulator "L")

Operation

(AL) ↔ (TL) (conversion of byte data)

Assembler format

XCH A, T

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

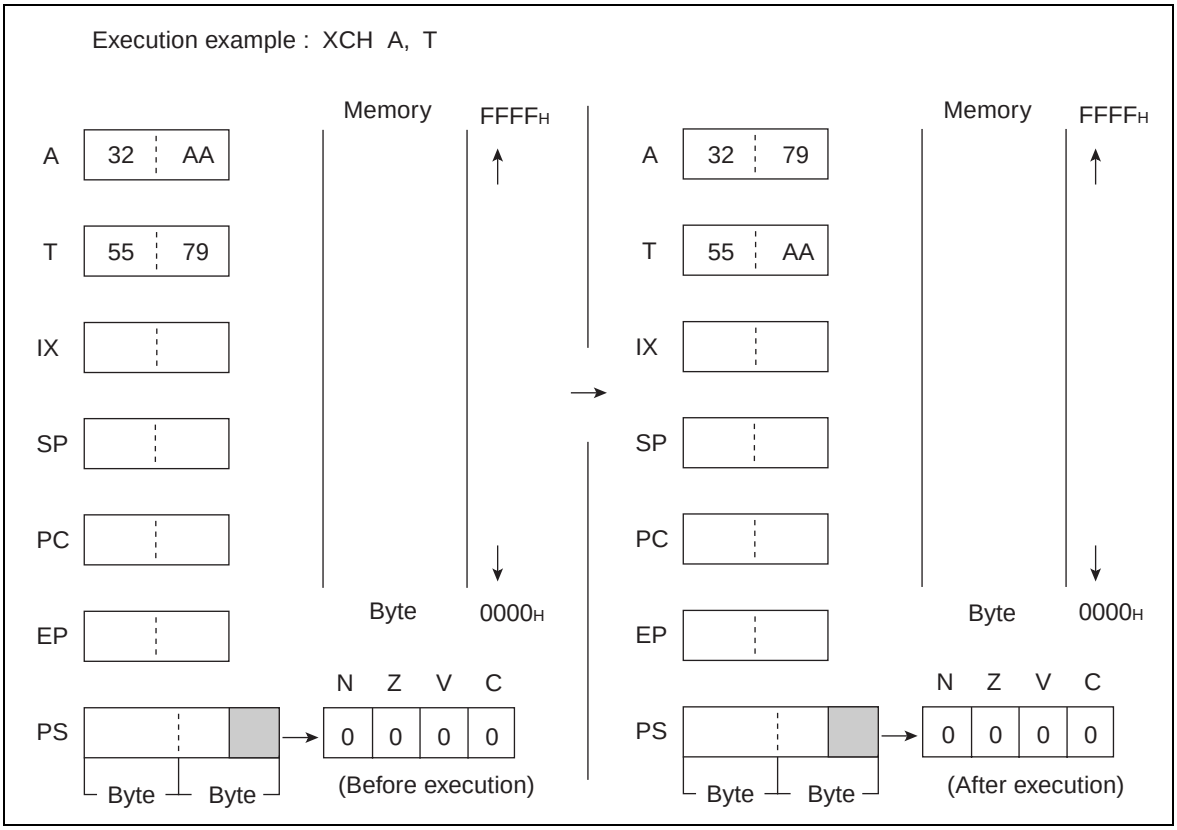
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: 42



6.79 XCHW (eXCHange Word Data Accumulator and Extrapointer)

Exchange the word data of A for that of EP.

■ XCHW (eXCHange Word Data Accumulator and Extrapointer)

Operation

(A) ↔ (EP) (conversion of word data)

Assembler format

XCHW A, EP

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

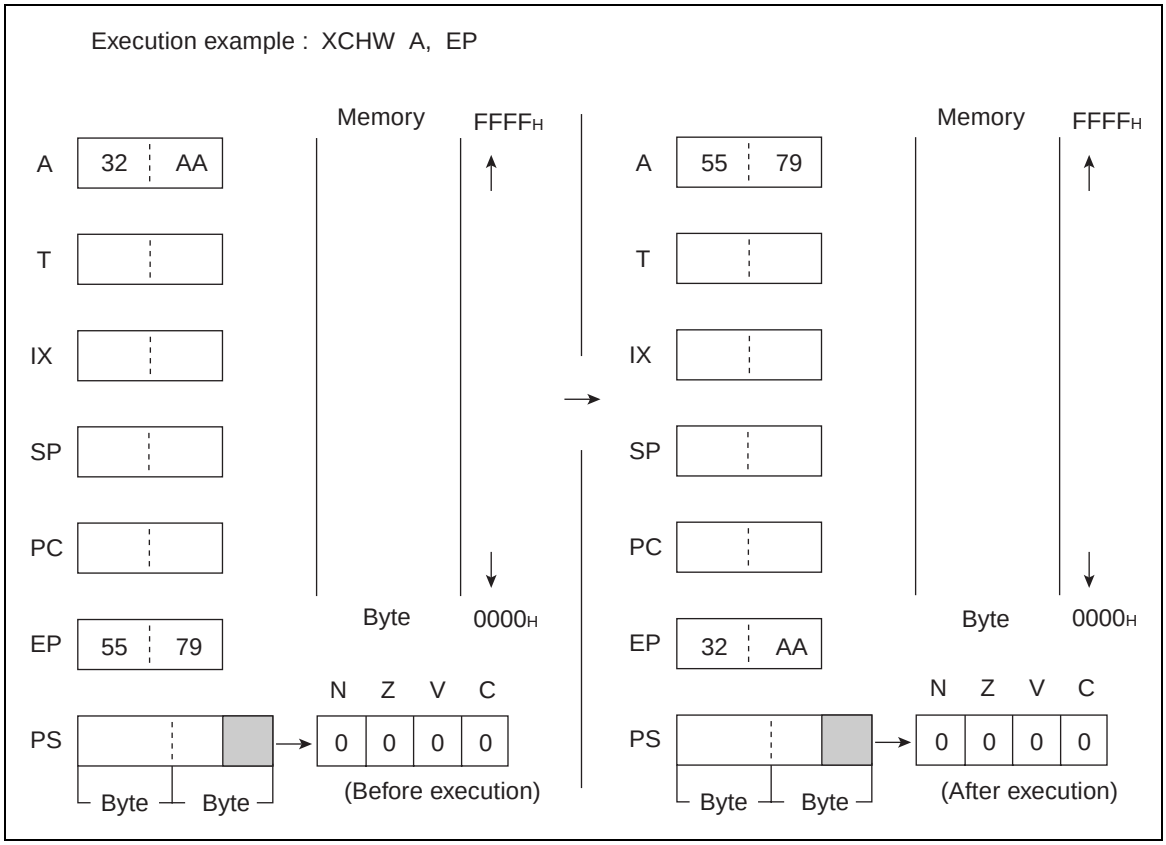
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: F7



6.80 XCHW (eXCHange Word Data Accumulator and Index Register)

Exchange the word data of A for that of IX.

■ XCHW (eXCHange Word Data Accumulator and Index Register)

Operation

(A) ↔ (IX) (conversion of word data)

Assembler format

XCHW A, IX

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

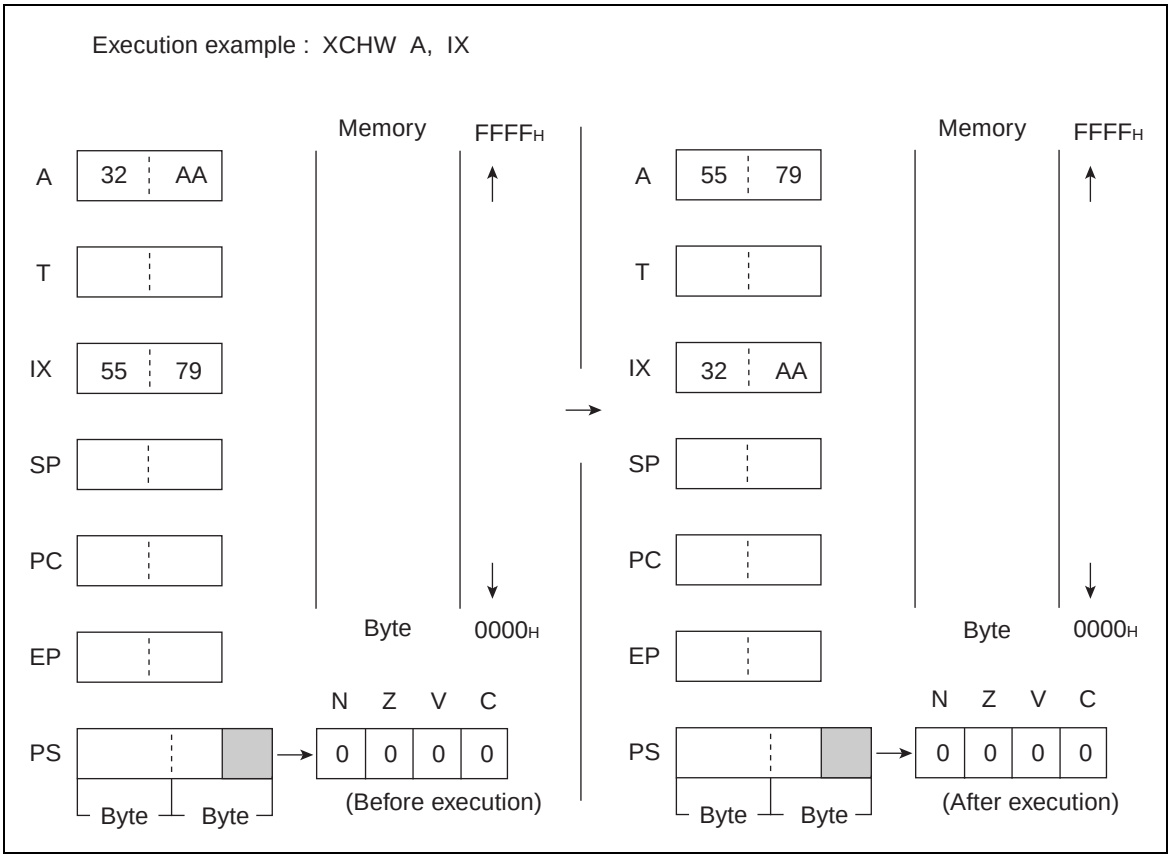
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: F6



6.81 XCHW (eXCHange Word Data Accumulator and Program Counter)

Exchange the word data of PC for that of A.

■ XCHW (eXCHange Word Data Accumulator and Program Counter)

Operation

(PC) ← (A) (word transfer)

(A) ← (PC) +1 (word addition, word transfer)

Assembler format

XCHW A, PC

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

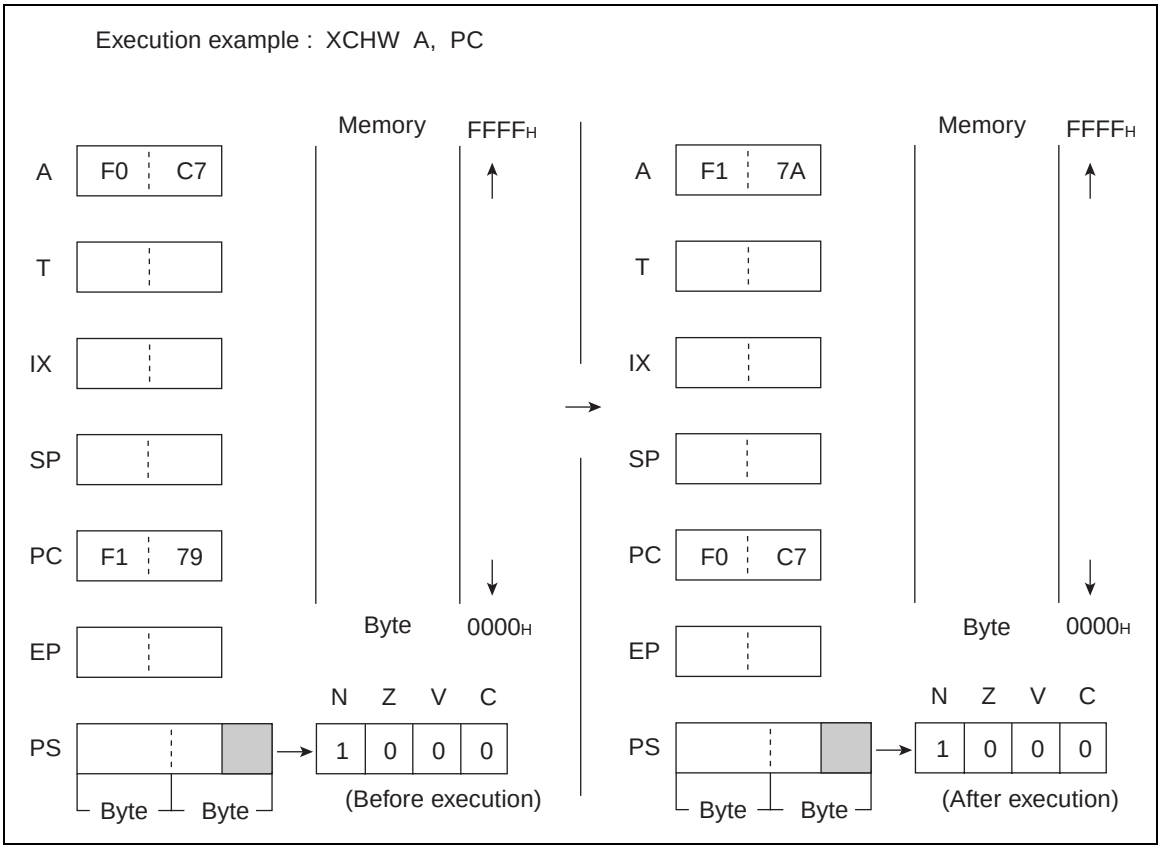
V: Not changed

C: Not changed

Number of execution cycles: 3

Byte count: 1

OP code: F4



6.82 XCHW (eXCHange Word Data Accumulator and Stack Pointer)

Exchange the word data of A for that of SP.

■ XCHW (eXCHange Word Data Accumulator and Stack Pointer)

Operation

(A) ↔ (SP) (conversion of word data)

Assembler format

XCHW A, SP

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

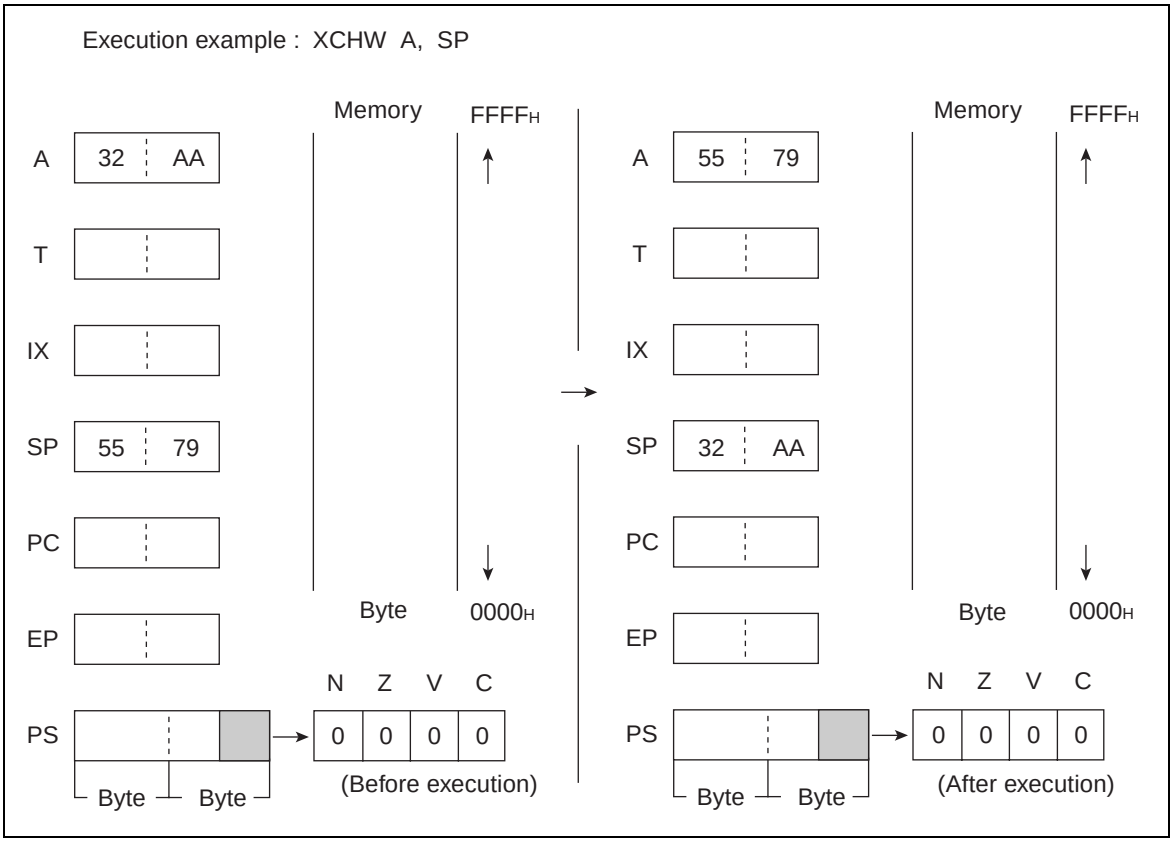
V: Not changed

C: Not changed

Number of execution cycles: 2

Byte count: 1

OP code: F5



6.83 XCHW (eXCHange Word Data Accumulator and Temporary Accumulator)

Exchange the word data of A for that of T.

■ XCHW (eXCHange Word Data Accumulator and Temporary Accumulator)

Operation

(A) ↔ (T) (conversion of word data)

Assembler format

XCHW A, T

Condition code (CCR)

N	Z	V	C
-	-	-	-

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

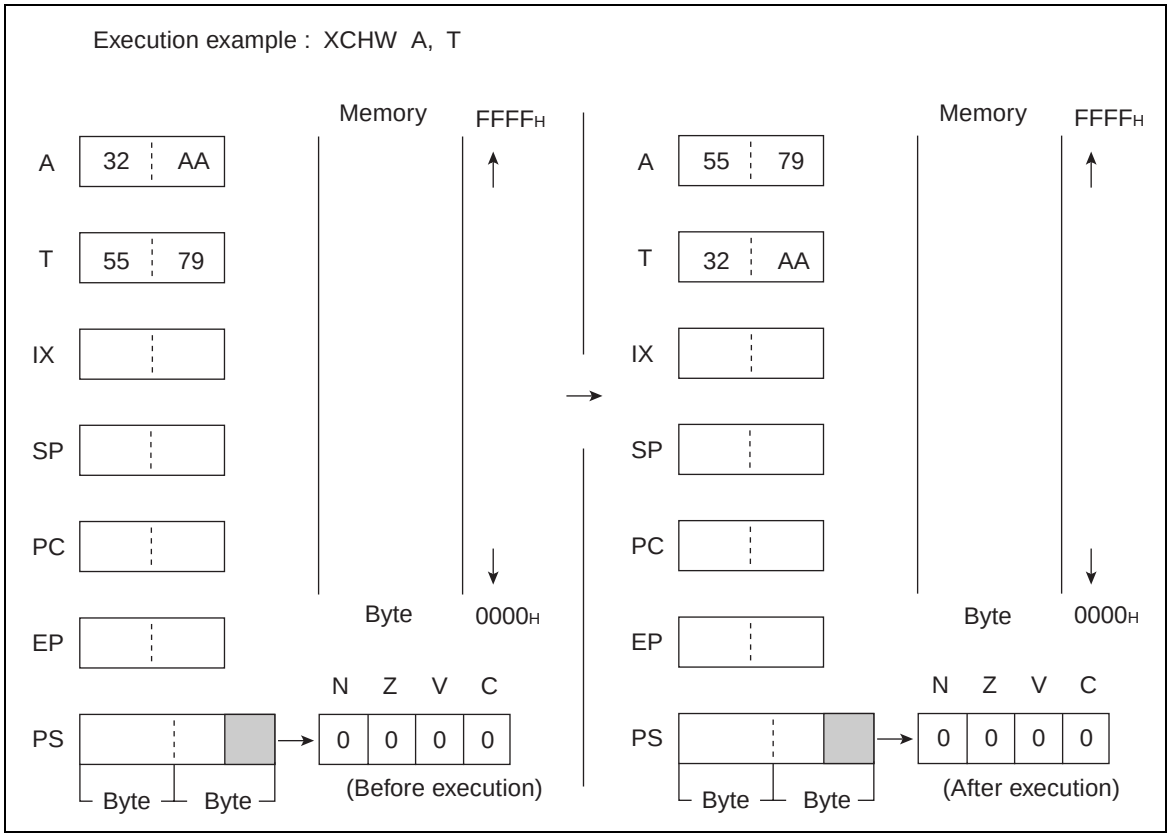
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: 43



6.84 XOR (eXclusive OR Byte Data of Accumulator and Temporary Accumulator to Accumulator)

Carry out the logical exclusive-OR on the byte data of AL and TL for every bit and return the results to AL. The contents of AH are not changed.

■ XOR (eXclusive OR Byte Data of Accumulator and Temporary Accumulator to Accumulator)

Operation

(AL) ← (AL) ∀ (TL) (byte logical exclusive-OR)

Assembler format

XOR A

Condition code (CCR)

N	Z	V	C
+	+	R	-

+: Changed by executing instruction

-: Not changed

R: Set to 0 by executing instruction

N: Set to 1 if the MSB of AL is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

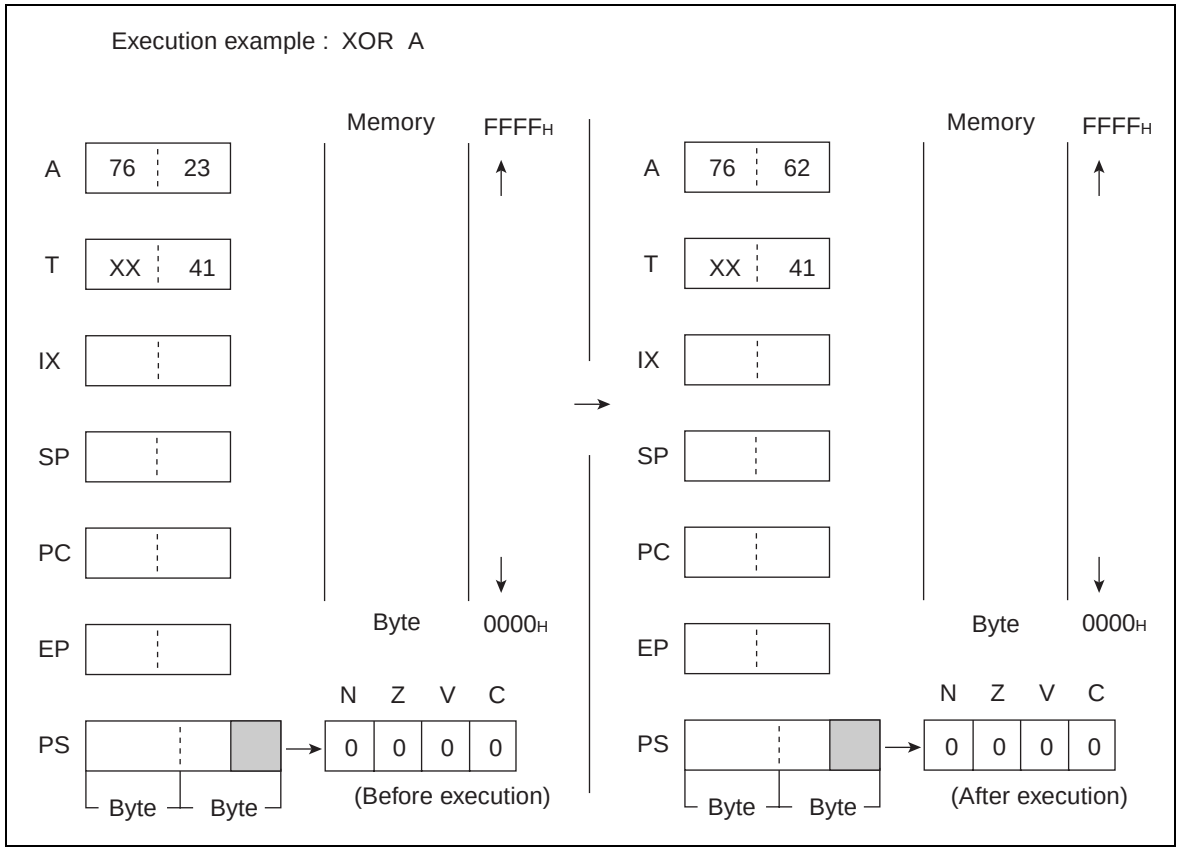
V: Always set to 0

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: 52



6.85 XOR (eXclusive OR Byte Data of Accumulator and Memory to Accumulator)

Carry out the logical exclusive-OR for the byte data of AL and EA memory (memory expressed in each type of addressing) for every bit and return the results to AL. The contents of AH are not changed.

■ XOR (eXclusive OR Byte Data of Accumulator and Memory to Accumulator)

Operation

(AL) ← (AL) ∀ (EA) (byte logical exclusive-OR)

Assembler format

XOR A, EA

Condition code (CCR)

N	Z	V	C
+	+	R	-

+: Changed by executing instruction

-: Not changed

R: Set to 0 by executing instruction

N: Set to 1 if the MSB of AL is 1 as the result of operation and set to 0 in other cases.

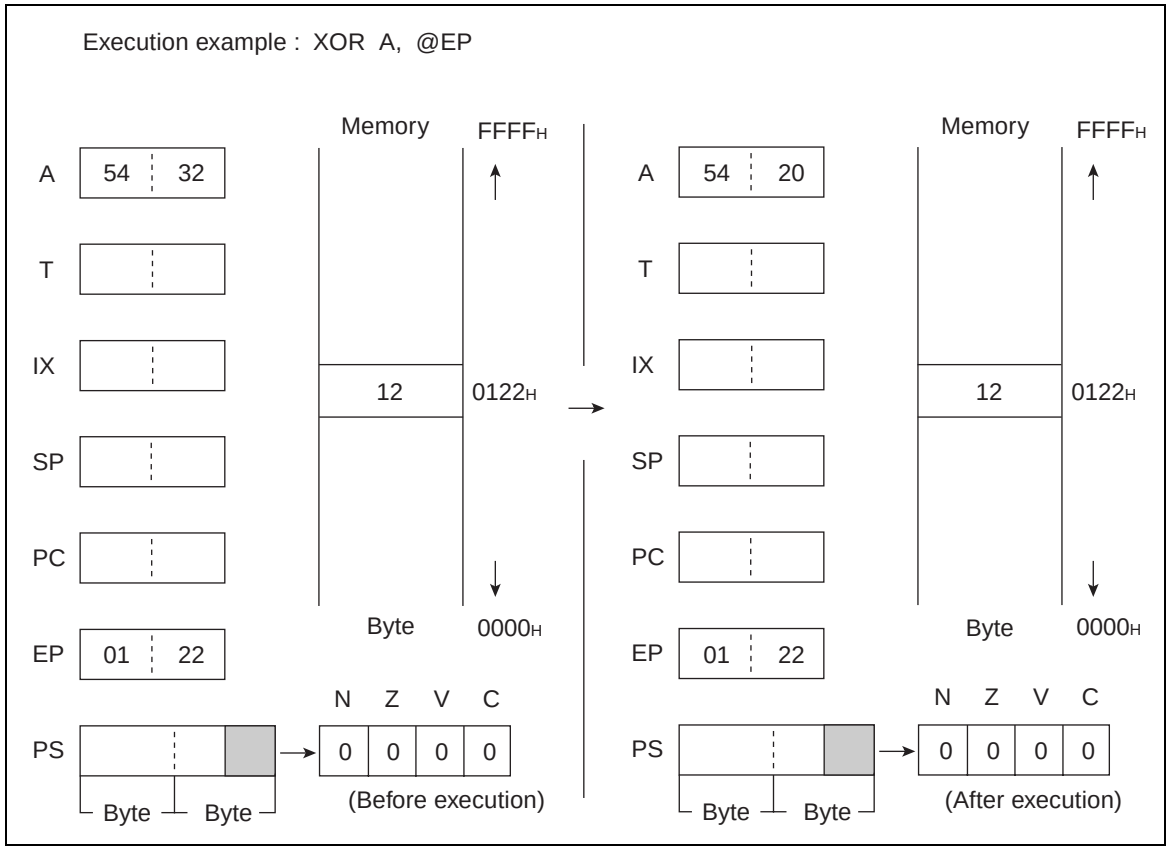
Z: Set to 1 if the result of operation is 00_H and set to 0 in other cases.

V: Always set to 0

C: Not changed

Table 6.85-1 Number of Execution Cycles / Byte Count / OP Code

EA	#d8	dir	@IX+off	@EP	Ri
Number of execution cycles	2	3	3	2	2
Byte count	2	2	2	1	1
OP code	54	55	56	57	58 to 5F



6.86 XORW (eXclusive OR Word Data of Accumulator and Temporary Accumulator to Accmulator)

Carry out the logical exclusive-OR on the word data of A and T for every bit and return the results to A.

■ XORW (eXclusive OR Word Data of Accumulator and Temporary Accumulator to Accmulator)

Operation

(A) ← (A) ∀ (T) (word logical exclusive-OR)

Assembler format

XORW A

Condition code (CCR)

N	Z	V	C
+	+	R	-

+: Changed by executing instruction

-: Not changed

R: Set to 0 by executing instruction

N: Set to 1 if the MSB of A is 1 as the result of operation and set to 0 in other cases.

Z: Set to 1 if the result of operation is 0000_H and set to 0 in other cases.

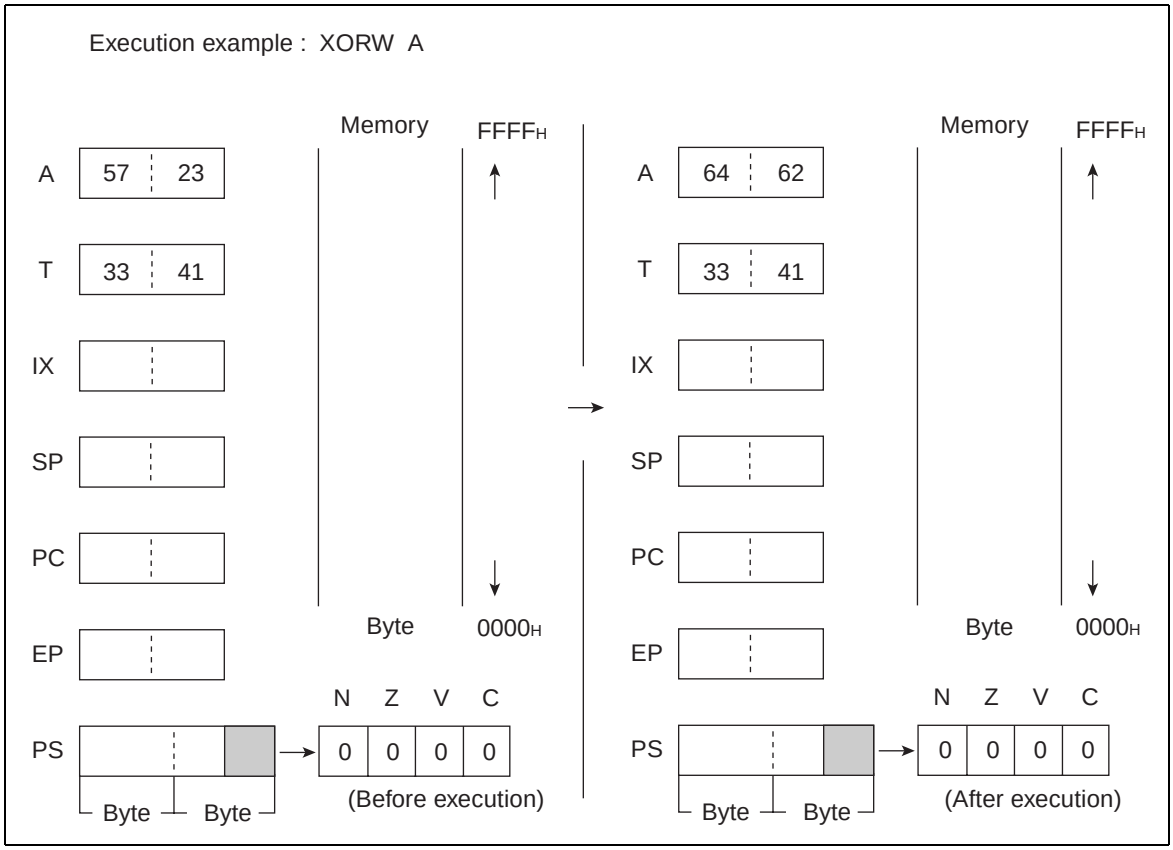
V: Always set to 0

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: 53



APPENDIX

The appendix contains instruction and bus operation lists and an instruction map.

APPENDIX A Instruction List

Appendix A contains lists of instructions used in the assembler.

A.1 F²MC-8FX CPU Instruction Overview

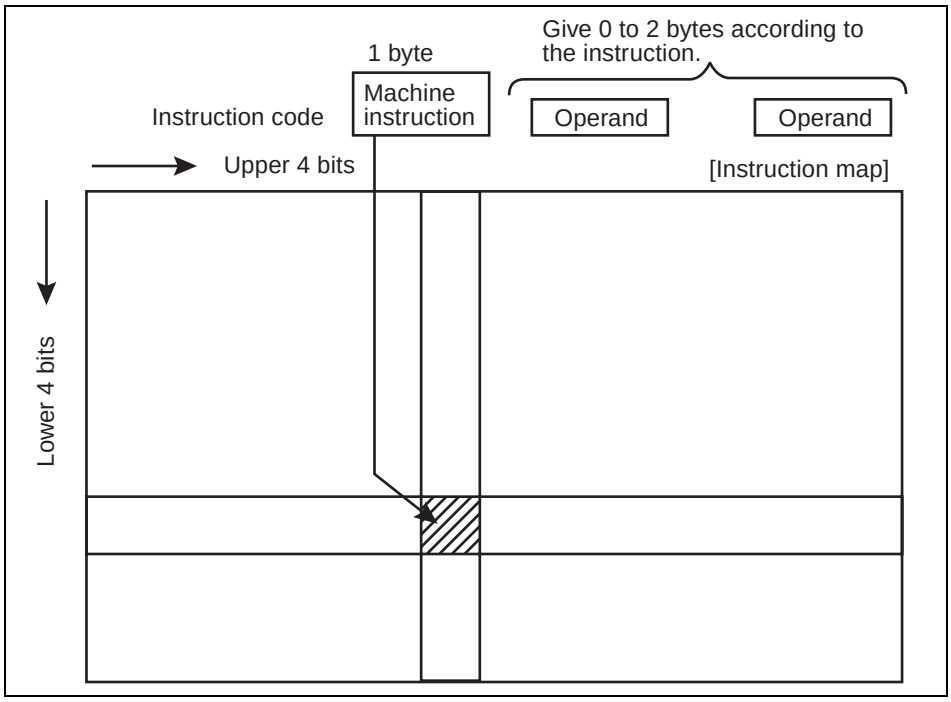
This section explains the F²MC-8FX CPU instructions.

■ F²MC-8FX CPU Instruction Overview

In the F²MC-8FX CPU, there is 140 kinds of one byte machine instruction (as the map, 256 bytes), and the instruction code is composed of the instruction and the operand following it.

Figure A.1-1 shows the instruction code and the correspondence of the instruction map.

Figure A.1-1 Correspondence between the Instruction Code and the Instruction Map



The following are enumerated as a feature of F²MC-8FX CPU instruction.

- The instruction is classified into 4 types: transfer, operation, branch, and others.
- There is various methods of address specification, and ten kinds of addressing can be selected by the selection of the instruction and the operand specification.
- It provides with the bit operation instruction, and the read modification write can operate.
- There is an instruction that directs special operation.

■ Sign of the Instruction List

Table A.1-1 explains the sign used by describing the instruction code in the table.

Table A.1-1 Sign of the Instruction List

Notation	Meaning
dir	Direct address (8 bits)
off	Offset (8 bits)
ext	Extended address (16 bits)
#vct	Vector table number (3 bits)
#d8	Immediate data (8 bits)
#d16	Immediate data (16 bits)
dir:b	Bit direct address (8 bits: 3 bits)
rel	Relative branch address (8 bits)
@	Register indirect (example: @A, @IX, @EP)
A	Accumulator (8-bit or 16-bit length is determined by instruction to be used.)
AH	Upper 8 bits of accumulator (8 bits)
AL	Lower 8 bits of accumulator (8 bits)
T	Temporary accumulator (8-bit or 16-bit length is determined by instruction to be used.)
TH	Upper 8 bits of temporary accumulator (8 bits)
TL	Lower 8 bits of temporary accumulator (8 bits)
IX	Index register (16 bits)
EP	Extra pointer (16 bits)
PC	Program counter (16 bits)
SP	Stack pointer (16 bits)
PS	Program status (16 bits)
dr	Accumulator or index register (16 bits)
CCR	Condition code register (8 bits)
RP	Register bank pointer (5 bits)
DP	Direct bank pointer (3 bits)
Ri	General-purpose register (8 bits, i = 0 to 7)
X	X indicates immediate data. (8-bit or 16-bit length is determined by instruction to be used.)
(X)	The contents of X are to be accessed. (8-bit or 16-bit length is determined by instruction to be used.)
((X))	The address indicated by the contents of X is to be accessed. (8-bit or 16-bit length is determined by instruction to be used.)

■ Item in Instruction Table

Table A.1-2 explains the item of instruction table.

Table A.1-2 Item in Instruction Table

Item	Description
NMEMONIC	The assembly description of the instruction is shown.
RD	The read of an internal bus is shown.
WR	The write of an internal bus is shown.
RMW	The read modification write signal of an internal bus is shown.
~	Cycle of the instruction number is shown. One instruction cycle is one machine cycle. Note: The instruction cycle number might be postponed one cycle by the immediately preceding instruction. Moreover, cycle of the instruction number might be extended in the access to the IO area.
#	The number of bytes for the instruction is shown.
Operation	The operation of the instruction is shown.
TL, TH, AH	The change in the content when TL, TH, and AH each instruction is executed is shown. The sign in the column shows the following respectively. • - : Do not change. • dH : Upper 8 bits of the data written in operation • AL, AH : Become the contents of AL or AH immediately before instruction. • 00 : Become 00.
N, Z, V, C	The flag changed when each instruction is executed is shown. The sign in the column shows the following respectively. • - : Do not change. • + : Change. • R : Become 0. • S : Become 1.
OP CODE	The code of the instruction is shown. When a pertinent instruction occupies two or more codes, it follows the following description rules. 48 to 4F: 48, 49, ..., 4F are shown.

A.2 Operation List

Table A.2-1 is the operation list for transfer instructions. Table A.2-2 is the operation list for operation instructions. Table A.2-3 is the operation list for branch instructions. Table A.2-4 is the operation list for other instructions.

■ Operation List

Table A.2-1 Operation List (for Transfer Instructions) (1/3)

No	MNEMONIC	~	#	OPERATION	TL	TH	AH	N Z V C	OP CODE
1	MOV dir, A	3	2	(dir) ← (A)	–	–	–	– – – –	45
2	MOV @IX+off, A	3	2	(IX)+off) ← (A)	–	–	–	– – – –	46
3	MOV ext, A	4	3	(ext) ← (A)	–	–	–	– – – –	61
4	MOV @EP, A	2	1	((EP)) ← (A)	–	–	–	– – – –	47
5	MOV Ri, A	2	1	(Ri) ← (A)	–	–	–	– – – –	48 to 4F
6	MOV A, #d8	2	2	(A) ← d8	AL	–	–	+ + – –	04
7	MOV A, dir	3	2	(A) ← (dir)	AL	–	–	+ + – –	05
8	MOV A, @IX+off	3	2	(A) ← ((IX)+off)	AL	–	–	+ + – –	06
9	MOV A, ext	4	3	(A) ← (ext)	AL	–	–	+ + – –	60
10	MOV A, @A	2	1	(A) ← ((A))	AL	–	–	+ + – –	92
11	MOV A, @EP	2	1	(A) ← ((EP))	AL	–	–	+ + – –	07
12	MOV A, Ri	2	1	(A) ← (Ri)	AL	–	–	+ + – –	08 to 0F
13	MOV dir, #d8	4	3	(dir) ← d8	–	–	–	– – – –	85
14	MOV @IX+off, #d8	4	3	((IX)+off) ← d8	–	–	–	– – – –	86
15	MOV @EP, #d8	3	2	((EP)) ← d8	–	–	–	– – – –	87
16	MOV Ri, #d8	3	2	(Ri) ← d8	–	–	–	– – – –	88 to 8F
17	MOVW dir, A	4	2	(dir) ← (AH), (dir+1)←(AL)	–	–	–	– – – –	D5
18	MOVW @IX+off, A	4	2	((IX)+off) ← (AH), ((IX)+off+1) ← (AL)	–	–	–	– – – –	D6

Table A.2-1 Operation List (for Transfer Instructions) (2/3)

No	MNEMONIC	~	#	OPERATION	TL	TH	AH	N Z V C	OP CODE
19	MOVW ext, A	5	3	(ext) ← (AH), (ext+1) ← (AL)	–	–	–	– – – –	D4
20	MOVW @EP, A	3	1	((EP)) ← (AH), ((EP)+1) ← (AL)	–	–	–	– – – –	D7
21	MOVW EP, A	1	1	(EP) ← (A)	–	–	–	– – – –	E3
22	MOVW A, #d16	3	3	(A) ← d16	AL	AH	dH	+ + – –	E4
23	MOVW A, dir	4	2	(AH) ← (dir), (AL) ← (dir+1)	AL	AH	dH	+ + – –	C5
24	MOVW A, @IX+off	4	2	(AH) ← ((IX)+off), (AL) ← ((IX)+off+1)	AL	AH	dH	+ + – –	C6
25	MOVW A, ext	5	3	(AH) ← (ext), (AL) ← (ext+1)	AL	AH	dH	+ + – –	C4
26	MOVW A, @A	3	1	(AH) ← ((A)), (AL) ← ((A)+1)	AL	AH	dH	+ + – –	93
27	MOVW A, @EP	3	1	(AH) ← ((EP)), (AL) ← ((EP)+1)	AL	AH	dH	+ + – –	C7
28	MOVW A, EP	1	1	(A) ← (EP)	–	–	dH	– – – –	F3
29	MOVW EP, #d16	3	3	(EP) ← d16	–	–	–	– – – –	E7
30	MOVW IX, A	1	1	(IX) ← (A)	–	–	–	– – – –	E2
31	MOVW A, IX	1	1	(A) ← (IX)	–	–	dH	– – – –	F2
32	MOVW SP, A	1	1	(SP) ← (A)	–	–	–	– – – –	E1
33	MOVW A, SP	1	1	(A) ← (SP)	–	–	dH	– – – –	F1
34	MOV @A, T	2	1	((A)) ← (T)	–	–	–	– – – –	82
35	MOVW @A, T	3	1	((A)) ← (TH), ((A)+1) ← (TL)	–	–	–	– – – –	83
36	MOVW IX, #d16	3	3	(IX) ← d16	–	–	–	– – – –	E6
37	MOVW A, PS	1	1	(A) ← (PS)	–	–	dH	– – – –	70
38	MOVW PS, A	1	1	(PS) ← (A)	–	–	–	+ + + +	71
39	MOVW SP, #d16	3	3	(SP) ← d16	–	–	–	– – – –	E5
40	SWAP	1	1	(AH) ↔ (AL)	–	–	AL	– – – –	10

Table A.2-1 Operation List (for Transfer Instructions) (3/3)

No	MNEMONIC	~	#	OPERATION	TL	TH	AH	N Z V C	OP CODE
41	SETB dir:b	4	2	(dir):b ← 1	–	–	–	– – – –	A8 to AF
42	CLRB dir:b	4	2	(dir):b ← 0	–	–	–	– – – –	A0 to A7
43	XCH A, T	1	1	(AL) ↔ (TL)	AL	–	–	– – – –	42
44	XCHW A, T	1	1	(A) ↔ (T)	AL	AH	dH	– – – –	43
45	XCHW A, EP	1	1	(A) ↔ (EP)	–	–	dH	– – – –	F7
46	XCHW A, IX	1	1	(A) ↔ (IX)	–	–	dH	– – – –	F6
47	XCHW A, SP	1	1	(A) ↔ (SP)	–	–	dH	– – – –	F5
48	MOVW A, PC	2	1	(A) ← (PC)	–	–	dH	– – – –	F0

Notes:

1. In byte transfer to A, T ← A is only low bytes.
2. The operands of an instruction with two or more operands should be stored in the order designated in MNEMONIC.

Table A.2-2 Operation List (for Operation Instructions) (1/3)

No	MNEMONIC	~	#	OPERATION	TL	TH	AH	NZVC	OP CODE
1	ADDC A, Ri	2	1	(A) ← (A)+(Ri)+C	–	–	–	+ + + +	28 to 2F
2	ADDC A, #d8	2	2	(A) ← (A)+d8+C	–	–	–	+ + + +	24
3	ADDC A, dir	3	2	(A) ← (A)+(dir)+C	–	–	–	+ + + +	25
4	ADDC A, @IX+off	3	2	(A) ← (A)+((IX)+off)+C	–	–	–	+ + + +	26
5	ADDC A, @EP	2	1	(A) ← (A)+((EP))+C	–	–	–	+ + + +	27
6	ADDCW A	1	1	(A) ← (A)+(T)+C	–	–	dH	+ + + +	23
7	ADDC A	1	1	(AL) ← (AL)+(TL)+C	–	–	–	+ + + +	22
8	SUBC A, Ri	2	1	(A) ← (A)-(Ri)-C	–	–	–	+ + + +	38 to 3F
9	SUBC A, #d8	2	2	(A) ← (A)-d8-C	–	–	–	+ + + +	34
10	SUBC A, dir	3	2	(A) ← (A)-(dir)-C	–	–	–	+ + + +	35
11	SUBC A, @IX+off	3	2	(A) ← (A)-((IX)+off)-C	–	–	–	+ + + +	36
12	SUBC A, @EP	2	1	(A) ← (A)-((EP))-C	–	–	–	+ + + +	37
13	SUBCW A	1	1	(A) ← (T)-(A)-C	–	–	dH	+ + + +	33
14	SUBC A	1	1	(AL) ← (TL)-(AL)-C	–	–	–	+ + + +	32

Table A.2-2 Operation List (for Operation Instructions) (2/3)

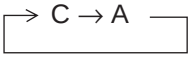
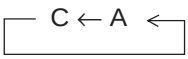
No	MNEMONIC	~	#	OPERATION	TL	TH	AH	NZVC	OP CODE
15	IINC Ri	3	1	(Ri) ← (Ri)+1	–	–	–	+ + + –	C8 to CF
16	INCW EP	1	1	(EP) ← (EP)+1	–	–	–	– – – –	C3
17	INCW IX	1	1	(IX) ← (IX)+1	–	–	–	– – – –	C2
18	INCW A	1	1	(A) ← (A)+1	–	–	dH	+ + – –	C0
19	DEC Ri	3	1	(Ri) ← (Ri)-1	–	–	–	+ + + –	D8 to DF
20	DECW EP	1	1	(EP) ← (EP)-1	–	–	–	– – – –	D3
21	DECW IX	1	1	(IX) ← (IX)-1	–	–	–	– – – –	D2
22	DECW A	1	1	(A) ← (A)-1	–	–	dH	+ + – –	D0
23	MULU A	8	1	(A) ← (AL)*(TL)	–	–	dH	– – – –	01
24	DIVU A	17	1	(A) ← (T)/(A), MOD → (T)	dL	dH	dH	– + – –	11
25	ANDW A	1	1	(A) ← (A) ∧ (T)	–	–	dH	+ + R –	63
26	ORW A	1	1	(A) ← (A) ∨ (T)	–	–	dH	+ + R –	73
27	XORW A	1	1	(A) ← (A) ∀ (T)	–	–	dH	+ + R –	53
28	CMP A	1	1	(TL)-(AL)	–	–	–	+ + + +	12
29	CMPW A	1	1	(T)- (A)	–	–	–	+ + + +	13
30	RORC A	1	1		–	–	–	+ + – +	03
31	ROLC A	1	1		–	–	–	+ + – +	02
32	CMP A, #d8	2	2	(A)- d8	–	–	–	+ + + +	14
33	CMP A, dir	3	2	(A)- (dir)	–	–	–	+ + + +	15
34	CMP A, @EP	2	1	(A)- ((EP))	–	–	–	+ + + +	17
35	CMP A, @IX+off	3	2	(A)- ((IX)+off)	–	–	–	+ + + +	16
36	CMP A, Ri	2	1	(A)- (Ri)	–	–	–	+ + + +	18 to 1F
37	DAA	1	1	decimal adjust for addition	–	–	–	+ + + +	84

Table A.2-2 Operation List (for Operation Instructions) (3/3)

No	MNEMONIC	~	#	OPERATION	TL	TH	AH	NZVC	OP CODE
38	DAS	1	1	decimal adjust for subtraction	–	–	–	+ + + +	94
39	XOR A	1	1	(A) ← (AL) ∀ (TL)	–	–	–	+ + R –	52
40	XOR A, #d8	2	2	(A) ← (AL) ∀ d8	–	–	–	+ + R –	54
41	XOR A, dir	3	2	(A) ← (AL) ∀ (dir)	–	–	–	+ + R –	55
42	XOR A, @EP	3	1	(A) ← (AL) ∀ ((EP))	–	–	–	+ + R –	57
43	XOR A, @IX+off	4	2	(A) ← (AL) ∀ ((IX)+off)	–	–	–	+ + R –	56
44	XOR A, Ri	2	1	(A) ← (AL) ∀ (Ri)	–	–	–	+ + R –	58 to 5F
45	AND A	1	1	(A) ← (AL) ^ (TL)	–	–	–	+ + R –	62
46	AND A, #d8	2	2	(A) ← (AL) ^ d8	–	–	–	+ + R –	64
47	AND A, dir	3	2	(A) ← (AL) ^ (dir)	–	–	–	+ + R –	65
48	AND A, @EP	2	1	(A) ← (AL) ^ ((EP))	–	–	–	+ + R –	67
49	AND A, @IX+off	3	2	(A) ← (AL) ^ ((IX)+off)	–	–	–	+ + R –	66
50	AND A, Ri	2	1	(A) ← (AL) ^ (Ri)	–	–	–	+ + R –	68 to 6F
51	OR A	1	1	(A) ← (AL) ∨ (TL)	–	–	–	+ + R –	72
52	OR A, #d8	2	2	(A) ← (AL) ∨ d8	–	–	–	+ + R –	74
53	OR A, dir	3	2	(A) ← (AL) ∨ (dir)	–	–	–	+ + R –	75
54	OR A,@EP	2	1	(A) ← (AL) ∨ ((EP))	–	–	–	+ + R –	77
55	OR A, @IX,off	3	2	(A) ← (AL) ∨ ((IX)+off)	–	–	–	+ + R –	76
56	OR A, Ri	2	1	(A) ← (AL) ∨ (Ri)	–	–	–	+ + R –	78 to 7F
57	CMP dir, #d8	4	3	(dir) - d8	–	–	–	+ + + +	95
58	CMP @EP, #d8	3	2	((EP))- d8	–	–	–	+ + + +	97
59	CMP @IX+off, #d8	4	3	((IX)+off) - d8	–	–	–	+ + + +	96
60	CMP Ri, #d8	3	2	(Ri) - d8	–	–	–	– – – –	98 to 9F
61	INCW SP	1	1	(SP) ← (SP) + 1	–	–	–	– – – –	C1
62	DECW SP	1	1	(SP) ← (SP) - 1	–	–	–	– – – –	D1

Table A.2-3 Operation List (for Branch Instructions)

No	MNEMONIC		~	#	OPERATION	TL	TH	AH	NZVC	OP CODE
1	BZ/BEQ rel	(divergence) (no divergence)	4 2	2	if Z=1 then PC ← PC+rel	–	–	–	– – – –	FD
2	BNZ/BNE rel	(divergence) (no divergence)	4 2	2	if Z=0 then PC ← PC+rel	–	–	–	– – – –	FC
3	BC/BLO rel	(divergence) (no divergence)	4 2	2	if C=1 then PC ← PC+rel	–	–	–	– – – –	F9
4	BNC/BHS rel	(divergence) (no divergence)	4 2	2	if C=0 then PC ← PC+rel	–	–	–	– – – –	F8
5	BN rel	(divergence) (no divergence)	4 2	2	if N=1 then PC ← PC+rel	–	–	–	– – – –	FB
6	BP rel	(divergence) (no divergence)	4 2	2	if N=0 then PC ← PC+rel	–	–	–	– – – –	FA
7	BLT rel	(divergence) (no divergence)	4 2	2	if V ∀ N=1 then PC ← PC+rel	–	–	–	– – – –	FF
8	BGE rel	(divergence) (no divergence)	4 2	2	if V ∀ N=0 then PC ← PC+rel	–	–	–	– – – –	FE
9	BBC dir:b, rel		5	3	if (dir:b)=0 then PC ← PC+rel	–	–	–	– + – –	B0 to B7
10	BBS dir:b, rel		5	3	if (dir:b)=1 then PC ← PC+rel	–	–	–	– + – –	B8 to BF
11	JMP @A		3	1	(PC) ← (A)	–	–	–	– – – –	E0
12	JMP ext		4	3	(PC) ← ext	–	–	–	– – – –	21
13	CALLV #vct		7	1	vector call	–	–	–	– – – –	E8 to EF
14	CALL ext		6	3	subroutine call	–	–	–	– – – –	31
15	XCHW A, PC		3	1	(PC) ← (A), (A) ← (PC)+1	–	–	dH	– – – –	F4
16	RET		6	1	return from subroutine	–	–	–	– – – –	20
17	RETI		8	1	return from interrupt	–	–	–	restore	30

Table A.2-4 Operation List (for Other Instructions)

No	MNEMONIC	~	#	OPERATION	TL	TH	AH	N Z V C	OP CODE
1	PUSHW A	4	1	(SP) ← (SP)-2, ((SP)) ← (A)	–	–	–	– – – –	40
2	POPW A	3	1	(A) ← ((SP)), (SP) ← (SP)+2	–	–	dH	– – – –	50
3	PUSHW IX	4	1	(SP) ← (SP)-2, ((SP)) ← (IX)	–	–	–	– – – –	41
4	POPW IX	3	1	(IX) ← ((SP)), (SP) ← (SP)+2	–	–	–	– – – –	51
5	NOP	1	1	No operation	–	–	–	– – – –	00
6	CLRC	1	1	(C) ← 0	–	–	–	– – – R	81
7	SETC	1	1	(C) ← 1	–	–	–	– – – S	91
8	CLRI	1	1	(I) ← 0	–	–	–	– – – –	80
9	SETI	1	1	(I) ← 1	–	–	–	– – – –	90

A.3 Flag Change Table

Table A.3-1 is the flag change table for transfer instructions. Table A.3-2 is the flag change table for operation instructions. Table A.3-3 is the flag change table for branch instructions. Table A.3-4 is the flag change table for other instructions.

■ Flag Change Table

Table A.3-1 Flag Change Table (for Transfer Instructions) (1/2)

Instruction	Flag change

MOV dir, A
MOV @IX+off, A
MOV ext, A
MOV @EP, A
MOV Ri, A
	N: Not changed
	Z: Not changed
	V: Not changed
	C: Not changed

MOV , #d8
MOV A, dir
MOV A, @IX+off
MOV A, ext
MOV A, @A
MOV A, @EP
MOV A, Ri
	N: Set to 1 if the transferred data is negative and set to 0 in other cases.
	Z: Set to 1 if the transferred data is 0 and set to 0 in other cases
	V: Not changed
	C: Not changed

MOV dir, #d8
MOV @IX+off, #d8
MOV @EP, #d8
MOV Ri, #d8
	N: Not changed
	Z: Not changed
	V: Not changed
	C: Not changed

MOVW dir, A
MOVW @IX+off, A
MOVW ext, A
MOVW @EP, A
	N: Not changed
	Z: Not changed
	V: Not changed
	C: Not changed

MOVW A, #d16
MOVW A, dir
MOVW A, @IX+off
MOVW A, ext
	N: Set to 1 if the transferred data is negative and set to 0 in other cases.
	Z: Set to 1 if the transferred data is 0 and set to 0 in other cases
	V: Not changed
	C: Not changed

MOVW A, @A MOVW A, @EP	

Table A.3-1 Flag Change Table (for Transfer Instructions) (2/2)

Instruction	Flag change

MOVW A, EP
MOVW EP, #d16
MOVW IX, A
MOVW A, IX
MOVW SP, A
MOVW A, SP
MOVW SP, #d16
	N: Not changed
	Z: Not changed
	V: Not changed
	C: Not changed

MOV @A, T MOVW @A, T	N: Not changed Z: Not changed V: Not changed C: Not changed

MOVW IX, #d16
MOVW A, PS
MOVW A, PC
JMP @A
	N: Not changed
	Z: Not changed
	V: Not changed
	C: Not changed

MOVW PS, A	N: Set to 1 if bit 3 of A is 1 and set to 0 if 0. Z: Set to 1 if bit 2 of A is 1 and set to 0 if 0. V: Set to 1 if bit 1 of A is 1 and set to 0 if 0. C: Set to 1 if bit 0 of A is 1 and set to 0 if 0.
SETB dir:b CLRB dir:b	N: Not changed Z: Not changed V: Not changed C: Not changed
SWAP XCH A, T	N: Not changed Z: Not changed V: Not changed C: Not changed
XCHW A, T XCHW A, EP XCHW A, IX XCHW A, SP XCHW A, PC	N: Not changed Z: Not changed V: Not changed C: Not changed

Table A.3-2 Flag Change Table (for Operation Instructions) (1/3)

Instruction	Flag change

ADDC A, Ri
ADDC A, #d8
ADDC A, dir
ADDC A, @IX+off
ADDC A, @EP
	N: Set to 1 if the result of operation is negative and set to 0 in other cases.
	Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.
	V: Set to 1 if an overflow occurs and set to 0 in other cases.
	C: Set to 1 if a carry occurs and set to 0 in other cases.

ADDC A ADDCW A	N: Set to 1 if the result of operation is negative and set to 0 in other cases. Z: Set to 1 if the result of operation is 0 and set to 0 in other cases. V: Set to 1 if an overflow occurs and set to 0 in other cases. C: Set to 1 if a carry occurs and set to 0 in other cases.

SUBC A, Ri
SUBC A, #d8
SUBC A, dir
SUBC A, @IX+off
SUBC A, @EP
	N: Set to 1 if the result of operation is negative and set to 0 in other cases.
	Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.
	V: Set to 1 if an overflow occurs and set to 0 in other cases.
	C: Set to 1 if a borrow occurs and set to 0 in other cases.

SUBC A SUBCW A	N: Set to 1 if the result of operation is negative and set to 0 in other cases. Z: Set to 1 if the result of operation is 0 and set to 0 in other cases. V: Set to 1 if an overflow occurs and set to 0 in other cases. C: Set to 1 if a borrow occurs and set to 0 in other cases.
INC Ri	N: Set to 1 if the result of operation is negative and set to 0 in other cases. Z: Set to 1 if the result of operation is 0 and set to 0 in other cases. V: Set to 1 if an overflow occurs and set to 0 in other cases. C: Not changed
INCW EP INCW IX INCW SP	N: Not changed Z: Not changed V: Not changed C: Not changed
INCW A	N: Set to 1 if the result of operation is negative and set to 0 in other cases. Z: Set to 1 if the result of operation is 0 and set to 0 in other cases. V: Not changed C: Not changed
DEC Ri	N: Set to 1 if the result of operation is negative and set to 0 in other cases. Z: Set to 1 if the result of operation is 0 and set to 0 in other cases. V: Set to 1 if an overflow occurs and set to 0 in other cases. C: Not changed

Table A.3-2 Flag Change Table (for Operation Instructions) (2/3)

Instruction	Flag change
DECW EP DECW IX DECW SP	N: Not changed Z: Not changed V: Not changed C: Not changed
DECW A	N: Set to 1 if the result of operation is negative and set to 0 in other cases. Z: Set to 1 if the result of operation is 0 and set to 0 in other cases. V: Not changed C: Not changed
MULU A	N: Not changed Z: Not changed V: Not changed C: Not changed
DIVU A	N: Not changed Z: Set to 1 if A before operation is 0000_H and set to 0 in other cases. V: Not changed C: Not changed
ANDW A	N: Set to 1 if the result of operation is negative and set to 0 in other cases. Z: Set to 1 if the result of operation is 0 and set to 0 in other cases. V: Always Set to 0 C: Not changed
AND A, #d8 AND A, dir AND A, @EP AND A, @IX+off AND A, Ri	N: Set to 1 if the result of operation is negative and set to 0 in other cases. Z: Set to 1 if the result of operation is 0 and set to 0 in other cases. V: Always set to 0 C: Not changed
ORW A	N: Set to 1 if the result of operation is negative and set to 0 in other cases. Z: Set to 1 if the result of operation is 0 and set to 0 in other cases. V: Always set to 0 C: Not changed
OR A, #d8 OR A, dir OR A, @EP OR A, @IX+off OR A, Ri	N: Set to 1 if the result of operation is negative and set to 0 in other cases. Z: Set to 1 if the result of operation is 0 and set to 0 in other cases. V: Always set to 0 C: Not changed

Table A.3-2 Flag Change Table (for Operation Instructions) (3/3)

Instruction	Flag change
XORW A	N: Set to 1 if the result of operation is negative and set to 0 in other cases. Z: Set to 1 if the result of operation is 0 and set to 0 in other cases. V: Always set to 0 C: Not changed
XOR A, #d8 XOR A, dir XOR A, @EP XOR A, @IX+off XOR A, Ri	N: Set to 1 if the result of operation is negative and set to 0 in other cases. Z: Set to 1 if the result of operation is 0 and set to 0 in other cases. V: Always set to 0 C: Not changed
CMP A	N: Set to 1 if the result of operation is negative and set to 0 in other cases. Z: Set to 1 if the result of operation is 0 and set to 0 in other cases. V: Set to 1 if an overflow occurs and set to 0 in other cases. C: Set to 1 if a borrow occurs and set to 0 in other cases.
CMPW A	N: Set to 1 if the result of operation is negative and set to 0 in other cases. Z: Set to 1 if the result of operation is 0 and set to 0 in other cases. V: Set to 1 if an overflow occurs and set to 0 in other cases. C: Set to 1 if a borrow occurs and set to 0 in other cases.

CMP A, #d8
CMP A, dir
CMP A, @EP
CMP A, @IX+off
CMP A, Ri
	N: Set to 1 if the result of operation is negative and set to 0 in other cases.
	Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.
	V: Set to 1 if an overflow occurs and set to 0 in other cases.
	C: Set to 1 if a borrow occurs and set to 0 in other cases.

CMP dir, #d8
CMP @EP #d8
CMP @IX+off, #d8
CMP Ri, #d8
	N: Set to 1 if the result of operation is negative and set to 0 in other cases.
	Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.
	V: Set to 1 if an overflow occurs and set to 0 in other cases.
	C: Set to 1 if a borrow occurs and set to 0 in other cases.

RORC A ROLC A	N: Set to 1 if the result of operation is negative and set to 0 in other cases. Z: Set to 1 if the result of operation is 0 and set to 0 in other cases. V: Not changed C: Enter bit 0 (when RORA) or bit 7 (when ROLA) of A before the operation.
DAA DAS	N: Set to 1 if the result of operation is negative and set to 0 in other cases. Z: Set to 1 if the result of operation is 0 and set to 0 in other cases. V: Set to 1 if an overflow occurs and set to 0 in other cases. C: Set to 1 if a carry (borrow) occurs and set to 0 in other cases.

Table A.3-3 Flag Change Table (for Branch Instructions)

Instruction	Flag change
BZ rel/BEQ rel BNZ rel/BNE rel BC rel/BLO rel BNC rel/BHS rel BN rel BP rel BLT rel BGE rel	N: Not changed Z: Not changed V: Not changed C: Not changed
JMP addr16	N: Not changed Z: Not changed V: Not changed C: Not changed
BBC dir:b, rel BBS dir:b, rel	N: Not changed Z: Set to 1 if bit b is 0 and set to 0 if 1. V: Not changed C: Not changed
CALL addr16 CALLV #vct RET	N: Not changed Z: Not changed V: Not changed C: Not changed
RETI	N: N value of saved CCR is entered. Z: Z value of saved CCR is entered. V: V value of saved CCR is entered. C: C value of saved CCR is entered.

Table A.3-4 Flag Change Table (for Other Instructions)

Instruction	Flag change
PUSHW A PUSHW IX	N: Not changed Z: Not changed V: Not changed C: Not changed
POPW A POPW IX	N: Not changed Z: Not changed V: Not changed C: Not changed
NOP	N: Not changed Z: Not changed V: Not changed C: Not changed
CLRC	N: Not changed Z: Not changed V: Not changed C: Become to 0
SETC	N: Not changed Z: Not changed V: Not changed C: Become to 1
CLRI	N: Not changed Z: Not changed V: Not changed C: Not changed I: Become to 0
SETI	N: Not changed Z: Not changed V: Not changed C: Not changed I: Become to 1

APPENDIX B Bus Operation List

Table B-1 is a bus operation list.

■ Bus Operation List

Table B-1 Bus Operation List (1/11)

CODE	MNEMONIC	~	Cycle	Address bus	Data bus	RD	WR	RMW
00 80 90 81 91	NOP CLRI SETI CLRC SETC	1	1	N +2	The following following instruction	1	0	0
10 12 22 32 42 52 62 72	SWAP CMP A ADDC A SUBC A XCH A, T XOR A AND A OR A	1	1	N +2	The following following instruction	1	0	0
13 23 33 43 53 63 73	CMPW A ADDCW A SUBCW A XCHW A, T XORW A ANDW A ORW A	1	1	N +2	The following following instruction	1	0	0

04
 	
14
24
34
54
64
74
	MOV A, #d8
	
	CMP A, #d8
	ADDC A, #d8
	SUBC A, #d8
	XOR A, #d8
	AND A, #d8
	OR A, #d8
		2	1
			
			2
				N +2
				
				N +3
					The following instruction
					
					The following following instruction
						1
						
						1
							0
							
							0
								0
								
								0

Table B-1 Bus Operation List (2/11)

CODE	MNEMONIC	~	Cycle	Address bus	Data bus	RD	WR	RMW
05	MOV A, dir	3	1	N +2	The following instruction	1	0	0
15	CMP A, dir		2	dir address	Data	1	0	0
25	ADDC A, dir		3	N +3	The following following instruction	1	0	0
35	SUBC A, dir							
55	XOR A, dir							
65	AND A, dir							
75	OR A, dir							
45	MOV dir, A	3	1	N +2	The following instruction	1	0	0
			2	dir address	Data	0	1	0
			3	N +3	The following following instruction	1	0	0
06	MOV A, @IX+off	3	1	N +2	The following instruction	1	0	0
16	CMP A, @IX+off		2	N +3	The following following instruction	1	0	0
26	ADDC A, @IX+off		3	(IX)+off address	Data	1	0	0
36	SUBC A, @IX+off							
56	XOR A, @IX+off							
66	AND A, @IX+off							
76	OR A, @IX+off							
46	MOV @IX+off, A	3	1	N +2	The following instruction	1	0	0
			2	N +3	The following following instruction	1	0	0
			3	(IX)+off address	Data	0	1	0
07	MOV A, @EP	2	1	N +2	The following following instruction	1	0	0
17	CMP A, @EP		2	(EP) address	Data	1	0	0
27	ADDC A, @EP							
37	SUBC A, @EP							
57	XOR A, @EP							
67	AND A, @EP							
77	OR A, @EP							
47	MOV @EP, A	2	1	N +2	The following following instruction	1	0	0
			2	(EP) address	Data	0	1	0

Table B-1 Bus Operation List (3/11)

CODE	MNEMONIC	~	Cycle	Address bus	Data bus	RD	WR	RMW
08 - 0F	MOV A, Ri	2	1	N +2	The following following instruction	1	0	0
18 - 1F	CMP A, Ri		2	Rn address	Data	1	0	0
28 - 2F	ADDC A, Ri							
38 - 3F	SUBC A, Ri							
58 - 5F	XOR A, Ri							
68 - 6F	AND A, Ri							
78 - 7F	OR A, Ri							
48 - 4F	MOV Ri, A	2	1	N +2	The following following instruction	1	0	0
			2	Rn address	Data	0	1	0
C0	INCW A	1	1	N +2	The following following instruction	1	0	0
D0	DECW A							
C1	INCW SP							
D1	DECW SP							
C2	INCW IX							
D2	DECW IX							
C3	INCW EP							
D3	DECW EP							
F0	MOVW A, PC	2	1	N +2	The following following instruction	1	0	0
			2	–	–	0	0	0
E1	MOVW SP, A	1	1	N +2	The following following instruction	1	0	0
F1	MOVW A, SP							
E2	MOVW IX, A							
F2	MOVW A, IX							
E3	MOVW EP, A							
F3	MOVW A, EP							
E0	JMP @A	3	1	N +2	Data of N +2	1	0	0
			2	Address divergence	The following instruction	1	0	0
			3	Address divergence +1	The following following instruction	1	0	0
F5	XCHW A, SP	1	1	N +2	The following following instruction	1	0	0
F6	XCHW A, IX							
F7	XCHW A, EP							

Table B-1 Bus Operation List (4/11)

CODE	MNEMONIC	~	Cycle	Address bus	Data bus	RD	WR	RMW
F4	XCHW A, PC	3	1	N +2	Data of N +2	1	0	0
			2	Address divergence	The following instruction	1	0	0
			3	Address divergence +1	The following following instruction	1	0	0
A0 - A7	CLRB dir:n	4	1	N +2	The following instruction	1	0	1
A8 - AF	SETB dir:n		2	dir address	Data	1	0	1
			3	dir address	Data	0	1	0
			4	N +3	The following following instruction	1	0	0
B0 - B7	BBC dir:n, rel	Divergence						
B8 - BF	BBS dir:n, rel	5	1	N +2	rel	1	0	0
			2	dir address	Data	1	0	0
			3	N +3	Data of N+3	1	0	0
			4	Address divergence	The following instruction	1	0	0
			5	Address divergence +1	The following following instruction	1	0	0
		No divergence						
		5	1	N +2	rel	1	0	0
			2	dir address	Data	1	0	0
			3	N +3	The following instruction	1	0	0
			4	–	–	0	0	0
			5	N +4	The following following instruction	1	0	0
60	MOV A, ext	4	1	N +2	ext (L byte)	1	0	0
			2	N +3	The following instruction	1	0	0
			3	ext address	Data	1	0	0
			4	N +4	The following following instruction	1	0	0
61	MOV ext, A	4	1	N +2	ext (L byte)	1	0	0
			2	N +3	The following instruction	1	0	0
			3	ext address	Data	0	1	0
			4	N +4	The following following instruction	1	0	0

Table B-1 Bus Operation List (5/11)

CODE	MNEMONIC	~	Cycle	Address bus	Data bus	RD	WR	RMW
C4	MOVW A, ext	5	1	N +2	ext (L byte)	1	0	0
			2	N +3	The following instruction	1	0	0
			3	ext address	Data (H byte)	1	0	0
			4	ext+1 address	Data (L byte)	1	0	0
			5	N +4	The following following instruction	1	0	0
D4	MOVW ext, A	5	1	N +2	ext (L byte)	1	0	0
			2	N +3	The following instruction	1	0	0
			3	ext address	Data (H byte)	0	1	0
			4	ext+1 address	Data (L byte)	0	1	0
			5	N +4	The following following instruction	1	0	0
C5	MOVW A, dir	4	1	N +2	The following instruction	1	0	0
			2	dir address	Data (H byte)	1	0	0
			3	dir+1 address	Data (L byte)	1	0	0
			4	N +3	The following following instruction	1	0	0
D5	MOVW dir, A	4	1	N +2	The following instruction	1	0	0
			2	dir address	Data (H byte)	0	1	0
			3	dir+1 address	Data (L byte)	0	1	0
			4	N +3	The following following instruction	1	0	0
C6	MOVW A, @IX+off	4	1	N +2	The following instruction	1	0	0
			2	N +3	The following following instruction	1	0	0
			3	(IX)+off address	Data (H byte)	1	0	0
			4	(IX)+off+1 address	Data (L byte)	1	0	0
D6	MOVW @IX+off, A	4	1	N +2	The following instruction	1	0	0
			2	N +3	The following following instruction	1	0	0
			3	(IX)+off address	Data (H byte)	0	1	0
			4	(IX)+off+1 address	Data (L byte)	0	1	0

Table B-1 Bus Operation List (6/11)

CODE	MNEMONIC	~	Cycle	Address bus	Data bus	RD	WR	RMW
C7	MOVW A, @EP	3	1	N +2	The following following instruction	1	0	0
			2	(EP) address	Data(H byte)	1	0	0
			3	(EP)+1 address	Data(L byte)	1	0	0
D7	MOVW @EP, A	3	1	N +2	The following following instruction	1	0	0
			2	(EP) address	Data(H byte)	0	1	0
			3	(EP)+1 address	Data(L byte)	0	1	0
85	MOV dir, #d8	4	1	N +2	#d8	1	0	0
			2	dir address	Data	0	1	0
			3	N +3	The following instruction	1	0	0
			4	N +4	The following following instruction	1	0	0
95	CMP dir, #d8	4	1	N +2	#d8	1	0	0
			2	dir address	Data	1	0	0
			3	N +3	The following instruction	1	0	0
86	MOV @IX+off, #d8	4	1	N +2	#d8	1	0	0
			2	N +3	The following instruction	1	0	0
			3	(IX)+off address	Data	0	1	0
			4	N +4	The following following instruction	1	0	0
96	CMP @IX+off, #d8	4	1	N +2	#d8	1	0	0
			2	N +3	The following instruction	1	0	0
			3	(IX)+off address	Data	1	0	0
			4	N +4	The following following instruction	1	0	0

Table B-1 Bus Operation List (7/11)

CODE	MNEMONIC	~	Cycle	Address bus	Data bus	RD	WR	RMW
87	MOV @EP, #d8	3	1	N +2	The following instruction	1	0	0
			2	(EP) address	Data	0	1	0
			3	N +3	The following following instruction	1	0	0
97	CMP @EP, #d8	3	1	N +2	The following instruction	1	0	0
			2	(EP) address	Data	1	0	0
			3	N +3	The following following instruction	1	0	0
88 - 8F	MOV Ri, #d8	3	1	N +2	The following instruction	1	0	0
			2	Rn address	Data	0	1	0
			3	N +3	The following following instruction	1	0	0
98 - 9F	CMP Ri, #d8	3	1	N +2	The following instruction	1	0	0
			2	Rn address	Data	1	0	0
			3	N +3	The following following instruction	1	0	0
82	MOV @A, T	2	1	N +2	The following following instruction	1	0	0
			2	(A) address	Data	0	1	0
92	MOV A, @A	2	1	N +2	The following following instruction	1	0	0
			2	(A) address	Data	1	0	0
83	MOVW @A, T	3	1	N +2	The following following instruction	1	0	0
			2	(A) address	Data (H byte)	0	1	0
			3	(A) +1 address	Data (L byte)	0	1	0

Table B-1 Bus Operation List (8/11)

CODE	MNEMONIC	~	Cycle	Address bus	Data bus	RD	WR	RMW
93	MOVW A, @A	3	1	N +2	The following following instruction	1	0	0
			2	(A) address	Data (H byte)	1	0	0
			3	(A) +1 address	Data (L byte)	1	0	0
E4	MOVW A, #d16	3	1	N +2	Data (L byte)	1	0	0
E5	MOVW SP, #d16		2	N +3	The following instruction	1	0	0
E6	MOVW IX, #d16		3	N +4	The following following instruction	1	0	0
E7	MOVW EP, #d16							
84	DAA	1	1	N +2	The following following instruction	1	0	0
94	DAS							
02	ROLC A							
03	RORC A							
70	MOVW A, PS							
71	MOVW PS, A							
C8 - CF	INC Ri	3	1	N +2	The following following instruction	1	0	1
D8 - DF	DEC Ri		2	Rn address	Data	1	0	1
			3	Rn address	Data	0	1	0
E8 - EF	CALLV #n	7	1	N +2	Data of N +2	1	0	0
			2	Vector address	Vector (H)	1	0	0
			3	Vector address +1	Vector (L)	1	0	0
			4	SP -1	Return address (L)	0	1	0
			5	SP -2	Return address (H)	0	1	0
			6	Address divergence ahead	The following instruction	1	0	0
			7	Address divergence ahead +1	The following following instruction	1	0	0

Table B-1 Bus Operation List (9/11)

CODE	MNEMONIC	~	Cycle	Address bus	Data bus	RD	WR	RMW
F8	BNC rel	Divergence						
F9	BC rel	4	1	N +2	Data of N +2	1	0	0
FA	BP rel		2	N +3	Data of N +3	1	0	0
FB	BN rel		3	Address divergence ahead	The following instruction	1	0	0
FC	BNZ rel		4	Address divergence ahead +1	The following following instruction	1	0	0
FD	BZ rel	No divergence						
FE	BGE rel	2	1	N +2	The following instruction	1	0	0
FF	BLT rel		2	N +3	The following following instruction	1	0	0
40	PUSHW A	4	1	N +2	The following following instruction	1	0	0
41	PUSHW IX		2	–	–	0	0	0
			3	SP -1	Save data (L)	0	1	0
			4	SP -2	Save data (H)	0	1	0
50	POPW A	3	1	N +2	The following following instruction	1	0	0
51	POPW IX		2	SP	Return data (H)	1	0	0
			3	SP +1	Return data (L)	1	0	0
20	RET	6	1	N +2	Data of N +2	1	0	0
			2	SP	Return address (H)	1	0	0
			3	SP +1	Return address (L)	1	0	0
			4	–	–	0	0	0
			5	Return address	The following instruction	1	0	0
			6	Return address +1	The following following instruction	1	0	0
30	RETI	8	1	N +2	Data of N +2	1	0	0
			2	SP	PSH (RP, DP)	1	0	0
			3	SP +1	PSL (CCR)	1	0	0
			4	SP +2	Return address (H)	1	0	0
			5	SP +3	Return address (L)	1	0	0
			6	–	–	0	0	0
			7	Return address	The following instruction	1	0	0
			8	Return address +1	The following following instruction	1	0	0

Table B-1 Bus Operation List (10/11)

CODE	MNEMONIC	~	Cycle	Address bus	Data bus	RD	WR	RMW
31	CALL ext	6	1	N +2	Address divergence ahead (L)	1	0	0
			2	–	–	0	0	0
			3	SP -1	Return address (L)	0	1	0
			4	SP -2	Return address (H)	0	1	0
			5	Address divergence ahead	The following instruction	1	0	0
			6	Address divergence ahead +1	The following following instruction	1	0	0
21	JMP ext	4	1	N +2	Address divergence ahead (L)	1	0	0
			2	–	–	0	0	0
			3	Address divergence ahead	The following instruction	1	0	0
			4	Address divergence ahead +1	The following following instruction	1	0	0
01	MULU A	8	1	N +2	The following following instruction	1	0	0
			2 to 8	–	–	0	0	0
11	DIVU A	17	1	N +2	The following following instruction	1	0	0
			2 to 17	–	–	0	0	0
–	RESET	7	1	–	–	0	0	0
			2	$0FFFD_H$	Mode data	1	0	0
			3	$0FFFE_H$	Reset vector (H)	1	0	0
			4	$0FFFF_H$	Reset vector (L)	1	0	0
			5	–	–	0	0	0
			6	Start address	The following instruction	1	0	0
			7	Start address +1	The following following instruction	1	0	0

Table B-1 Bus Operation List (11/11)

CODE	MNEMONIC	~	Cycle	Address bus	Data bus	RD	WR	RMW
–	INTERRUPT	9	1	N +2	Data of N +2	1	0	0
			2	Vector address	Vector (H)	1	0	0
			3	Vector address +1	Vector (L)	1	0	0
			4	SP -1	Return address (L)	0	1	0
			5	SP -2	Return address (H)	0	1	0
			6	SP -3	PSL (CCR)	0	1	0
			7	SP -4	PSH (RP, DP)	0	1	0
			8	Address divergence ahead	The following instruction	1	0	0
			9	Address divergence ahead +1	The following following instruction	1	0	0

-: Invalid bus cycle
N: Address where instruction under execution is stored
Note:
The cycle of the instruction might be extended by the immediately preceding instruction by one cycle.
Moreover, cycle of the instruction number might be extended in the access to the IO area.

APPENDIX C Instruction Map

Table C-1 is an instruction map.

■ Instruction Map

Table C-1 Instruction Map

LH	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	SWAP	RET	RETI	PUSHW A	POPW A	MOV A,ext	MOVW A,PS	CLRI	SETI	CLRB dir:0	BBC dir :0,rel	INCW A	DECW A	JMP @A	MOVW A,PC
1	MULU A	DIVU A	JMP addr16	CALL addr16	PUSHW IX	POPW IX	MOV ext,A	MOVW PS,A	CLRC	SETC	CLRB dir:1	BBC dir :1,rel	INCW SP	DECW SP	MOVW SP,A	MOVW A,SP
2	ROLC A	CMP A	ADDC A	SUBC A	XCH A,T	XOR A	AND A	OR A	MOV @A,T	MOV A,@A	CLRB dir:2	BBC dir :2,rel	INCW IX	DECW IX	MOVW IX,A	MOVW A,IX
3	RORC A	CMPW A	ADDCW A	SUBCW A	XCHW A,T	XORW A	ANDW A	ORW A	MOVW @A,T	MOVW A,@A	CLRB dir:3	BBC dir :3,rel	INCW EP	DECW EP	MOVW EP,A	MOVW A,EP
4	MOV A,#d8	CMP A,#d8	ADDC A,#d8	SUBC A,#d8	□	XOR A,#d8	AND A,#d8	OR A,#d8	DAA	DAS	CLRB dir:4	BBC dir :4,rel	MOVW A,ext	MOVW ext,A	MOVW A,#d16	XCHW A,PC
5	MOV A,dir	CMP A,dir	ADDC A,dir	SUBC A,dir	MOV dir,A	XOR A,dir	AND A,dir	OR A,dir	MOV dir,#d8	CMP dir,#d8	CLRB dir:5	BBC dir :5,rel	MOVW A,dir	MOVW dir,A	MOVW SP,#d16	XCHW A,SP
6	MOV A, @IX+d	CMP A, @IX+d	ADDC A, @IX+d	SUBC A, @IX+d	MOV @IX+d,A	XOR A, @IX+d	AND A, @IX+d	OR A, @IX+d	MOV @ IX+d,#d8	CMP @ IX+d,#d8	CLRB dir:6	BBC dir :6,rel	MOVW A,@IX+d	MOVW @IX+d,A	MOVW IX,#d16	XCHW A,IX
7	MOV A,@EP	CMP A,@EP	ADDC A,@EP	SUBC A,@EP	MOV @EP,A	XOR A,@EP	AND A,@EP	OR A,@EP	MOV @EP,#d8	CMP @EP,#d8	CLRB dir:7	BBC dir :7,rel	MOVW A,@EP	MOVW @EP,A	MOVW EP,#d16	XCHW A,EP
8	MOV A,R0	CMP A,R0	ADDC A,R0	SUBC A,R0	MOV R0,A	XOR A,R0	AND A,R0	OR A,R0	MOV R0,#d8	CMP R0,#d8	SETB dir:0	BBS dir :0,rel	INC R0	DEC R0	CALLV #0	BNC rel
9	MOV A,R1	CMP A,R1	ADDC A,R1	SUBC A,R1	MOV R1,A	XOR A,R1	AND A,R1	OR A,R1	MOV R1,#d8	CMP R1,#d8	SETB dir:1	BBS dir :1,rel	INC R1	DEC R1	CALLV #1	BC rel
A	MOV A,R2	CMP A,R2	ADDC A,R2	SUBC A,R2	MOV R2,A	XOR A,R2	AND A,R2	OR A,R2	MOV R2,#d8	CMP R2,#d8	SETB dir:2	BBS dir :2,rel	INC R2	DEC R2	CALLV #2	BP rel
B	MOV A,R3	CMP A,R3	ADDC A,R3	SUBC A,R3	MOV R3,A	XOR A,R3	AND A,R3	OR A,R3	MOV R3,#d8	CMP R3,#d8	SETB dir:3	BBS dir :3,rel	INC R3	DEC R3	CALLV #3	BN rel
C	MOV A,R4	CMP A,R4	ADDC A,R4	SUBC A,R4	MOV R4,A	XOR A,R4	AND A,R4	OR A,R4	MOV R4,#d8	CMP R4,#d8	SETB dir:4	BBS dir :4,rel	INC R4	DEC R4	CALLV #4	BNZ rel
D	MOV A,R5	CMP A,R5	ADDC A,R5	SUBC A,R5	MOV R5,A	XOR A,R5	AND A,R5	OR A,R5	MOV R5,#d8	CMP R5,#d8	SETB dir:5	BBS dir :5,rel	INC R5	DEC R5	CALLV #5	BZ rel
E	MOV A,R6	CMP A,R6	ADDC A,R6	SUBC A,R6	MOV R6,A	XOR A,R6	AND A,R6	OR A,R6	MOV R6,#d8	CMP R6,#d8	SETB dir:6	BBS dir :6,rel	INC R6	DEC R6	CALLV #6	BGE rel
F	MOV A,R7	CMP A,R7	ADDC A,R7	SUBC A,R7	MOV R7,A	XOR A,R7	AND A,R7	OR A,R7	MOV R7,#d8	CMP R7,#d8	SETB dir:7	BBS dir :7,rel	INC R7	DEC R7	CALLV #7	BLT rel

INDEX

The index follows on the next page.
This is listed in alphabetic order.

Index

CM26-00301-2E

FUJITSU SEMICONDUCTOR • CONTROLLER MANUAL

F^2MC-8FX
8-BIT MICROCONTROLLER
PROGRAMMING MANUAL

February 2008 the second edition

Published **FUJITSU LIMITED** Electronic Devices

Edited Strategic Business Development Dept.