

**Please note that Cypress is an Infineon Technologies Company.**

The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

**Continuity of document content**

The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

**Continuity of ordering part numbers**

Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.



# F<sup>2</sup>MC-8FX Programming Specifications

Document Number: 002-07004 Rev. \*B

Cypress Semiconductor  
198 Champion Court  
San Jose, CA 95134-1709  
[www.cypress.com](http://www.cypress.com)

## Copyrights

© Cypress Semiconductor Corporation, 2006-2019. This document is the property of Cypress Semiconductor Corporation and its subsidiaries ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress shall have no liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. CYPRESS DOES NOT REPRESENT, WARRANT, OR GUARANTEE THAT CYPRESS PRODUCTS, OR SYSTEMS CREATED USING CYPRESS PRODUCTS, WILL BE FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (collectively, "Security Breach"). Cypress disclaims any liability relating to any Security Breach, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from any Security Breach. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. "High-Risk Device" means any device or system whose failure could cause personal injury, death, or property damage. Examples of High-Risk Devices are weapons, nuclear installations, surgical implants, and other medical devices. "Critical Component" means any component of a High-Risk Device whose failure to perform can be reasonably expected to cause, directly or indirectly, the failure of the High-Risk Device, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from any use of a Cypress product as a Critical Component in a High-Risk Device. You shall indemnify and hold Cypress, its directors, officers, employees, agents, affiliates, distributors, and assigns harmless from and against all claims, costs, damages, and expenses, arising out of any claim, including claims for product liability, personal injury or death, or property damage arising from any use of a Cypress product as a Critical Component in a High-Risk Device. Cypress products are not intended or authorized for use as a Critical Component in any High-Risk Device except to the limited extent that (i) Cypress's published data sheet for the product explicitly states Cypress has qualified the product for use in a specific High-Risk Device, or (ii) Cypress has given you advance written authorization to use the product as a Critical Component in the specific High-Risk Device and you have signed a separate indemnification agreement.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit [cypress.com](http://cypress.com). Other names and brands may be claimed as property of their respective owners.

# Contents



|  |           |
|--|-----------|
| <b>1. Outline And Configuration Example Of F<sup>2</sup>MC-8FX CPU</b>                       | <b>6</b>  |
| 1.1 Outline of F <sup>2</sup> MC-8FX CPU   | 7         |
| 1.2 Configuration Example of Device Using F <sup>2</sup> MC-8FX CPU                          | 8         |
| <b>2. Memory Space</b>   | <b>9</b>  |
| 2.1 CPU Memory Space   | 10        |
| 2.2 Memory Space and Addressing  | 11        |
| 2.2.1 Data Area  | 13        |
| 2.2.2 Program Area   | 15        |
| 2.2.3 Arrangement of 16-bit Data in Memory Space   | 17        |
| <b>3. Registers</b>  | <b>18</b> |
| 3.1 F <sup>2</sup> MC-8FX Registers  | 19        |
| 3.2 Program Counter (PC) and Stack Pointer (SP)  | 20        |
| 3.3 Accumulator (A) and Temporary Accumulator (T)  | 21        |
| 3.3.1 How To Use The Temporary Accumulator (T)   | 23        |
| 3.3.2 Byte Data Transfer and Operation of Accumulator (A) and Temporary Accumulator (T)      | 24        |
| 3.4 Program Status (PS)  | 26        |
| 3.5 Index Register (IX) and Extra Pointer (EP)   | 29        |
| 3.6 Register Banks   | 30        |
| 3.7 Direct Banks   | 31        |
| <b>4. Interrupt Processing</b>   | <b>32</b> |
| 4.1 Outline of Interrupt Operation   | 33        |
| 4.2 Interrupt Enable/Disable and Interrupt Priority Functions                                | 34        |
| 4.3 Creating an Interrupt Processing Program   | 35        |
| 4.4 Multiple Interrupt   | 37        |
| 4.5 Reset Operation  | 38        |
| <b>5. CPU Software Architecture</b>  | <b>39</b> |
| 5.1 Types of Addressing Modes  | 40        |
| 5.2 Special Instructions   | 44        |
| <b>6. Detailed Rules For Execution Instructions</b>  | <b>48</b> |
| 6.1 ADDC (ADD Byte Data of Accumulator and Temporary Accumulator with Carry to Accumulator)  | 49        |
| 6.2 ADDC (ADD Byte Data of Accumulator and Memory with Carry to Accumulator)                 | 51        |
| 6.3 ADDCW (ADD Word Data of Accumulator and Temporary Accumulator with Carry to Accumulator) | 53        |

|      |   |     |
|------|---|-----|
| 6.4  | AND (AND Byte Data of Accumulator and Temporary Accumulator to Accumulator).....        | 55  |
| 6.5  | AND (AND Byte Data of Accumulator and Memory to Accumulator) .....                      | 57  |
| 6.6  | ANDW (AND Word Data of Accumulator and Temporary Accumulator to Accumulator).....       | 59  |
| 6.7  | BBC (Branch if Bit is Clear) .....  | 61  |
| 6.8  | BBS (Branch if Bit is Set).....   | 63  |
| 6.9  | BC (Branch relative if C=1)/BLO (Branch if LOWER).....                                  | 65  |
| 6.10 | BGE (Branch Great or Equal: relative if larger than or equal to Zero) .....             | 67  |
| 6.11 | BLT (Branch Less Than zero: relative if < Zero).....                                    | 69  |
| 6.12 | BN (Branch relative if N = 1).....  | 71  |
| 6.13 | BNZ (Branch relative if Z = 0)/BNE (Branch if Not Equal) .....                          | 73  |
| 6.14 | BNC (Branch relative if C = 0)/BHS (Branch if Higher or Same) .....                     | 75  |
| 6.15 | BP (Branch relative if N = 0: PLUS).....  | 77  |
| 6.16 | BZ (Branch relative if Z = 1)/BEQ (Branch if Equal) .....                               | 79  |
| 6.17 | CALL (CALL subroutine).....   | 81  |
| 6.18 | CALLV (CALL Vectored subroutine).....   | 83  |
| 6.19 | CLRB (Clear direct Memory Bit) .....  | 85  |
| 6.20 | CLRC (Clear Carry flag) .....   | 87  |
| 6.21 | CLRI (CLear Interrupt flag).....  | 89  |
| 6.22 | CMP (CoMPare Byte Data of Accumulator and Temporary Accumulator) .....                  | 91  |
| 6.23 | CMP (CoMPare Byte Data of Accumulator and Memory) .....                                 | 93  |
| 6.24 | CMP (CoMPare Byte Data of Immediate Data and Memory) .....                              | 95  |
| 6.25 | CMPW (CoMPare Word Data of Accumulator and Temporary Accumulator) .....                 | 97  |
| 6.26 | DAA (Decimal Adjust for Addition).....  | 99  |
| 6.27 | DAS (Decimal Adjust for Subtraction).....   | 102 |
| 6.28 | DEC (DECrement Byte Data of General-purpose Register).....                              | 104 |
| 6.29 | DECW (DECrement Word Data of Accumulator) .....   | 106 |
| 6.30 | DECW (DECrement Word Data of Extra Pointer).....  | 108 |
| 6.31 | DECW (DECrement Word Data of Index Pointer) .....                                       | 110 |
| 6.32 | DECW (DECrement Word Data of Stack Pointer) .....                                       | 112 |
| 6.33 | DIVU (DIVide Unsigned).....   | 114 |
| 6.34 | INC (INCrement Byte Data of General-purpose Register).....                              | 116 |
| 6.35 | INCW (INCrement Word Data of Accumulator) .....   | 118 |
| 6.36 | INCW (INCrement Word Data of Extra Pointer) .....                                       | 120 |
| 6.37 | INCW (INCrement Word Data of Index Register).....                                       | 122 |
| 6.38 | INCW (INCrement Word Data of Stack Pointer).....  | 124 |
| 6.39 | JMP (JuMP to address pointed by Accumulator).....                                       | 126 |
| 6.40 | JMP (JuMP to effective Address) .....   | 128 |
| 6.41 | MOV (MOVE Byte Data from Temporary Accumulator to Address Pointed by Accumulator).....  | 130 |
| 6.42 | MOV (MOVE Byte Data from Memory to Accumulator).....                                    | 132 |
| 6.43 | MOV (MOVE Immediate Byte Data to Memory) .....  | 134 |
| 6.44 | MOV (MOVE Byte Data from Accumulator to memory).....                                    | 136 |
| 6.45 | MOVW (MOVE Word Data from Temporary Accumulator to Address Pointed by Accumulator)..... | 138 |
| 6.46 | MOVW (MOVE Word Data from Memory to Accumulator).....                                   | 140 |
| 6.47 | MOVW (MOVE Word Data from Extra Pointer to Accumulator) .....                           | 142 |
| 6.48 | MOVW (MOVE Word Data from Index Register to Accumulator).....                           | 144 |
| 6.49 | MOVW (MOVE Word Data from Program Status Register to Accumulator) .....                 | 146 |
| 6.50 | MOVW (MOVE Word Data from Program Counter to Accumulator) .....                         | 148 |
| 6.51 | MOVW (MOVE Word Data from Stack Pointer to Accumulator) .....                           | 150 |
| 6.52 | MOVW (MOVE Word Data from Accumulator to Memory).....                                   | 152 |

|                                    |   |            |
|------------------------------------|---|------------|
| 6.53                               | MOVW (MOVE Word Data from Accumulator to Extra Pointer) .....   | 154        |
| 6.54                               | MOVW (MOVE Immediate Word Data to Extra Pointer) .....  | 156        |
| 6.55                               | MOVW (MOVE Word Data from Accumulator to Index Register) .....  | 158        |
| 6.56                               | MOVW (MOVE Immediate Word Data to Index Register) .....   | 160        |
| 6.57                               | MOVW (MOVE Word data from Accumulator to Program Status Register) .....                                 | 162        |
| 6.58                               | MOVW (MOVE Immediate Word Data to Stack Pointer) .....  | 164        |
| 6.59                               | MOVW (MOVE Word data from Accumulator to Stack Pointer) .....   | 166        |
| 6.60                               | MULU (MULTIply Unsigned) .....  | 168        |
| 6.61                               | NOP (NoOPeration) .....   | 170        |
| 6.62                               | OR (OR Byte Data of Accumulator and Temporary Accumulator to Accumulator) ..                            | 172        |
| 6.63                               | OR (OR Byte Data of Accumulator and Memory to Accumulator) .....  | 174        |
| 6.64                               | ORW (OR Word Data of Accumulator and Temporary Accumulator<br>to Accumulator) .....                     | 176        |
| 6.65                               | PUSHW (PUSH Word Data of Inherent Register to Stack Memory) .....                                       | 178        |
| 6.66                               | POPW (POP Word Data of Inherent Register from Stack Memory) .....                                       | 180        |
| 6.67                               | RET (RETurn from subroutine) .....  | 182        |
| 6.68                               | RETI (RETurn from Interrupt) .....  | 184        |
| 6.69                               | ROL (Rotate Byte Data of Accumulator with Carry to Left) .....  | 186        |
| 6.70                               | ROR (Rotate Byte Data of Accumulator with Carry to Right) .....   | 188        |
| 6.71                               | SUBC (SUBtract Byte Data of Accumulator from Temporary Accumulator with Carry to<br>Accumulator) .....  | 190        |
| 6.72                               | SUBC (SUBtract Byte Data of Memory from Accumulator with Carry<br>to Accumulator) .....                 | 192        |
| 6.73                               | SUBCW (SUBtract Word Data of Accumulator from Temporary Accumulator<br>with Carry to Accumulator) ..... | 194        |
| 6.74                               | SETB (Set Direct Memory Bit) .....  | 196        |
| 6.75                               | SETC (SET Carry flag) .....   | 198        |
| 6.76                               | SETI (SET Interrupt flag) .....   | 200        |
| 6.77                               | SWAP (SWAP Byte Data Accumulator "H" and Accumulator "L") .....   | 202        |
| 6.78                               | XCH (eXCHange Byte Data Accumulator "L" and Temporary Accumulator "L") .....                            | 204        |
| 6.79                               | XCHW (eXCHange Word Data Accumulator and Extrapointer) .....  | 206        |
| 6.80                               | XCHW (eXCHange Word Data Accumulator and Index Register) .....  | 208        |
| 6.81                               | XCHW (eXCHange Word Data Accumulator and Program Counter) .....   | 210        |
| 6.82                               | XCHW (eXCHange Word Data Accumulator and Stack Pointer) .....   | 212        |
| 6.83                               | XCHW (eXCHange Word Data Accumulator and Temporary Accumulator) .....                                   | 214        |
| 6.84                               | XOR (eXclusive OR Byte Data of Accumulator and Temporary Accumulator<br>to Accumulator) .....           | 216        |
| 6.85                               | XOR (eXclusive OR Byte Data of Accumulator and Memory to Accumulator) .....                             | 218        |
| 6.86                               | XORW (eXclusive OR Word Data of Accumulator and Temporary Accumulator<br>to Accumulator) .....          | 220        |
| <b>Major Changes .....</b>         |   | <b>222</b> |
| <b>A. Instruction List .....</b>   |   | <b>223</b> |
| <b>B. Bus Operation List .....</b> |   | <b>245</b> |
| <b>C. Instruction Map .....</b>    |   | <b>260</b> |
| <b>Revision History .....</b>      |   | <b>262</b> |

# 1. Outline And Configuration Example Of F<sup>2</sup>MC-8FX CPU



This chapter outlines the F<sup>2</sup>MC-8FX CPU and explains its configuration by example.

1.1 Outline of F<sup>2</sup>MC-8FX CPU

1.2 Configuration Example of Device Using F<sup>2</sup>MC-8FX CPU

## 1.1 Outline of F<sup>2</sup>MC-8FX CPU

The F<sup>2</sup>MC-8FX CPU is a high-performance 8-bit CPU designed for the embedded control of various industrial and OA equipment.

### Outline of F<sup>2</sup>MC-8FX CPU

The F<sup>2</sup>MC-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 F<sup>2</sup>MC-8FX CPU is upper compatibility CPU of the F<sup>2</sup>MC-8L CPU, and the instruction cycle number is shortened, the division instruction is strengthened, and a direct area is enhanced.

### F<sup>2</sup>MC-8FX CPU Features

The F<sup>2</sup>MC-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:  $8 \times 8 = 16$  bits,  $16/16 = 16$  bits
- Interrupt priorities : 4 levels



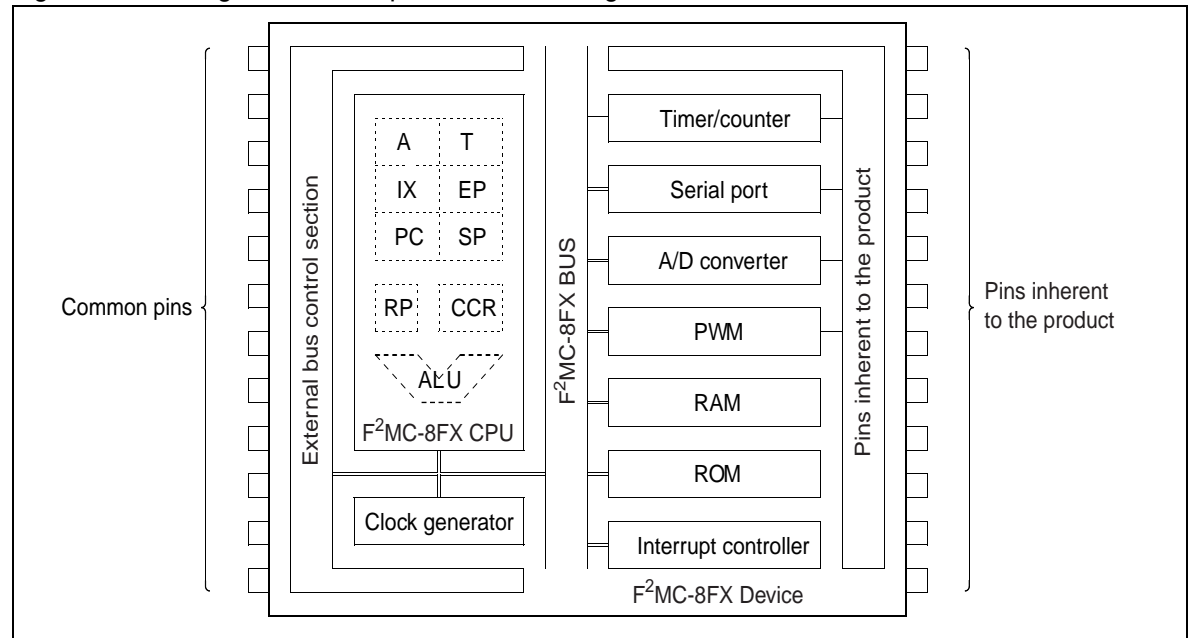
## 1.2 Configuration Example of Device Using F<sup>2</sup>MC-8FX CPU

The CPU, ROM, RAM and various resources for each F<sup>2</sup>MC-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 F<sup>2</sup>MC-8FX CPU

Figure 1-1. shows a configuration example of a device using the F<sup>2</sup>MC-8FX CPU.

Figure 1-1. Configuration Example of Device Using F<sup>2</sup>MC-8FX CPU



## 2. Memory Space



This chapter explains the F<sup>2</sup>MC-8FX CPU memory space.

2.1 CPU Memory Space

2.2 Memory Space and Addressing

## 2.1 CPU Memory Space

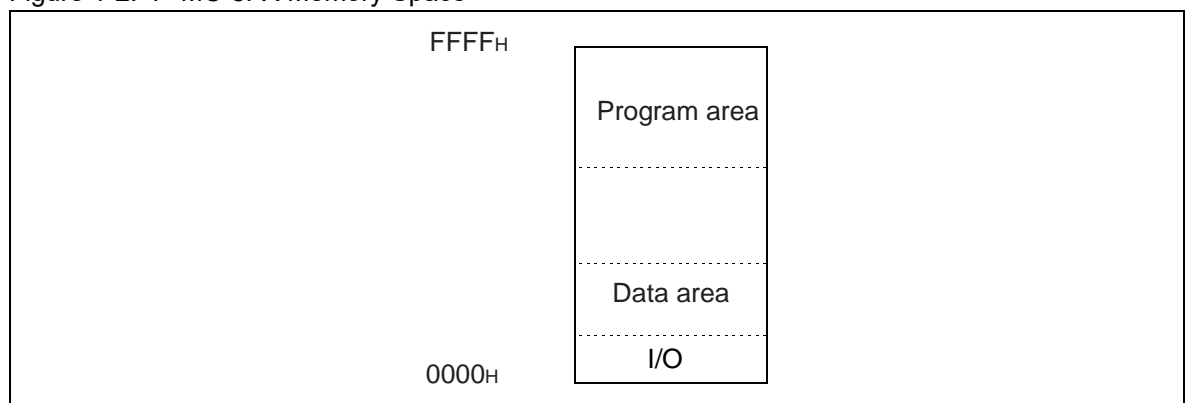
All of the data, program, and I/O areas managed by the F<sup>2</sup>MC-8FX CPU are assigned to the 64 Kbyte memory space of the F<sup>2</sup>MC-8FX CPU. The CPU can access each resource by indicating its address on the 16-bit address bus.

### CPU Memory Space

Figure 1-2. shows the address configuration of the F<sup>2</sup>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 1-2. F<sup>2</sup>MC-8FX Memory Space



## 2.2 Memory Space and Addressing

In addressing by the F<sup>2</sup>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<sup>2</sup>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<sub>H</sub> to 7F<sub>H</sub> are always 0000<sub>H</sub> to 007F<sub>H</sub>. The accesses of operand address 80<sub>H</sub> to FF<sub>H</sub> are mapped to 0080<sub>H</sub> to 047F<sub>H</sub> 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<sub>H</sub> to 7F<sub>H</sub> are always 0000<sub>H</sub> to 007F<sub>H</sub>. The accesses of operand address 80<sub>H</sub> to FF<sub>H</sub> are mapped to 0080<sub>H</sub> to 047F<sub>H</sub> 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.

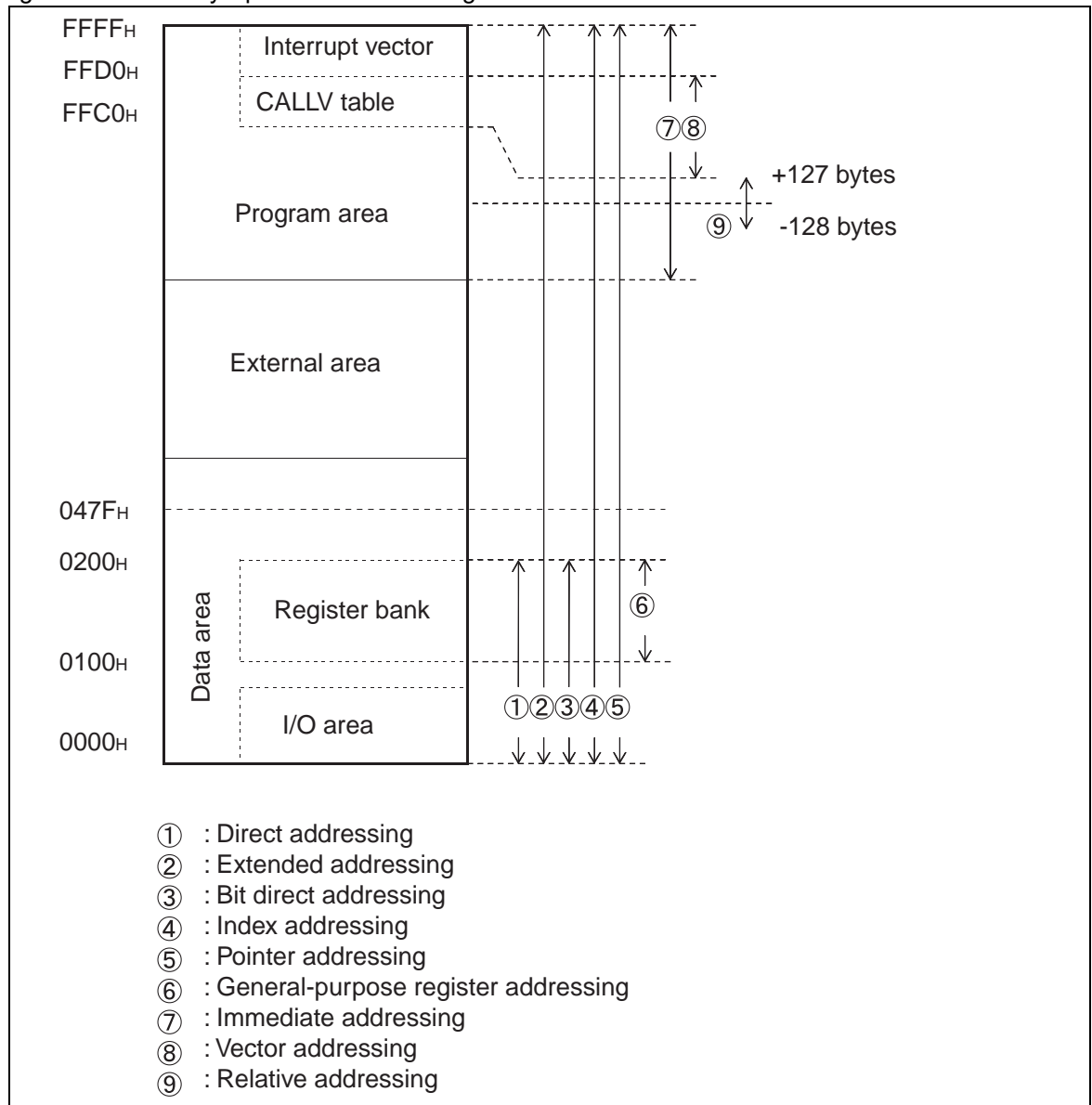
[Structure] [← OP code: table →]

- 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-1-2. shows the memory space accessible by each addressing mode.

Figure 1-3. Memory Space and Addressing



## 2.2.1 Data Area

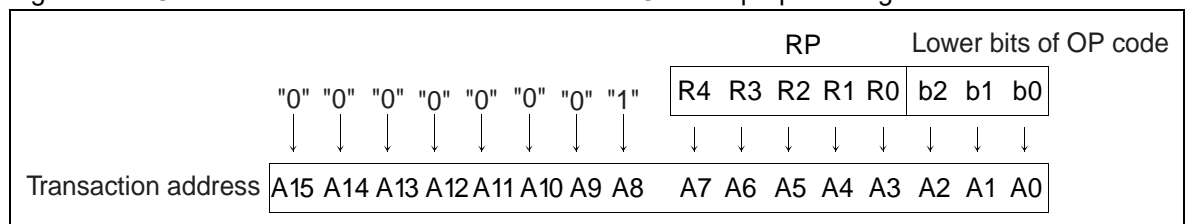
The F<sup>2</sup>MC-8FX CPU data area can be divided into the following three for each purpose:

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

### General-Purpose Register Bank Area

The general-purpose register bank area in the F<sup>2</sup>MC-8FX CPU is assigned to 0100<sub>H</sub> to 01FF<sub>H</sub>. The general-purpose register numbers are converted to the actual addresses according to the conversion rule shown in Figure 1-4. by using the register bank pointer (RP) and the lower 3 bits of the OP code.

Figure 1-4. Conversion Rule for Actual Addresses of General-purpose Register Bank Area



### Stack Area

The stack area in the F<sup>2</sup>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 1-5. and Figure 1-6. give examples of saving data in the stack area and returning data from it.

Figure 1-5. Example of Saving Data in Stack Area

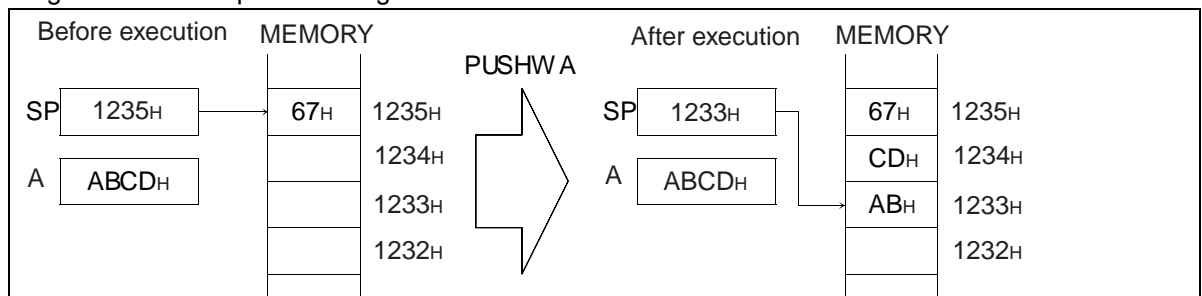
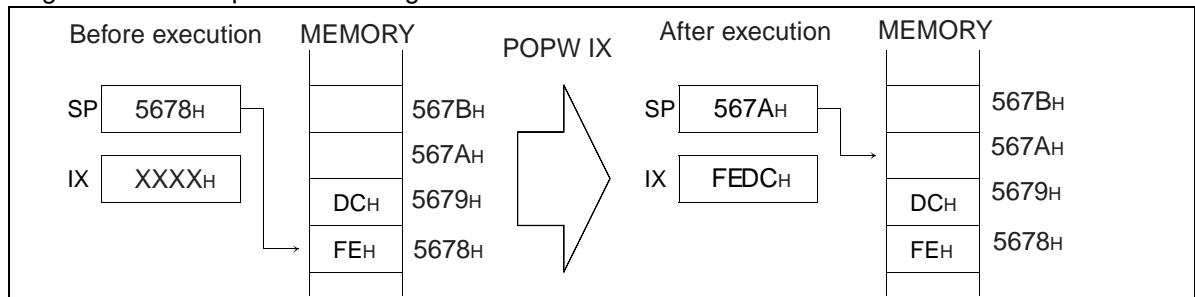


Figure 1-6. Example of Returning Data from Stack Area



## Direct Area

The direct area in the F<sup>2</sup>MC-8FX CPU is located at the lower side of the memory space or the 1152 bytes from 0000<sub>H</sub> to 047F<sub>H</sub> 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<sub>H</sub> to 007F<sub>H</sub> can be used at any time as a direct area. 0080<sub>H</sub> to 047F<sub>H</sub> 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 2-1. 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 2-1. Conversion Rule for Actual Address of Direct Addressing and Bit Direct Addressing

| Operand address                    | Direct bank pointer (DP) | Actual address                         |
|------------------------------------|--------------------------|--|
| 00 <sub>H</sub> to 7F <sub>H</sub> |                          | 0000 <sub>H</sub> to 007F <sub>H</sub> |
| 80 <sub>H</sub> to FF <sub>H</sub> | 000                      | 0080 <sub>H</sub> to 00FF <sub>H</sub> |
|                                    | 001                      | 0100 <sub>H</sub> to 017F <sub>H</sub> |
|                                    | 010                      | 0180 <sub>H</sub> to 01FF <sub>H</sub> |
|                                    | 011                      | 0200 <sub>H</sub> to 027F <sub>H</sub> |
|                                    | 100                      | 0280 <sub>H</sub> to 02FF <sub>H</sub> |
|                                    | 101                      | 0300 <sub>H</sub> to 037F <sub>H</sub> |
|                                    | 110                      | 0380 <sub>H</sub> to 03FF <sub>H</sub> |
|                                    | 111                      | 0400 <sub>H</sub> to 047F <sub>H</sub> |

## 2.2.2 Program Area

The program area in the F<sup>2</sup>MC-8FX CPU includes the following two:

- Vector call instruction table
- Reset and interrupt vector table

### Vector Call Instruction Table

FFC0<sub>H</sub> to FFCF<sub>H</sub> of the memory space is used as the vector call instruction table. The vector call instruction for the F<sup>2</sup>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. indicates the correspondence of the vector numbers with the jump address table.

Table 2-2. CALLV Jump Address Table

| CALLV<br>#k | Jump address table |                   |
|-------------|--------------------|-------------------|
|             | Upper address      | Lower address     |
| #0          | FFC0 <sub>H</sub>  | FFC1 <sub>H</sub> |
| #1          | FFC2 <sub>H</sub>  | FFC3 <sub>H</sub> |
| #2          | FFC4 <sub>H</sub>  | FFC5 <sub>H</sub> |
| #3          | FFC6 <sub>H</sub>  | FFC7 <sub>H</sub> |
| #4          | FFC8 <sub>H</sub>  | FFC9 <sub>H</sub> |
| #5          | FFCA <sub>H</sub>  | FFCB <sub>H</sub> |
| #6          | FFCC <sub>H</sub>  | FFCD <sub>H</sub> |
| #7          | FFCE <sub>H</sub>  | FFCF <sub>H</sub> |

### Reset and Interrupt Vector Table

FFCC<sub>H</sub> to FFFF<sub>H</sub> of the memory space is used as the table indicating the starting address of an interrupt or reset Table 2-3. indicates the correspondence between the interrupt numbers or resets and the reference table.

FFFC<sub>H</sub>: Reserved

Table 2-3. Reset and Interrupt Vector Table

| Interrupt No. | Table address     |                   |  | Interrupt No. | Table address     |                   |
|---------------|-------------------|-------------------|--|---------------|-------------------|-------------------|
|               | Upper data        | Lower data        |  |               | Upper data        | Lower data        |
| Reset         | FFFE <sub>H</sub> | FFFF <sub>H</sub> |  | #11           | FFE4 <sub>H</sub> | FFE5 <sub>H</sub> |
|               | FFFC <sub>H</sub> | FFFD <sub>H</sub> |  | #12           | FFE2 <sub>H</sub> | FFE3 <sub>H</sub> |
| #0            | FFFA <sub>H</sub> | FFFB <sub>H</sub> |  | #13           | FFE0 <sub>H</sub> | FFE1 <sub>H</sub> |
| #1            | FFF8 <sub>H</sub> | FFF9 <sub>H</sub> |  | #14           | FFDE <sub>H</sub> | FFDF <sub>H</sub> |



Table 2-3. Reset and Interrupt Vector Table (*continued*)

| Interrupt No. | Table address     |                   |  | Interrupt No. | Table address     |                   |
|---------------|-------------------|-------------------|--|---------------|-------------------|-------------------|
|               | Upper data        | Lower data        |  |               | Upper data        | Lower data        |
| #2            | FFF6 <sub>H</sub> | FFF7 <sub>H</sub> |  | #15           | FFDC <sub>H</sub> | FFDD <sub>H</sub> |
| #3            | FFF4 <sub>H</sub> | FFF5 <sub>H</sub> |  | #16           | FFDA <sub>H</sub> | FFDB <sub>H</sub> |
| #4            | FFF2 <sub>H</sub> | FFF3 <sub>H</sub> |  | #17           | FFD8 <sub>H</sub> | FFD9 <sub>H</sub> |
| #5            | FFF0 <sub>H</sub> | FFF1 <sub>H</sub> |  | #18           | FFD6 <sub>H</sub> | FFD7 <sub>H</sub> |
| #6            | FFFE <sub>H</sub> | FFFF <sub>H</sub> |  | #19           | FFD4 <sub>H</sub> | FFD5 <sub>H</sub> |
| #7            | FFEC <sub>H</sub> | FFFD <sub>H</sub> |  | #20           | FFD2 <sub>H</sub> | FFD3 <sub>H</sub> |
| #8            | FFEA <sub>H</sub> | FFFB <sub>H</sub> |  | #21           | FFD0 <sub>H</sub> | FFD1 <sub>H</sub> |
| #9            | FFE8 <sub>H</sub> | FFF9 <sub>H</sub> |  | #22           | FFCE <sub>H</sub> | FFCF <sub>H</sub> |
| #10           | FFE6 <sub>H</sub> | FFE7 <sub>H</sub> |  | #23           | FFCC <sub>H</sub> | FFCD <sub>H</sub> |

FFFD<sub>H</sub>: 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

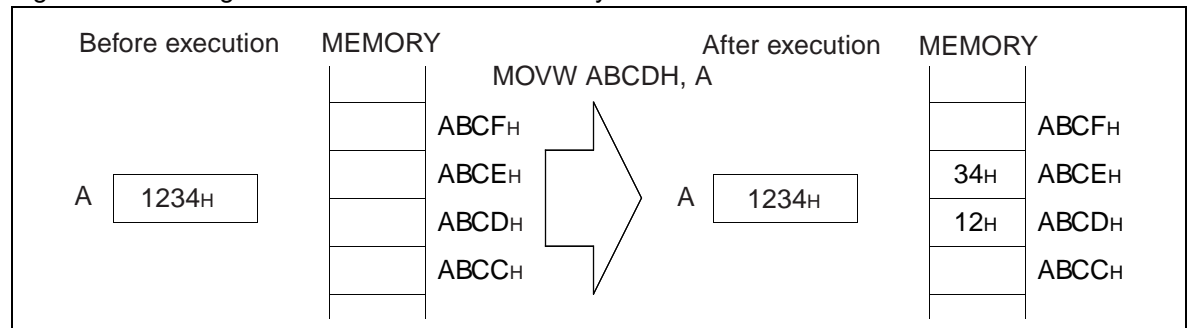
### 2.2.3 Arrangement of 16-bit Data in Memory Space

The F<sup>2</sup>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

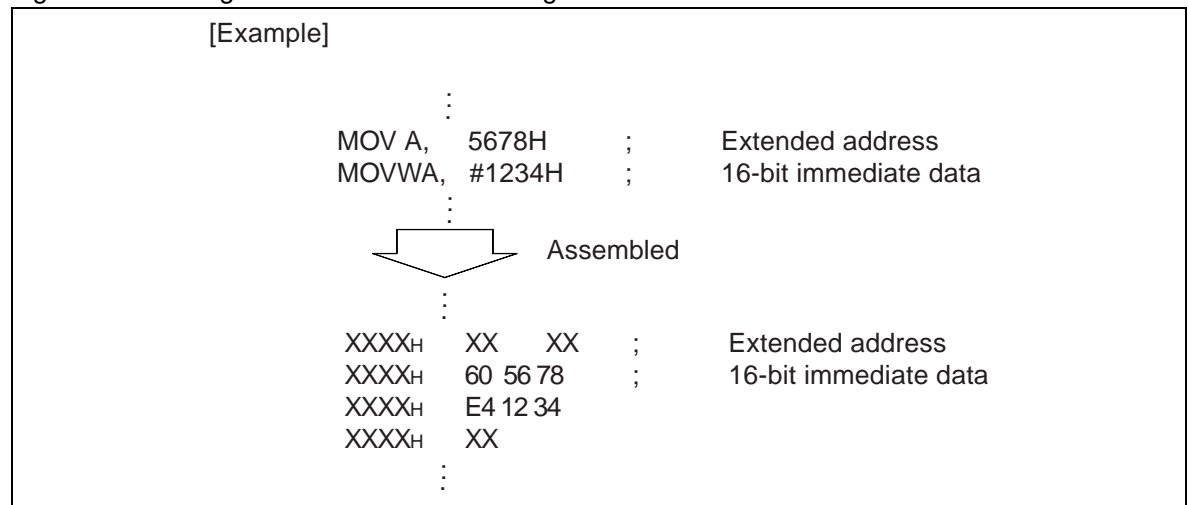
As shown in Figure 1-7., the F<sup>2</sup>MC-8FX CPU treats 16-bit data in the memory as upper data if it is written at the first location having a lower address and as lower data if it is written at the next location after that.

Figure 1-7. 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 1-8..

Figure 1-8. Arrangement of 16-bit Data during Instruction Execution



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

## 3. Registers



This chapter explains the F<sup>2</sup>MC-8FX dedicated registers and general-purpose registers.

### 3.1 F<sup>2</sup>MC-8FX Registers

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

### 3.3 Accumulator (A) and Temporary Accumulator (T)

### 3.4 Program Status (PS)

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

### 3.6 Register Banks

### 3.7 Direct Banks

### 3.1 F<sup>2</sup>MC-8FX Registers

In the F<sup>2</sup>MC-8FX series, there are two types of registers: dedicated registers in the CPU, and general-purpose registers in memory.

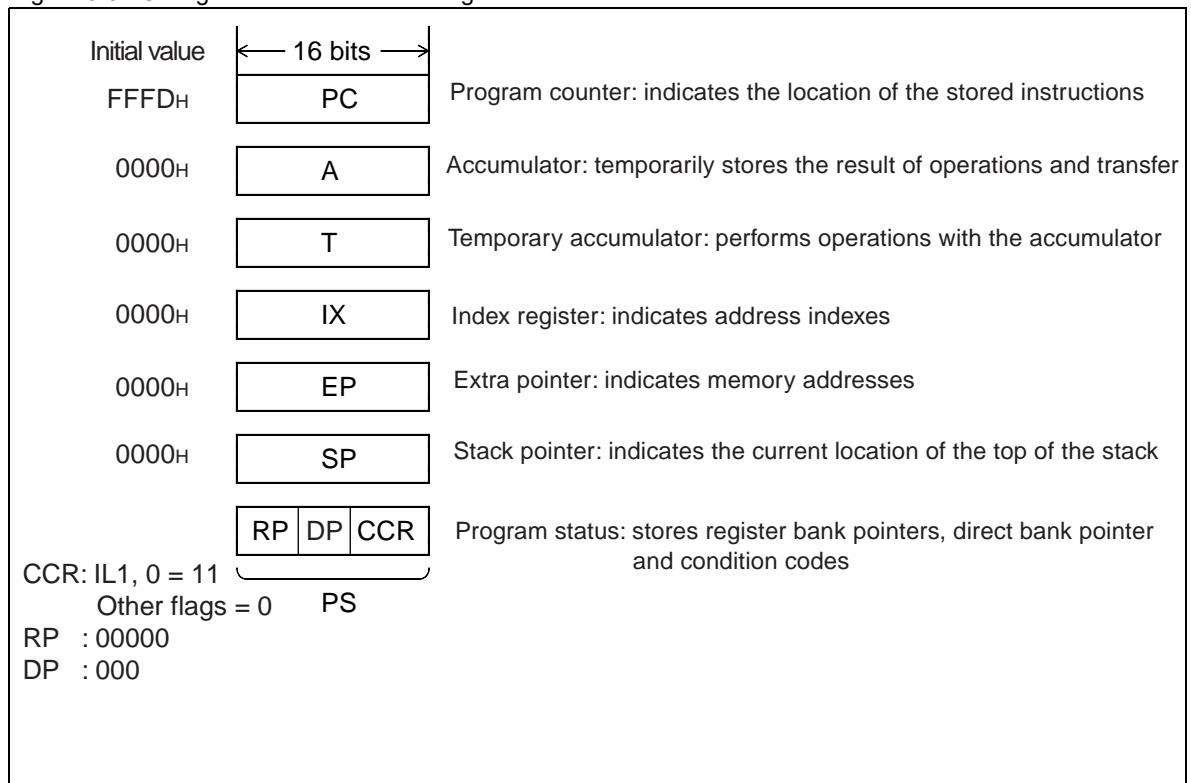
#### F<sup>2</sup>MC-8FX Dedicated Registers

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 3-9. shows the configuration of seven dedicated registers.

Figure 3-9. Configuration of Dedicated Registers



#### F<sup>2</sup>MC-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.

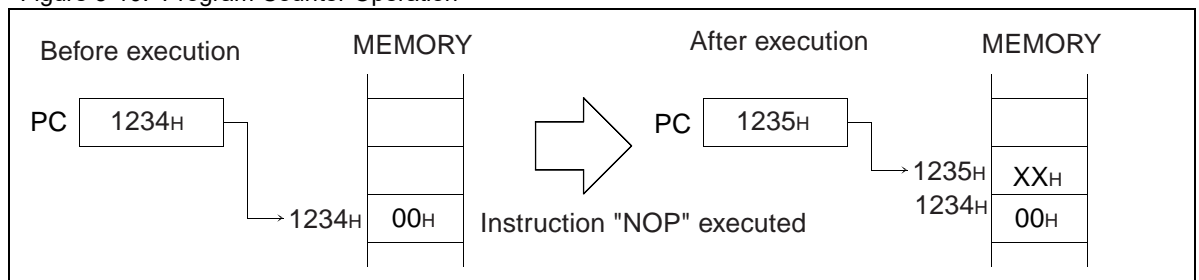
The program counter (PC) indicates the address of the location at which the instruction currently being executed is stored.

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 3-10. shows the operation of the program counter (PC).

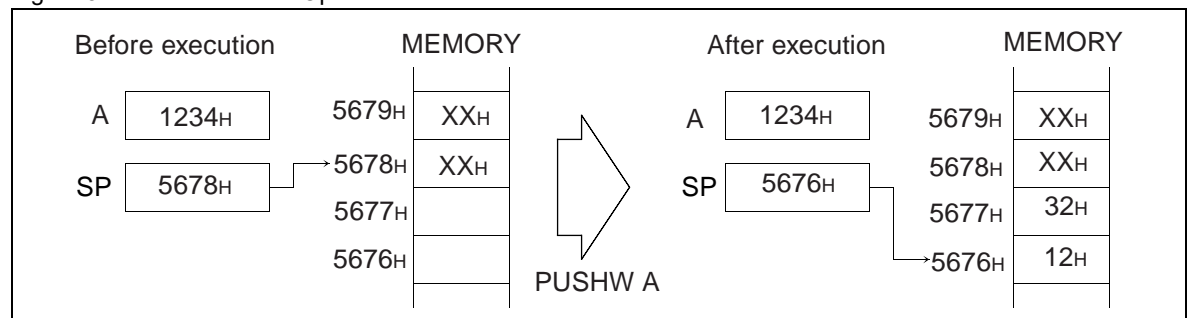
Figure 3-10. Program Counter Operation



### Stack Pointer (SP)

Figure 3-11. shows the operation of the stack pointer (SP).

Figure 3-11. 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 3-12.. For 8-bit operation only the lower 8 bits are used as shown in Figure 3-13..

Figure 3-12. Accumulator (A) Operation (16-bit Operation)

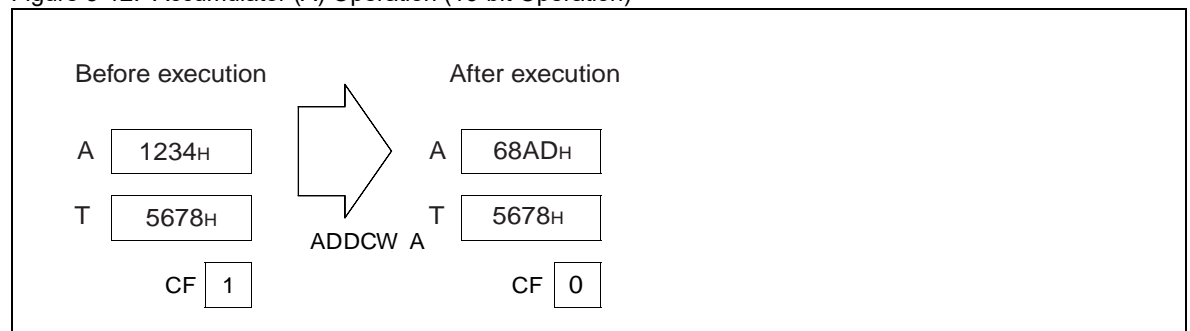
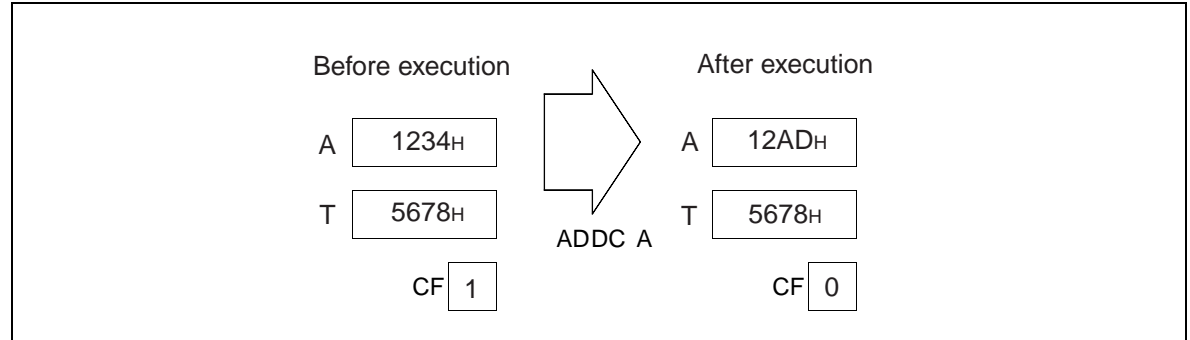


Figure 3-13. Accumulator (A) Operation (8-bit Operation)



#### 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 3-14.. 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 3-15.. Although all 16-bits are used as the operand for 16-bit operations as shown in Figure 3-16., only the lower 8 bits are used for 8-bit operations as shown in Figure 3-17..

Figure 3-14. Data Transfer between Accumulator (A) and Temporary Accumulator (T) (16-bit Transfer)

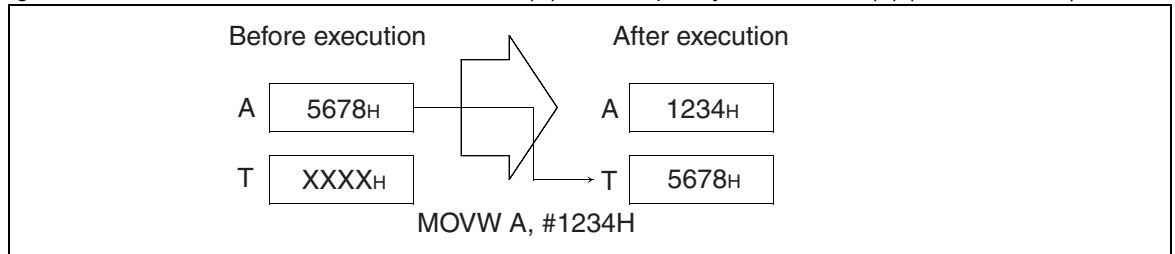


Figure 3-15. Data Transfer between Accumulator (A) and Temporary Accumulator (T) (8-bit Transfer)

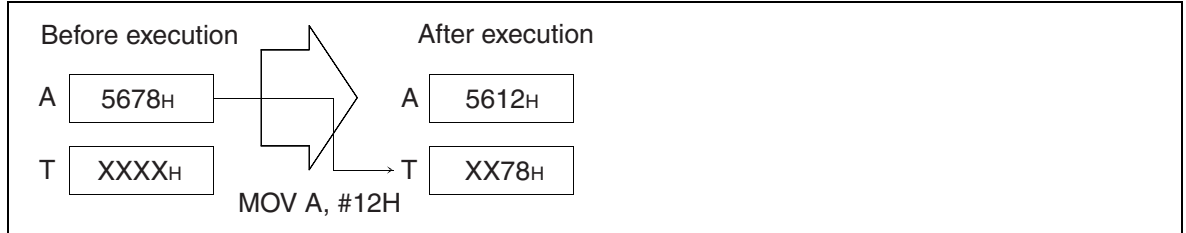


Figure 3-16. Operations between Accumulator (A) and Temporary Accumulator (T) (16-bit Operations)

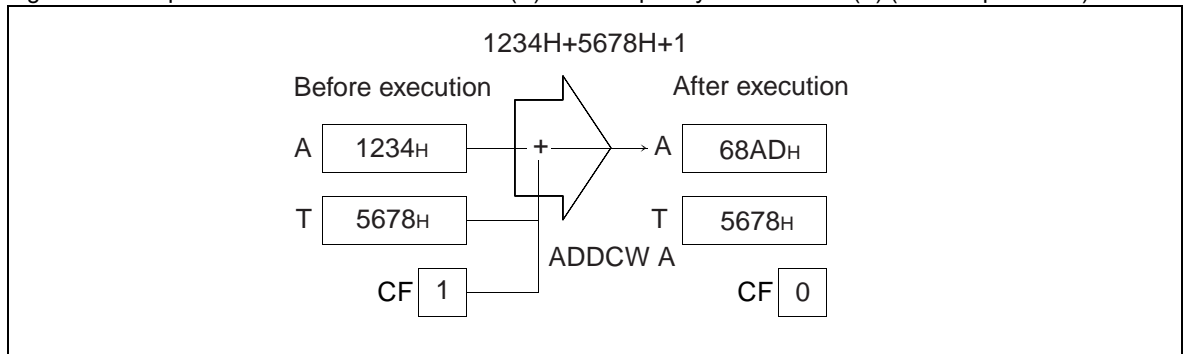
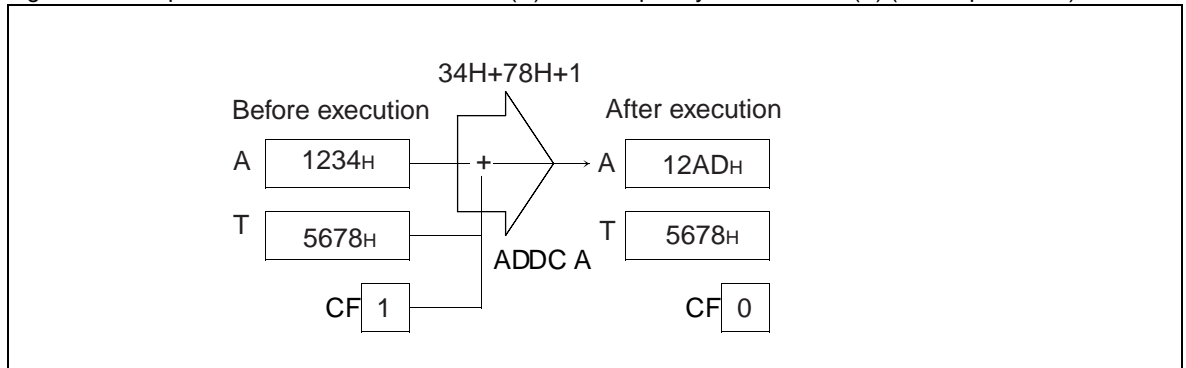


Figure 3-17. Operations between Accumulator (A) and Temporary Accumulator (T) (8-bit Operations)



### 3.3.1 How To Use The Temporary Accumulator (T)

The F<sup>2</sup>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<sup>2</sup>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<sub>H</sub> and 0042<sub>H</sub> is shown below.

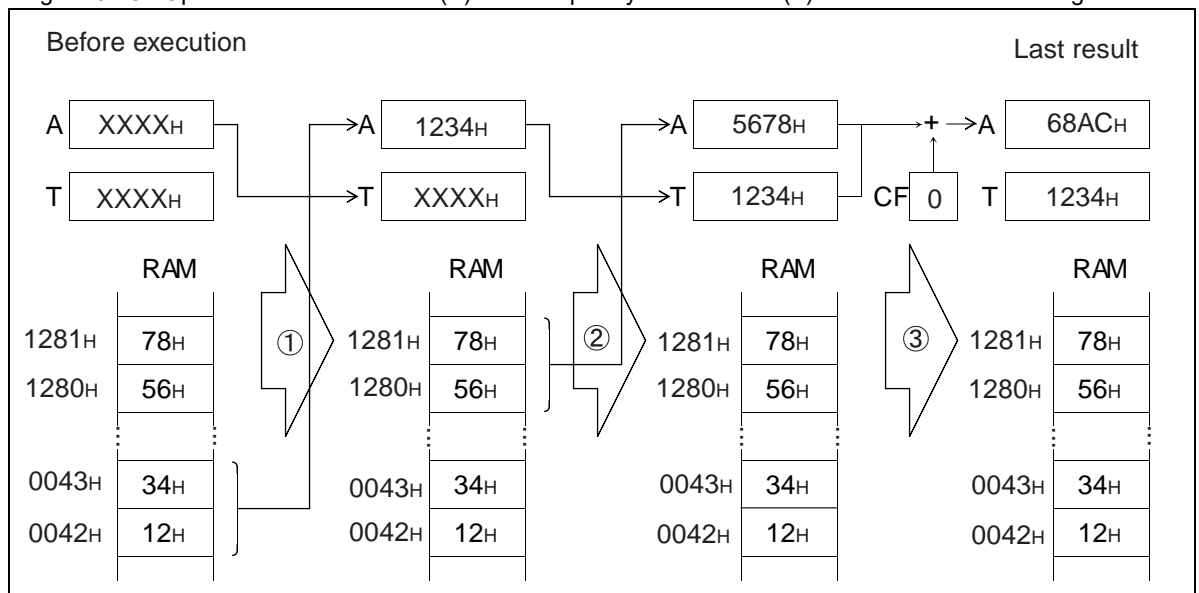
```

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

```

Figure 3-18. shows the operation for the accumulator and temporary accumulator when the above example is executed.

Figure 3-18. 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.

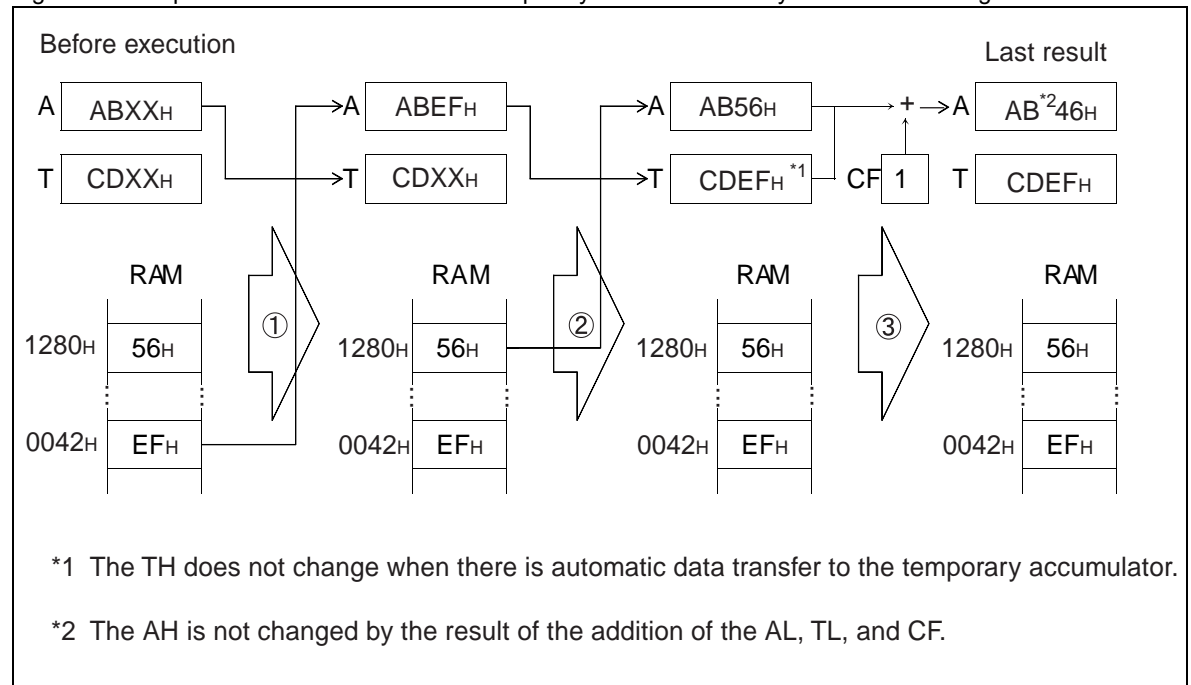
#### Example of Operation of Accumulator (A) and Temporary Accumulator (T) in Byte Data Processing

An example of addition with carry of 8-bit data stored at addresses 1280<sub>H</sub> and 0042<sub>H</sub> is shown below.

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

Figure 3-19. shows the operation of the accumulator and temporary accumulator when the above example is executed.

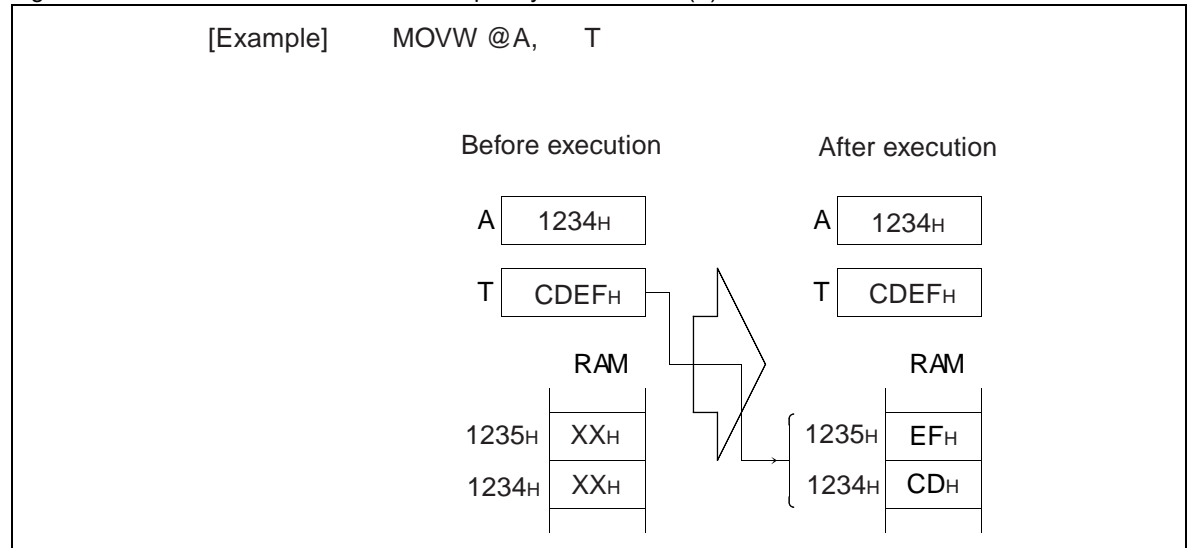
Figure 3-19. Operation of Accumulator and Temporary Accumulator in Byte Data Processing



## 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 3-20. 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<sub>H</sub>. So it is possible to make read and write accesses to them by an access to address 0078<sub>H</sub>.

#### Structure of Program Status (PS)

Figure 3-21. 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 3-22..

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. 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 3-21. Structure of Program Status (PS)

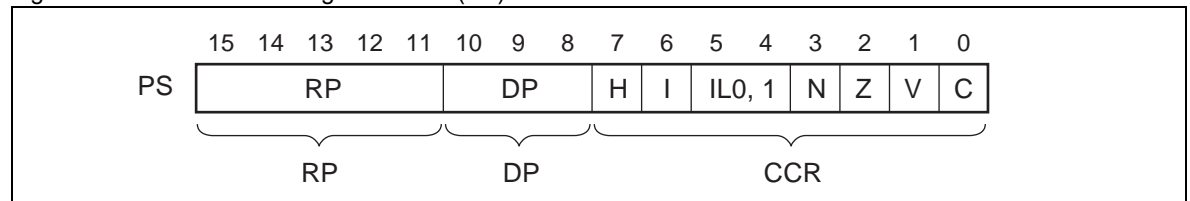


Figure 3-22. Conversion Rule for Actual Address of General-purpose Register Area

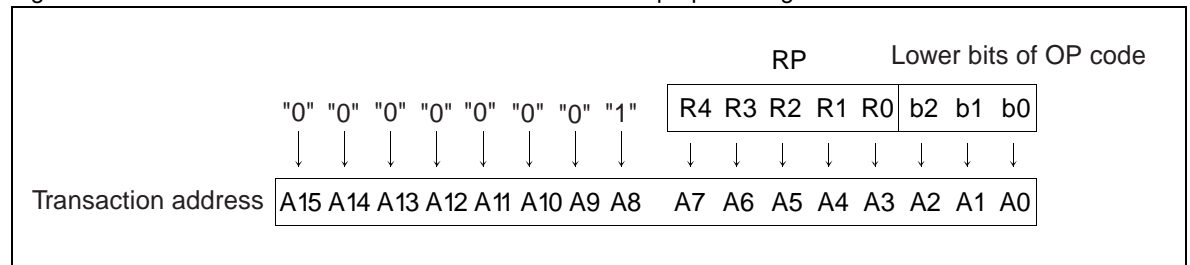


Table 3-4. Conversion Rule for Actual Address of Direct Addressing and Bit Direct Addressing

| Operand address                    | Direct bank pointer (DP) | Actual address                         |
|------------------------------------|--------------------------|--|
| 00 <sub>H</sub> to 7F <sub>H</sub> |                          | 0000 <sub>H</sub> to 007F <sub>H</sub> |
| 80 <sub>H</sub> to FF <sub>H</sub> | 000                      | 0080 <sub>H</sub> to 00FF <sub>H</sub> |
|                                    | 001                      | 0100 <sub>H</sub> to 017F <sub>H</sub> |
|                                    | 010                      | 0180 <sub>H</sub> to 01FF <sub>H</sub> |
|                                    | 011                      | 0200 <sub>H</sub> to 027F <sub>H</sub> |
|                                    | 100                      | 0280 <sub>H</sub> to 02FF <sub>H</sub> |
|                                    | 101                      | 0300 <sub>H</sub> to 037F <sub>H</sub> |
|                                    | 110                      | 0380 <sub>H</sub> to 03FF <sub>H</sub> |
|                                    | 111                      | 0400 <sub>H</sub> to 047F <sub>H</sub> |

## 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<br><br>Lowest |
| 0   | 1   | 1               |  |
| 1   | 0   | 2               |  |
| 1   | 1   | 3               |  |

### 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<sub>H</sub>. So it is possible to make read and write accesses to them by an access to address 0078<sub>H</sub>, 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.

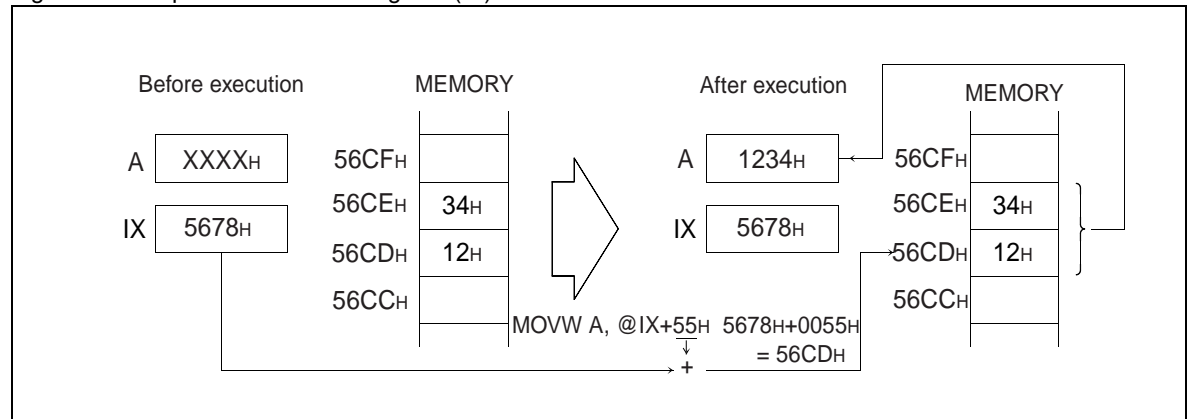
The index register (IX) adds an 8-bit offset value with its sign to generate the address stored by the operand.

The extra pointer (EP) indicates the address stored by the operand.

#### Index Register (IX)

Figure 3-23. indicates the operation of the index register.

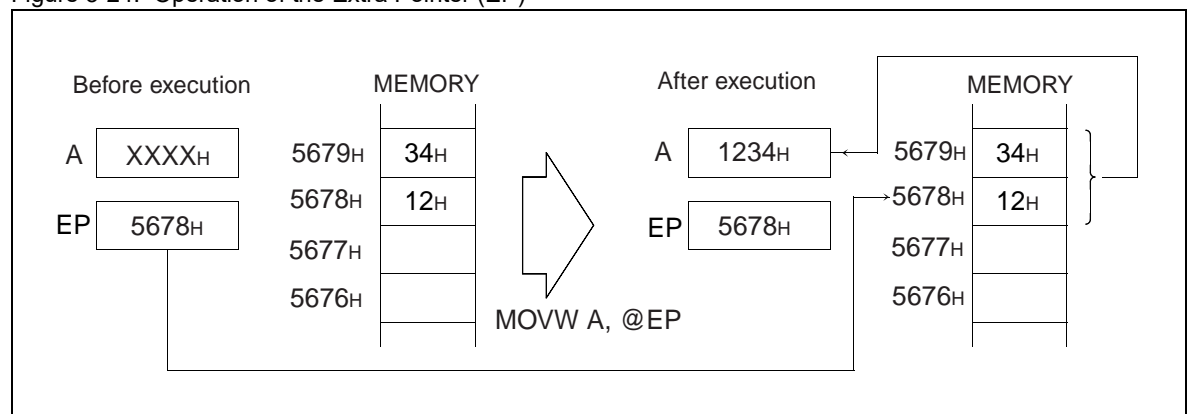
Figure 3-23. Operation of Index Register (IX)



#### Extra Pointer (EP)

Figure 3-24. shows the operation of the extra pointer.

Figure 3-24. Operation of the Extra Pointer (EP)



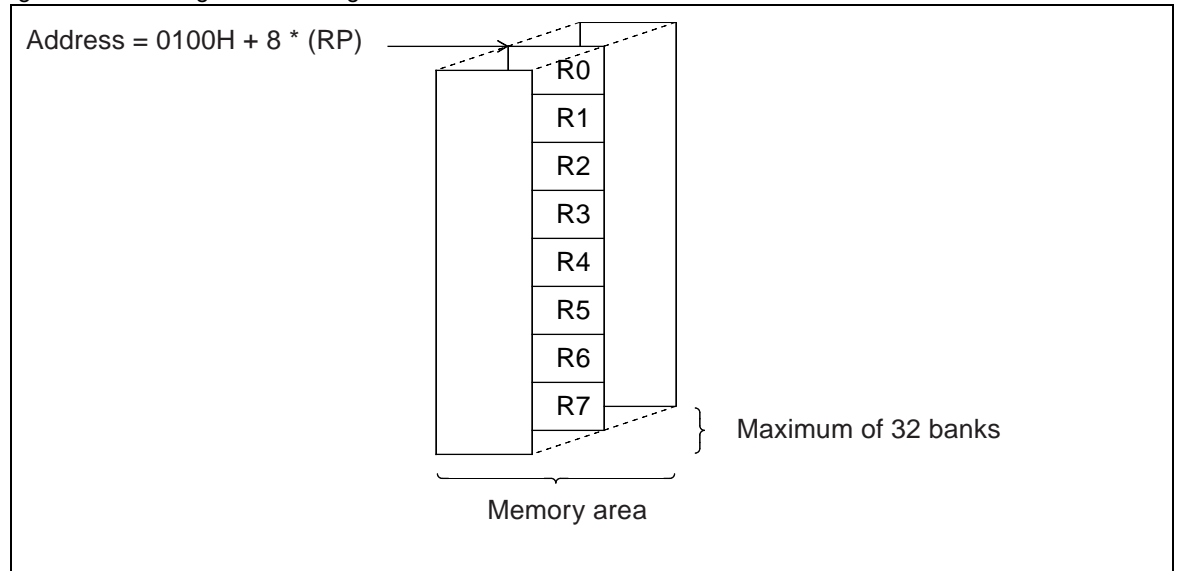
## 3.6 Register Banks

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 3-25. shows the configuration of the register bank.

Figure 3-25. Configuration of Register Bank



## 3.7 Direct Banks

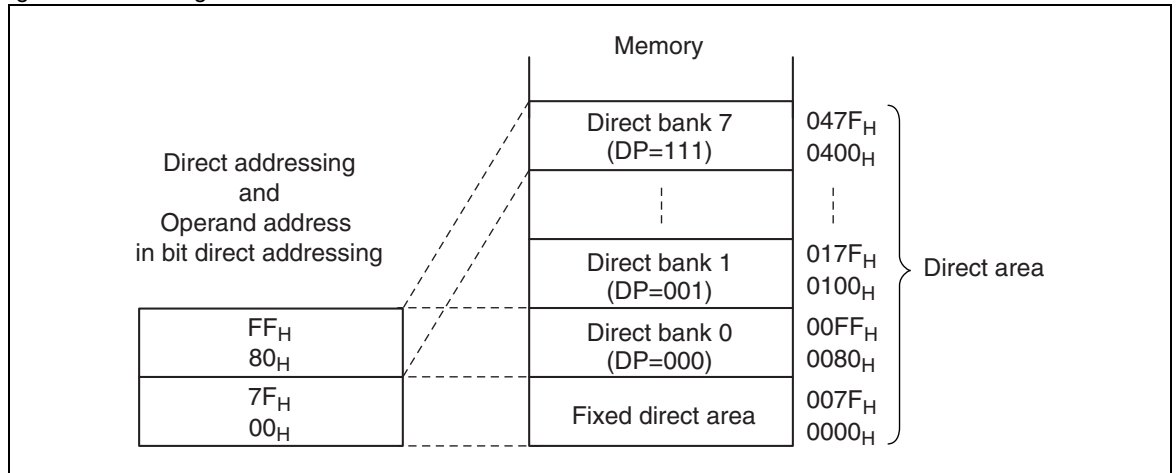
The direct bank is in 0080<sub>H</sub> to 047F<sub>H</sub> of direct area, and composed of 128 bytes × 8 banks. The access that uses direct addressing and bit direct addressing in operand address 80<sub>H</sub> to FF<sub>H</sub> can be extended to 8 direct banks according to the value of the direct bank pointer (DP). The current bank is indicated by the direct bank pointer (DP).

### Direct Bank

Figure 3-26. shows the configuration of a direct bank.

The access that uses direct addressing and bit direct addressing in operand address 80<sub>H</sub> to FF<sub>H</sub> 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<sub>H</sub> to 7F<sub>H</sub> is not affected by the value of the direct bank pointer (DP). This access is directed to fixed direct area 0000<sub>H</sub> to 007F<sub>H</sub>.

Figure 3-26. Configuration of Direct Bank





## 4. Interrupt Processing



This chapter explains the functions and operation of F<sup>2</sup>MC-8FX interrupt processing.

- 4.1 Outline of Interrupt Operation
- 4.2 Interrupt Enable/Disable and Interrupt Priority Functions
- 4.3 Creating an Interrupt Processing Program
- 4.4 Multiple Interrupt
- 4.5 Reset Operation

## 4.1 Outline of Interrupt Operation

F<sup>2</sup>MC-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.

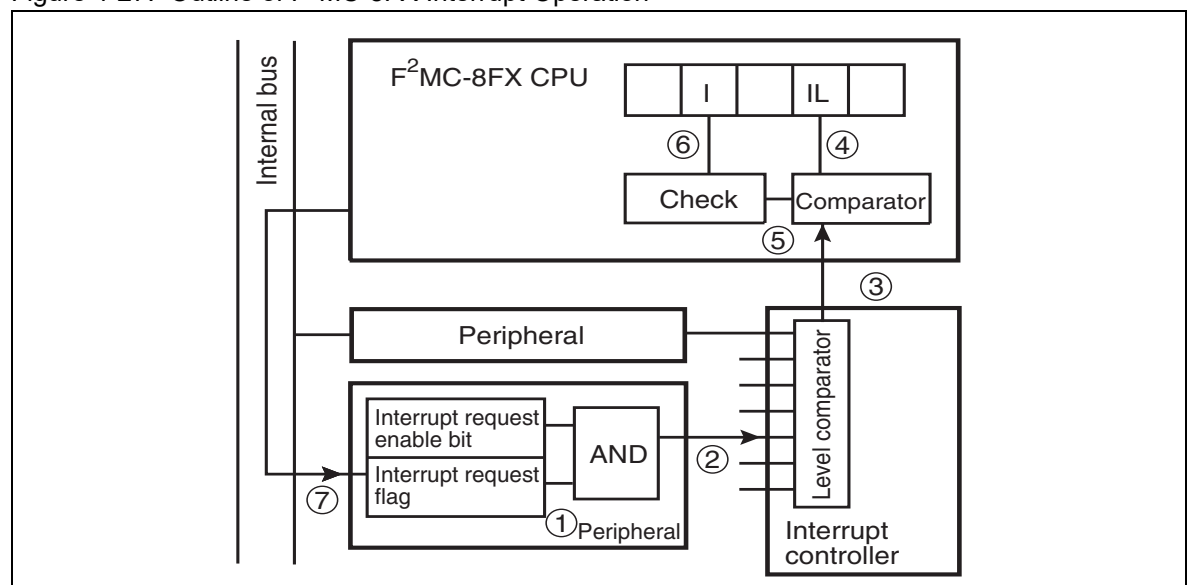
### Outline of Interrupt Operation

In the F<sup>2</sup>MC-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 4-27. shows the flow diagram of F<sup>2</sup>MC-8FX interrupt operation.

Figure 4-27. Outline of F<sup>2</sup>MC-8FX Interrupt Operation



## 4.2 Interrupt Enable/Disable and Interrupt Priority Functions

In the F<sup>2</sup>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

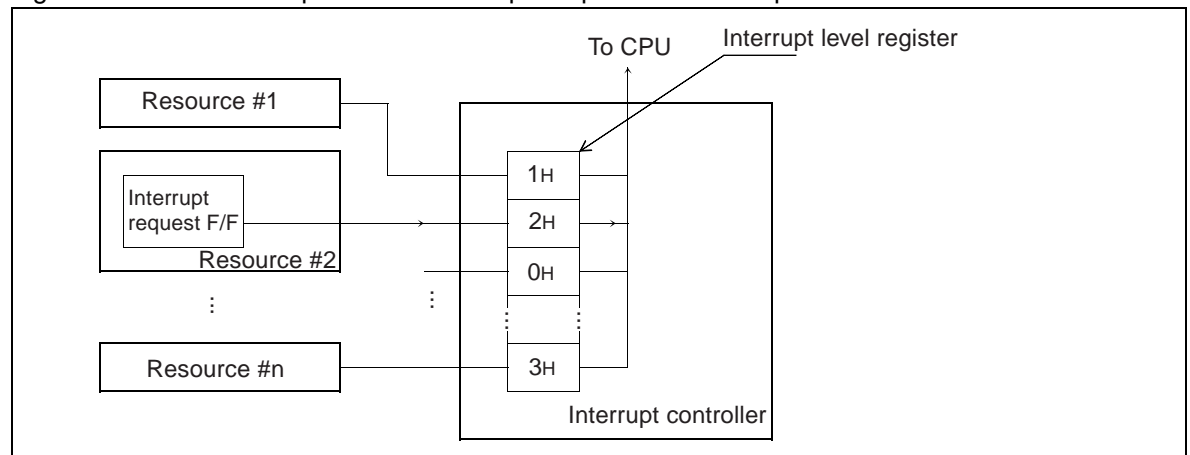
- Request enable check by interrupt enable flags in resources
- This is a function to enable/disable a request at the interrupt source. If interrupt enable flags in resources are enabled, interrupt request signals are sent from resources to the interrupt controller. This function is used for controlling 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 4-28., 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 4-28. Relationship between Interrupt Request and Interrupt Level in Resources



### 4.3 Creating an Interrupt Processing Program

In the F<sup>2</sup>MC-8FX series, basically, interrupt requests from resources are issued by hardware and cleared by software.

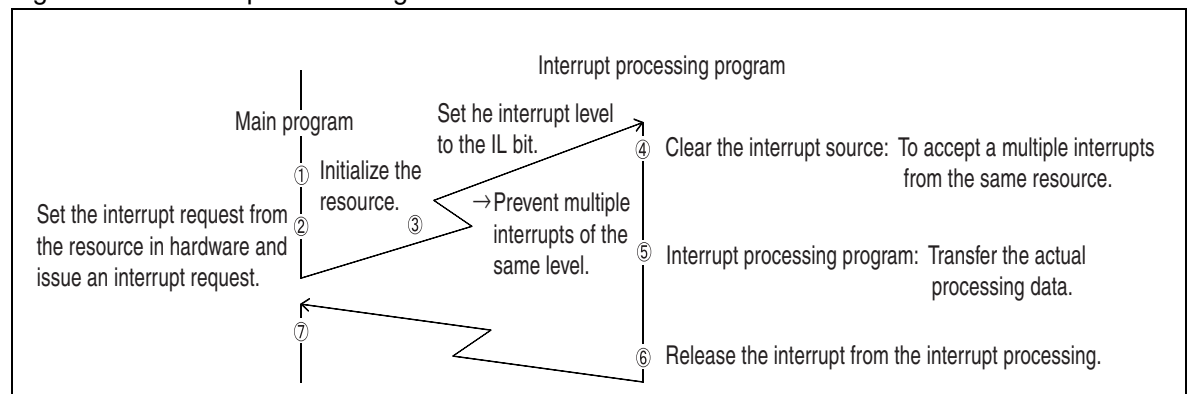
#### Creating an Interrupt Processing Program

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 4-29..

The numbers in the figure correspond to the numbers above.

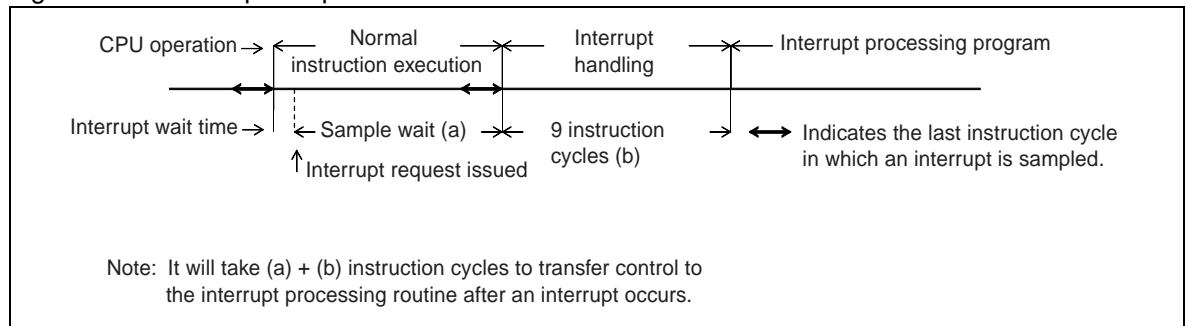
Figure 4-29. Interrupt Processing Flow



The time to transfer control to the interrupt processing routine after the occurrence of an interrupt 3 in Figure 4-29.) is 9 instruction cycles. An interrupt can only be processed in the last cycle of each instruction. The time shown in Figure 4-30. 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 4-30. Interrupt Response Time



## 4.4 Multiple Interrupt

The F<sup>2</sup>MC-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

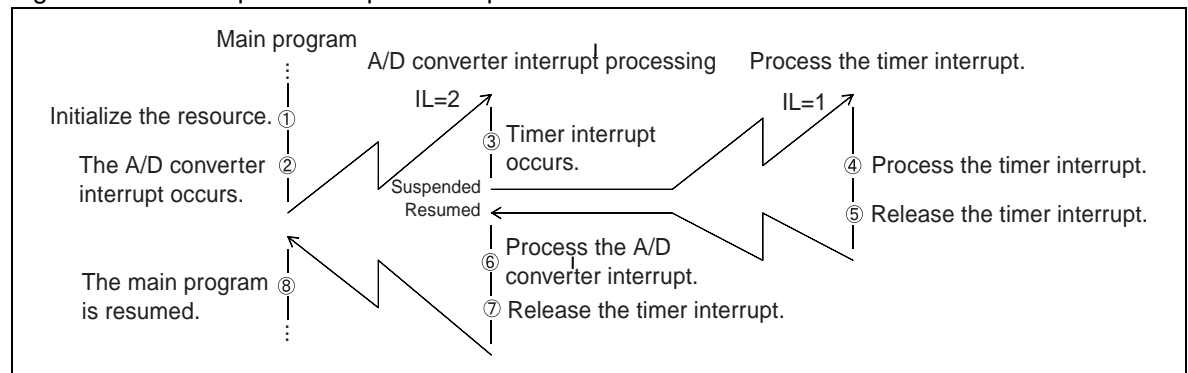
A specific example is given below.

- 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 4-31..

Figure 4-31. 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<sup>2</sup>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<sub>H</sub>, FFFF<sub>H</sub>).

### Reset Operation

A reset affects:

- Accumulator, temporary accumulator: Initializes to 0000<sub>H</sub>
- Stack pointer: Initializes to 0000<sub>H</sub>
- Extra pointer, index register: Initializes to 0000<sub>H</sub>
- 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.

## 5. CPU Software Architecture



This chapter explains the instructions for the F<sup>2</sup>MC-8FX CPU.

5.1 Types of Addressing Modes

5.2 Special Instructions



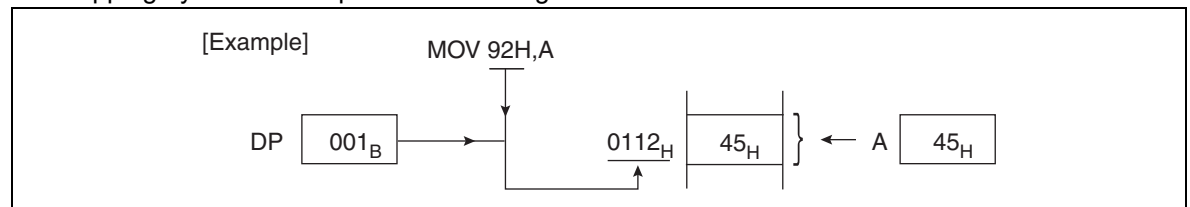
## 5.1 Types of Addressing Modes

The F<sup>2</sup>MC-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<sub>H</sub> to 047F<sub>H</sub>. In this addressing, when the operand address is 00<sub>H</sub> to 7F<sub>H</sub>, it accesses 0000<sub>H</sub> to 007F<sub>H</sub>. Moreover, when the operand address is 80<sub>H</sub> to FF<sub>H</sub>, the access is good to 0080<sub>H</sub> to 047F<sub>H</sub> at the mapping by direct bank pointer DP setting.

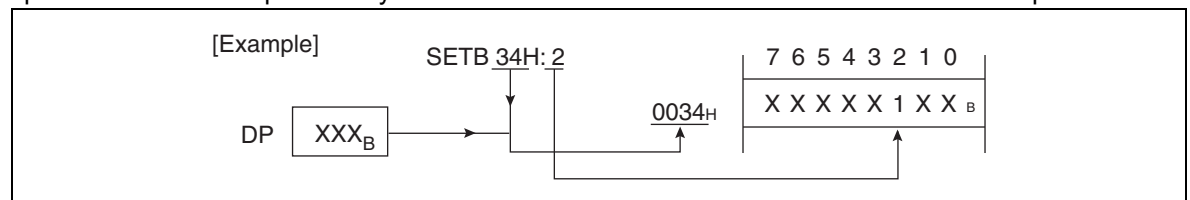


### Extended Addressing (ext)

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.

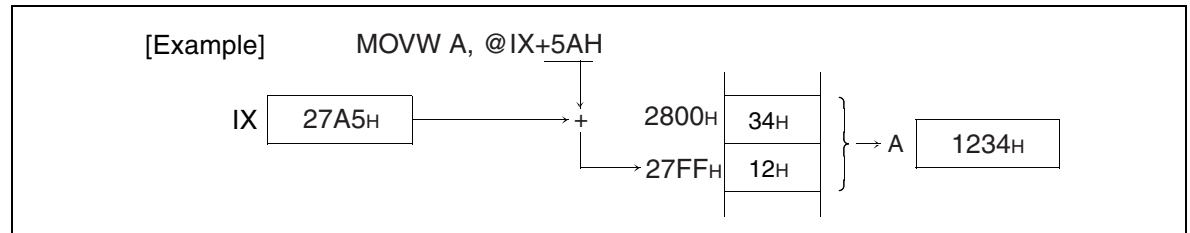
### Bit Direct Addressing (dir:b)

This addressing mode, indicated as "dir:b" in the instruction list, is used for bit-by-bit access of the direct area from 0000<sub>H</sub> to 047F<sub>H</sub>. In this addressing, when the operand address is 00<sub>H</sub> to 7F<sub>H</sub>, it accesses 0000<sub>H</sub> to 007F<sub>H</sub>. Moreover, when the operand address is 80<sub>H</sub> to FF<sub>H</sub>, the access is good to 0080<sub>H</sub> to 047F<sub>H</sub> 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.



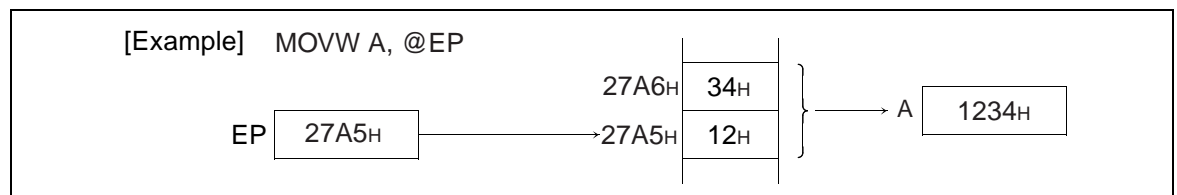
## Index Addressing (@IX+off)

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.



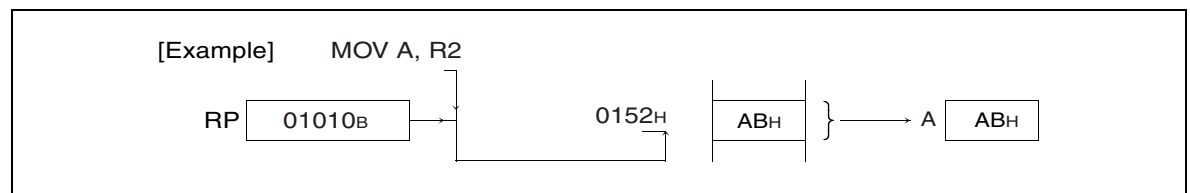
## Pointer Addressing (@EP)

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.



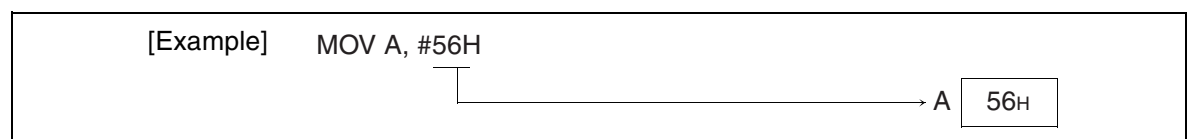
## General-Purpose Register Addressing (Ri)

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.

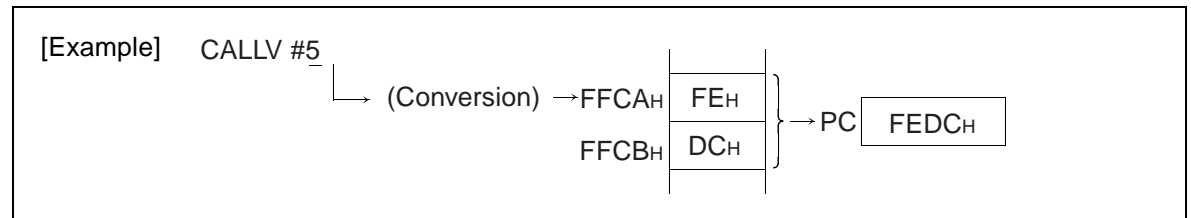


## Vector Addressing (#k)

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 5-5. are created.

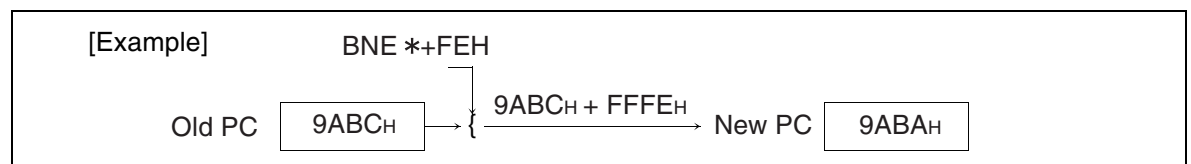
Table 5-5. Jump Address Table

| #k | Address table (upper jump address: lower jump address) |
|----|--|
| 0  | FFC0 <sub>H</sub> :FFC1 <sub>H</sub>                   |
| 1  | FFC2 <sub>H</sub> :FFC3 <sub>H</sub>                   |
| 2  | FFC4 <sub>H</sub> :FFC5 <sub>H</sub>                   |
| 3  | FFC6 <sub>H</sub> :FFC7 <sub>H</sub>                   |
| 4  | FFC8 <sub>H</sub> :FFC9 <sub>H</sub>                   |
| 5  | FFCA <sub>H</sub> :FFCB <sub>H</sub>                   |
| 6  | FFCC <sub>H</sub> :FFCD <sub>H</sub>                   |
| 7  | FFCE <sub>H</sub> :FFCF <sub>H</sub>                   |



## Relative Addressing (rel)

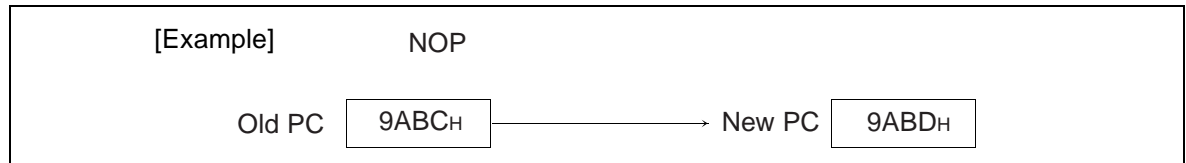
This addressing mode, indicated as "rel" in the instruction list, is used for branching to the 128-byte area across the program counter (PC). In this addressing mode, the contents of the operand are added with their sign, to the program counter. The result is stored in the program counter.



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.



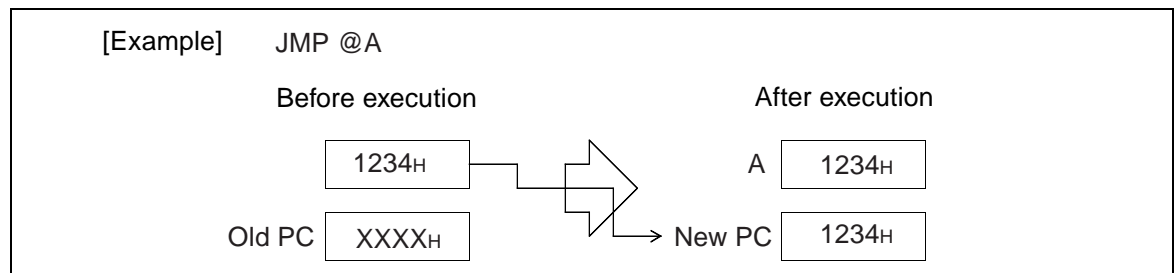
## 5.2 Special Instructions

In the F<sup>2</sup>MC-8FX series, the following six special instructions are available:

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

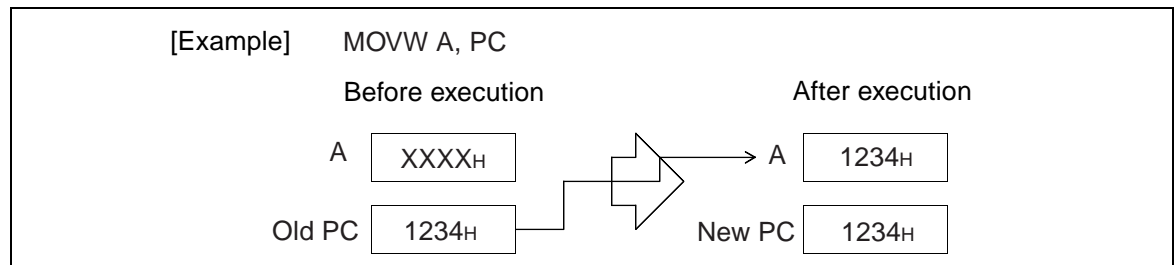
### JMP @A

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

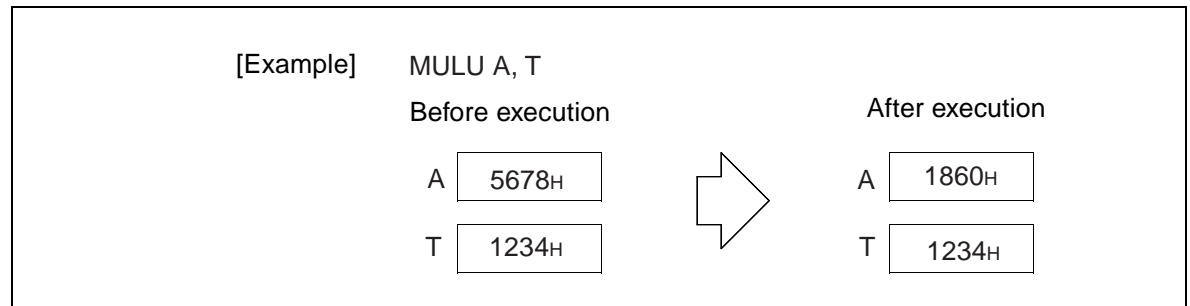
This instruction is used for performing the opposite operation to JMP @A. In other words, it stores the contents of the program counter (PC) in the accumulator (A). When this instruction is executed in the main routine and a specific subroutine is to be called, make sure that the contents of the accumulator 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<sub>H</sub> stored in the accumulator agrees with that of the address where the instruction code next to MOVW A, PC is stored.

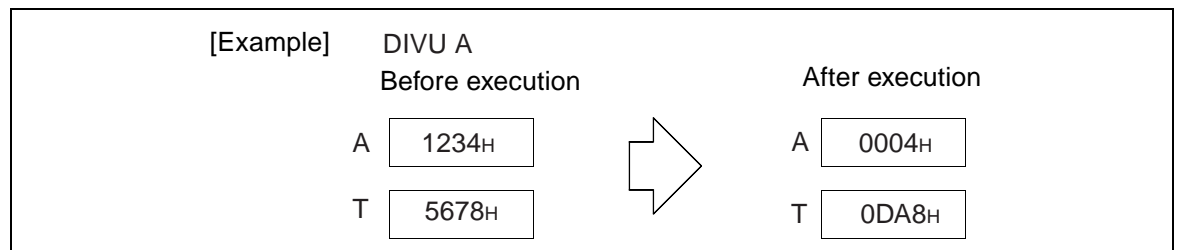
## MULU A

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.



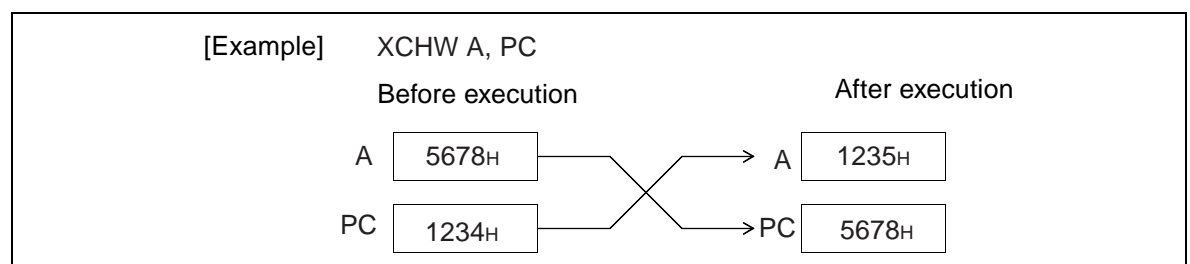
## DIVU A

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<sub>H</sub>, Z flag is 1 as 0 division. At this time, the operation result is not guaranteed.



## XCHW A, PC

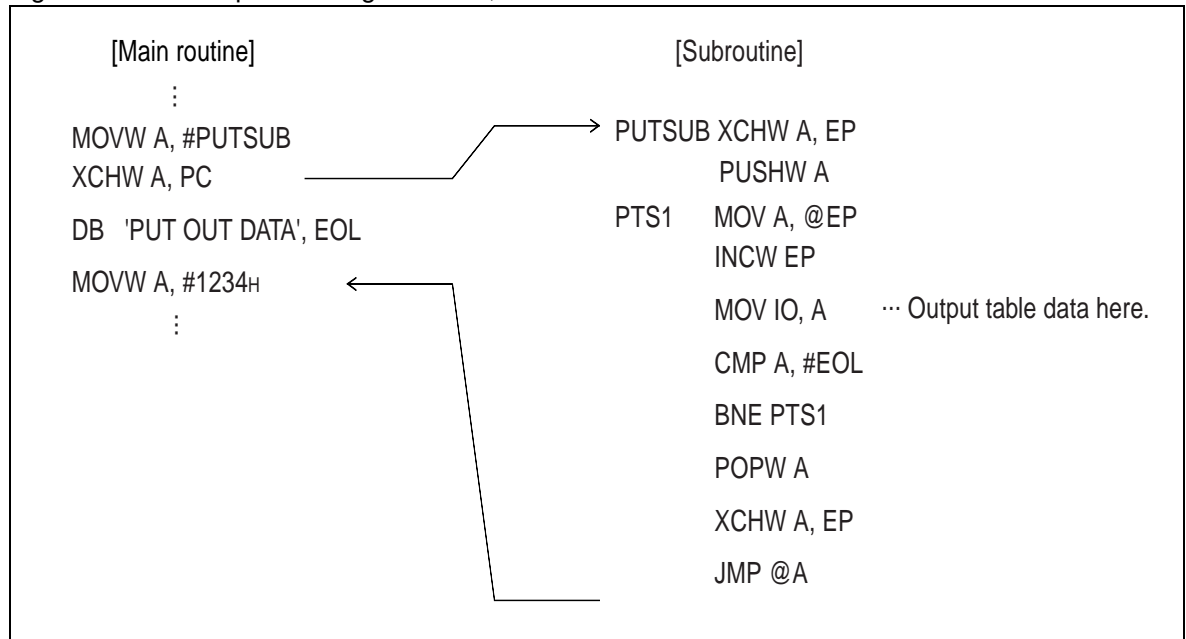
This instruction is used for exchanging the contents of the accumulator (A) for those of the program counter (PC). As a result, the program branches to the address indicated by the contents of the original accumulator and the contents of the current accumulator become the value of the address next to 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<sub>H</sub> agrees with that of the address where the instruction code next to XCHW A, PC is stored. Consequently, 1235<sub>H</sub> not 1234<sub>H</sub> is indicated.

Figure 5-32. Example of Using XCHW A, PC

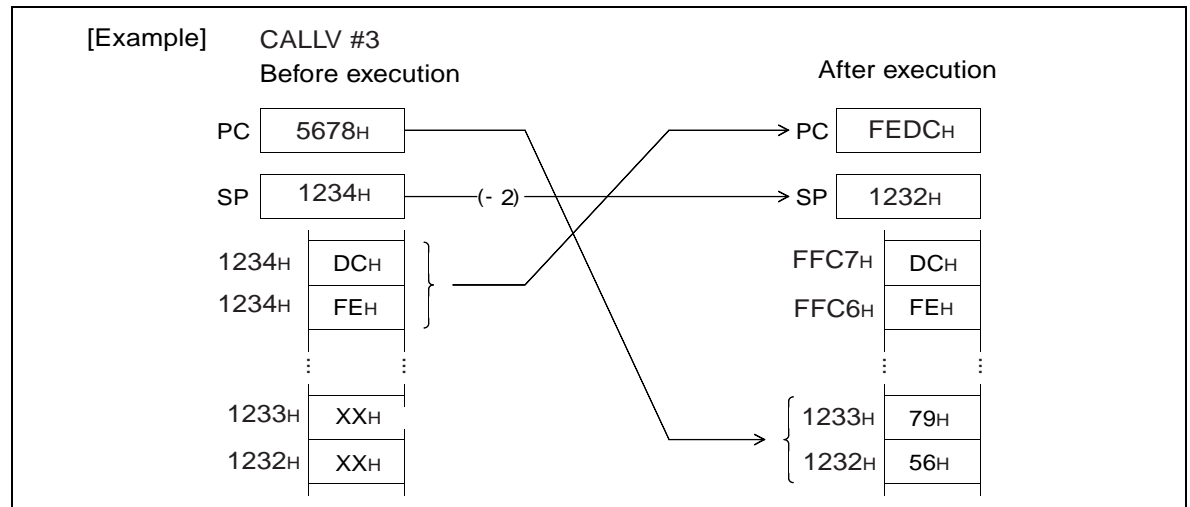


## CALLV #k

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 table addresses listed in Table 5-6. 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 5-6. Jump Address Table

| #k | Address table (upper jump address : lower jump address) |
|----|---|
| 0  | FFC0 <sub>H</sub> :FFC1 <sub>H</sub>                    |
| 1  | FFC2 <sub>H</sub> :FFC3 <sub>H</sub>                    |
| 2  | FFC4 <sub>H</sub> :FFC5 <sub>H</sub>                    |
| 3  | FFC6 <sub>H</sub> :FFC7 <sub>H</sub>                    |
| 4  | FFC8 <sub>H</sub> :FFC9 <sub>H</sub>                    |
| 5  | FFCA <sub>H</sub> :FFCB <sub>H</sub>                    |
| 6  | FFCC <sub>H</sub> :FFCD <sub>H</sub>                    |
| 7  | FFCE <sub>H</sub> :FFCF <sub>H</sub>                    |





## 6. Detailed Rules For Execution Instructions



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

All execution instructions 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 "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

$(AL) \leftarrow (AL) + (TL) + (C)$  (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<sub>H</sub> and set to 0 in other cases.

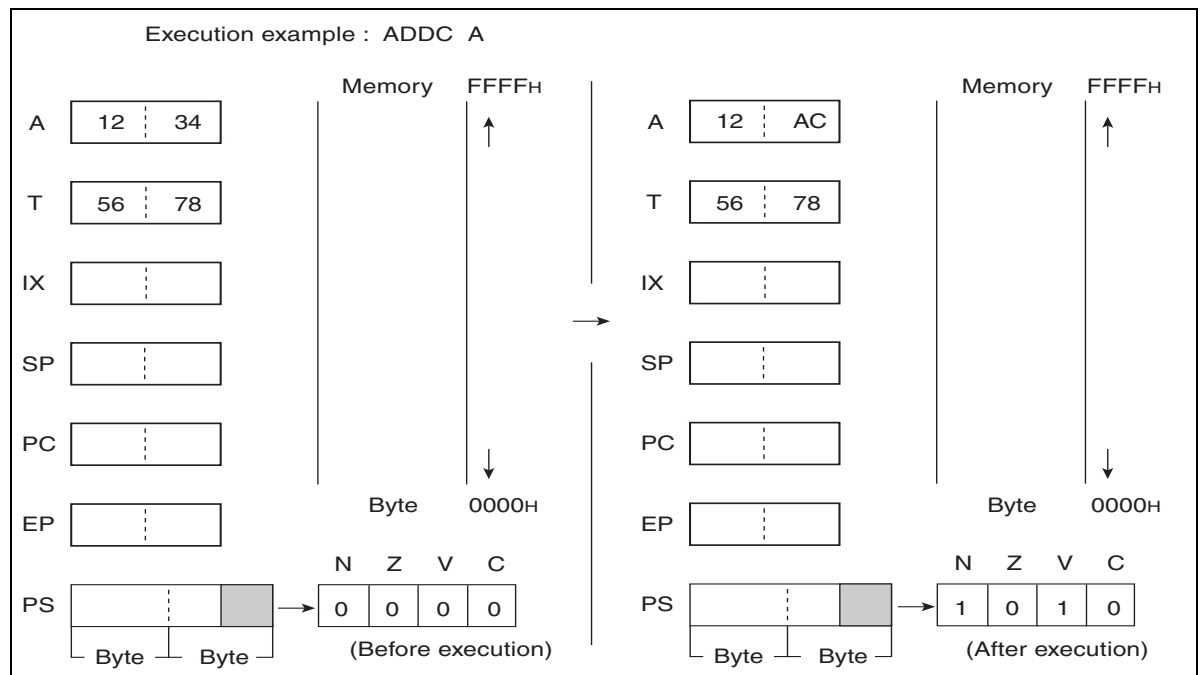
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

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

$(AL) \leftarrow (AL) + (EA) + (C)$  (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.

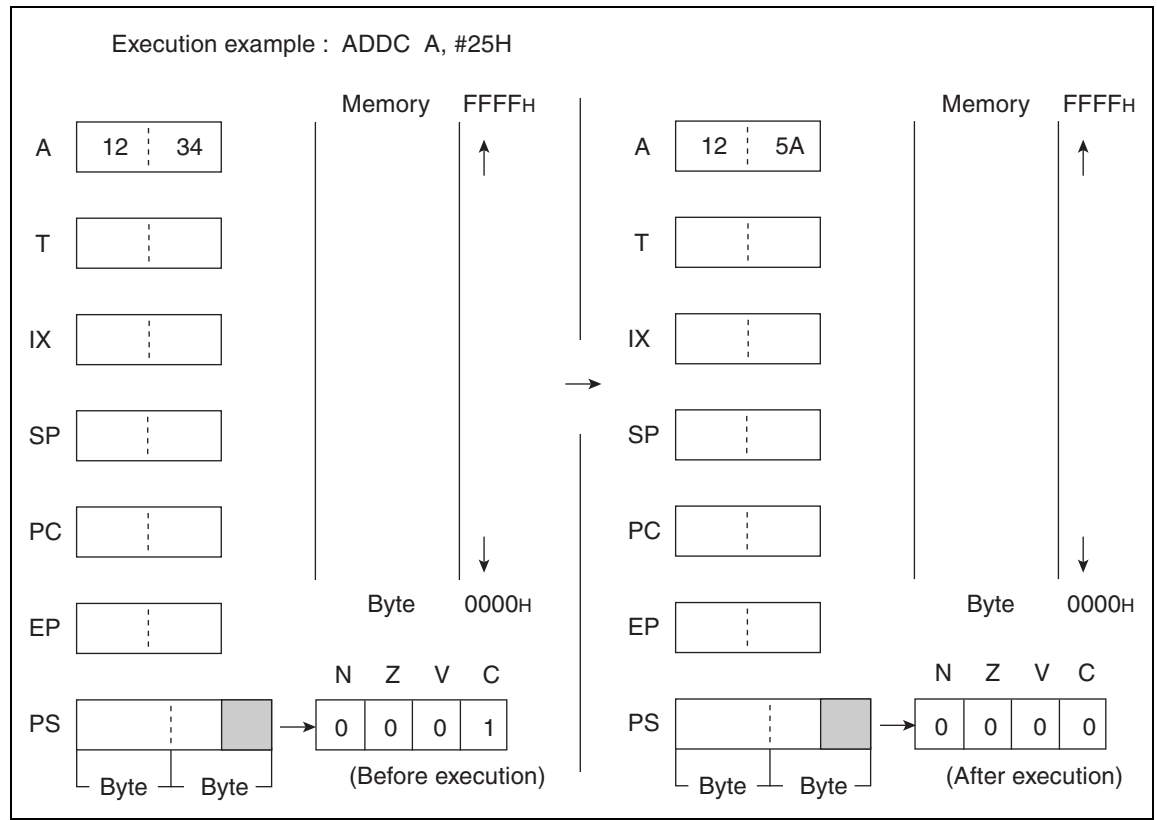
Z: Set to 1 if the result of operation is 00<sub>H</sub> and set to 0 in other cases.

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 6-7. 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

$(A) \leftarrow (A) + (T) + (C)$  (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<sub>H</sub> and set to 0 in other cases.

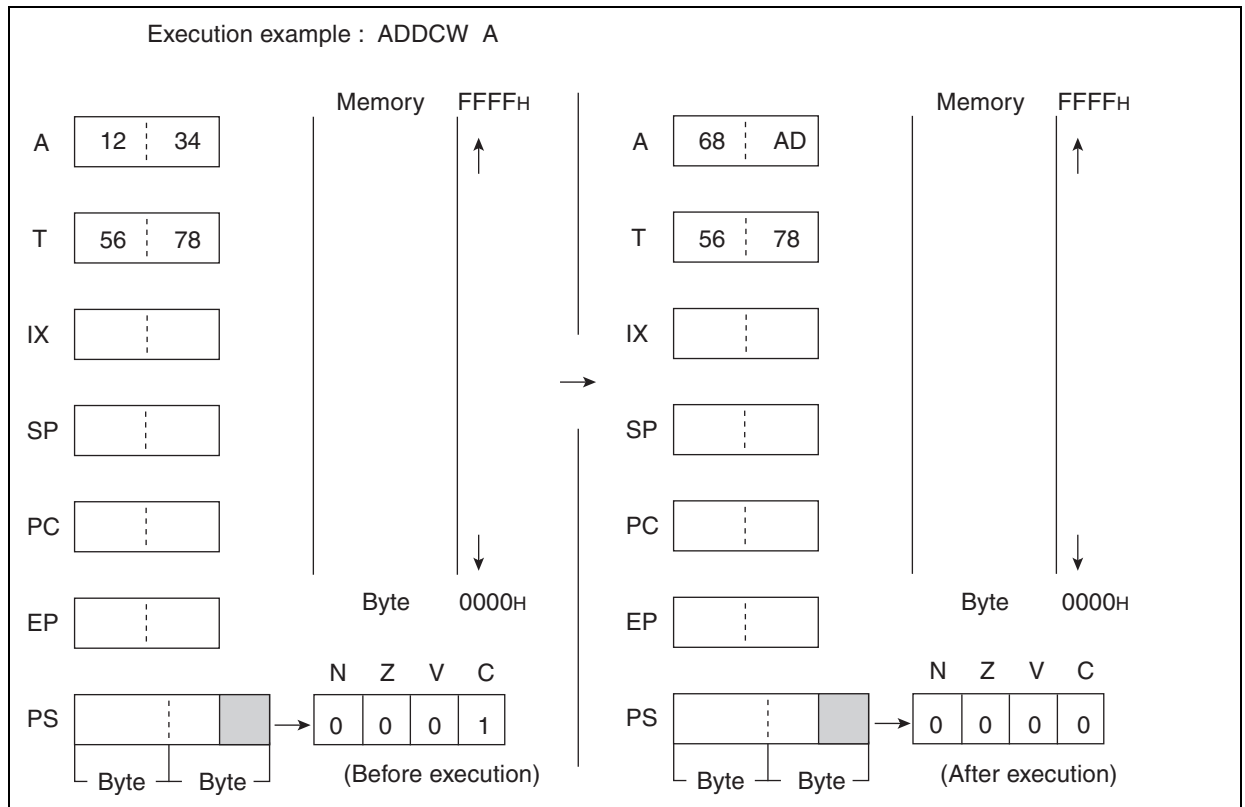
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

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

$(AL) \leftarrow (AL) \wedge (TL)$  (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<sub>H</sub> and set to 0 in other cases.

V: Always set to 0

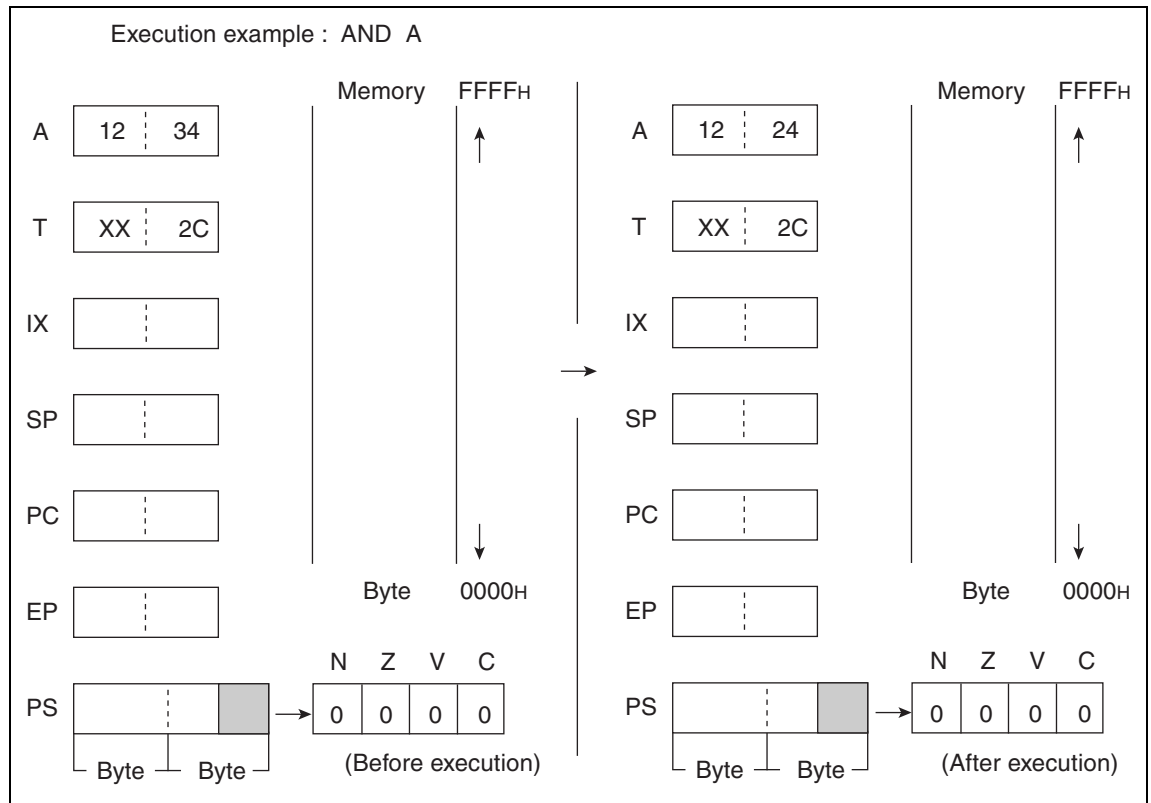
C: Not changed

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

$(AL) \leftarrow (AL) \wedge (EA)$  (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.

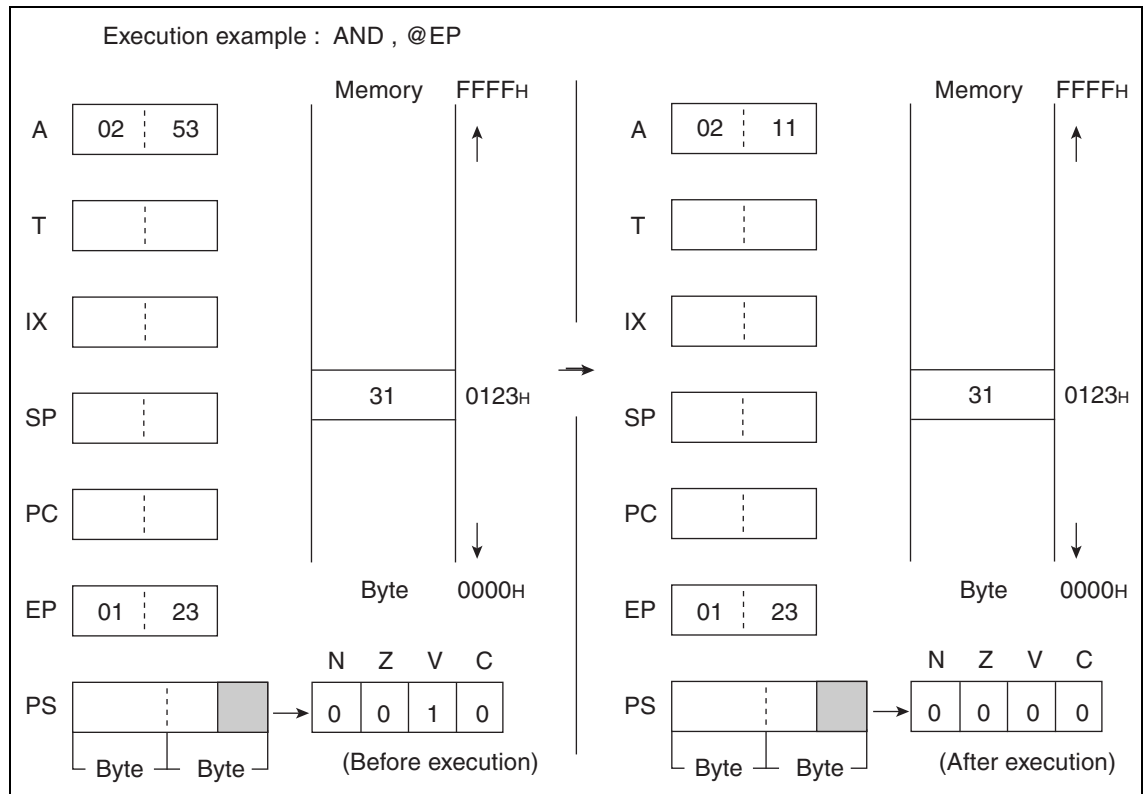
Z: Set to 1 if the result of operation is 00<sub>H</sub> and set to 0 in other cases.

V: Always set to 0

C: Not changed

Table 6-8. 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

$(A) \leftarrow (A) \wedge (T)$  (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<sub>H</sub> and set to 0 in other cases.

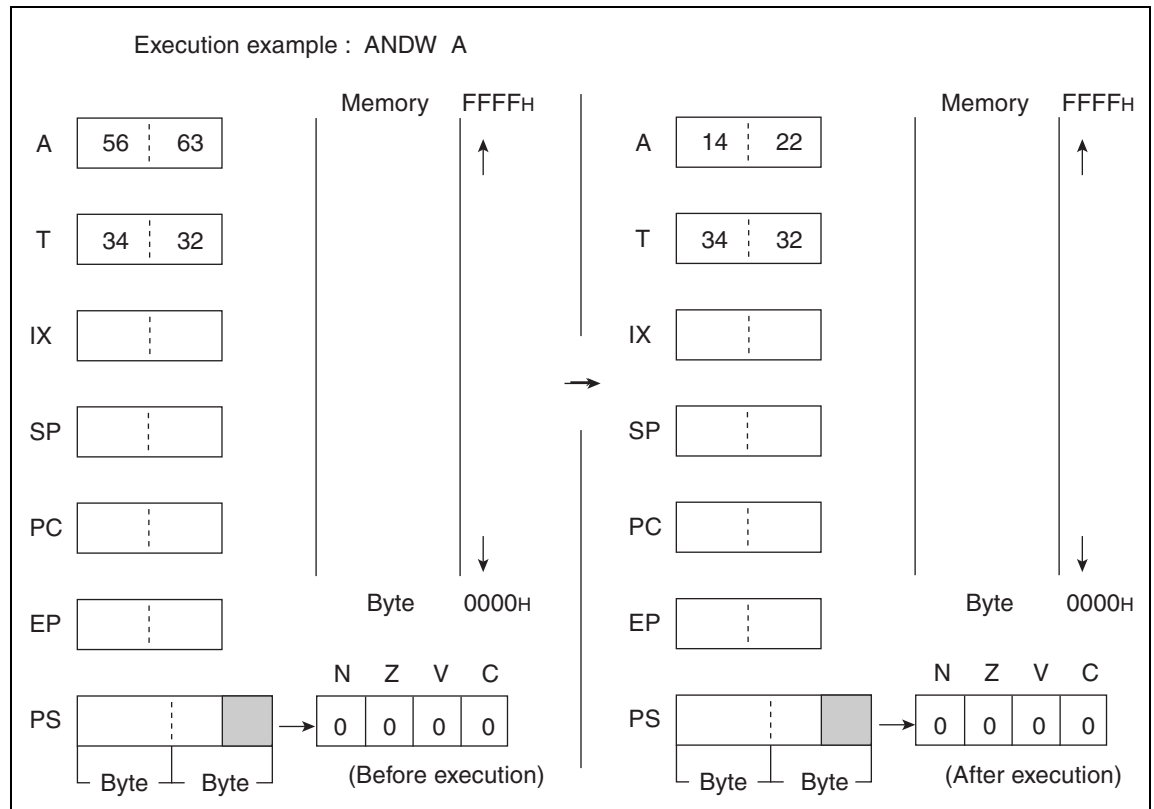
V: Always set to 0

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:  $(PC) \leftarrow (PC) + 3 + \text{rel}$  (Word addition)

(bit)b = 1:  $(PC) \leftarrow (PC) + 3$  (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.

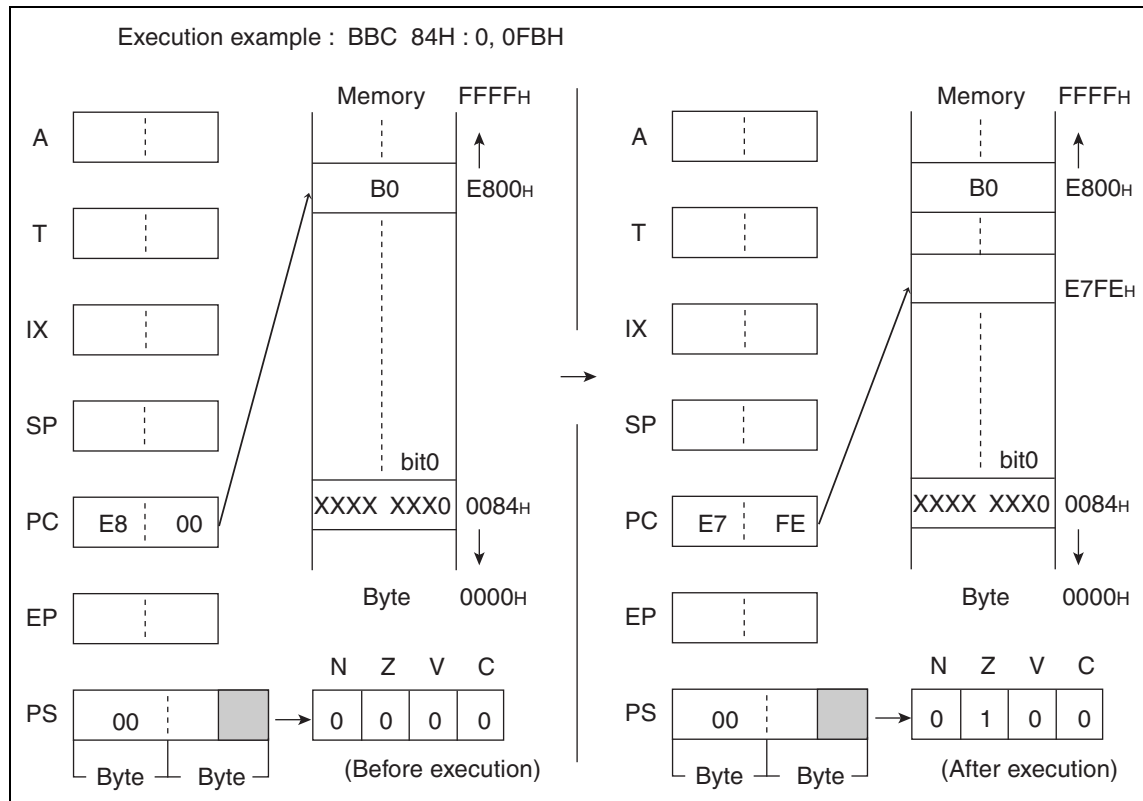
V: Not changed

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:  $(PC) \leftarrow (PC) + 3$  (Word addition)

(bit)b = 1:  $(PC) \leftarrow (PC) + 3 + \text{rel}$  (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

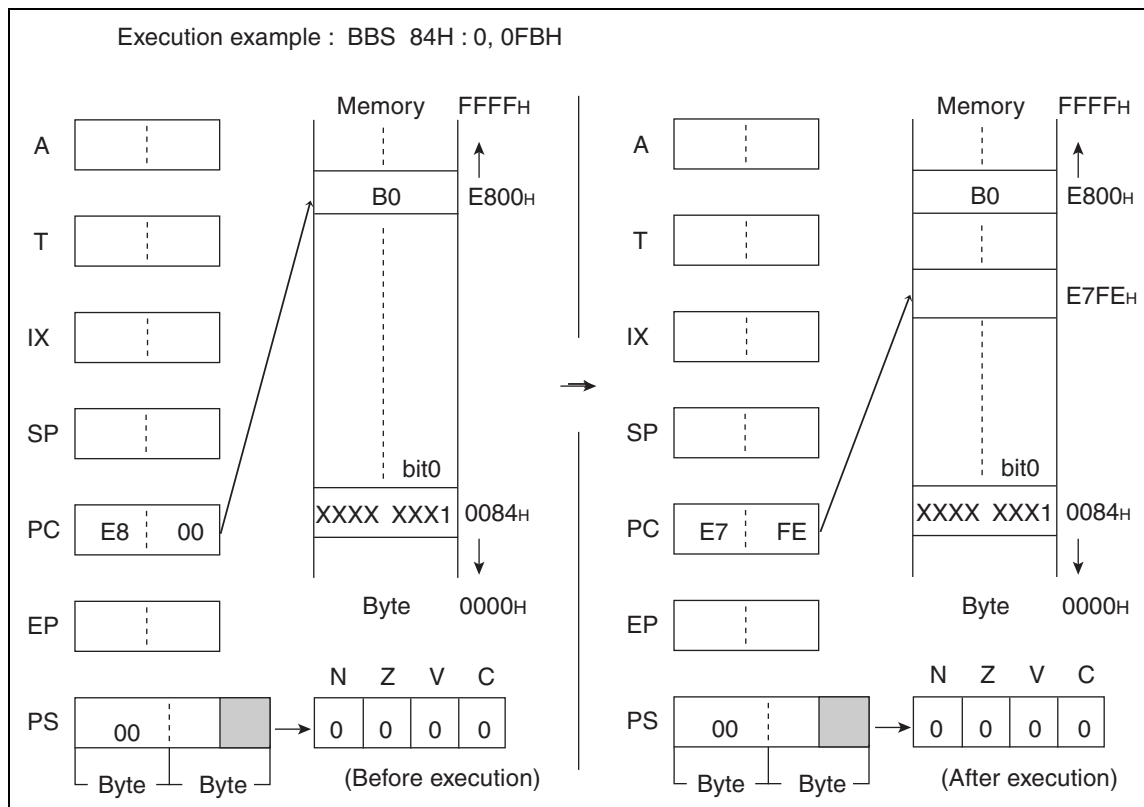
C: Not changed

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:  $(PC) \leftarrow (PC) + 2$  (Word addition)

(C) = 1:  $(PC) \leftarrow (PC) + 2 + \text{rel}$  (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

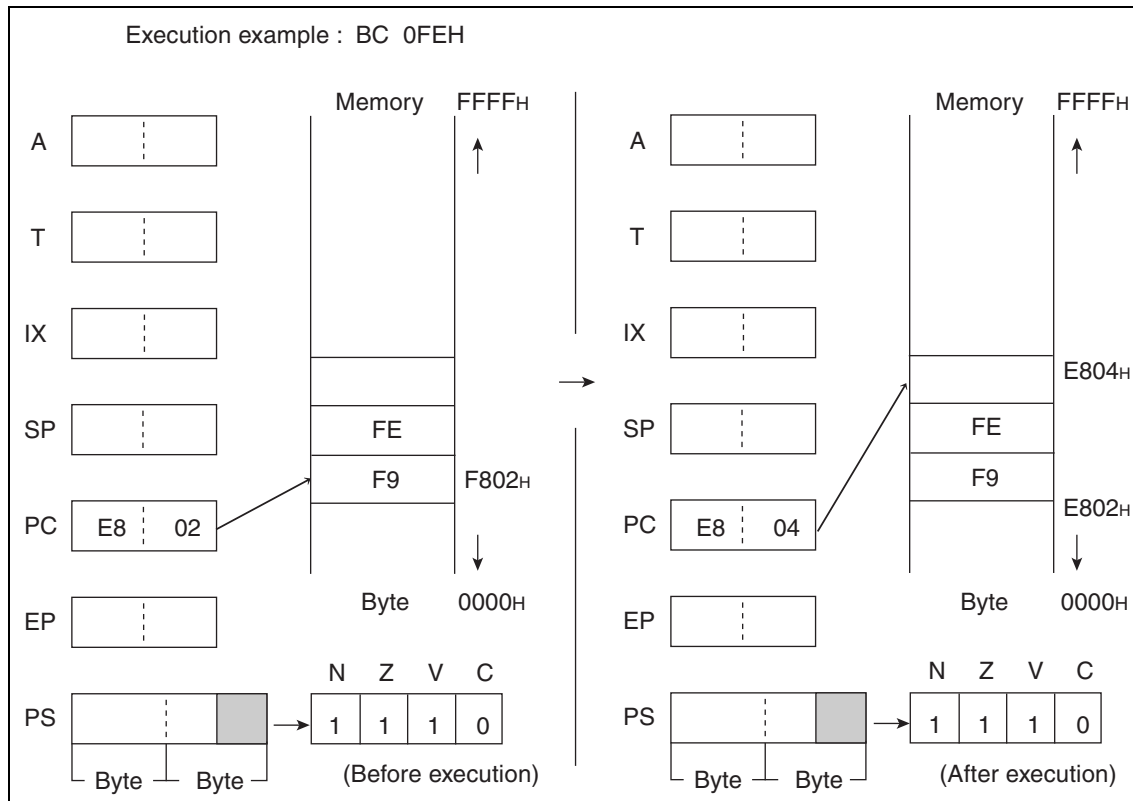
V: Not changed

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

$(V) \vee (N) = 1: (PC) \leftarrow (PC) + 2$  (Word addition)

$(V) \vee (N) = 0: (PC) \leftarrow (PC) + 2 + \text{rel}$  (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

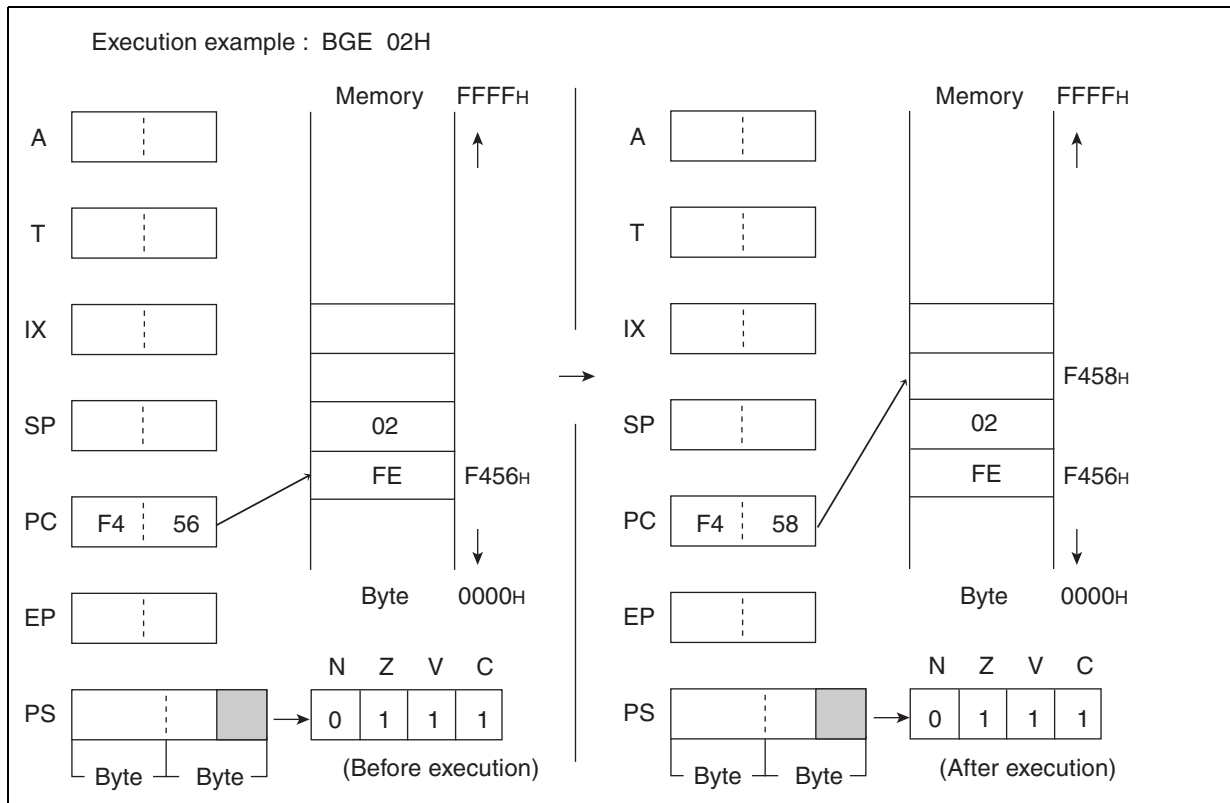
V: Not changed

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

$(V) \vee (N) = 0: (PC) \leftarrow (PC) + 2$  (Word addition)

$(V) \vee (N) = 1: (PC) \leftarrow (PC) + 2 + \text{rel}$  (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

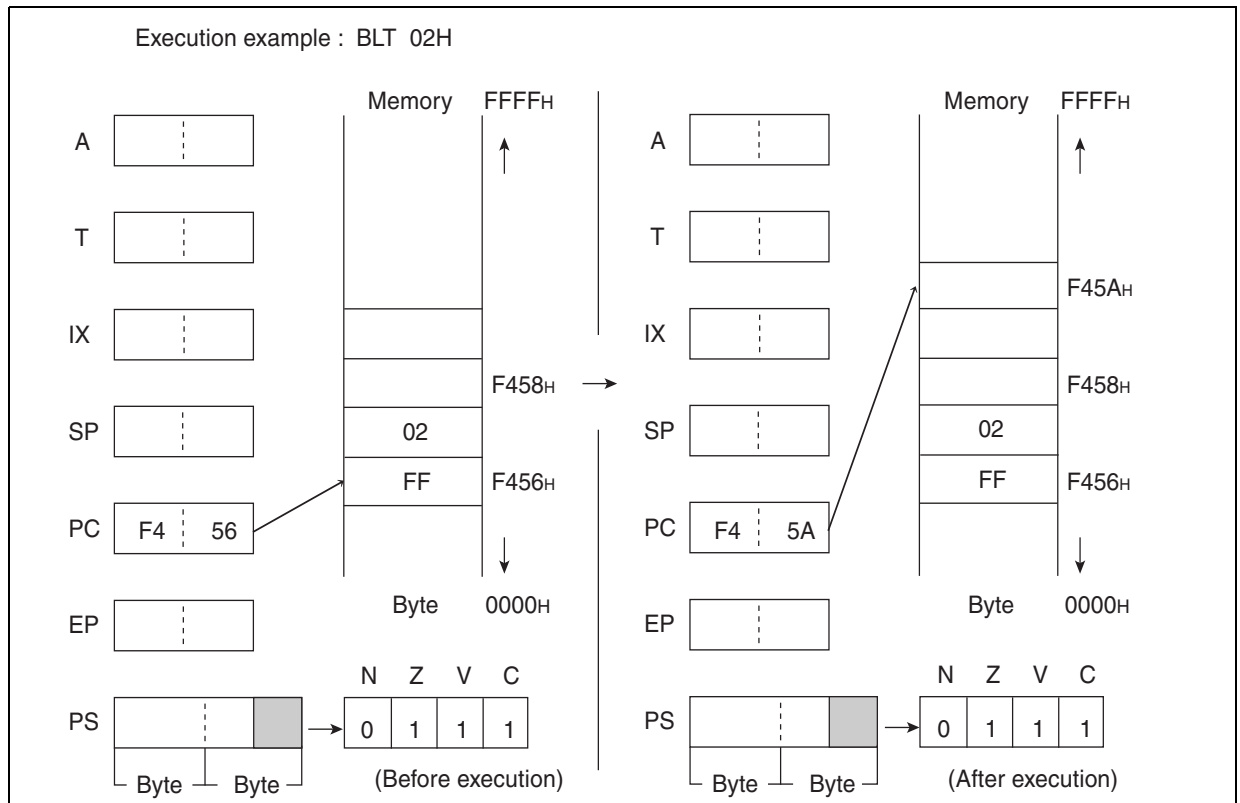
V: Not changed

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:  $(PC) \leftarrow (PC) + 2$  (Word addition)

N = 1:  $(PC) \leftarrow (PC) + 2 + \text{rel}$  (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

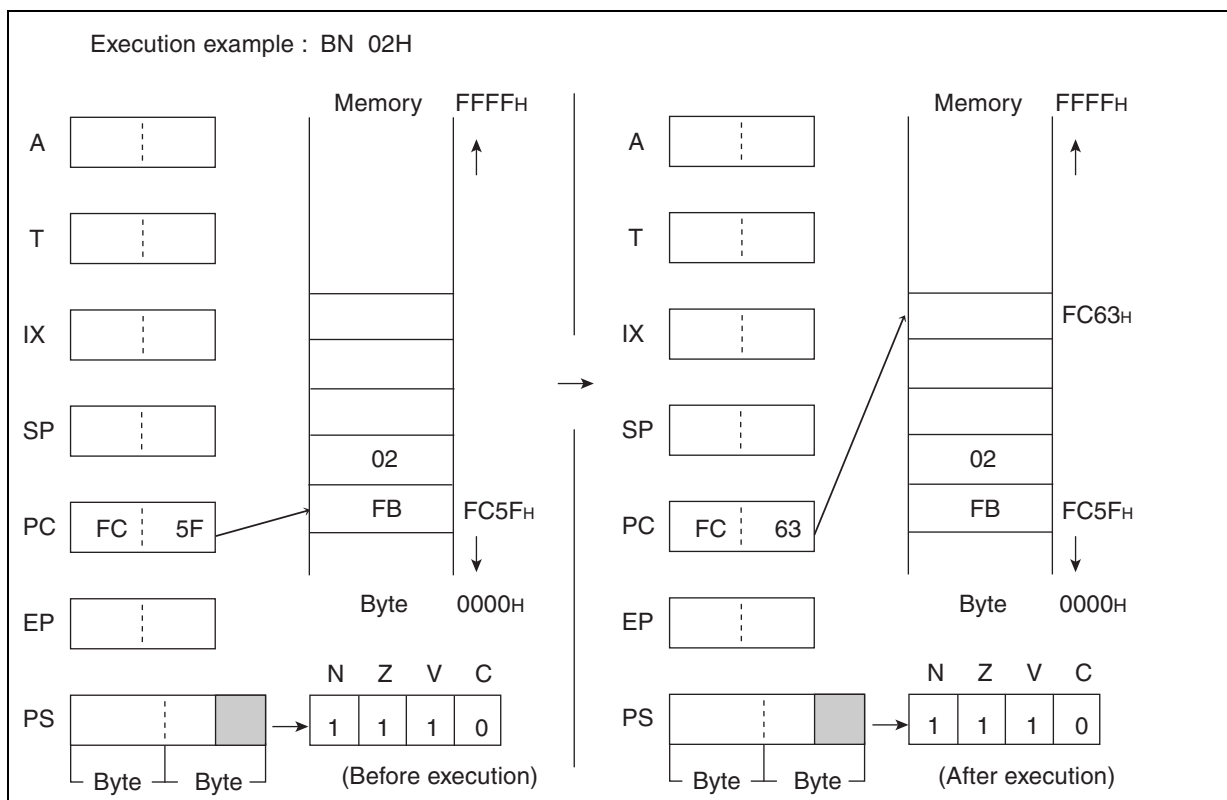
C: Not changed

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:  $(PC) \leftarrow (PC) + 2$  (Word addition)

(Z) = 0:  $(PC) \leftarrow (PC) + 2 + \text{rel}$  (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

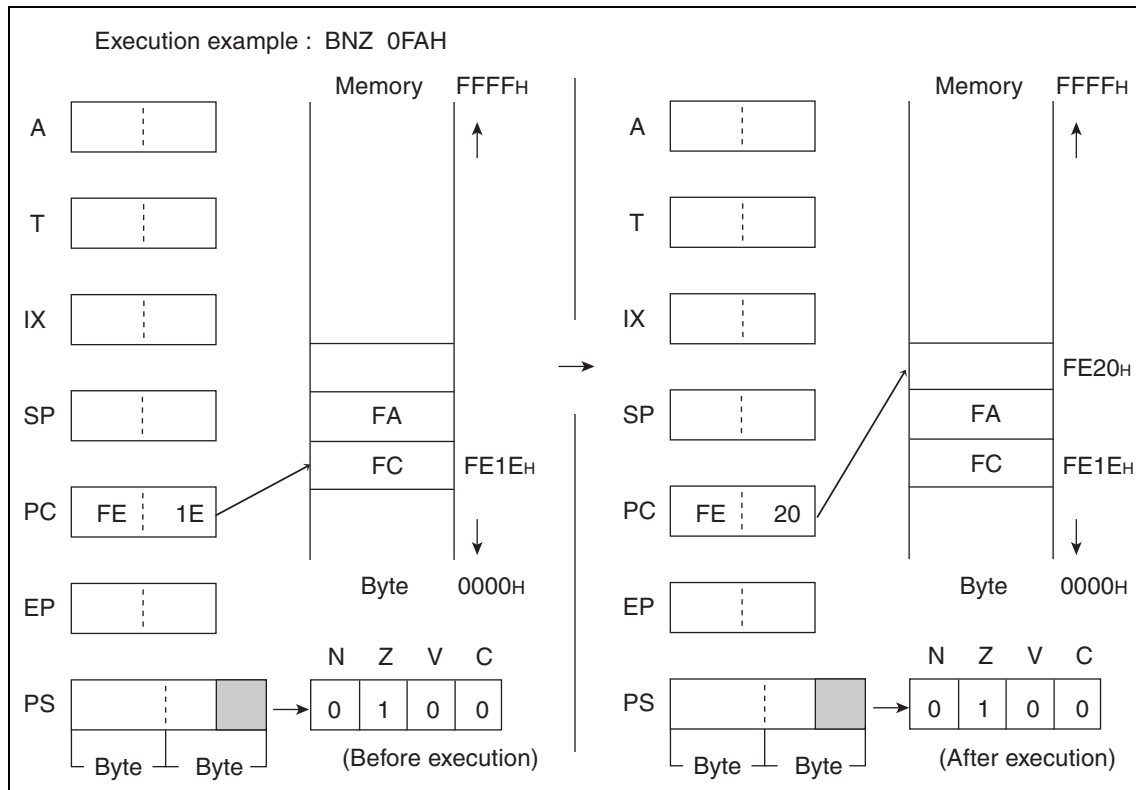
V: Not changed

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)  $\leftarrow$  (PC) + 2 (Word addition)

(C) = 0: (PC)  $\leftarrow$  (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

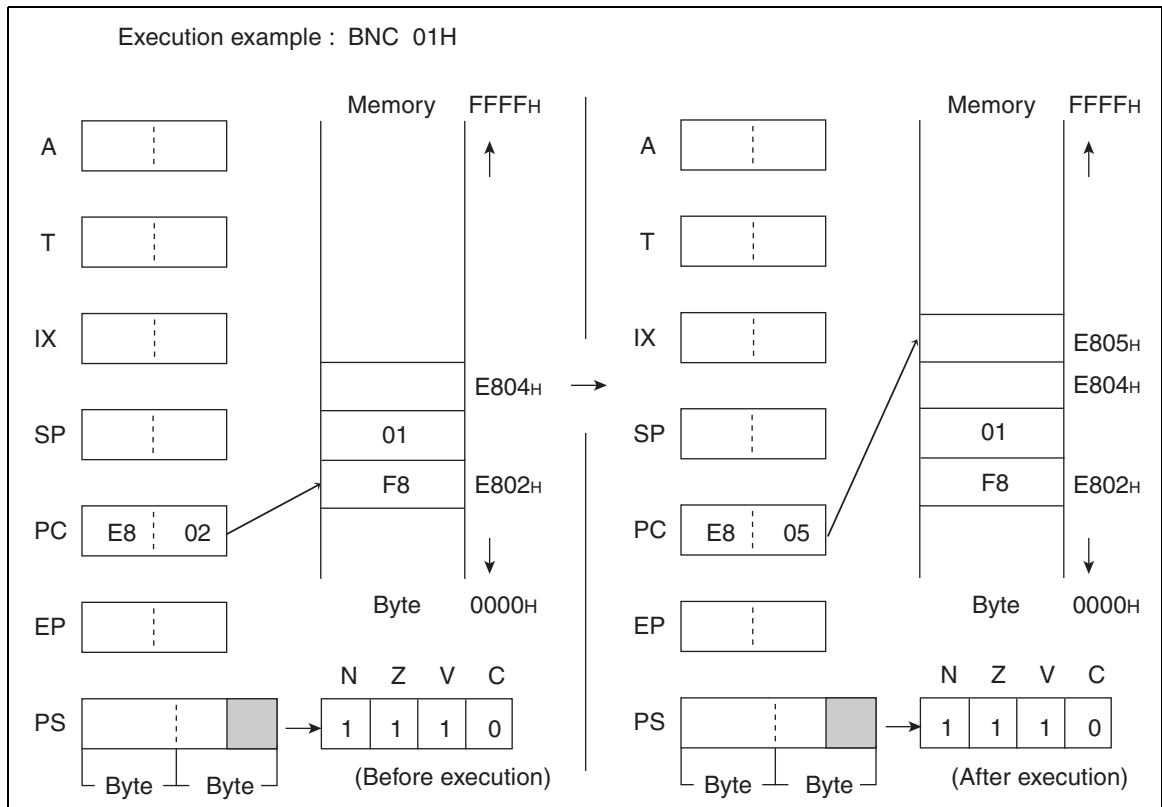
V: Not changed

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:  $(PC) \leftarrow (PC) + 2$  (Word addition)

(N) = 1:  $(PC) \leftarrow (PC) + 2 + \text{rel}$  (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

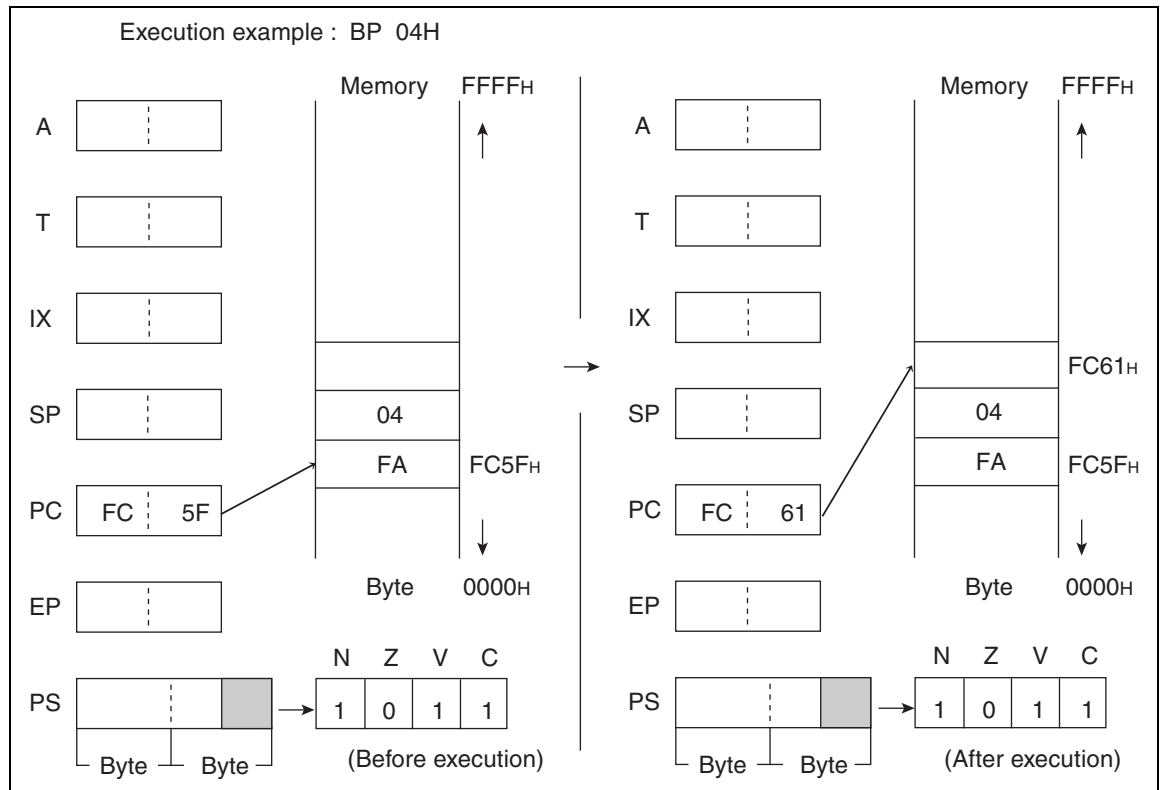
V: Not changed

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:  $(PC) \leftarrow (PC) + 2$  (Word addition)

(Z) = 1:  $(PC) \leftarrow (PC) + 2 + \text{rel}$  (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

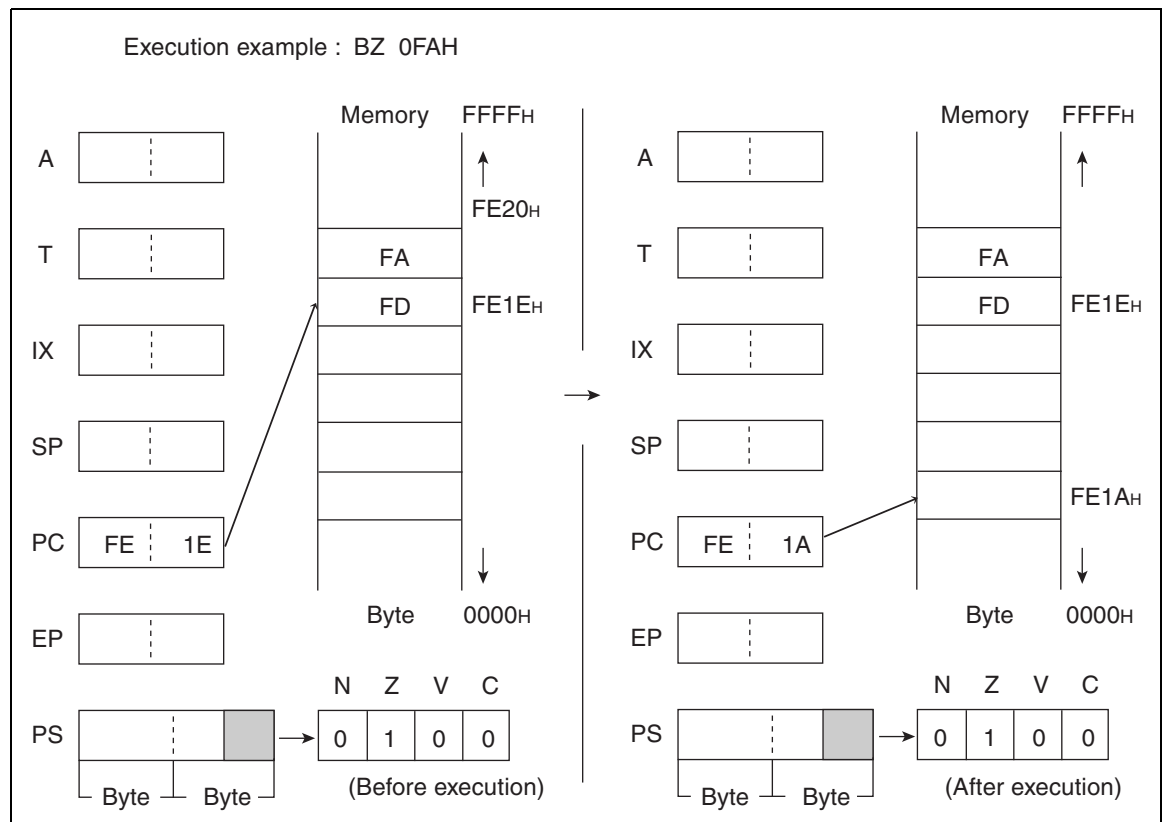
C: Not changed

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

$(SP) \leftarrow (SP) - 2$  (Word subtraction),  $((SP)) \leftarrow (PC)$  (Word transfer)

$(PC) \leftarrow \text{ext}$

Assembler format

CALL ext

Condition code (CCR)

| N | Z | V | C |
|---|---|---|---|
| - | - | - | - |

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

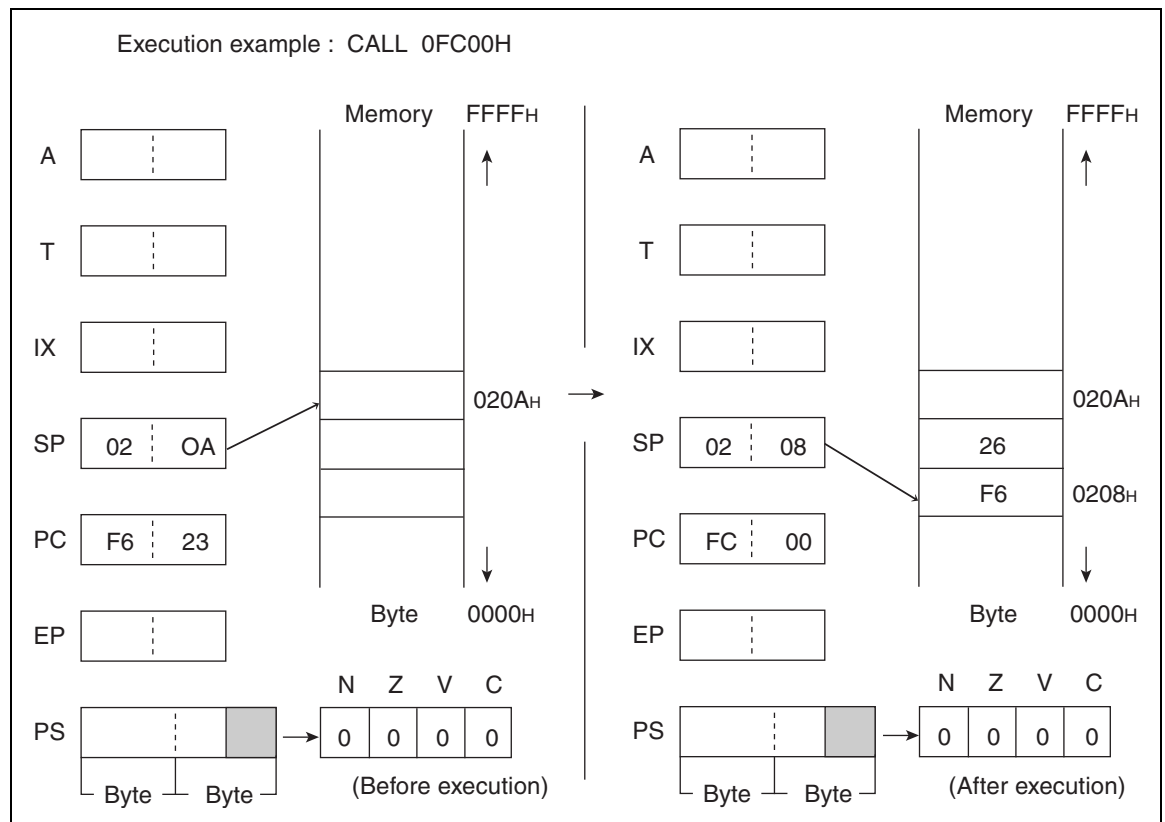
V: Not changed

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

$(SP) \leftarrow (SP) - 2$  (Word subtraction),  $((SP)) \leftarrow (PC)$  (Word transfer)

$(PC) \leftarrow (VA)$

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

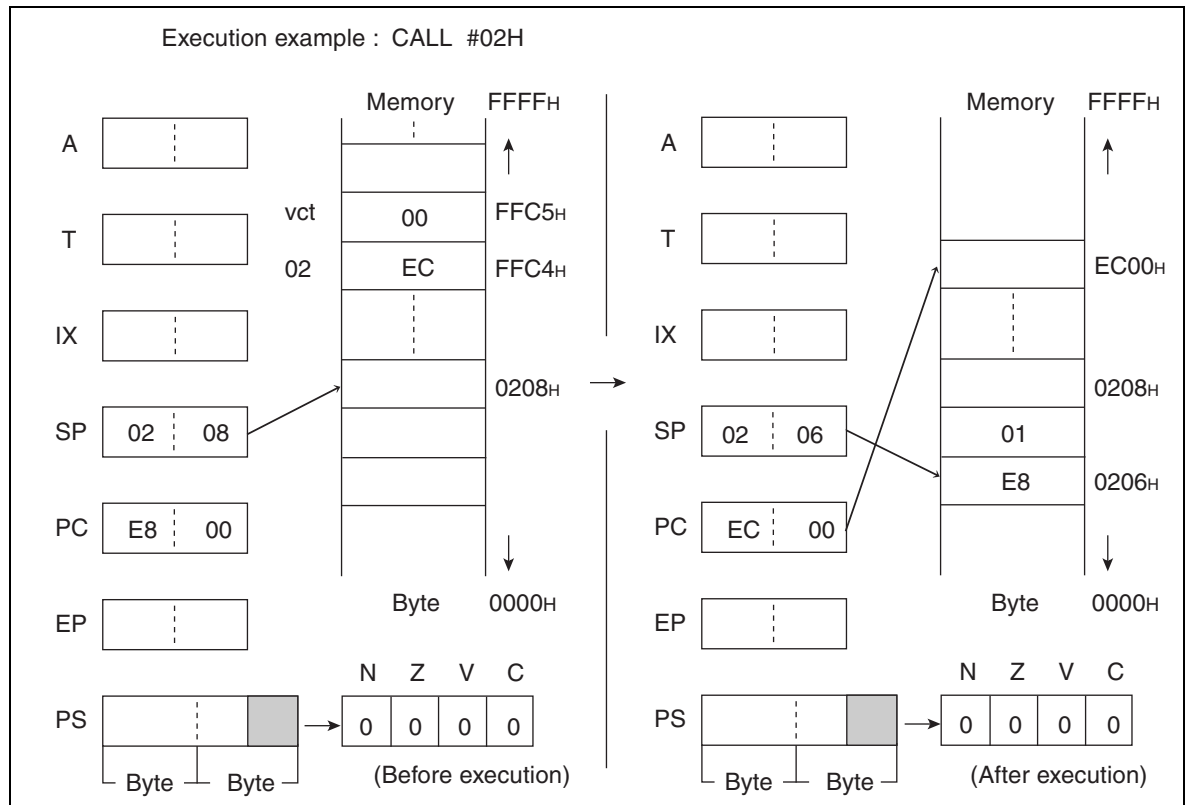


Table 6-9. Call Storage Address of Vector Call Instruction

| Vector address (VA) |                   | Instruction |
|---------------------|-------------------|-------------|
| Lower address       | Upper address     |             |
| FFCE <sub>H</sub>   | FFCF <sub>H</sub> | CALL#7      |
| FFCC <sub>H</sub>   | FFCD <sub>H</sub> | CALL#6      |
| FFCA <sub>H</sub>   | FFCB <sub>H</sub> | CALL#5      |
| FFC8 <sub>H</sub>   | FFC9 <sub>H</sub> | CALL#4      |
| FFC6 <sub>H</sub>   | FFC7 <sub>H</sub> | CALL#3      |
| FFC4 <sub>H</sub>   | FFC5 <sub>H</sub> | CALL#2      |
| FFC2 <sub>H</sub>   | FFC3 <sub>H</sub> | CALL#1      |
| FFC0 <sub>H</sub>   | FFC1 <sub>H</sub> | CALL#0      |

## 6.19 CLR B (Clear direct Memory Bit)

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

### CLR B (Clear direct Memory Bit)

Operation

(dir:b) ← 0

Assembler format

CLR B dir:b

Condition code (CCR)

| N | Z | V | C |
|---|---|---|---|
| - | - | - | - |

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

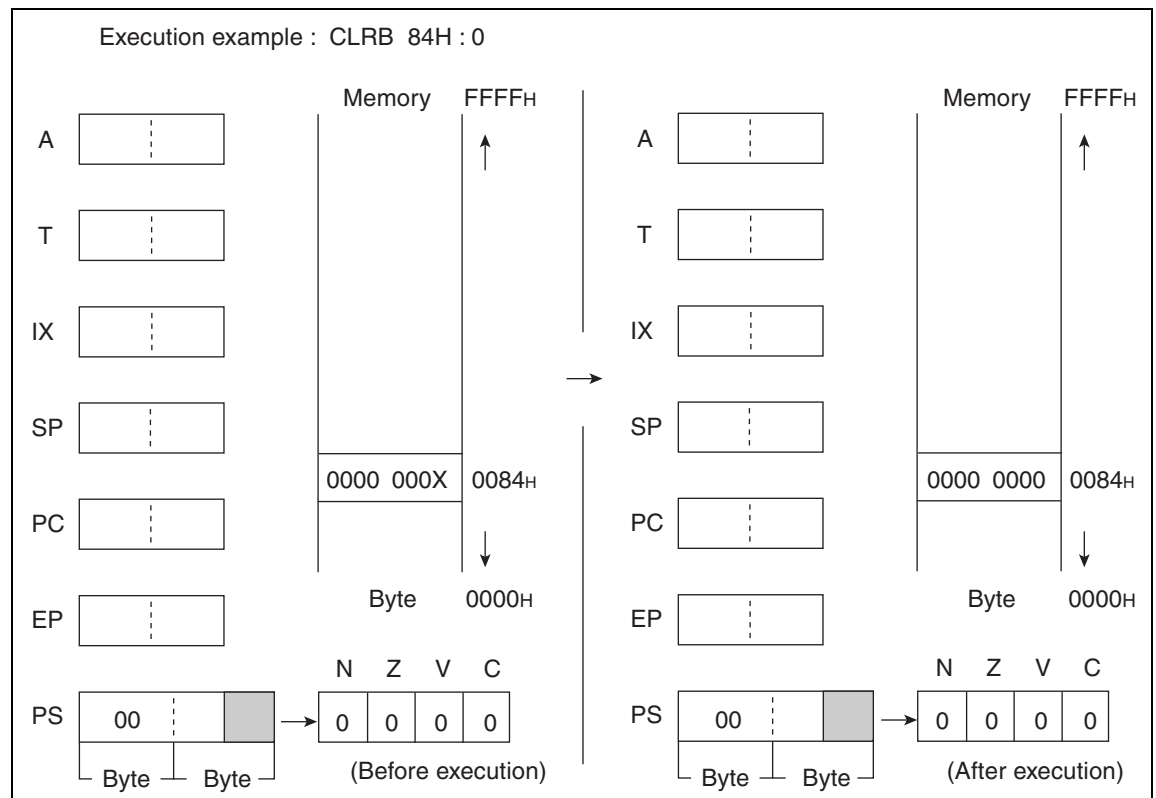
V: Not changed

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

$(C) \leftarrow 0$

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

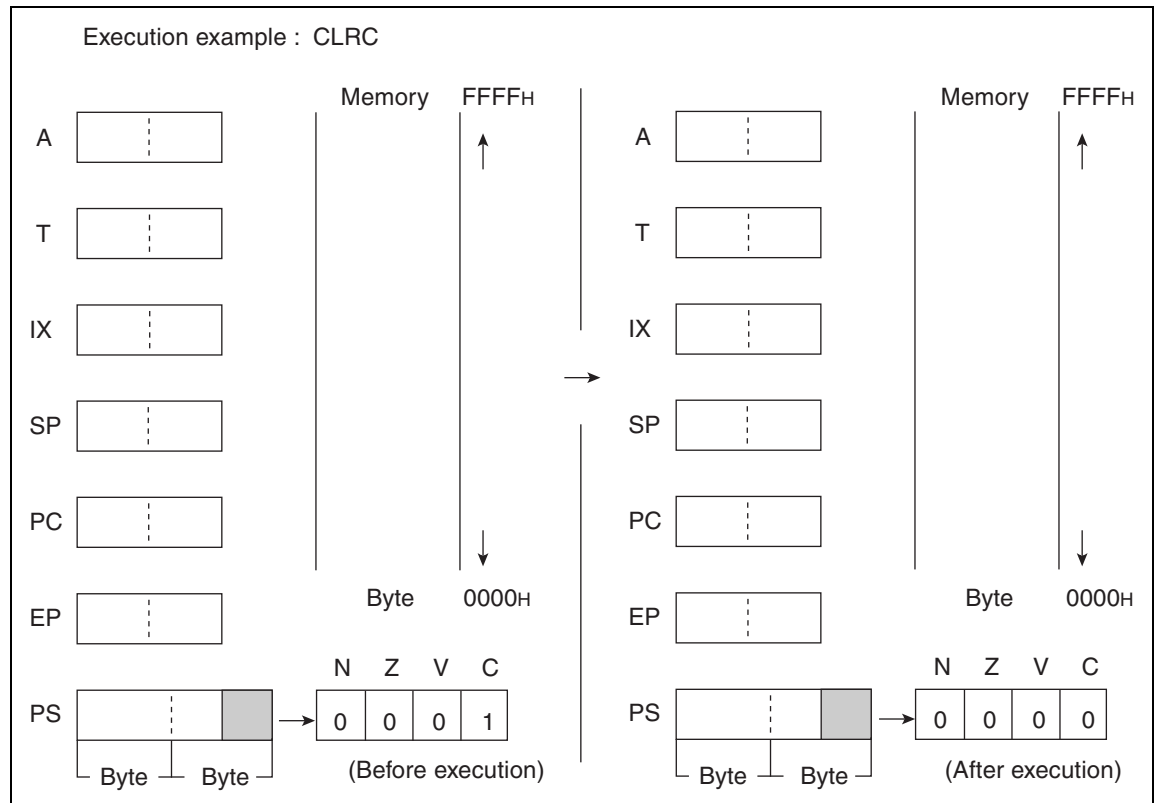
C: Set to 0.

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

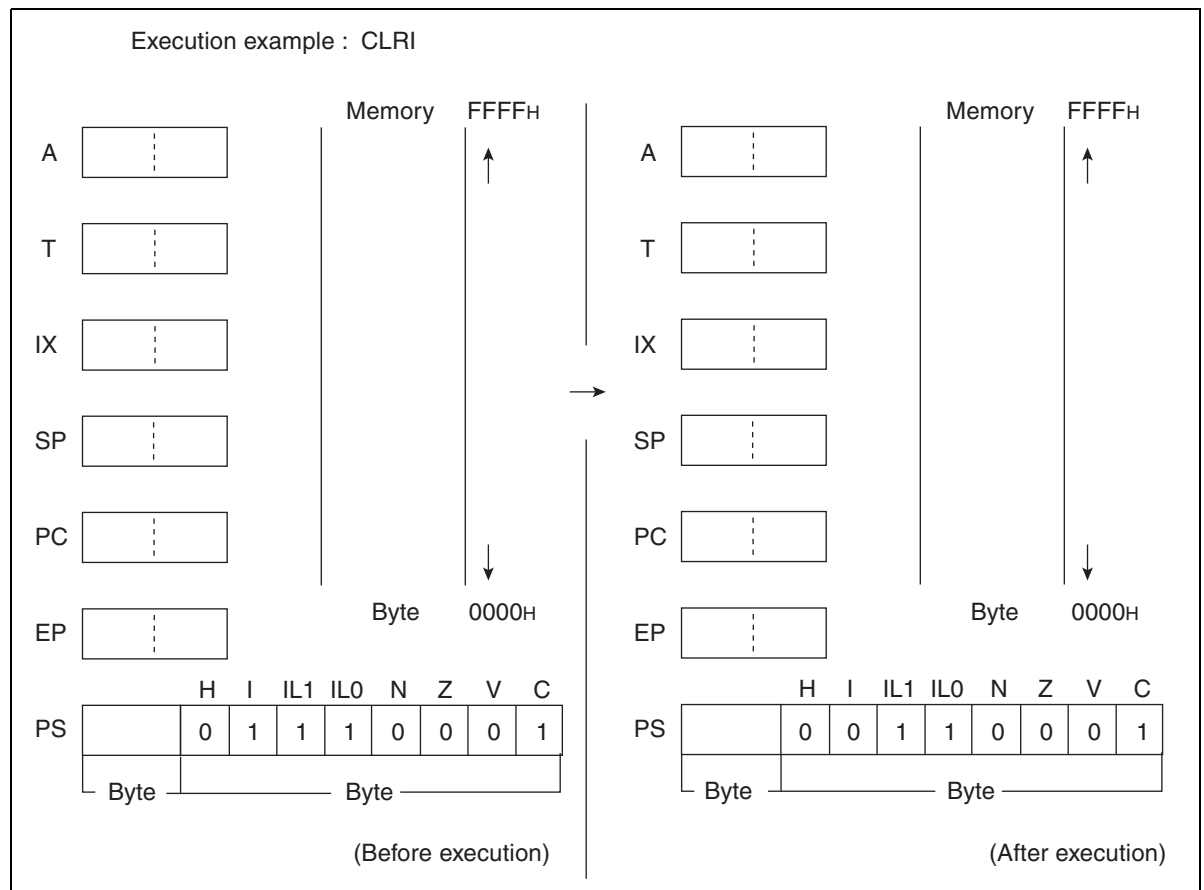
V: Not changed

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<sub>H</sub> and set to 0 in other cases.

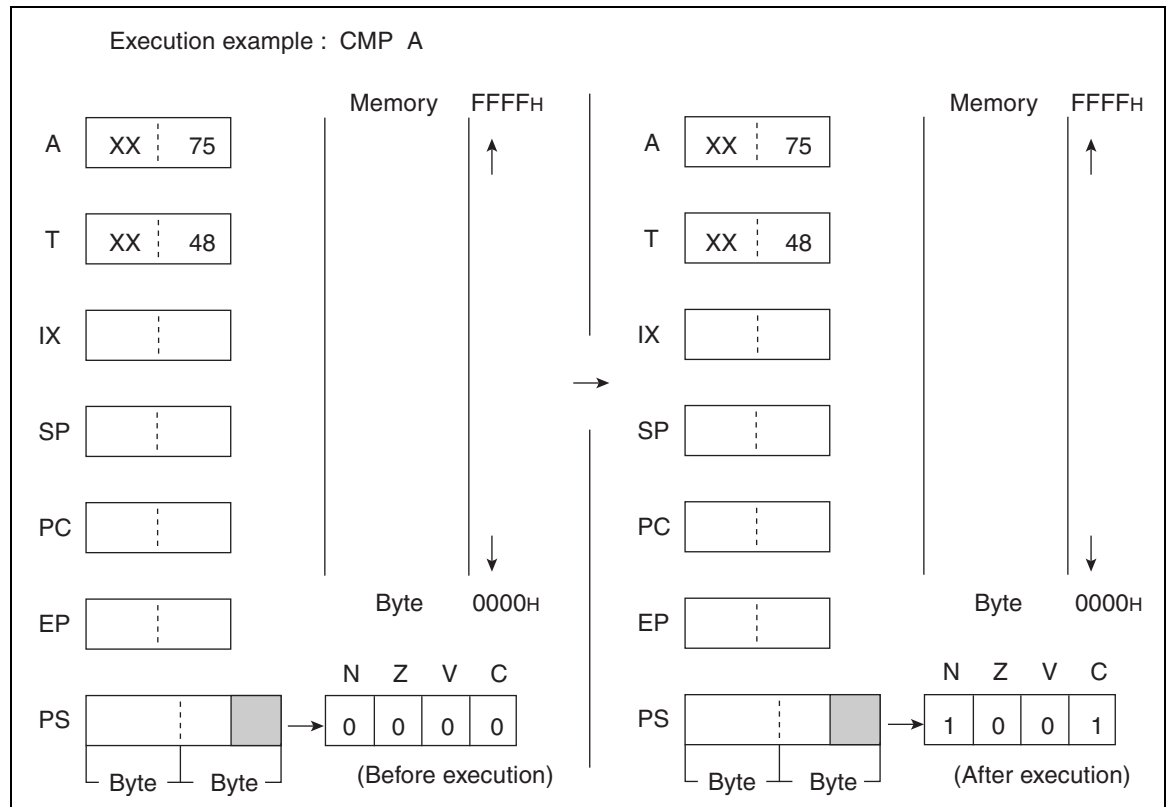
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

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.

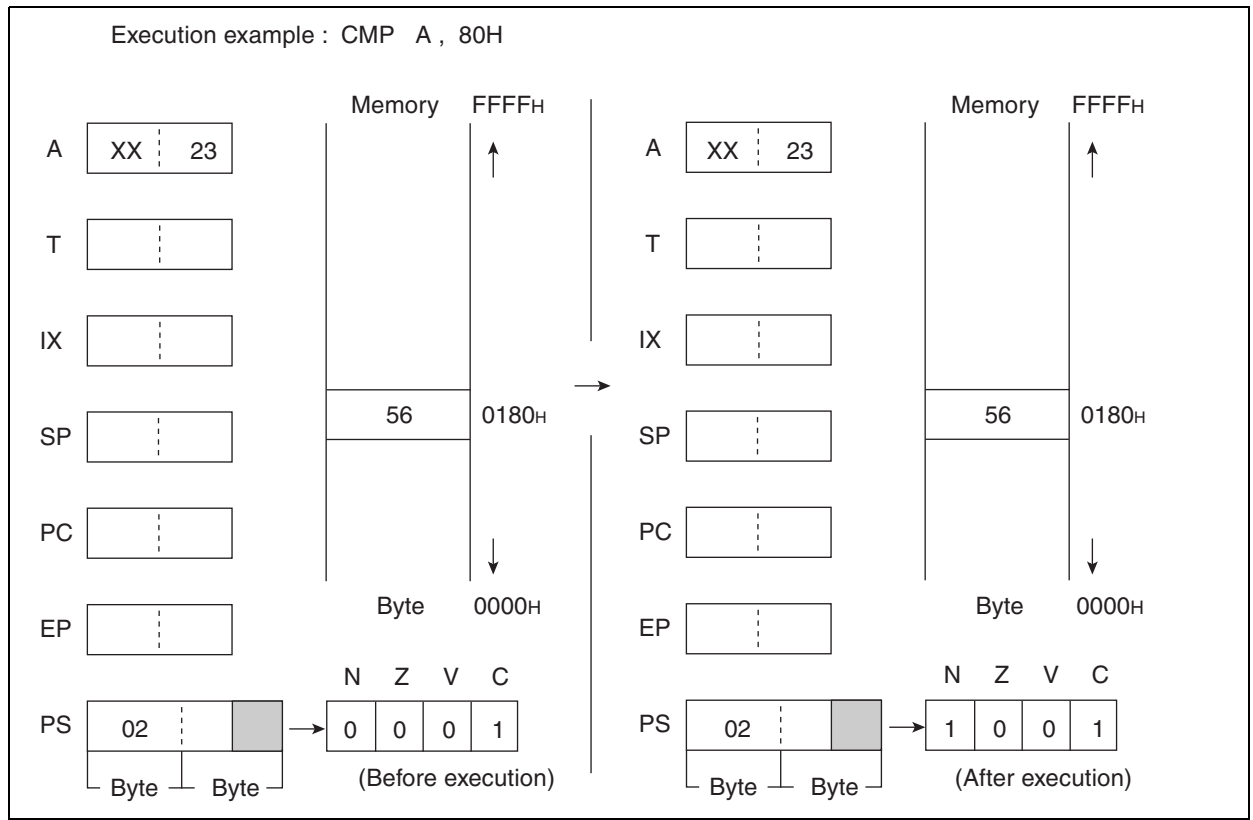
Z: Set to 1 if the result of operation is 00<sub>H</sub> and set to 0 in other cases.

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 6-10. 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<sub>H</sub> and set to 0 in other cases.

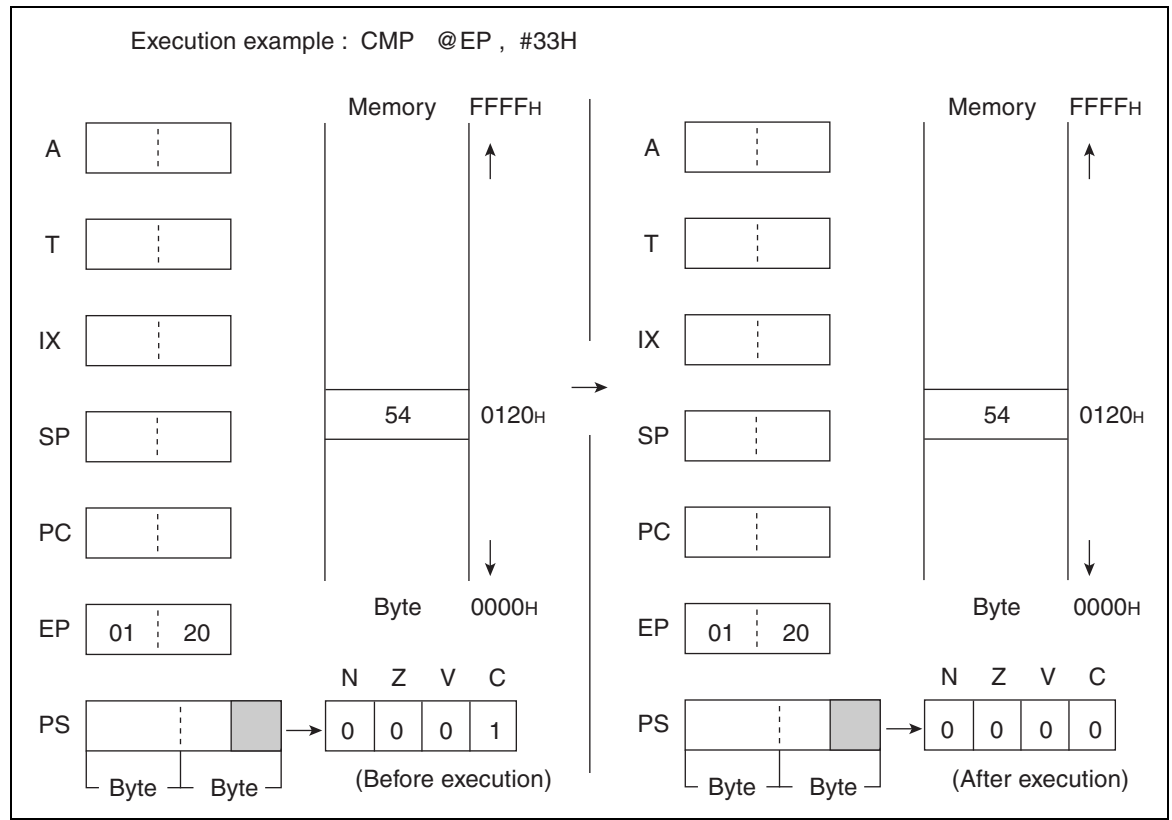
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

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

Table 6-11. 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<sub>H</sub> and set to 0 in other cases.

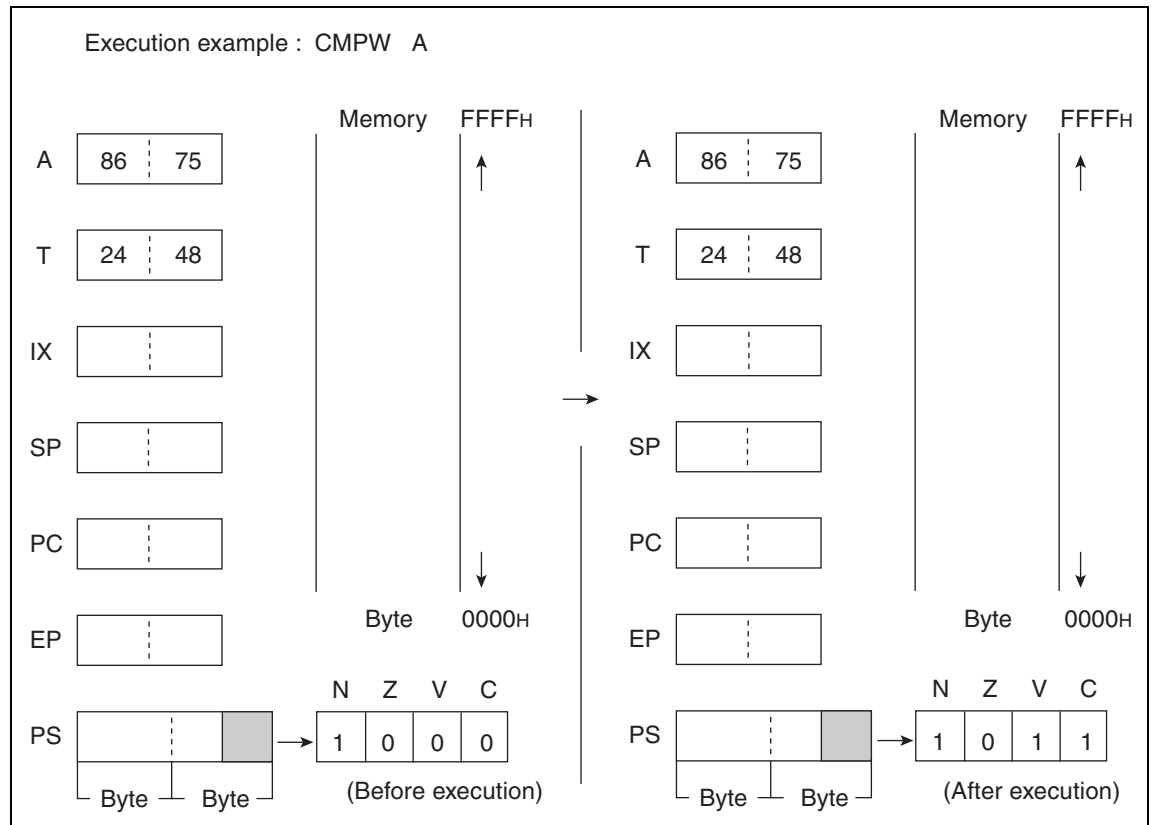
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

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

$(AL) \leftarrow (AL) + 6 \text{ or } 60H \text{ or } 66H$

(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<sub>H</sub> 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

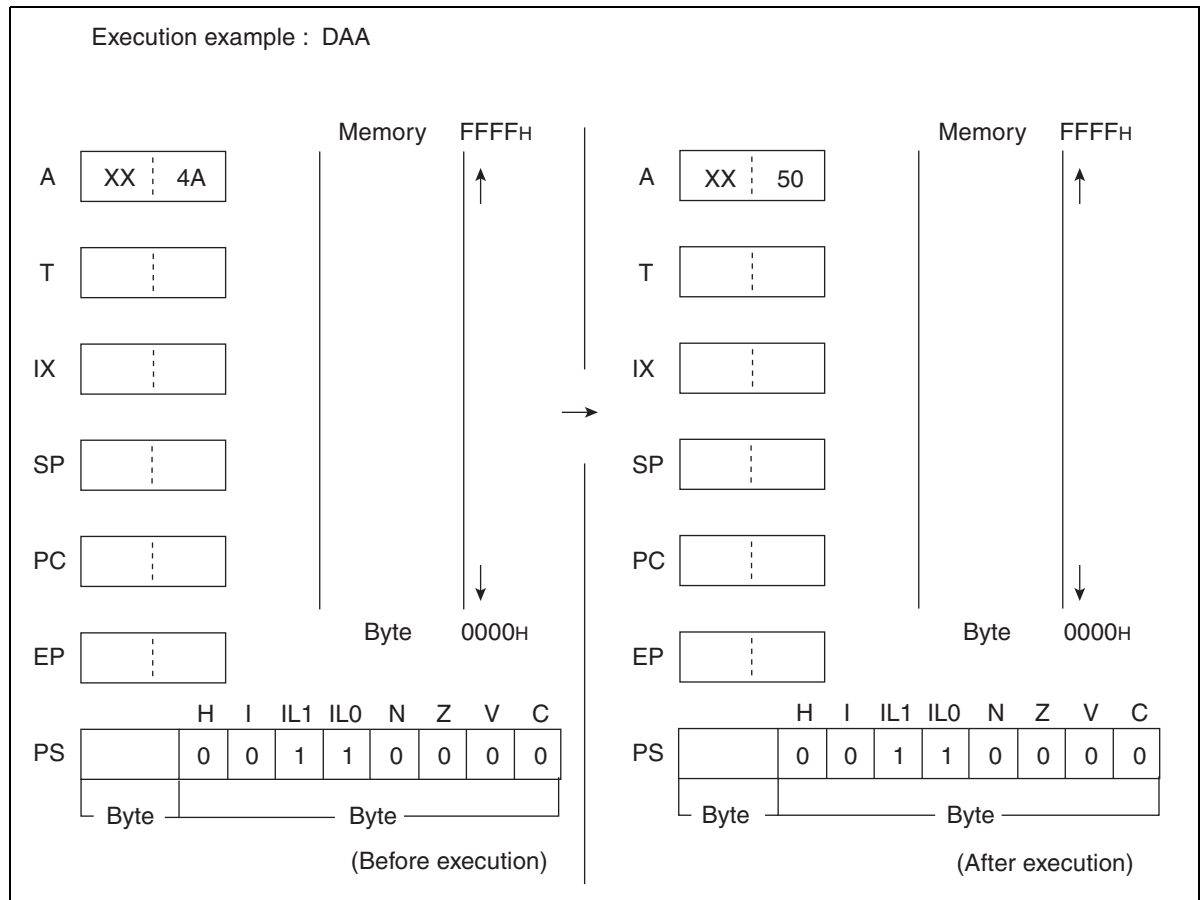


Table 6-12. Decimal Adjustment Table (DAA)

| C-flag | AL<br>(bit7 to bit4) | H-flag | AL<br>(bit3 to bit0) | Correction<br>value | C-flag after<br>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 6-13. Execution Example

| Mnemonic     | AL | C | H |
|--------------|----|---|---|
| MOV A, #75H  | 75 | 0 | × |
| ADDC A, #25H | 9A | 0 | 0 |

Table 6-13. Execution Example (*continued*)

| <b>Mnemonic</b> | <b>AL</b> | <b>C</b> | <b>H</b> |
|-----------------|-----------|----------|----------|
| 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

$(AL) \leftarrow (AL) - 6 \text{ or } 60H \text{ or } 66H$

(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<sub>H</sub> 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

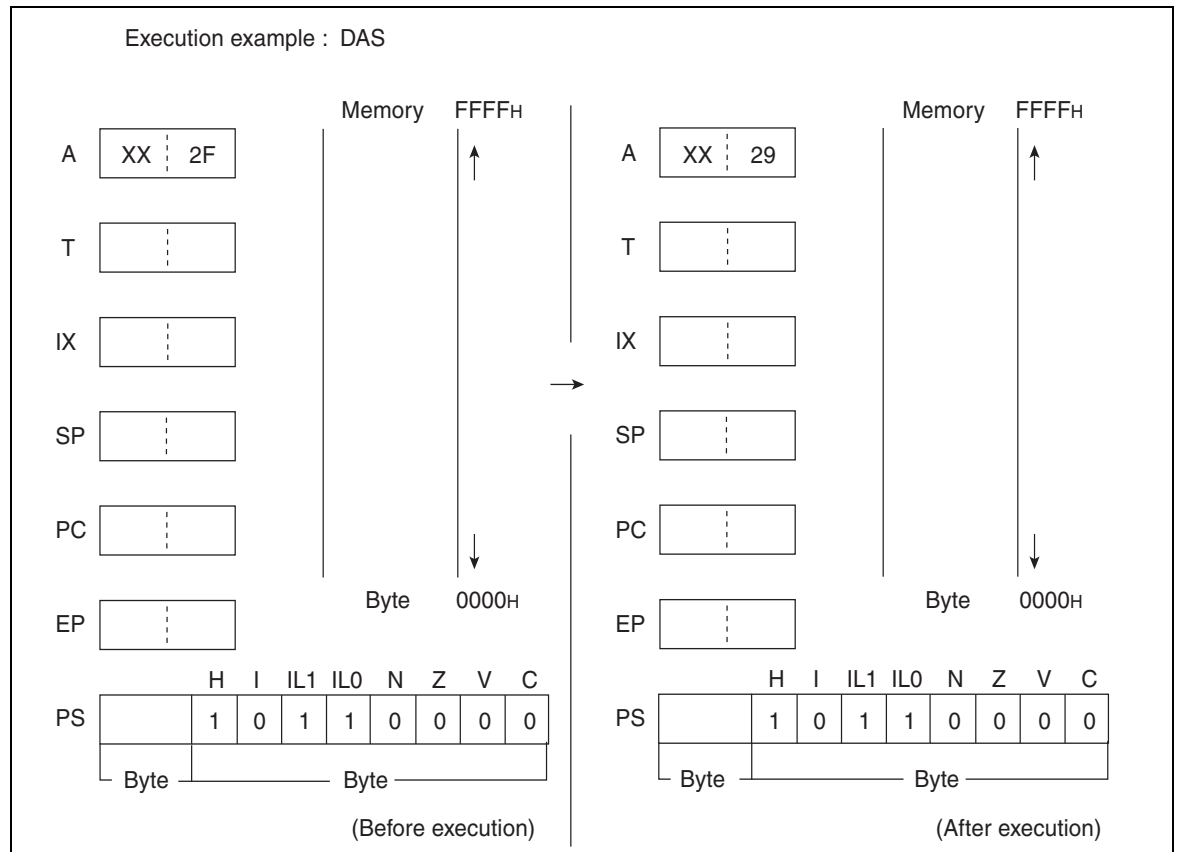


Table 6-14. Decimal Adjustment Table (DAS)

| 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 6-15. 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

$(Ri) \leftarrow (Ri) - 1$  (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<sub>H</sub> and set to 0 in other cases.

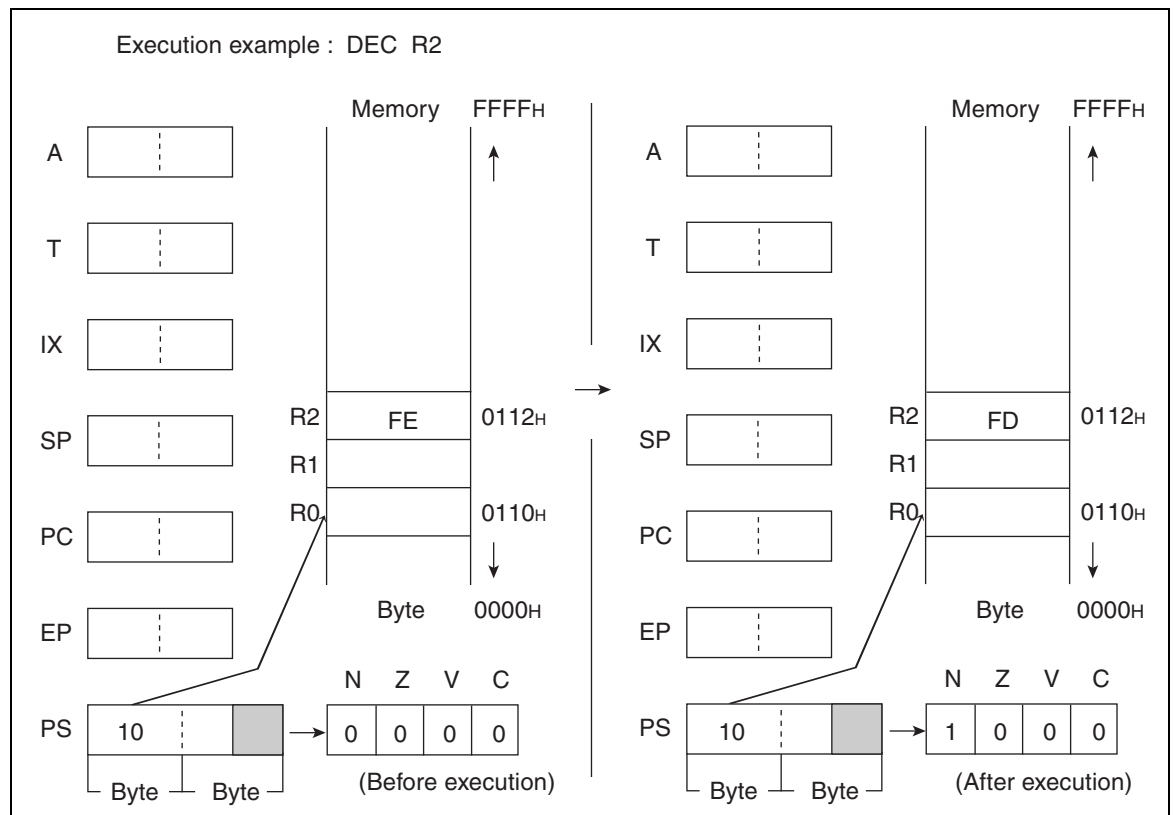
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

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

$(A) \leftarrow (A) - 1$  (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<sub>H</sub> and set to 0 in other cases.

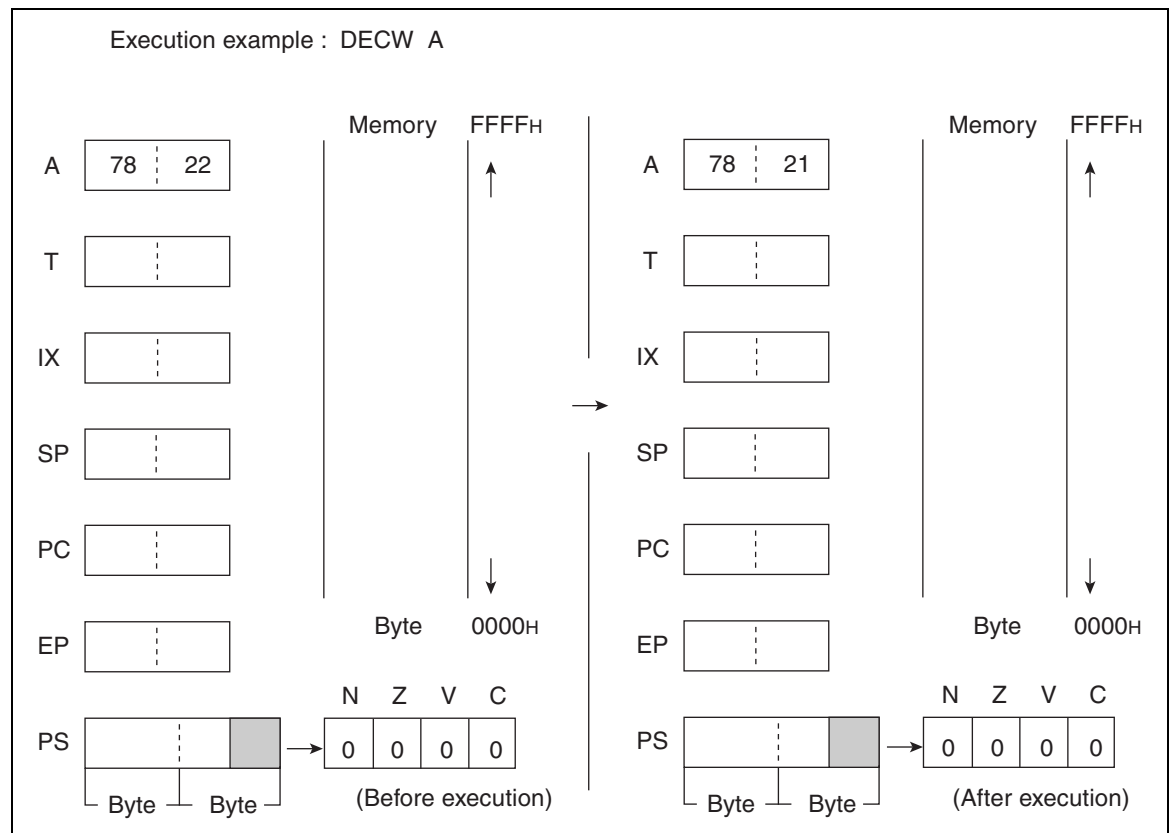
V: Not changed

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

$(EP) \leftarrow (EP) - 1$  (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

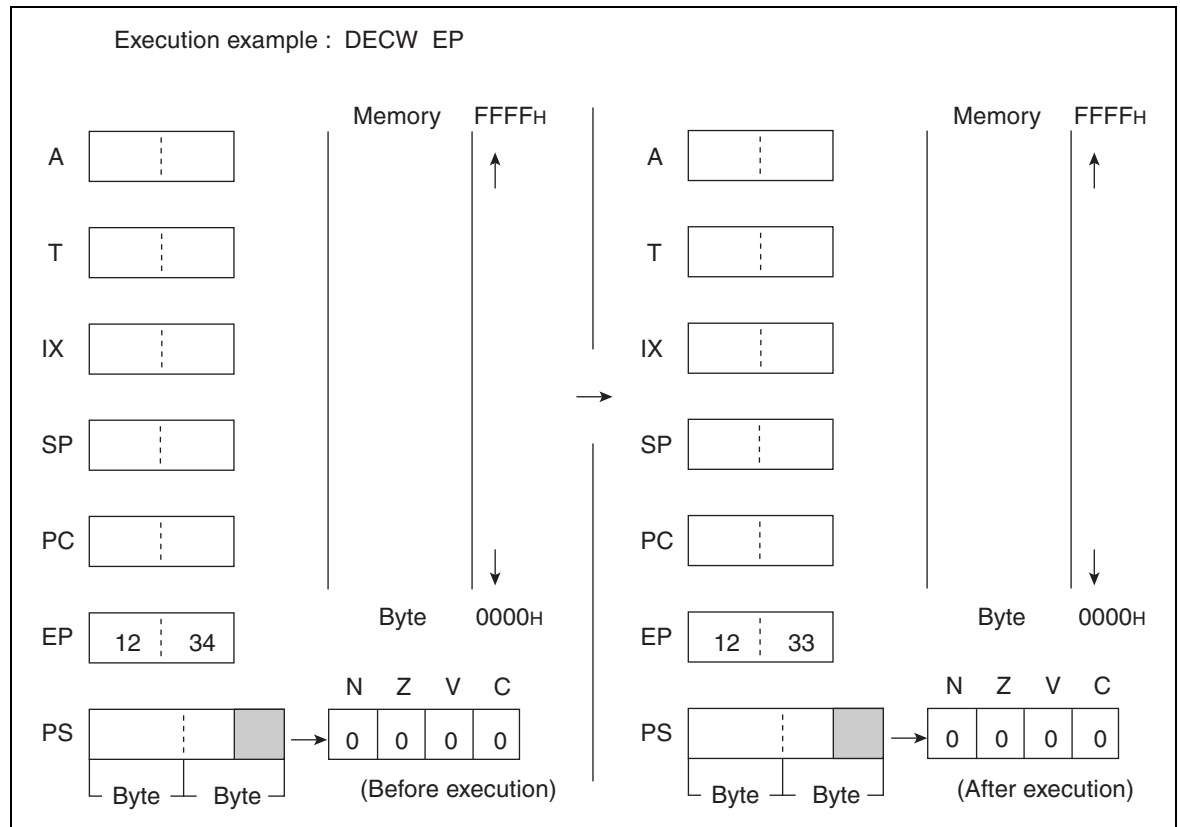
V: Not changed

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

$(IX) \leftarrow (IX) - 1$  (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

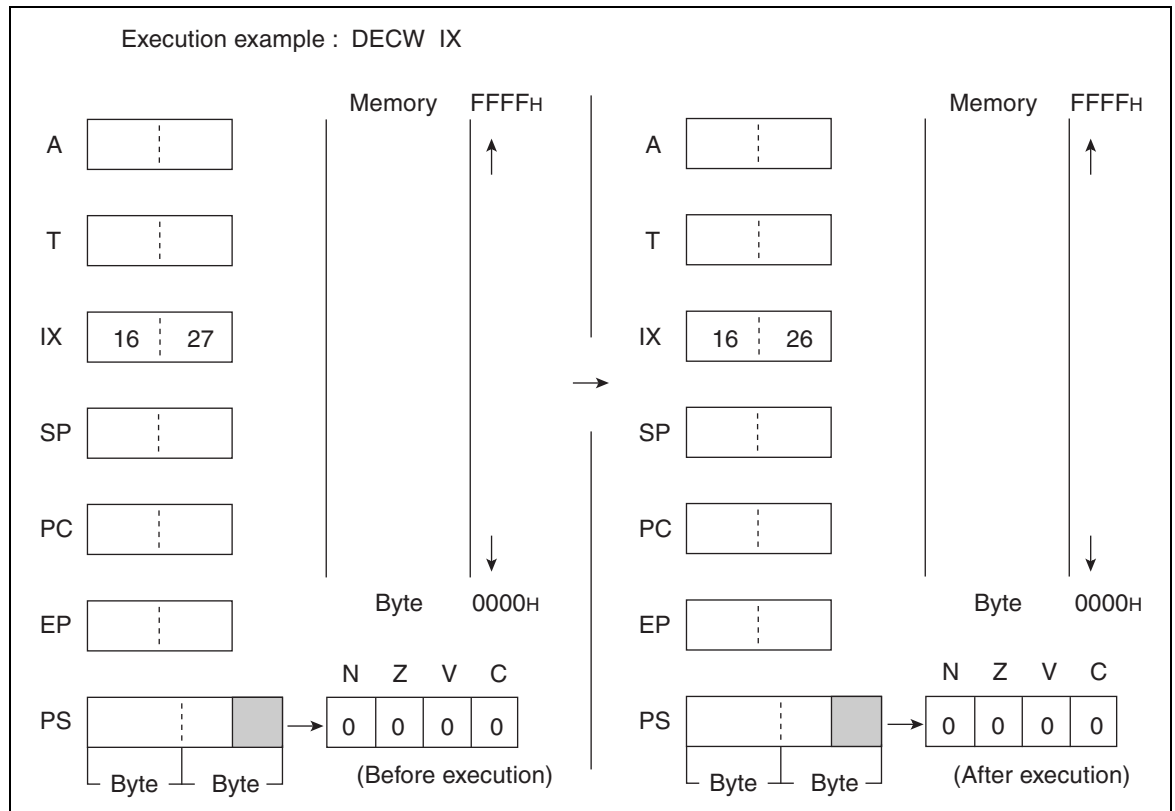
V: Not changed

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

$(SP) \leftarrow (SP) - 1$  (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

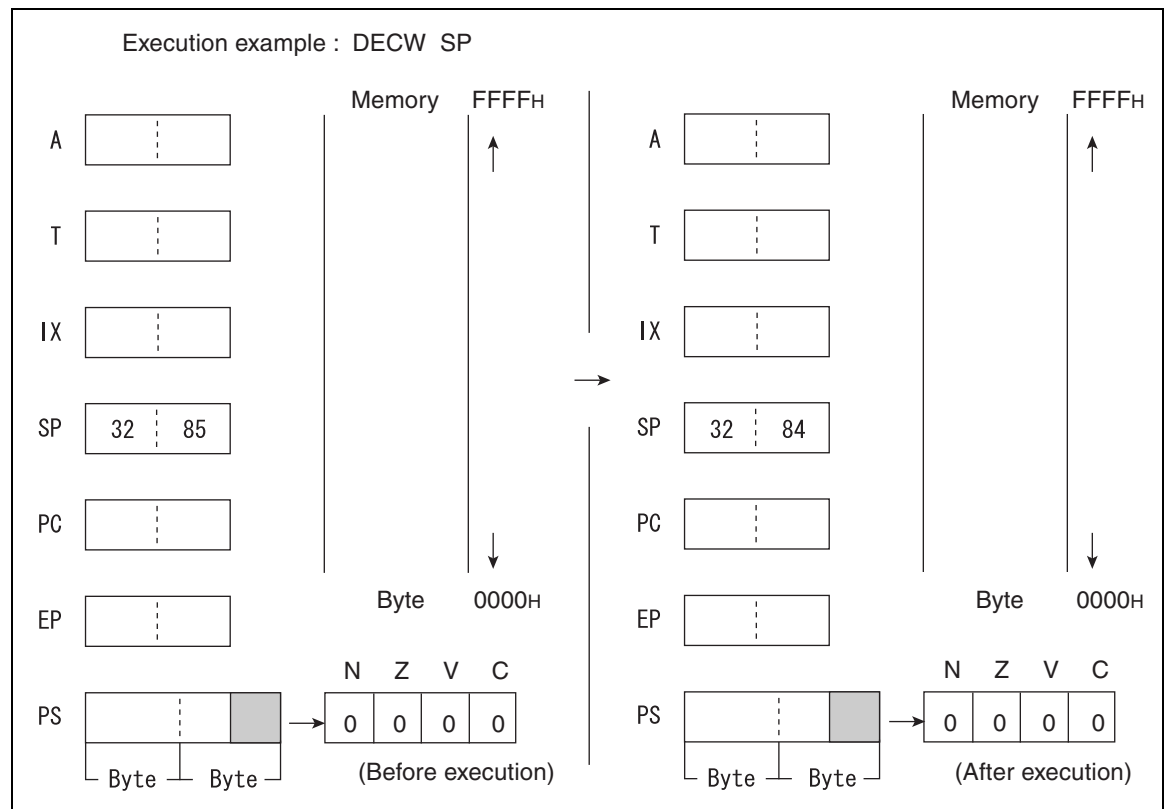
V: Not changed

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)  $\leftarrow$  (T) / (A)

Remainder (T)  $\leftarrow$  (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<sub>H</sub> and set to 0 in other cases.

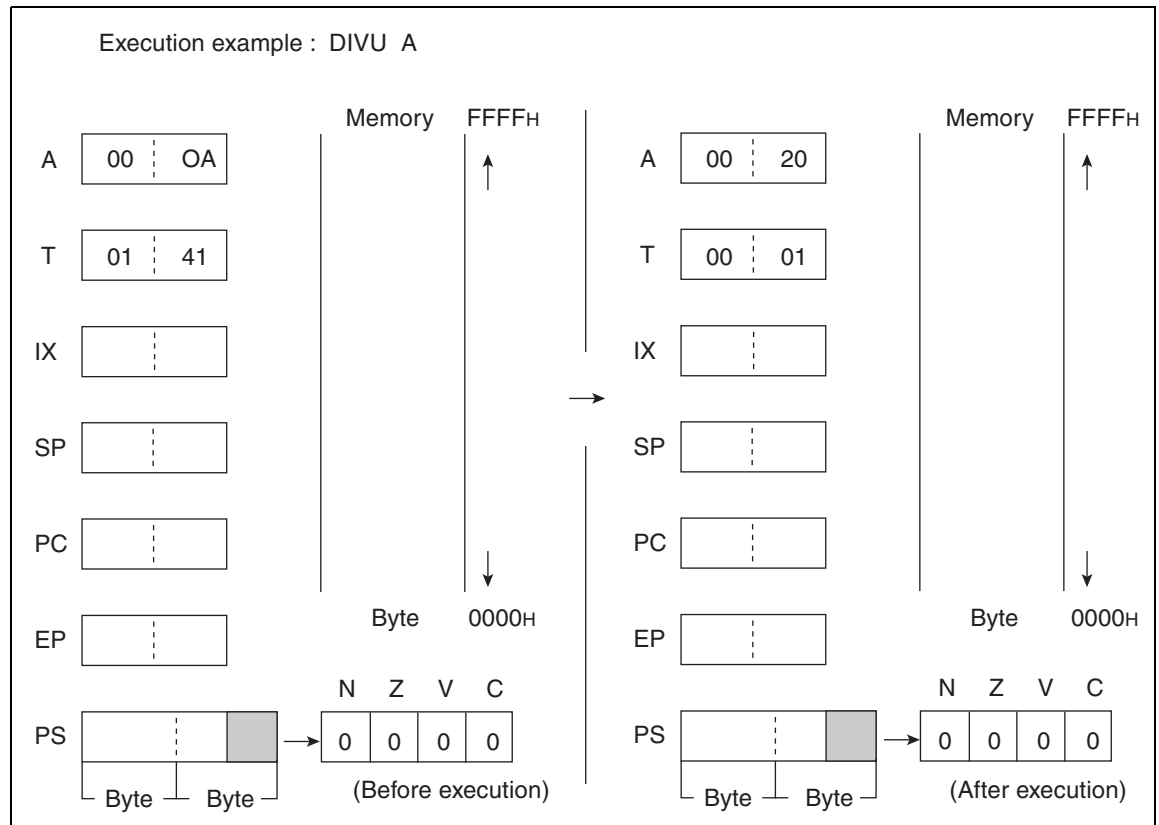
V: Not changed

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

$(Ri) \leftarrow (Ri) + 1$  (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<sub>H</sub> and set to 0 in other cases.

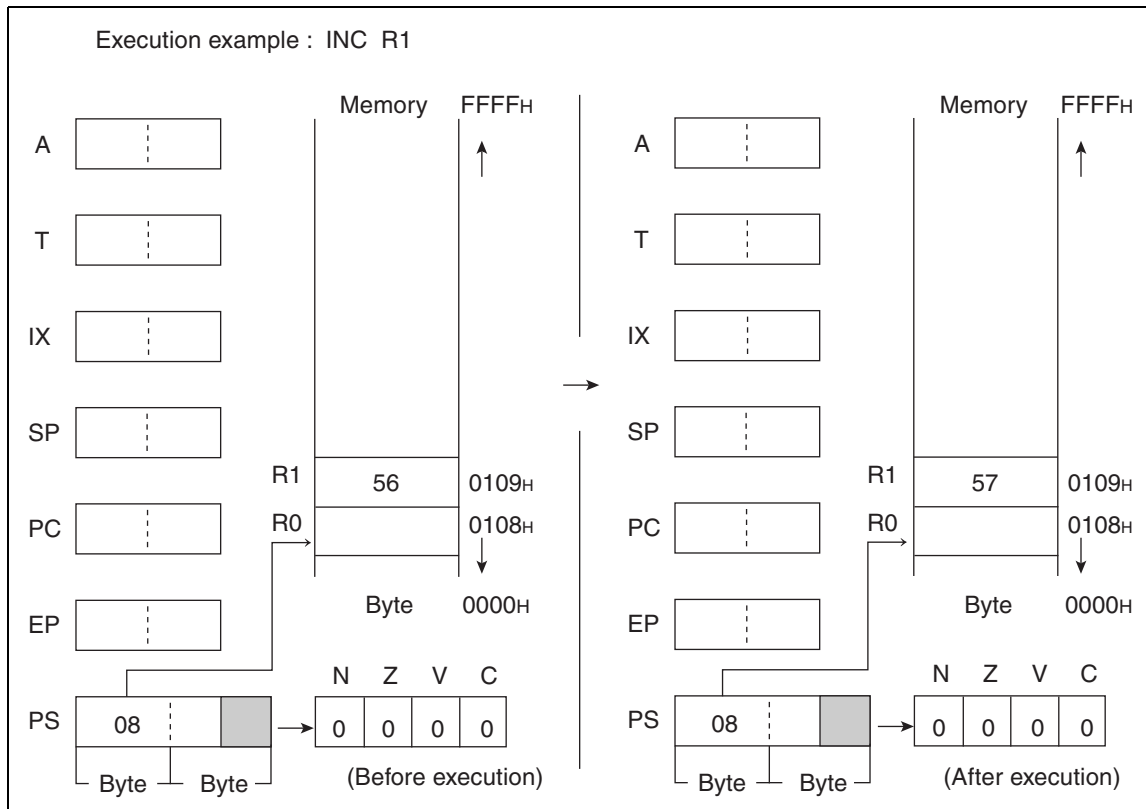
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

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

$(A) \leftarrow (A) + 1$  (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<sub>H</sub> and set to 0 in other cases.

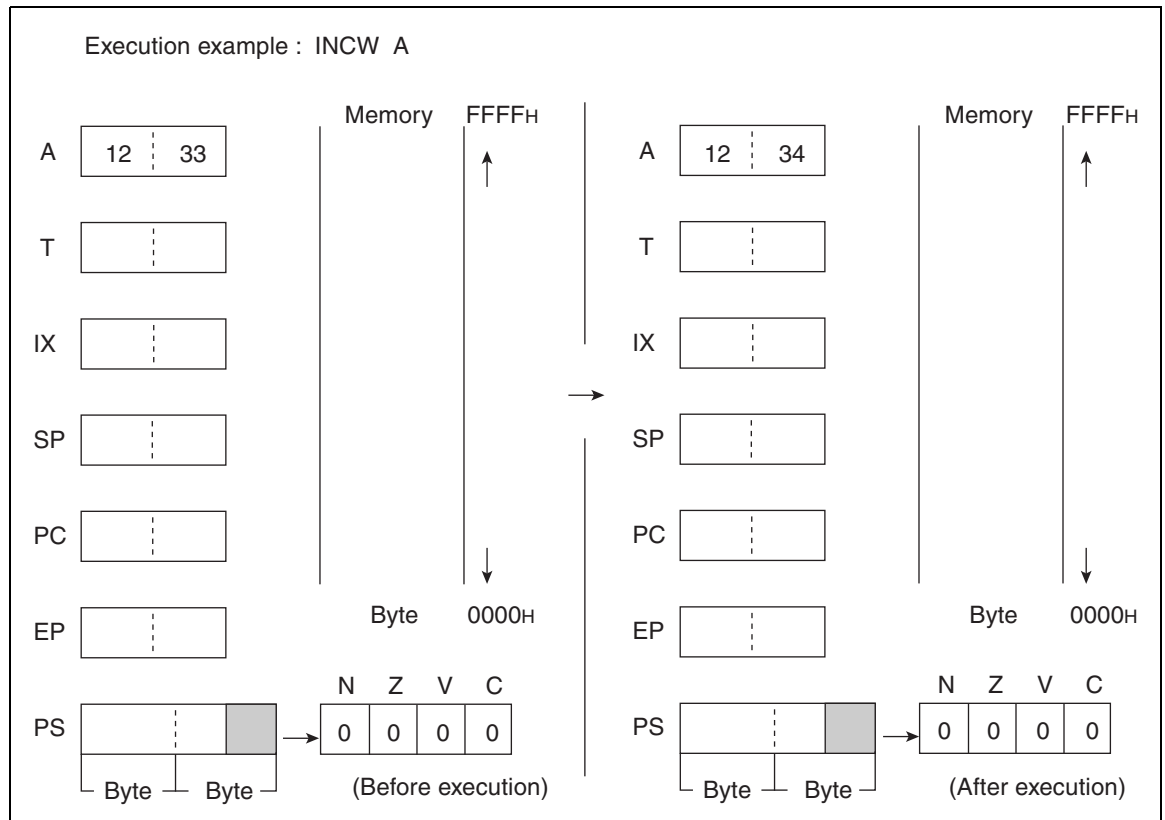
V: Not changed

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

$(EP) \leftarrow (EP) + 1$  (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

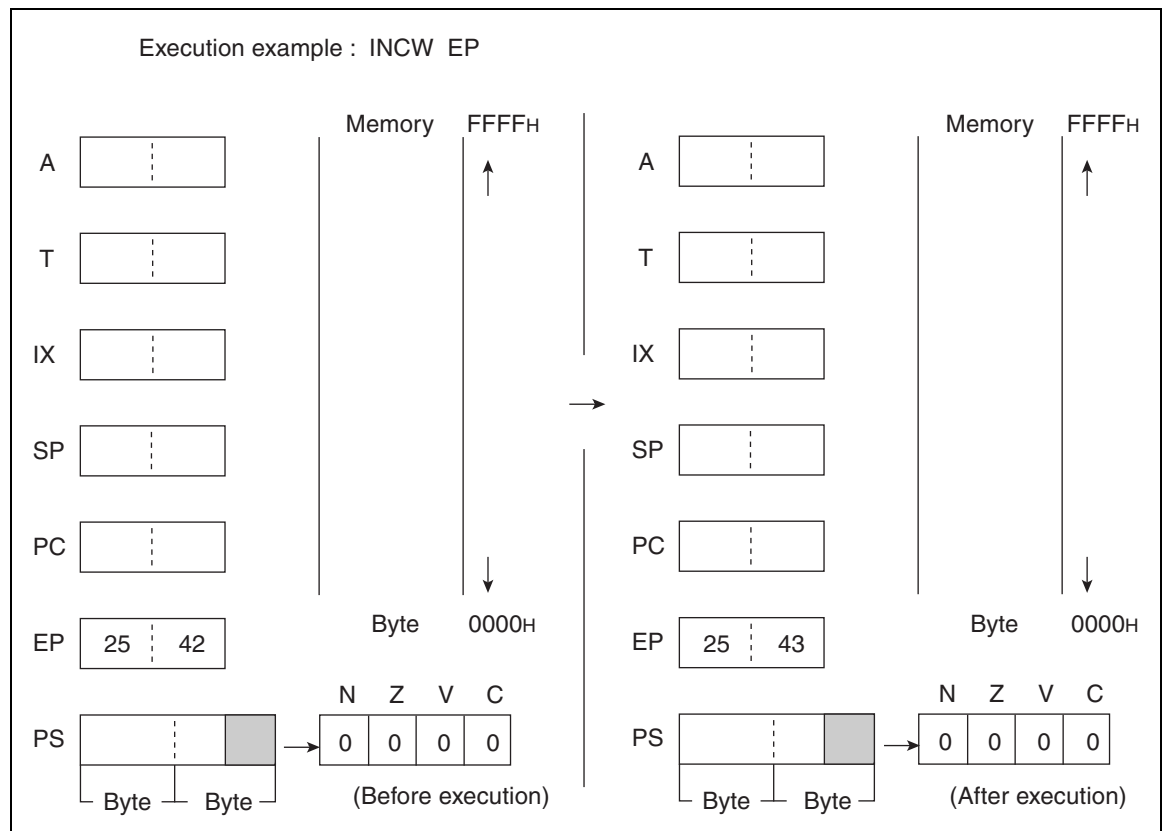
V: Not changed

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

$(IX) \leftarrow (IX) + 1$  (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

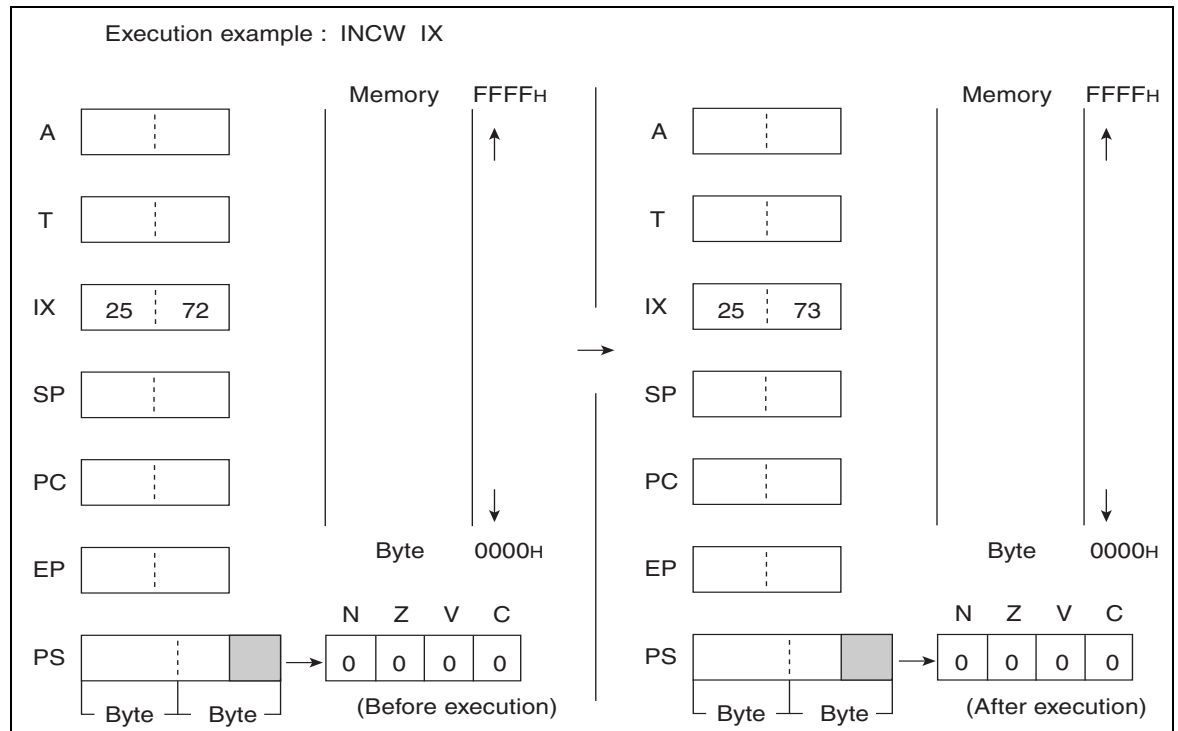
V: Not changed

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

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

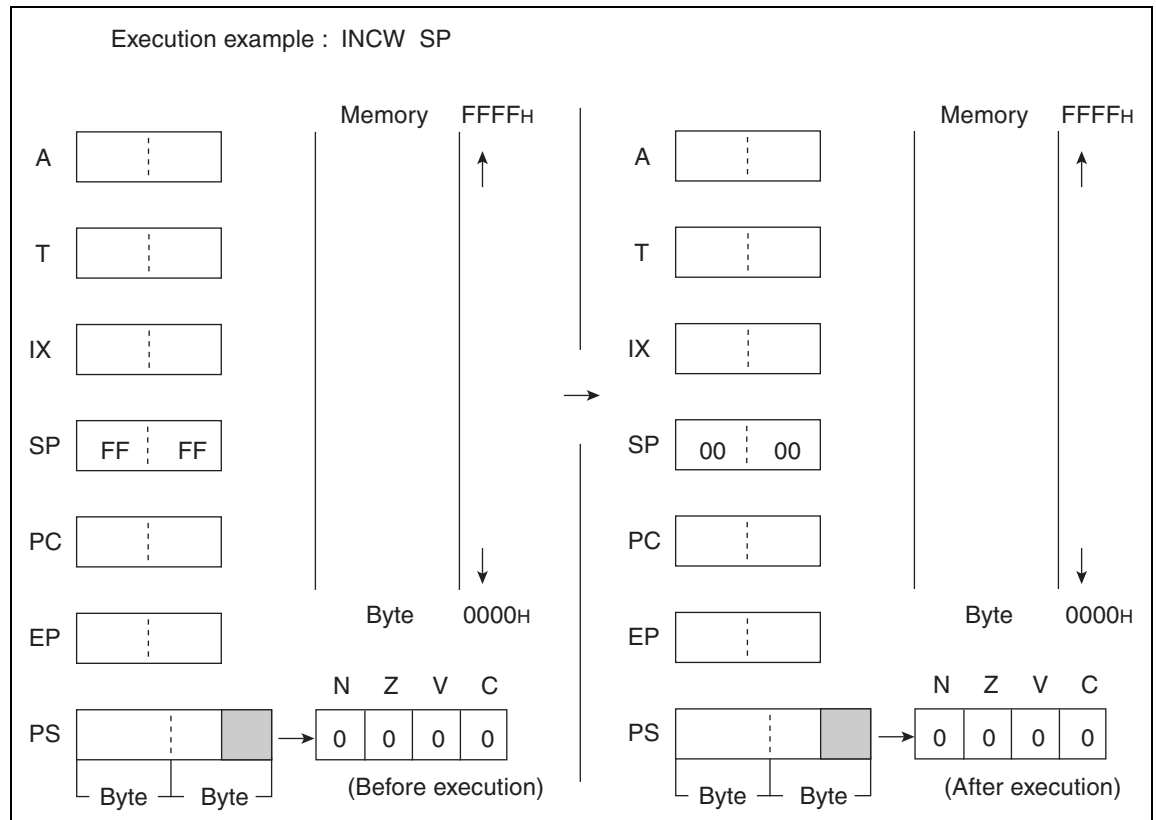
V: Not changed

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

$(PC) \leftarrow (A)$  (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

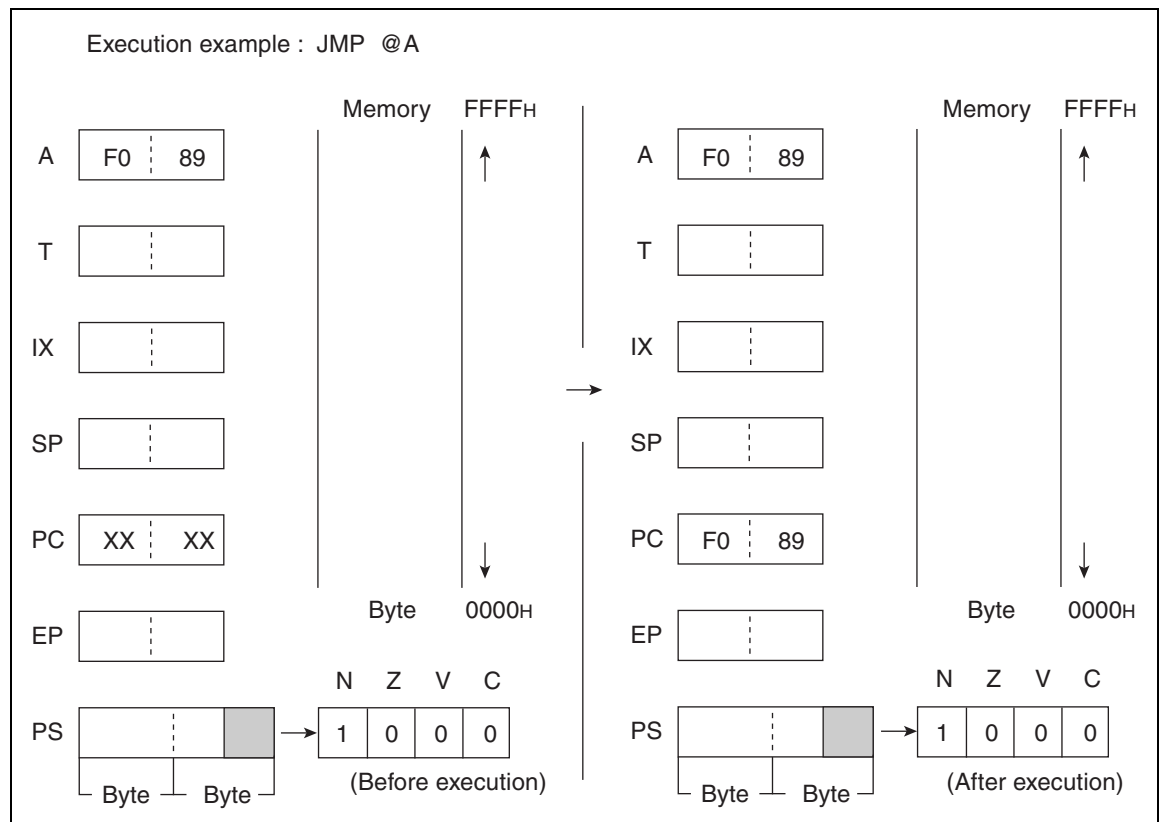
V: Not changed

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

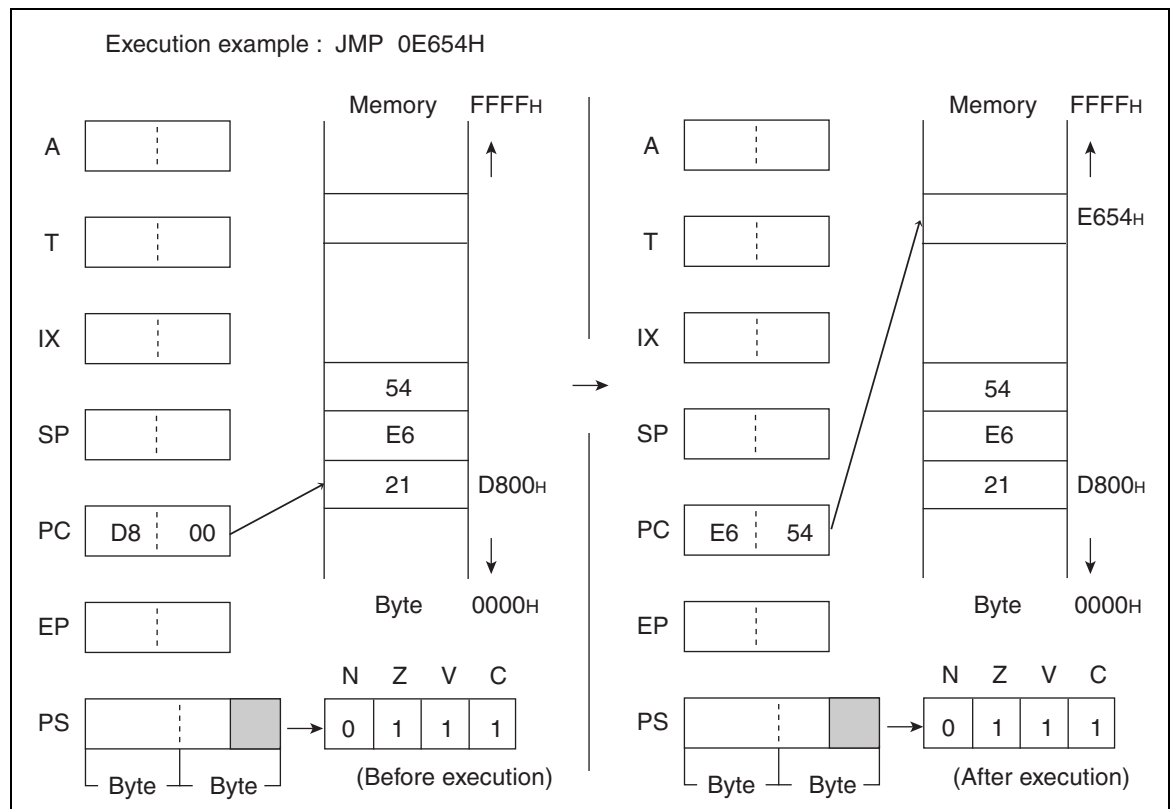
V: Not changed

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

$((A)) \leftarrow T$  (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

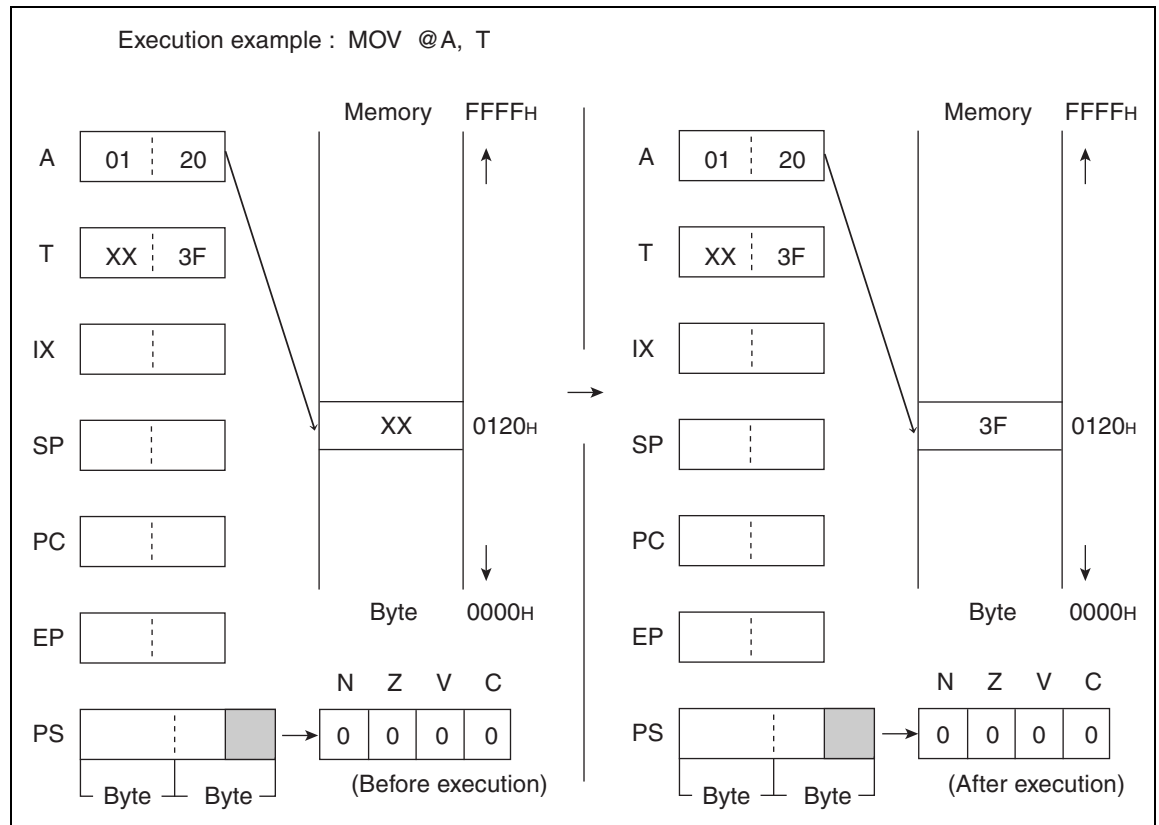
V: Not changed

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.

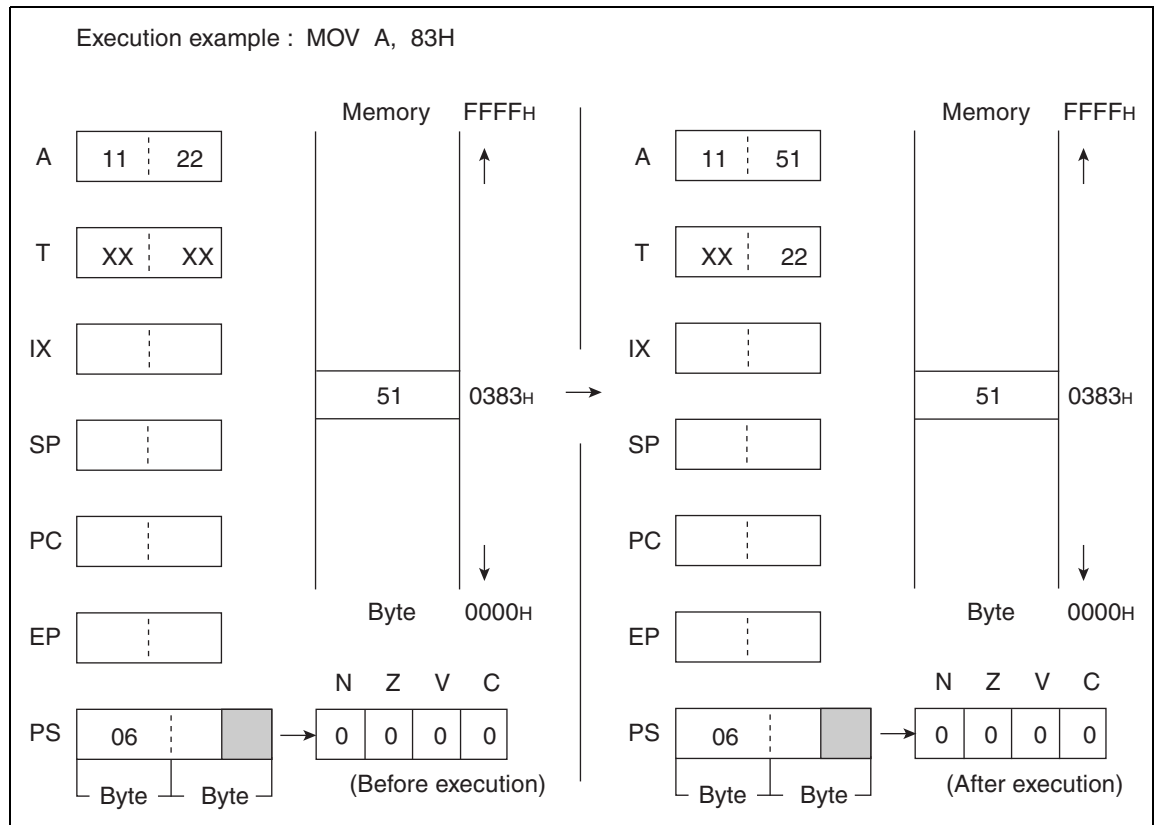
Z: Set to 1 if transferred data is 00<sub>H</sub> and set to 0 in other cases.

V: Not changed

C: Not changed

Table 6-16. 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

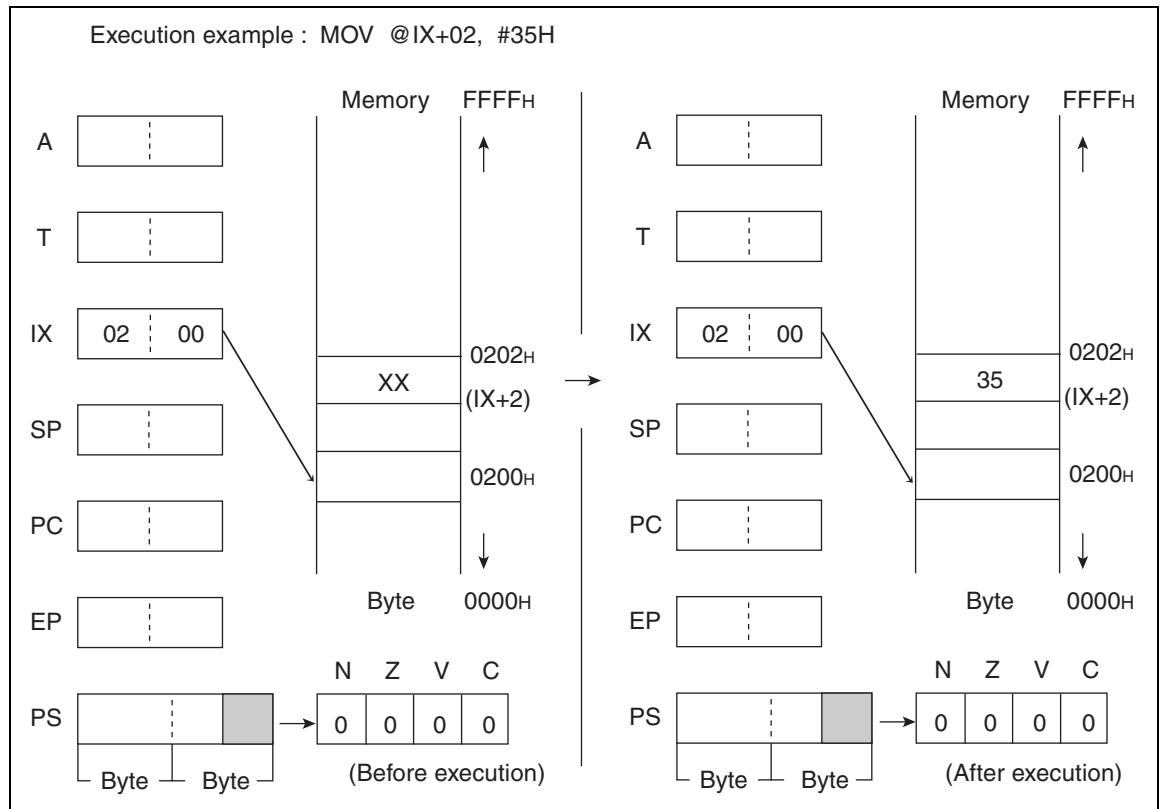
Z: Not changed

V: Not changed

C: Not changed

Table 6-17. 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

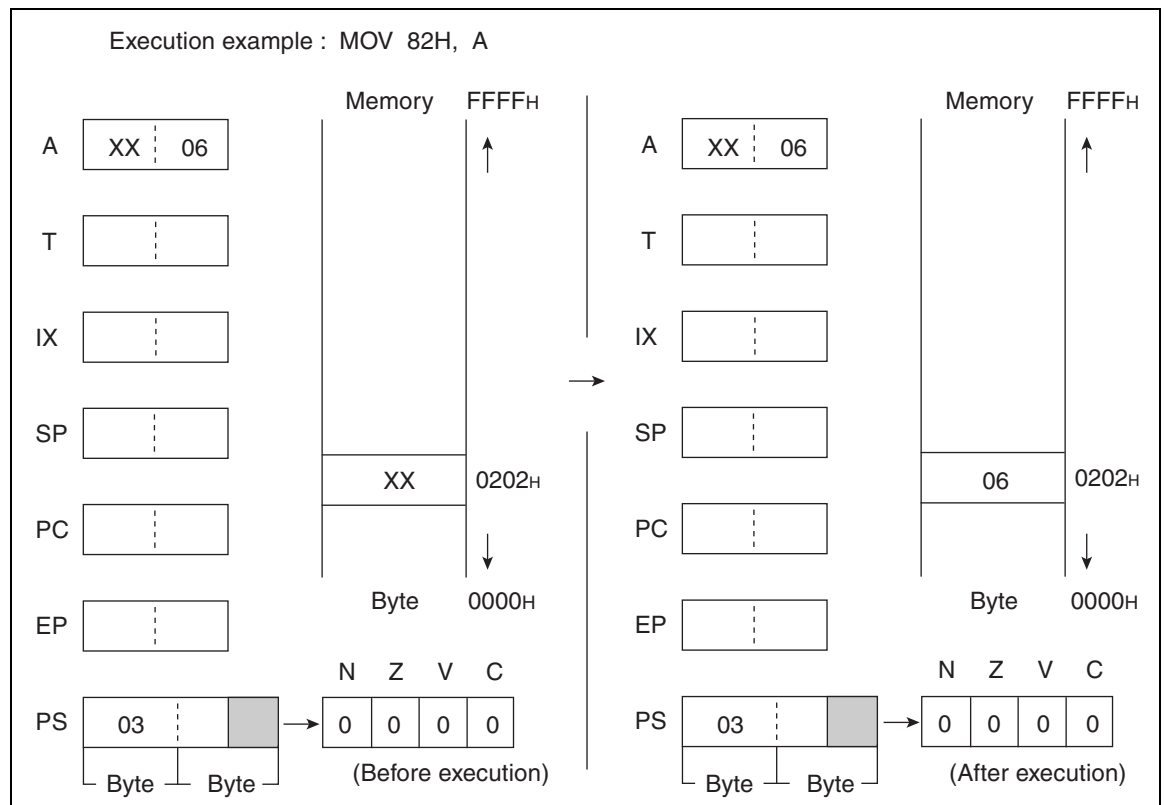
Z: Not changed

V: Not changed

C: Not changed

Table 6-18. 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

$((A)) \leftarrow (T)$  (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

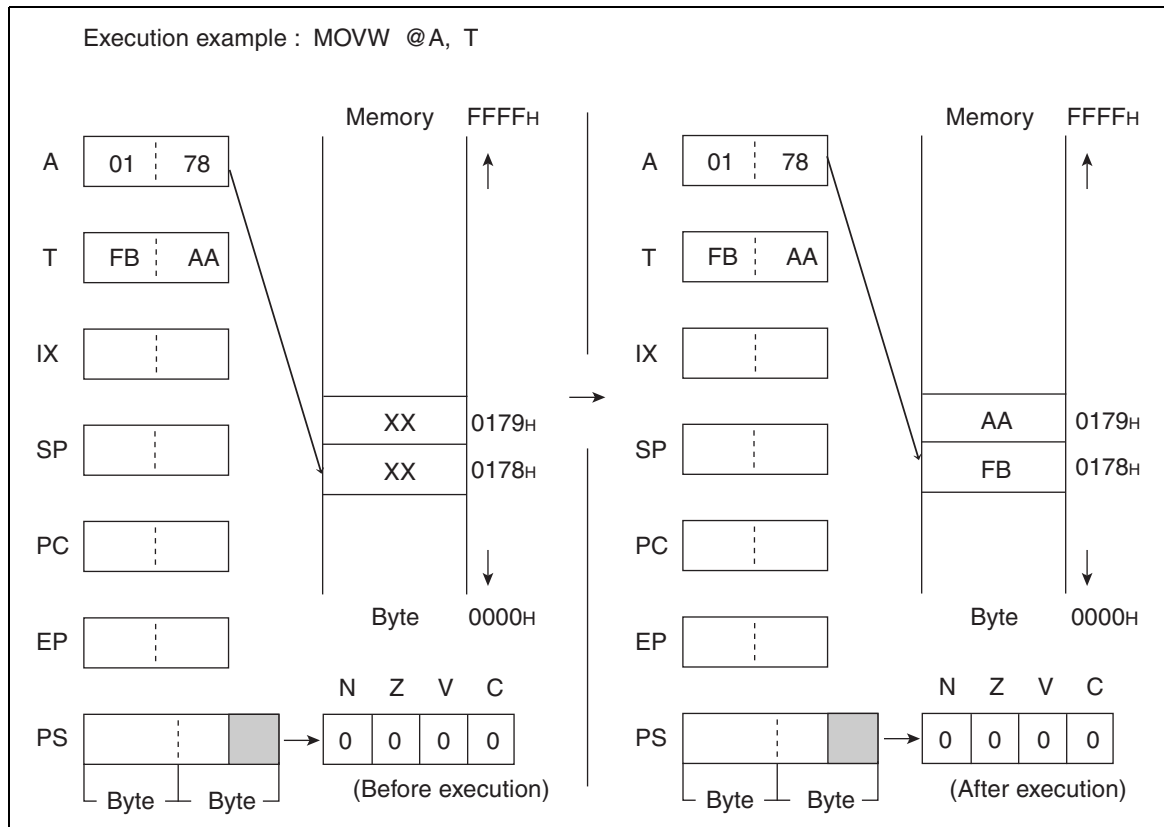
V: Not changed

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.

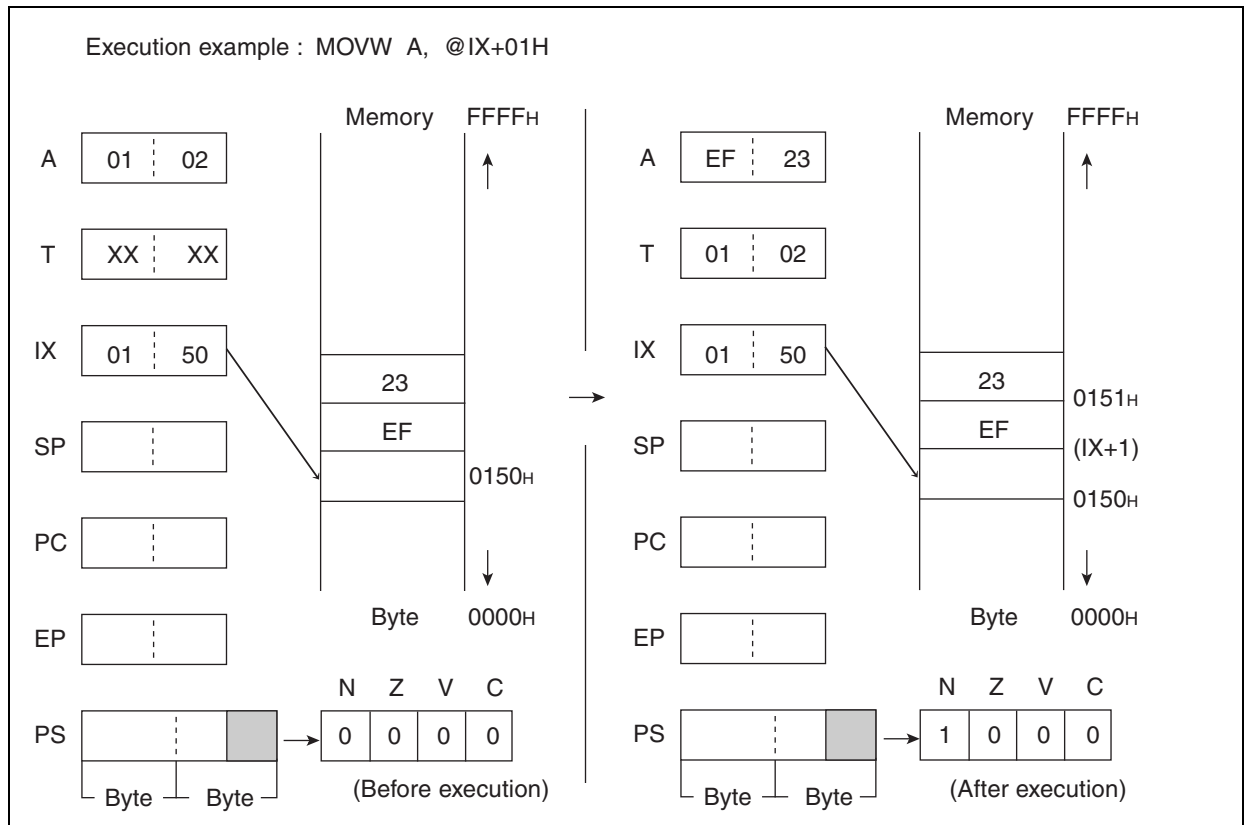
Z: Set to 1 if transferred data is 0000<sub>H</sub> and set to 0 in other cases.

V: Not changed

C: Not changed

Table 6-19. 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

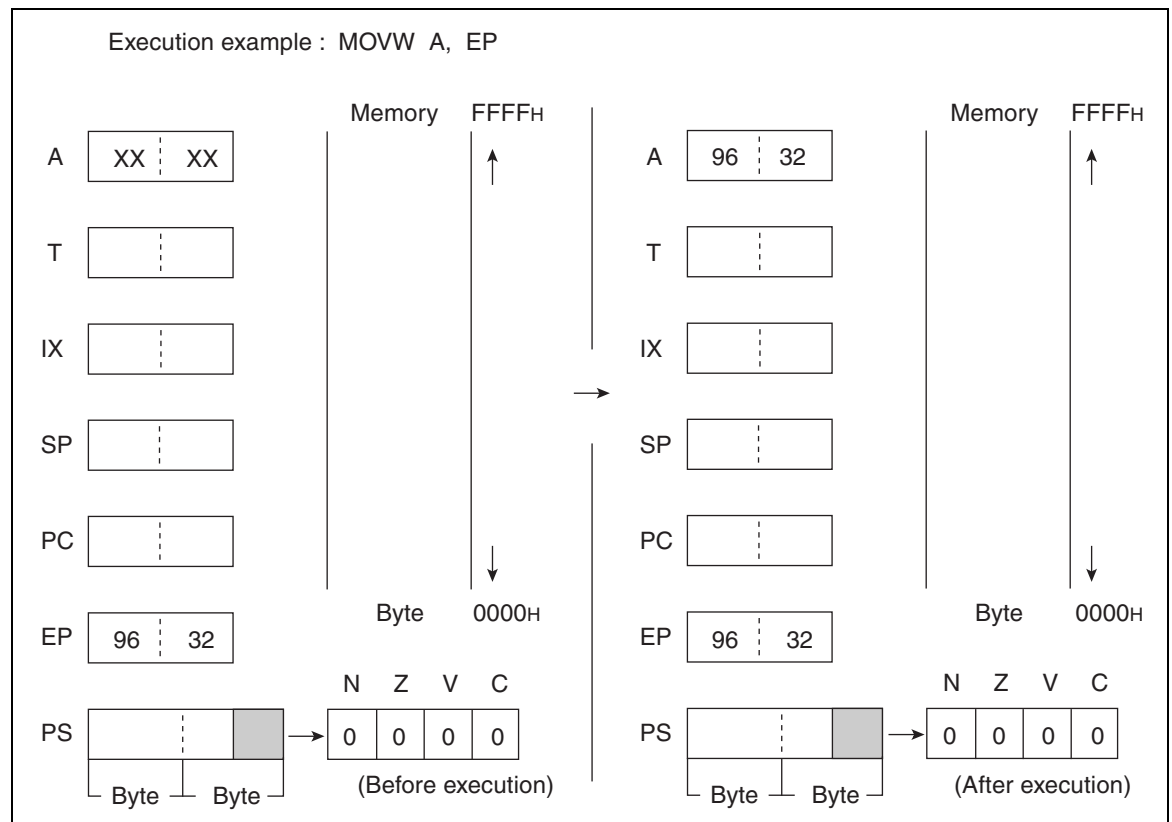
V: Not changed

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

$(A) \leftarrow (IX)$  (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

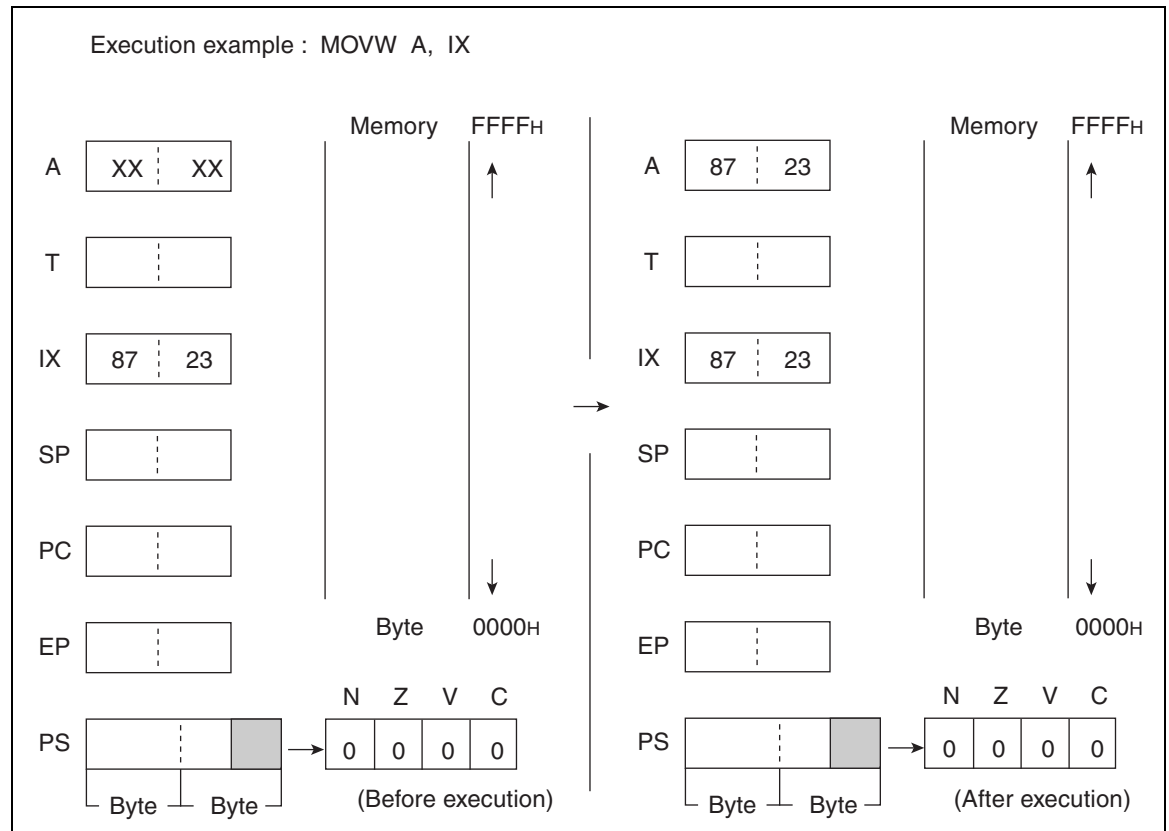
V: Not changed

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

$(A) \leftarrow (PS)$  (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

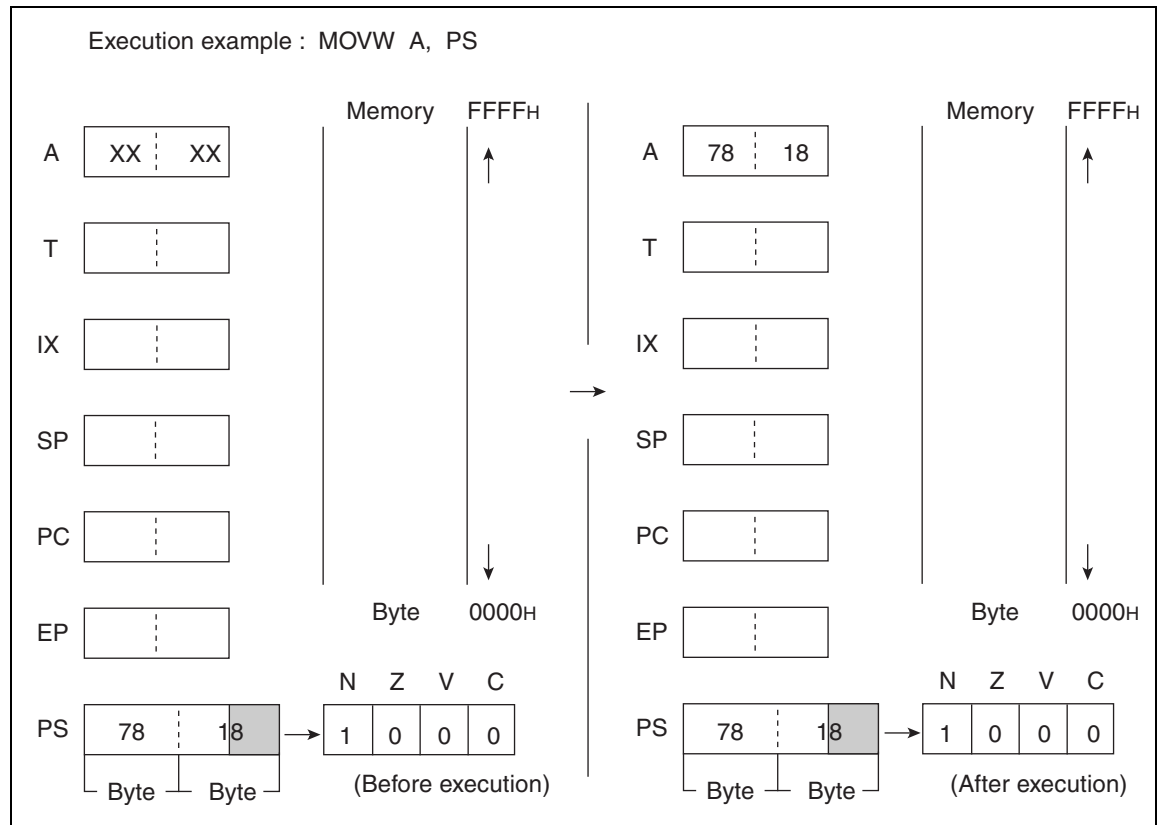
V: Not changed

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

$(A) \leftarrow (PC)$  (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

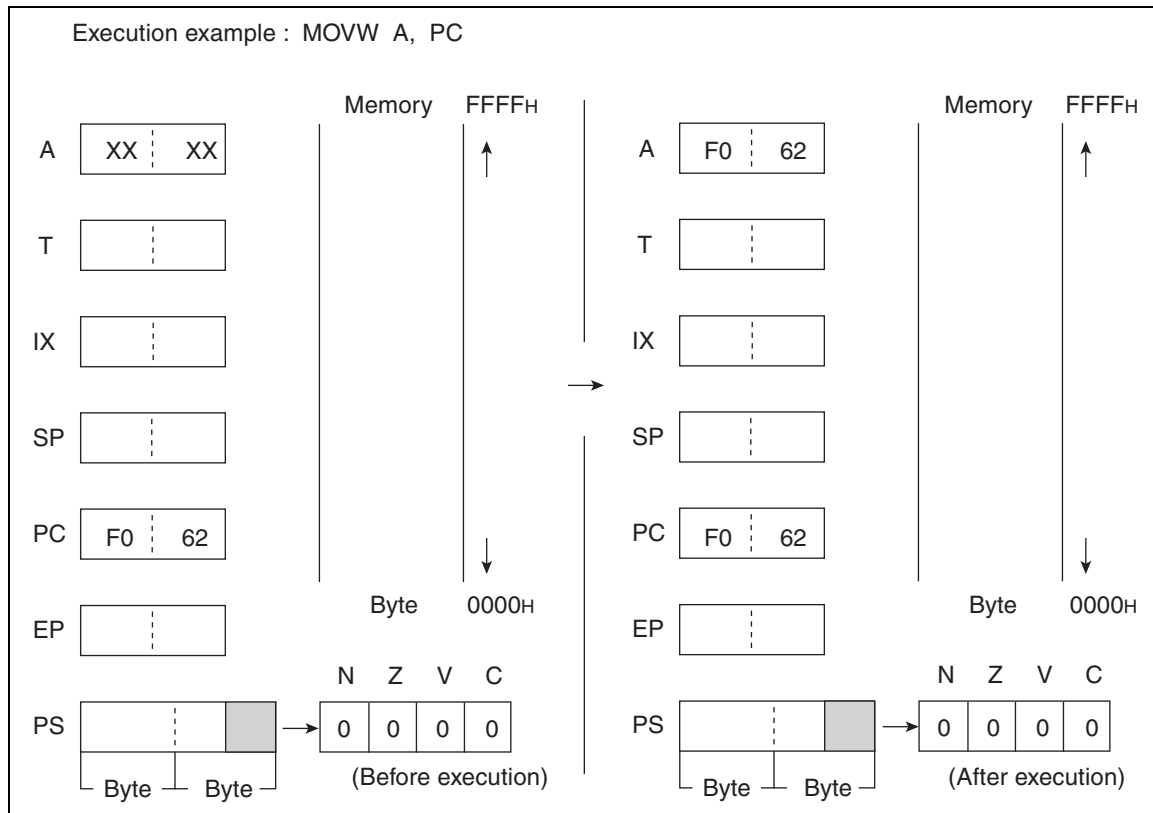
V: Not changed

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

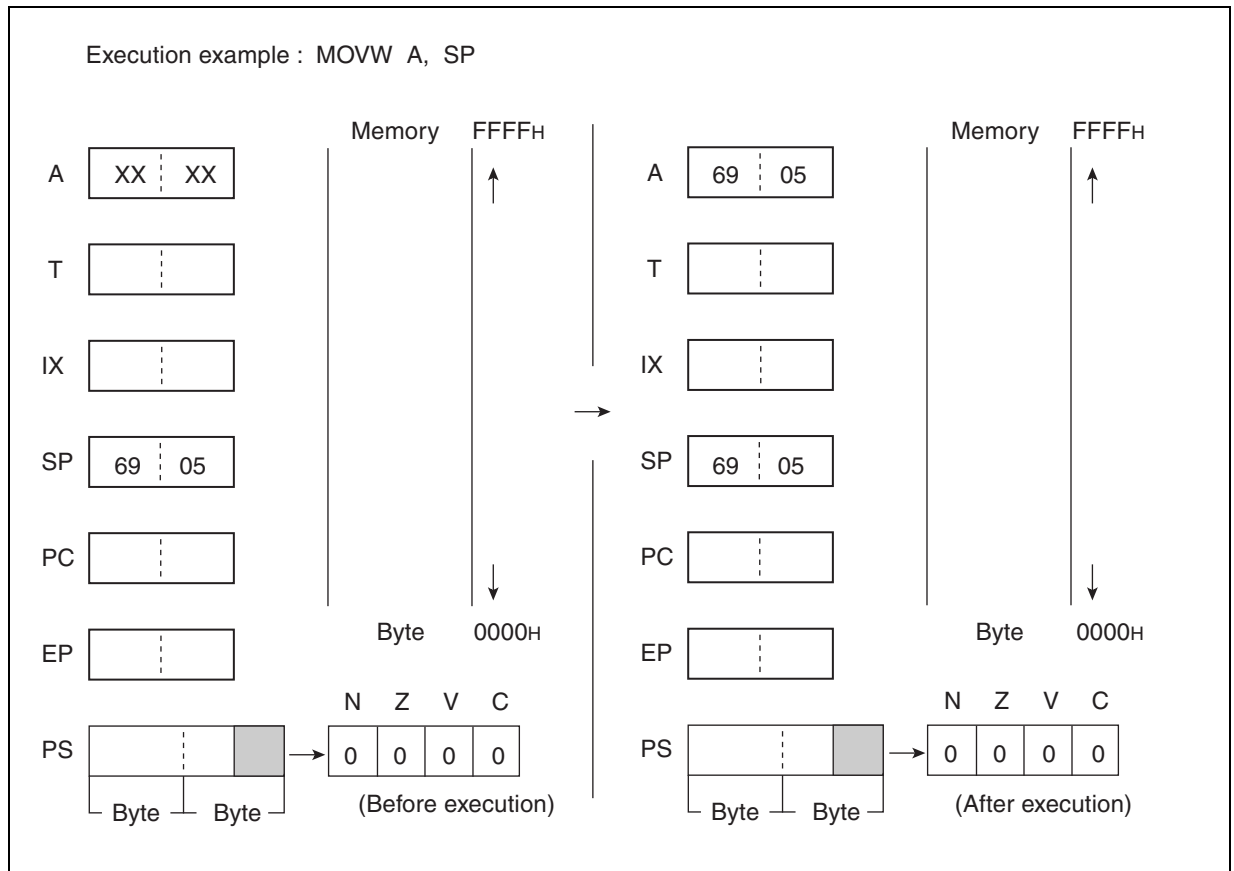
V: Not changed

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

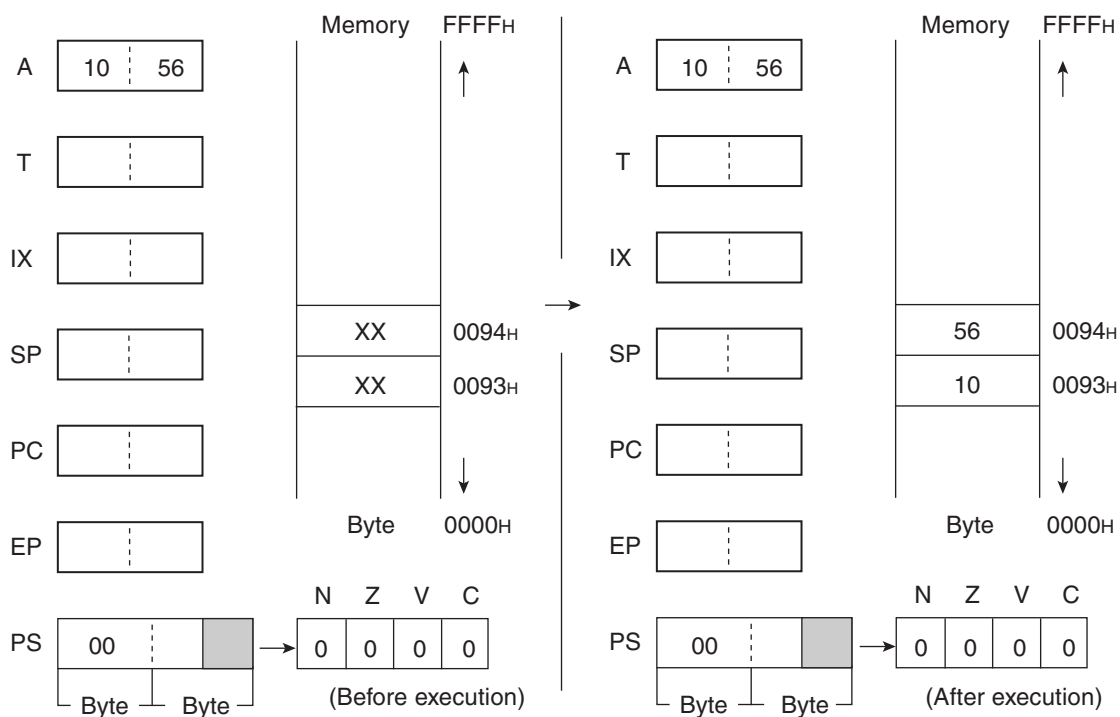
V: Not changed

C: Not changed

Table 6-20. 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

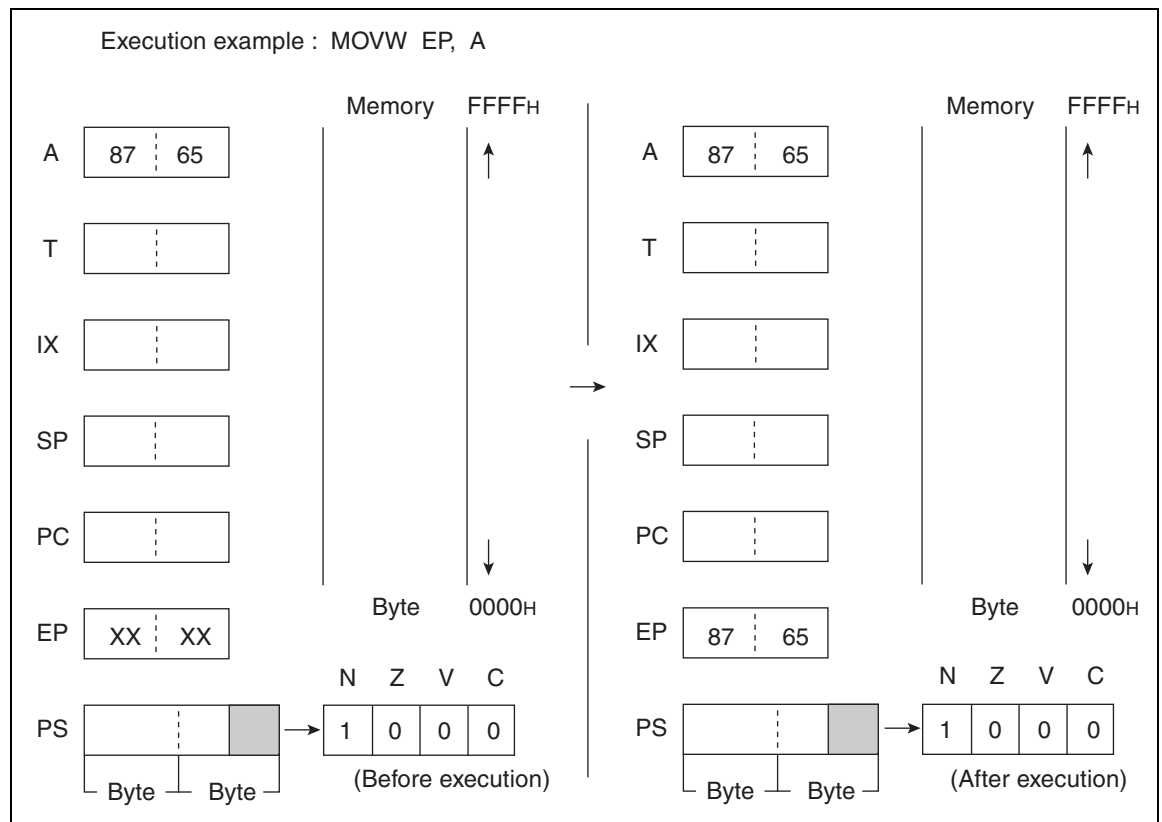
V: Not changed

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

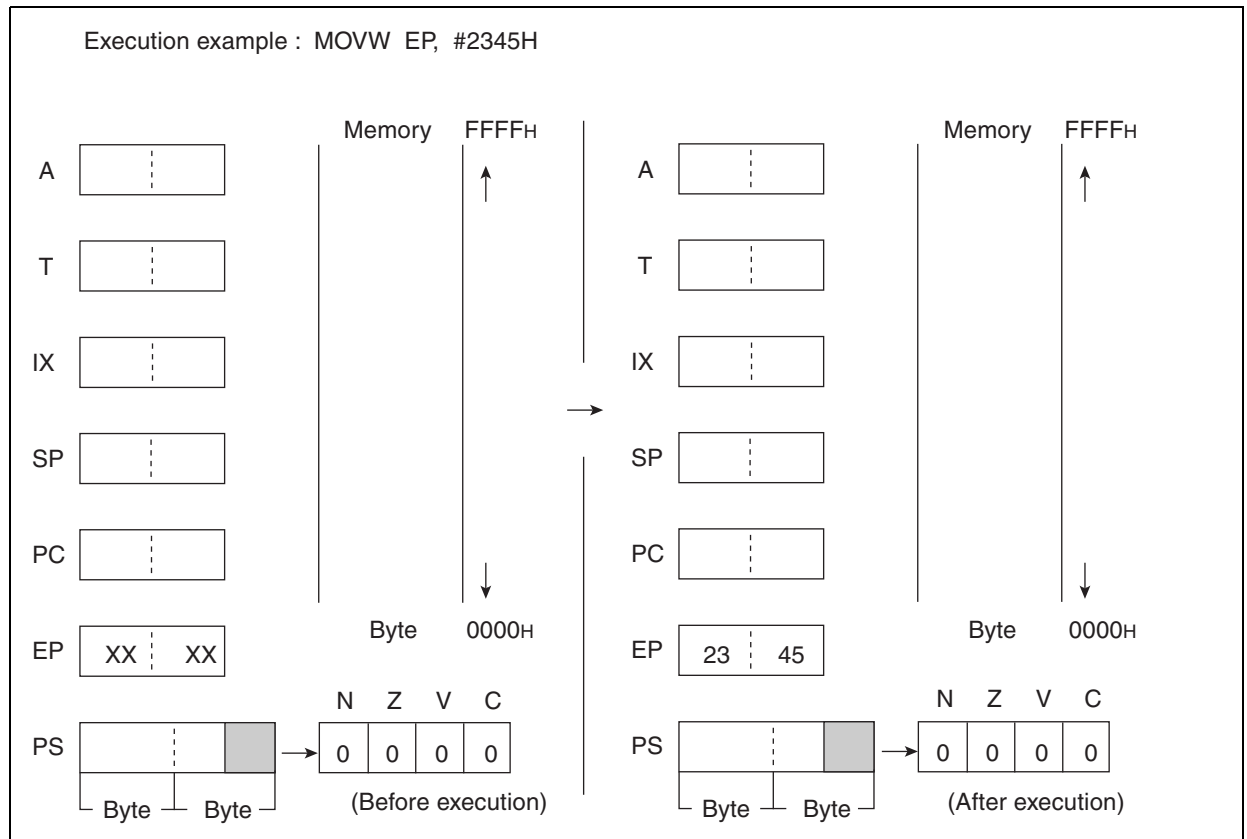
V: Not changed

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

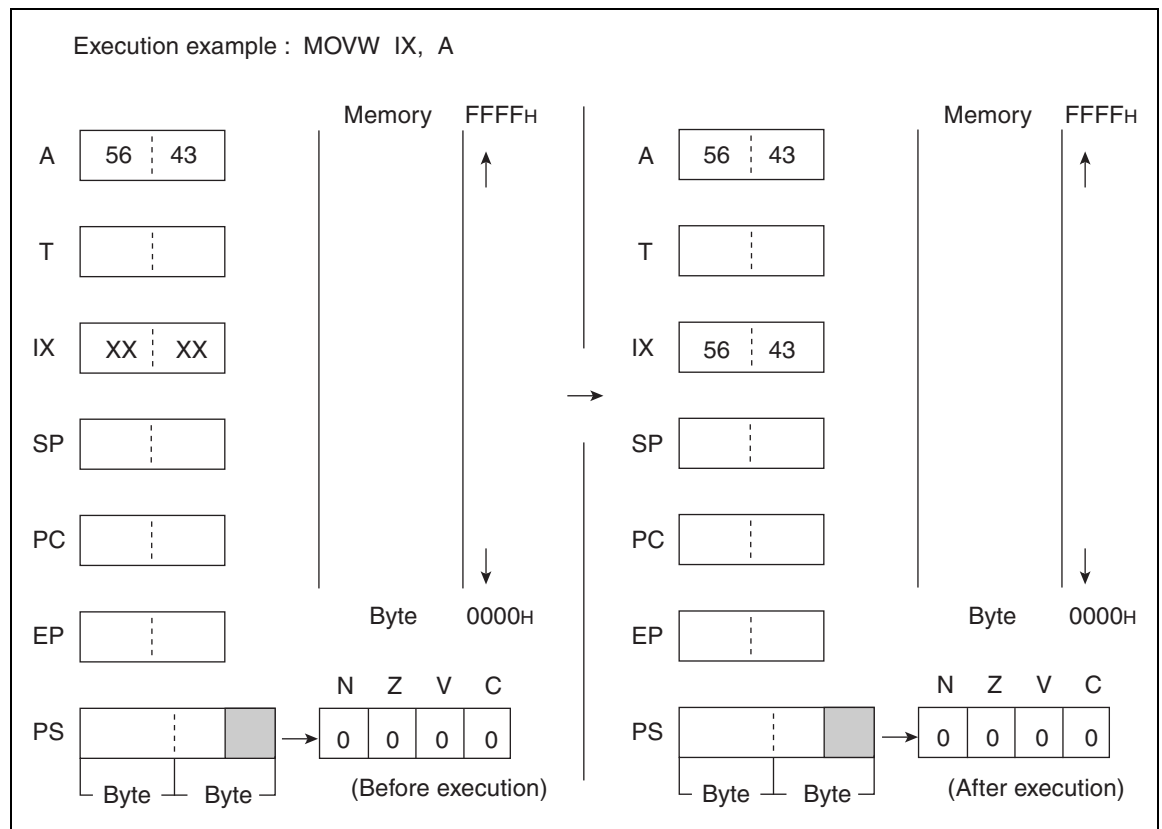
V: Not changed

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

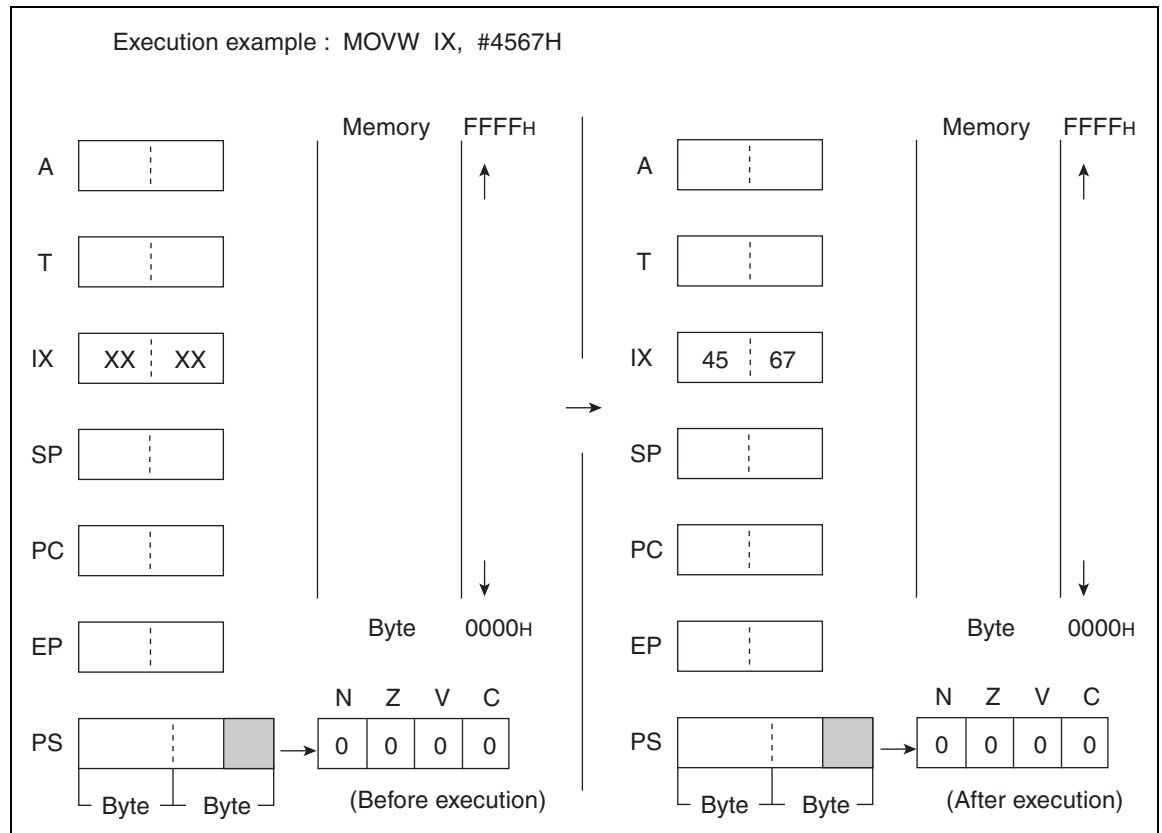
V: Not changed

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

$(PS) \leftarrow (A)$  (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

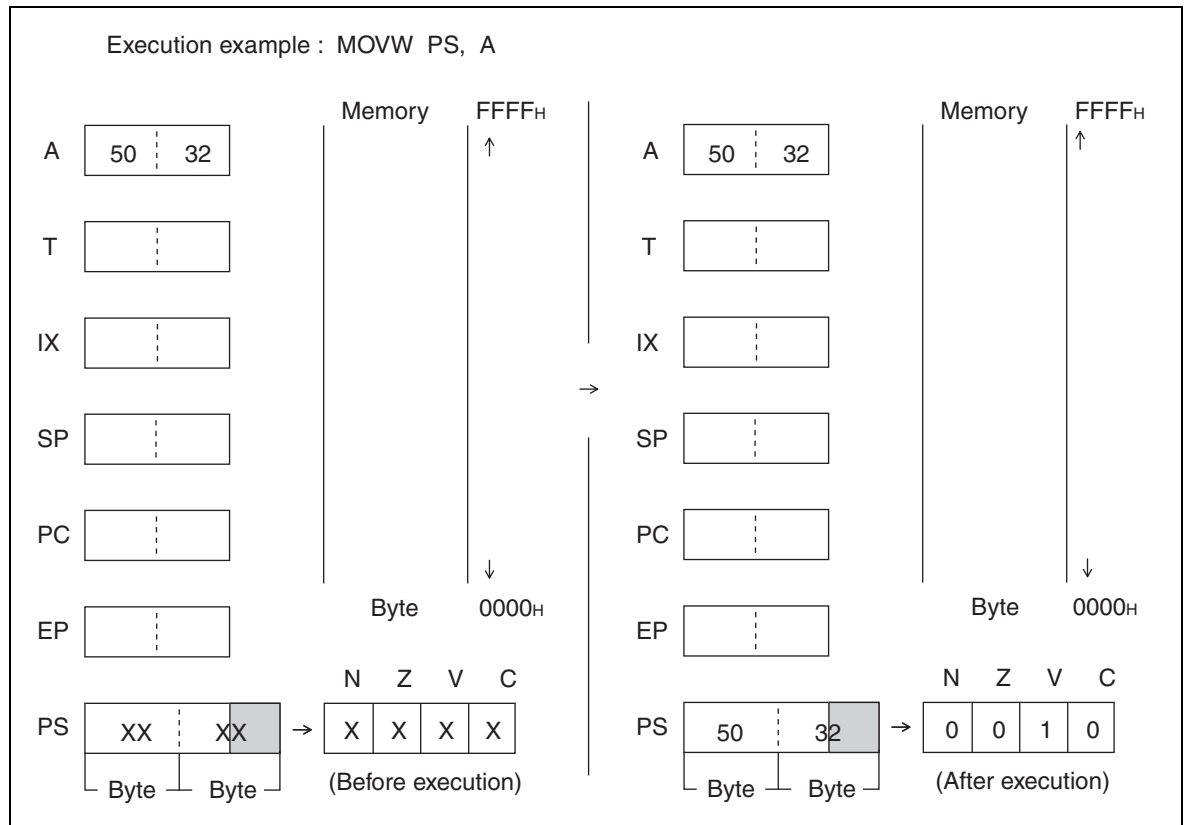
V: Become the value for lower bit 1 of A

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

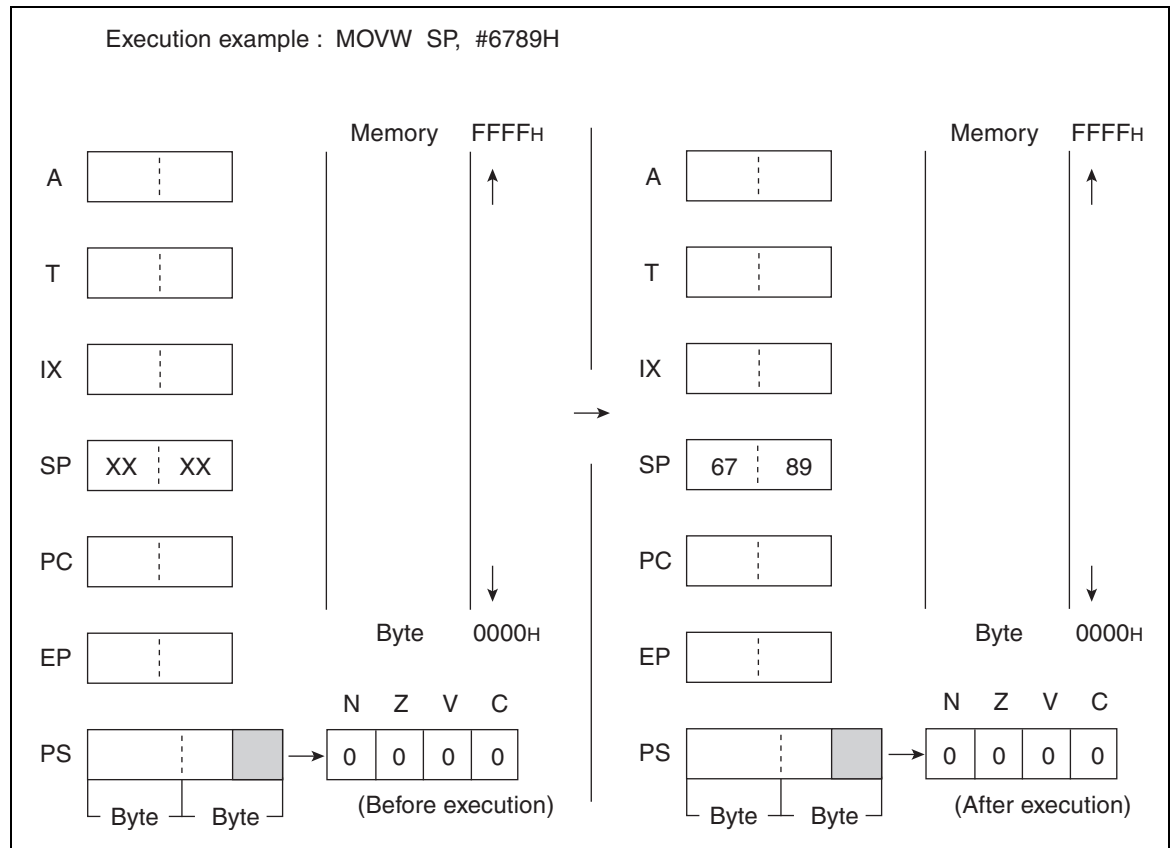
V: Not changed

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

$(SP) \leftarrow (A)$  (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

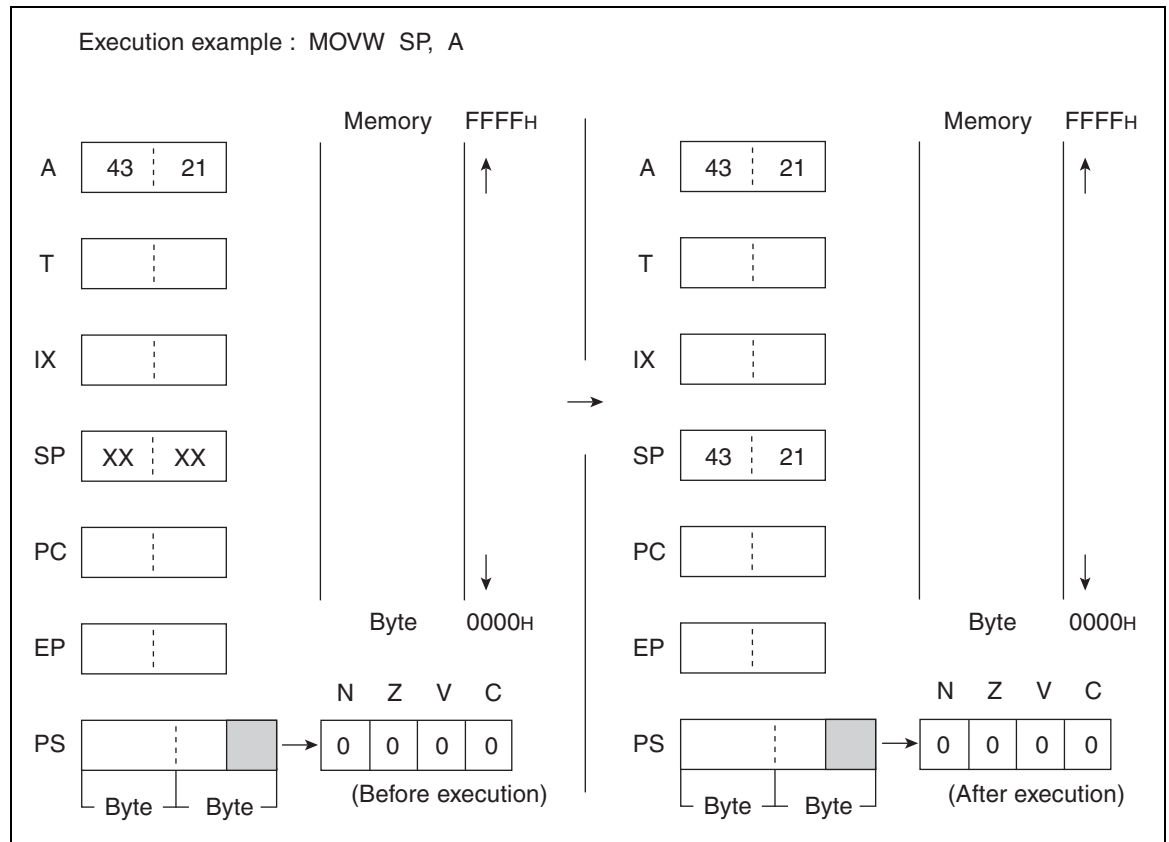
V: Not changed

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

$(A) \leftarrow (AL) * (TL)$

Assembler format

MULU A

Condition code (CCR)

| N | Z | V | C |
|---|---|---|---|
| - | - | - | - |

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

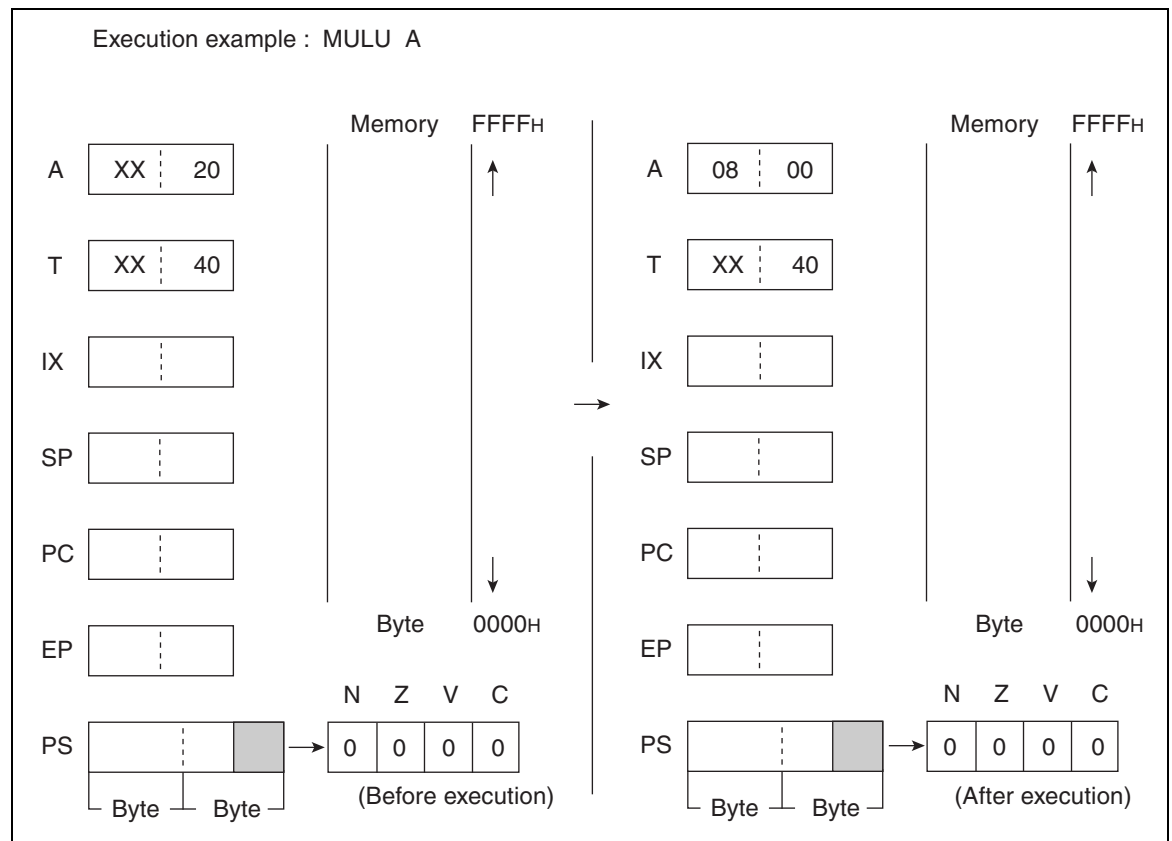
V: Not changed

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

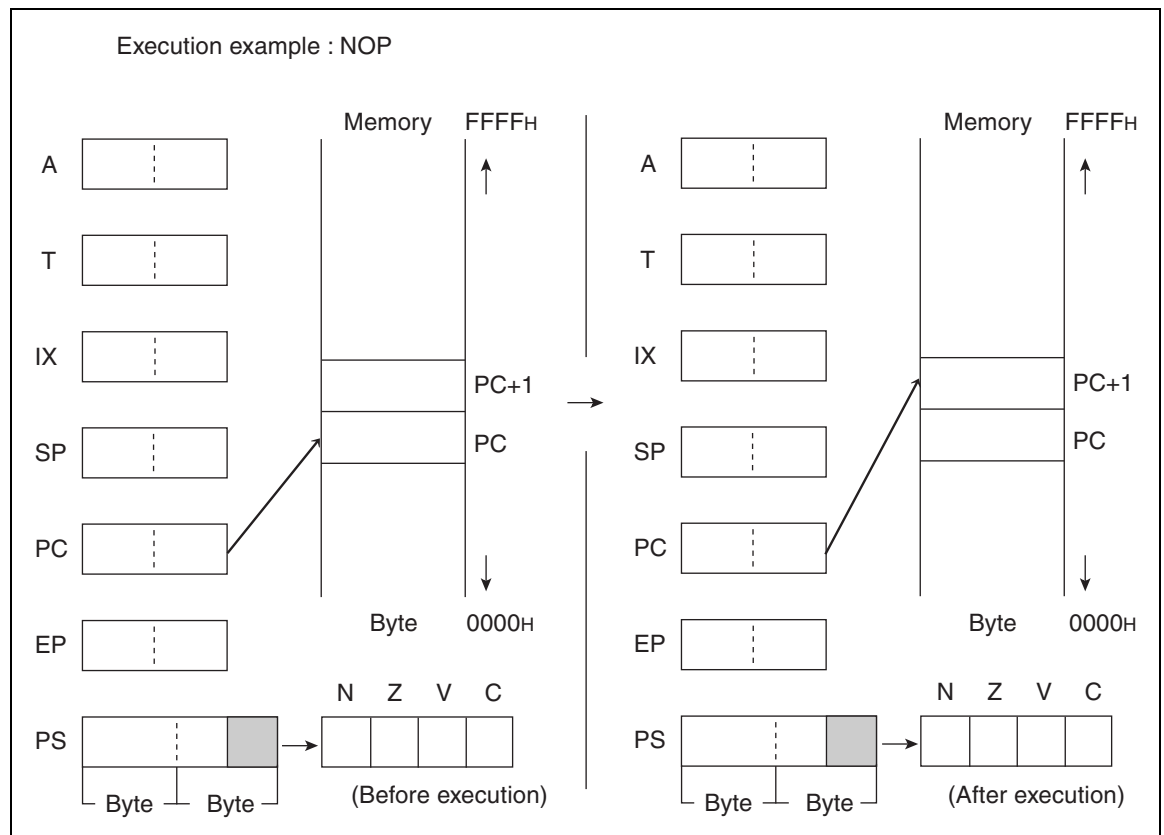
V: Not changed

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

$(AL) \leftarrow (AL) \vee (TL)$  (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<sub>H</sub> and set to 0 in other cases.

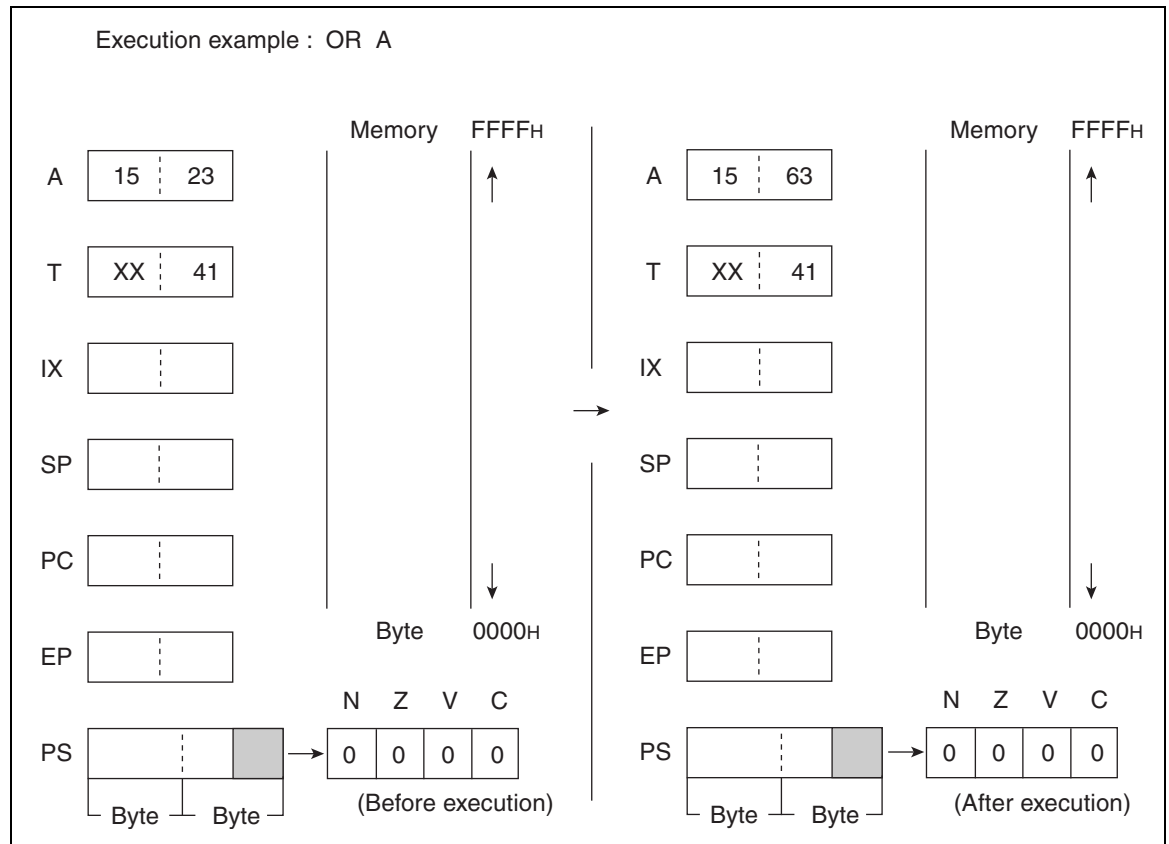
V: Always set to 0

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

$(AL) \leftarrow (AL) \vee (EA)$  (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.

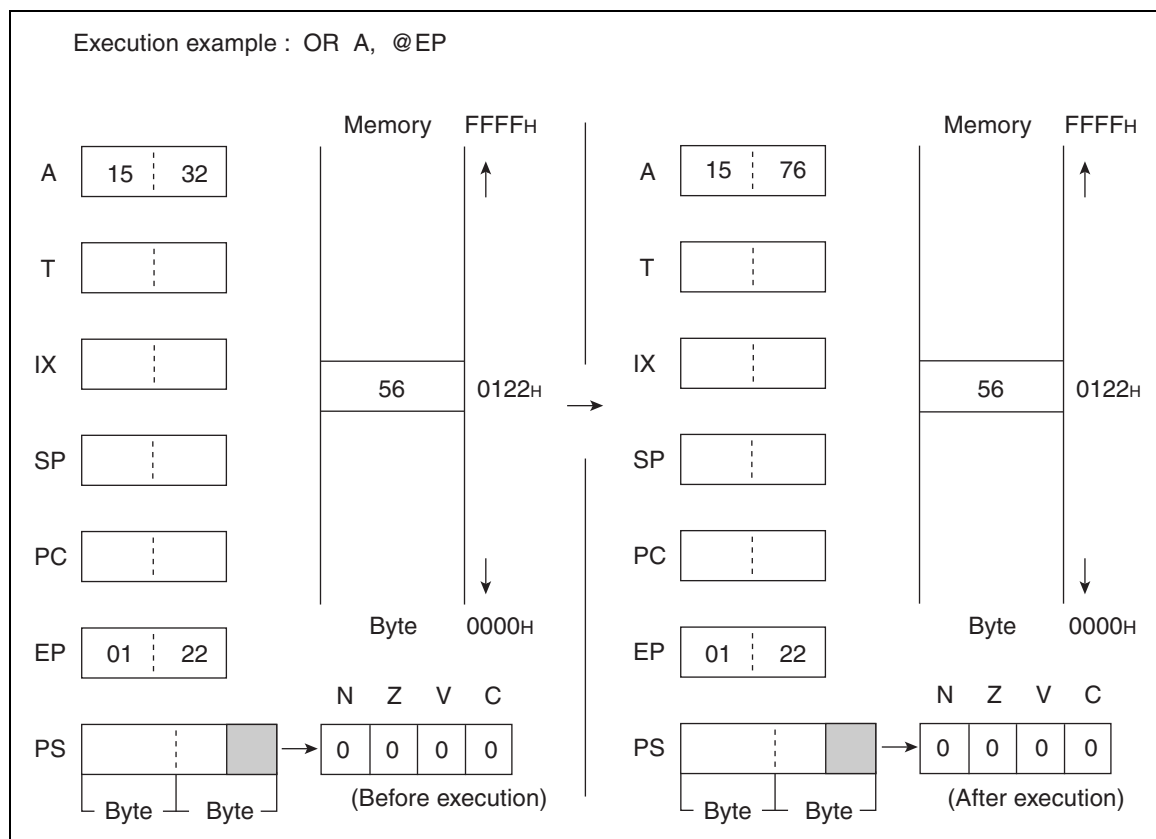
Z: Set to 1 if the result of operation is 00<sub>H</sub> and set to 0 in other cases.

V: Always set to 0

C: Not changed

Table 6-21. 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

$(A) \leftarrow (A) \vee (T)$  (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<sub>H</sub> and set to 0 in other cases.

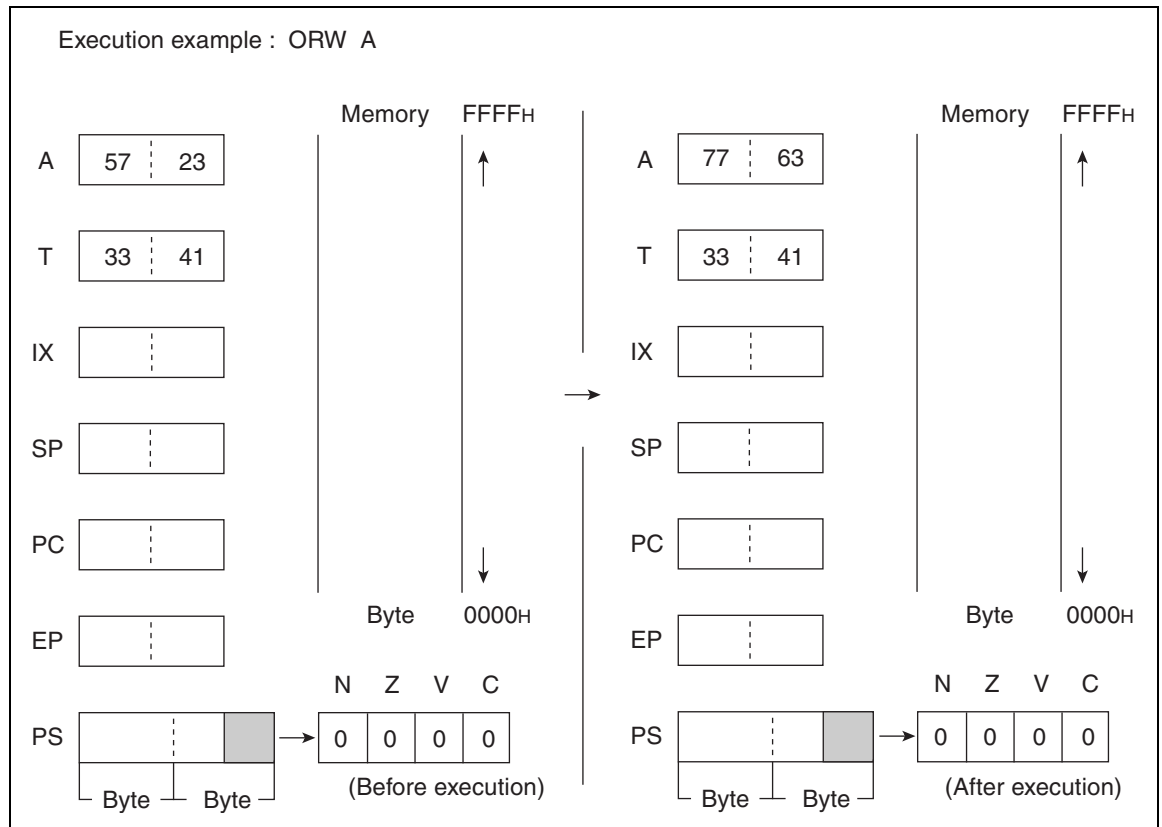
V: Always set to 0

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

$(SP) \leftarrow (SP) - 2$  (Word subtraction)

$((SP)) \leftarrow (dr)$  (Word transfer)

Assembler format

PUSHW dr

Condition code (CCR)

| N | Z | V | C |
|---|---|---|---|
| - | - | - | - |

+: Changed by executing instruction

-: Not changed

N: Not changed

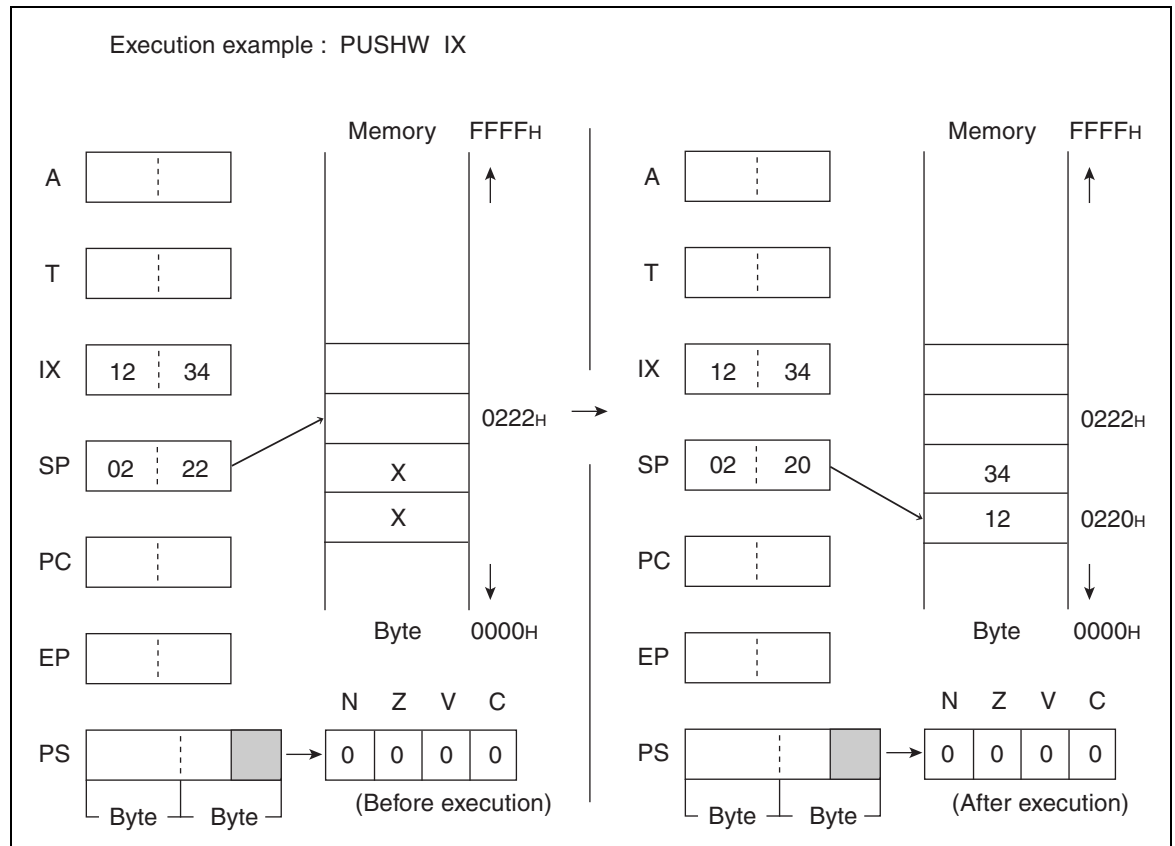
Z: Not changed

V: Not changed

C: Not changed

Table 6-22. 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 Inherent 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 Inherent Register from Stack Memory)

Operation

$(dr) \leftarrow ((SP))$  (Word transfer)

$(SP) \leftarrow (SP) + 2$  (Word addition)

Assembler format

POPW dr

Condition code (CCR)

| N | Z | V | C |
|---|---|---|---|
| - | - | - | - |

+: Changed by executing instruction

-: Not changed

N: Not changed

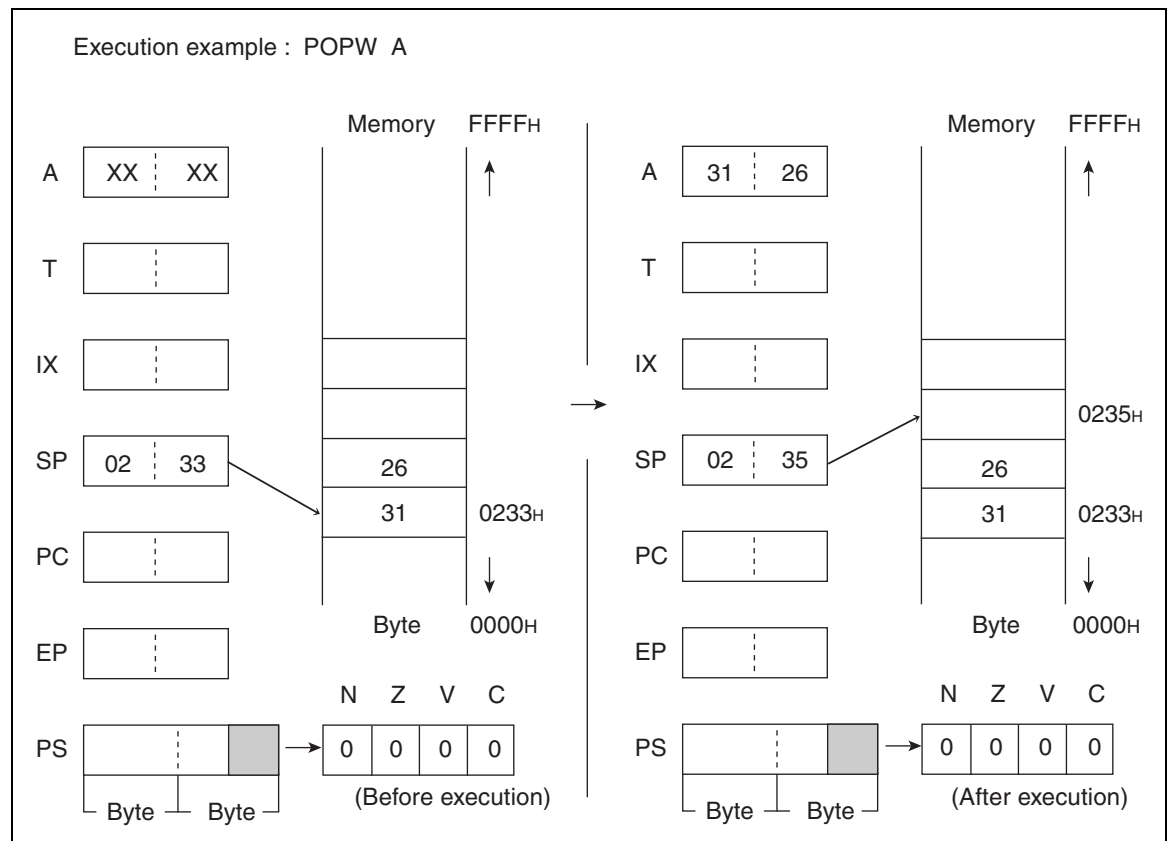
Z: Not changed

V: Not changed

C: Not changed

Table 6-23. 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

$(PC) \leftarrow ((SP))$  (Word transfer)

$(SP) \leftarrow (SP) + 2$  (Word addition)

Assembler format

RET

Condition code (CCR)

| N | Z | V | C |
|---|---|---|---|
| - | - | - | - |

+: Changed by executing instruction

-: Not changed

N: Not changed

Z: Not changed

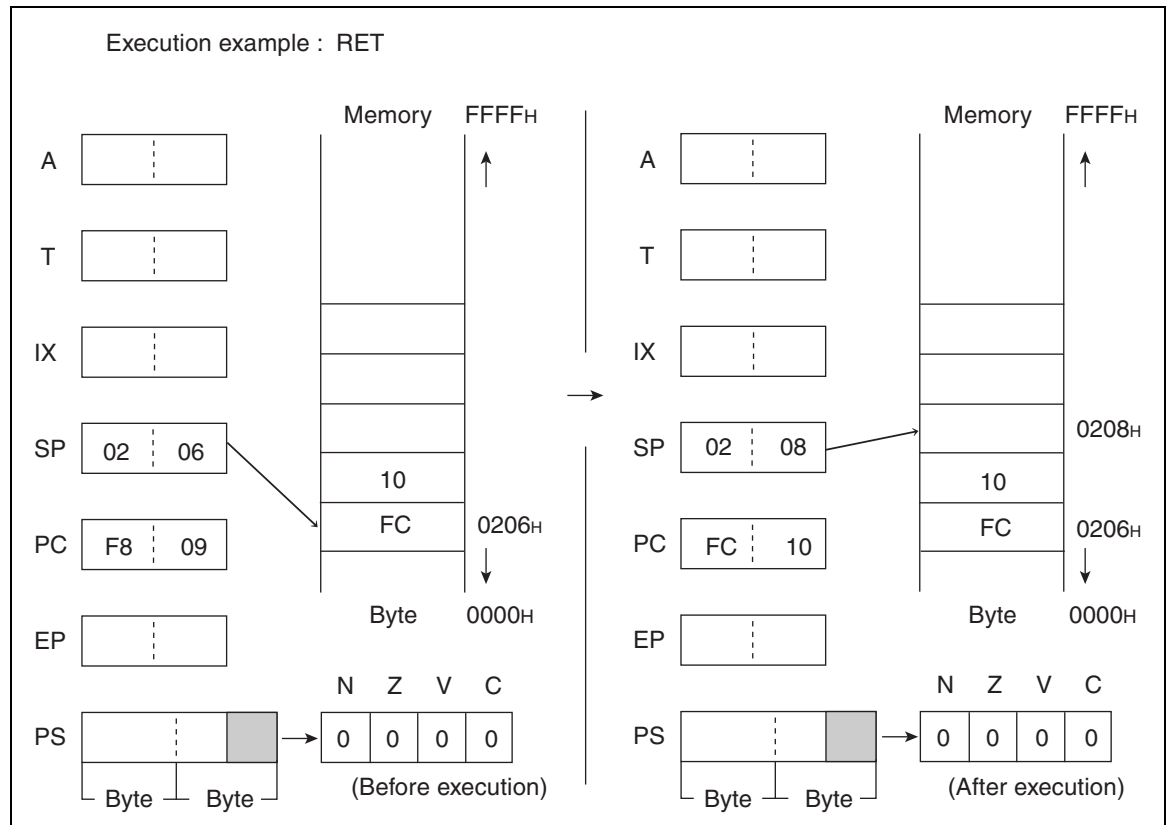
V: Not changed

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

$(PS) \leftarrow ((SP)), (PC) \leftarrow ((SP + 2))$  (Word transfer)

$(SP) \leftarrow (SP) + 4$  (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.

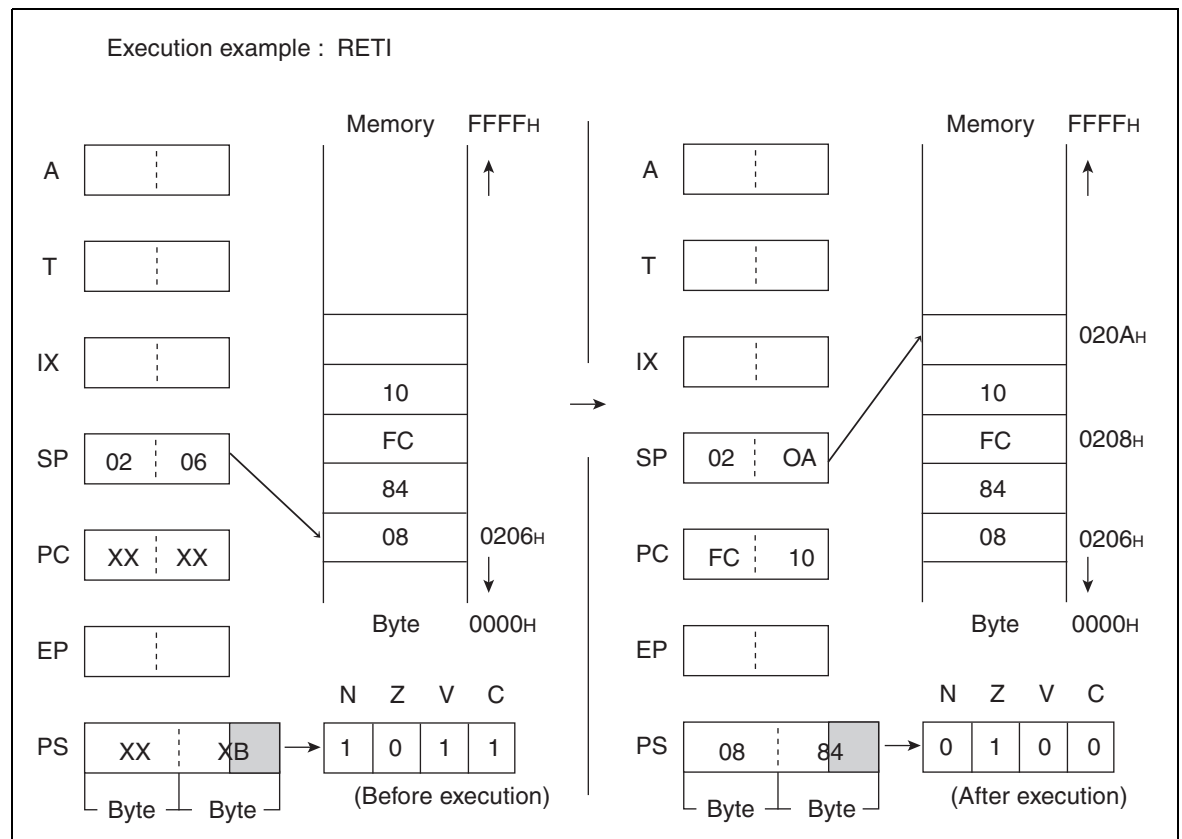
V: Become to the saved value of V.

C: Become to the saved value of C.

Number of execution cycles: 8

Byte count: 1

OP code: 30

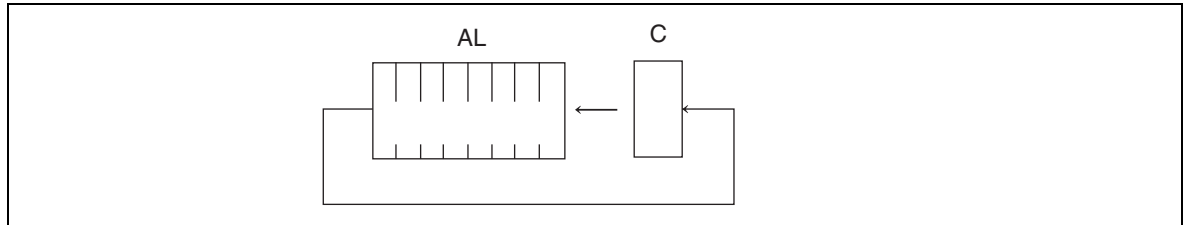


## 6.69 ROLC (Rotate Byte Data of Accumulator with Carry to Left)

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<sub>H</sub> and set to 0 in other cases.

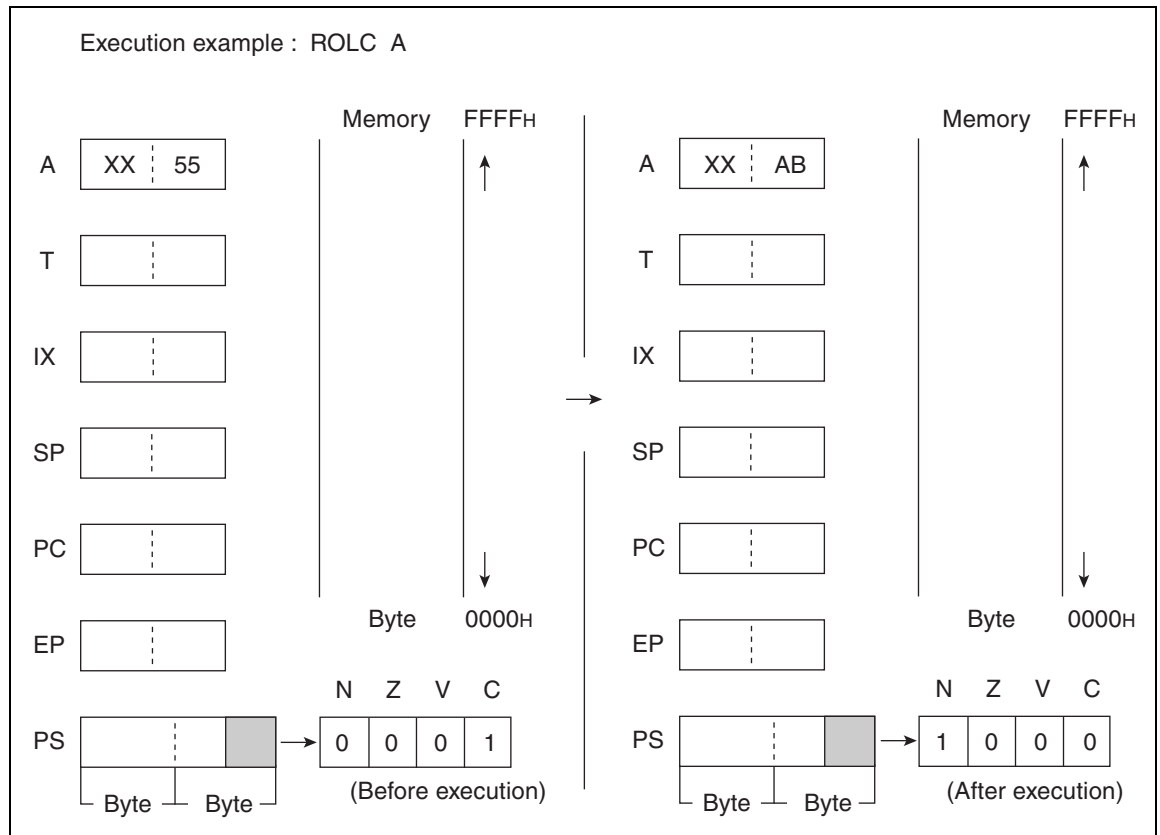
V: Not changed

C: Enter Bit 7 of A before shift.

Number of execution cycle: 1

Byte count: 1

OP code: 02

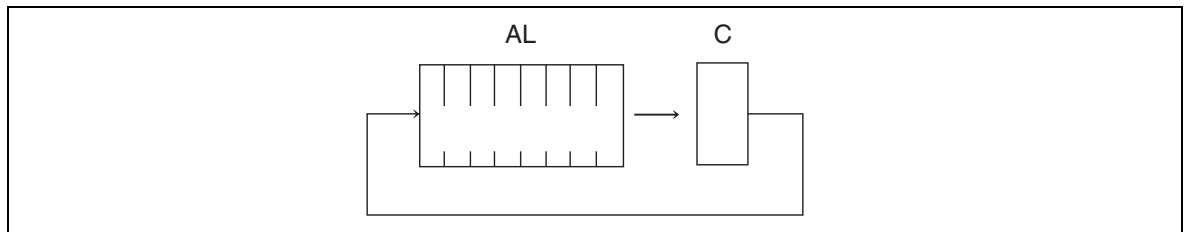


## 6.70 RORC (Rotate Byte Data of Accumulator with Carry to Right)

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<sub>H</sub> and set to 0 in other cases.

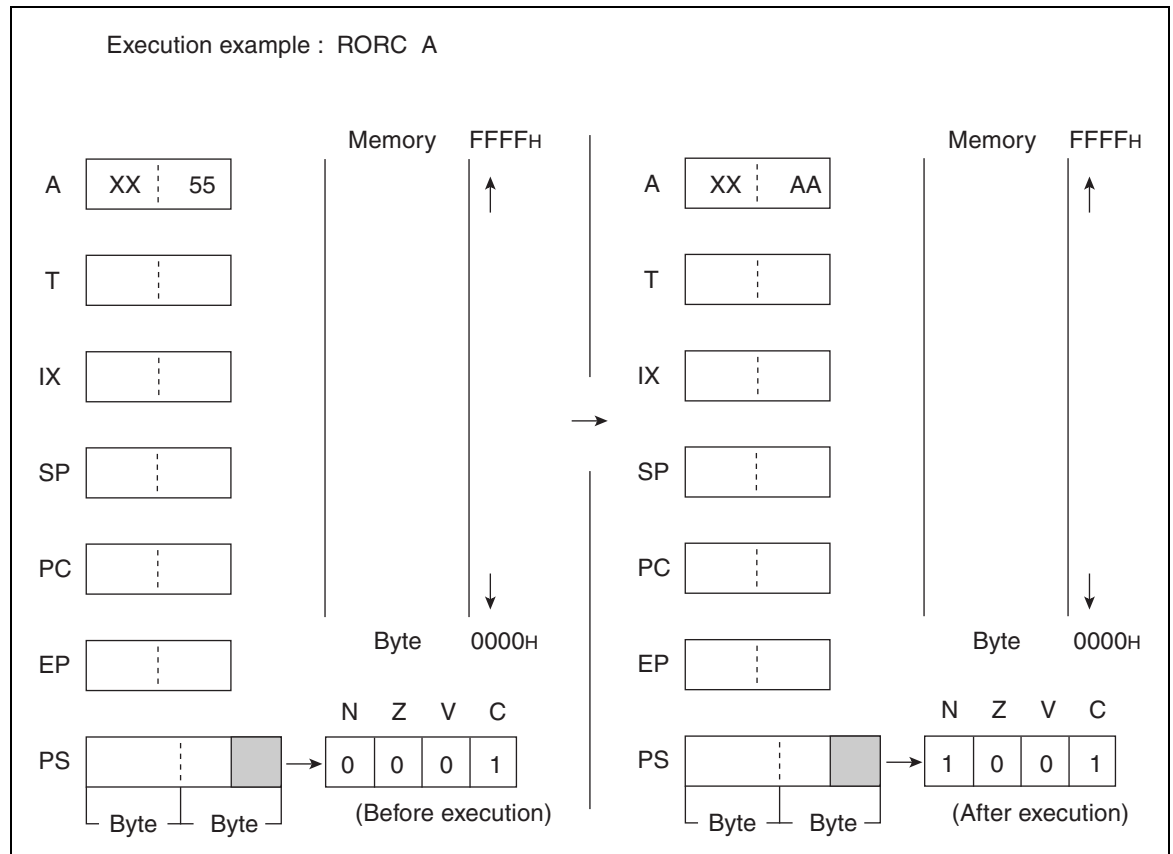
V: Not changed

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

$(AL) \leftarrow (TL) - (AL) - C$  (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<sub>H</sub> and set to 0 in other cases.

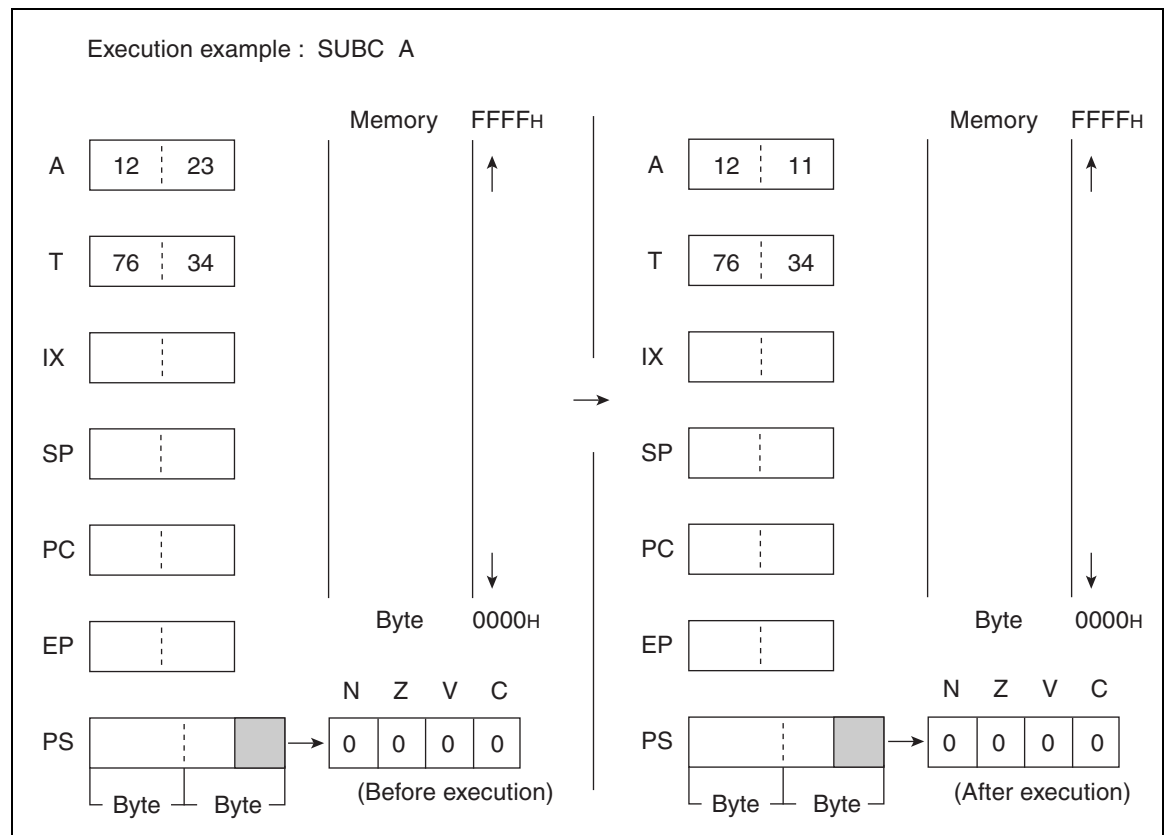
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

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

$(AL) \leftarrow (AL) - (EA) - C$  (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.

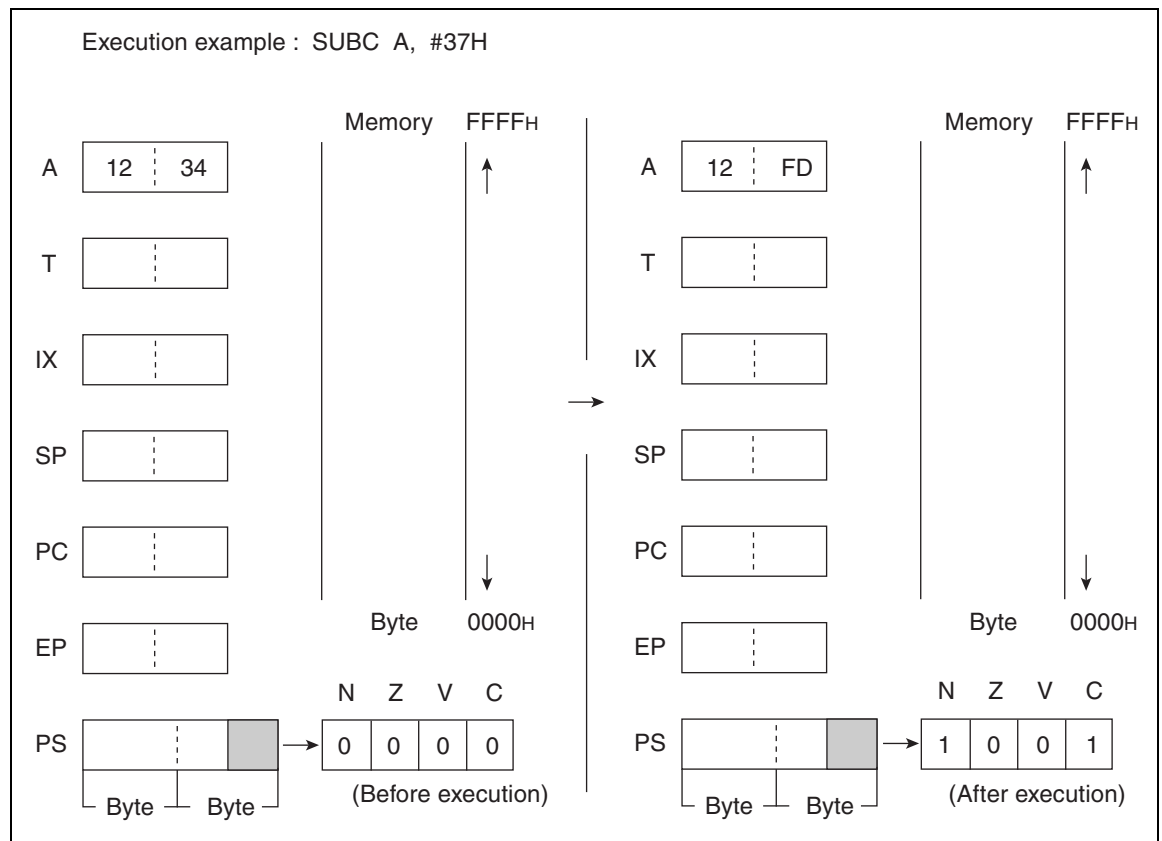
Z: Set to 1 if the result of operation is 00<sub>H</sub> and set to 0 in other cases.

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. 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

$(AL) \leftarrow (T) - (A) - C$  (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<sub>H</sub> and set to 0 in other cases.

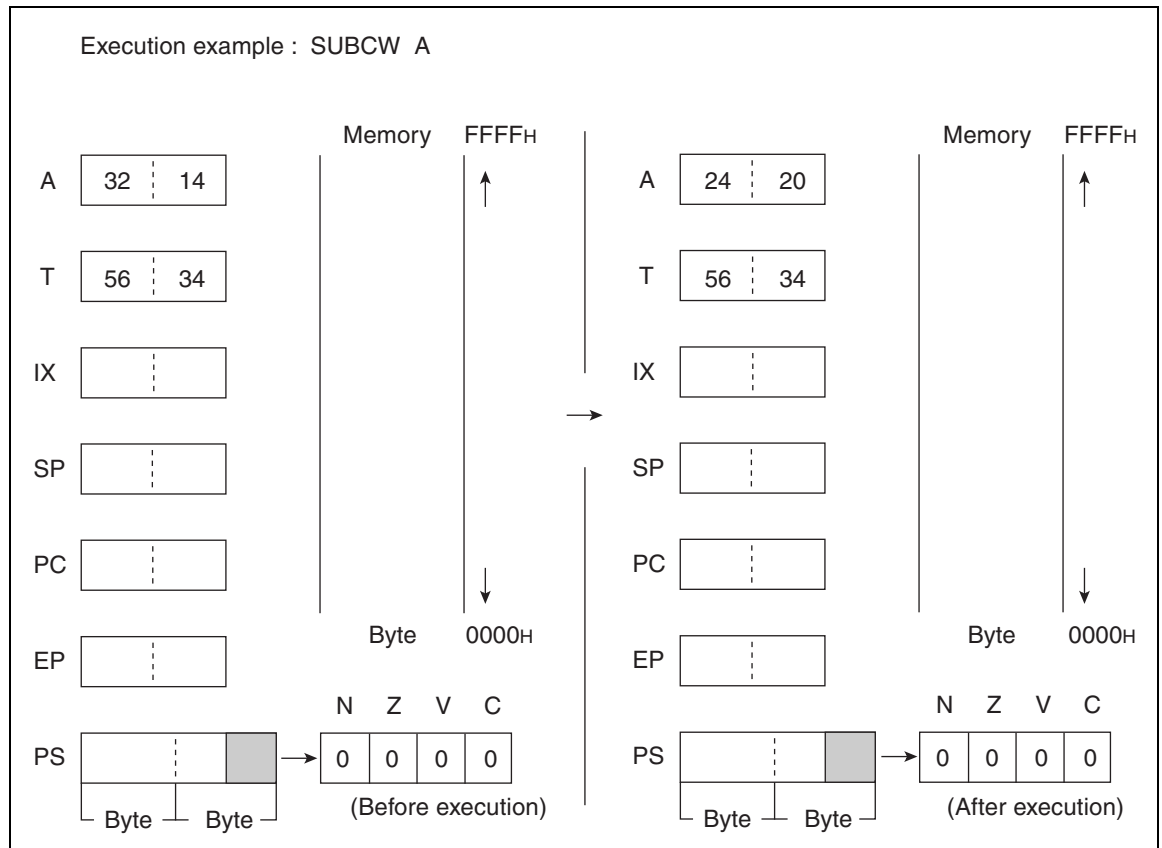
V: Set to 1 if an overflow occurs as the result of operation and set to 0 in other cases.

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

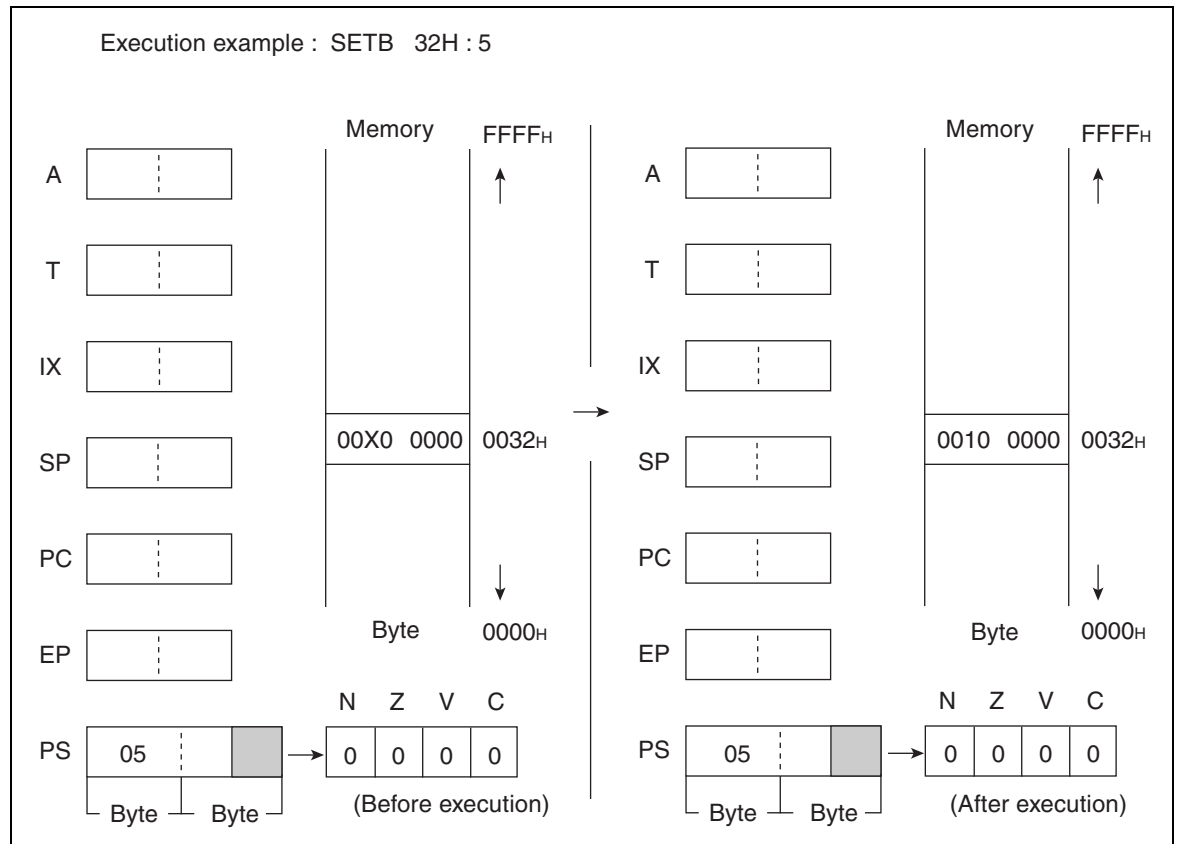
V: Not changed

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

$(C) \leftarrow 1$

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

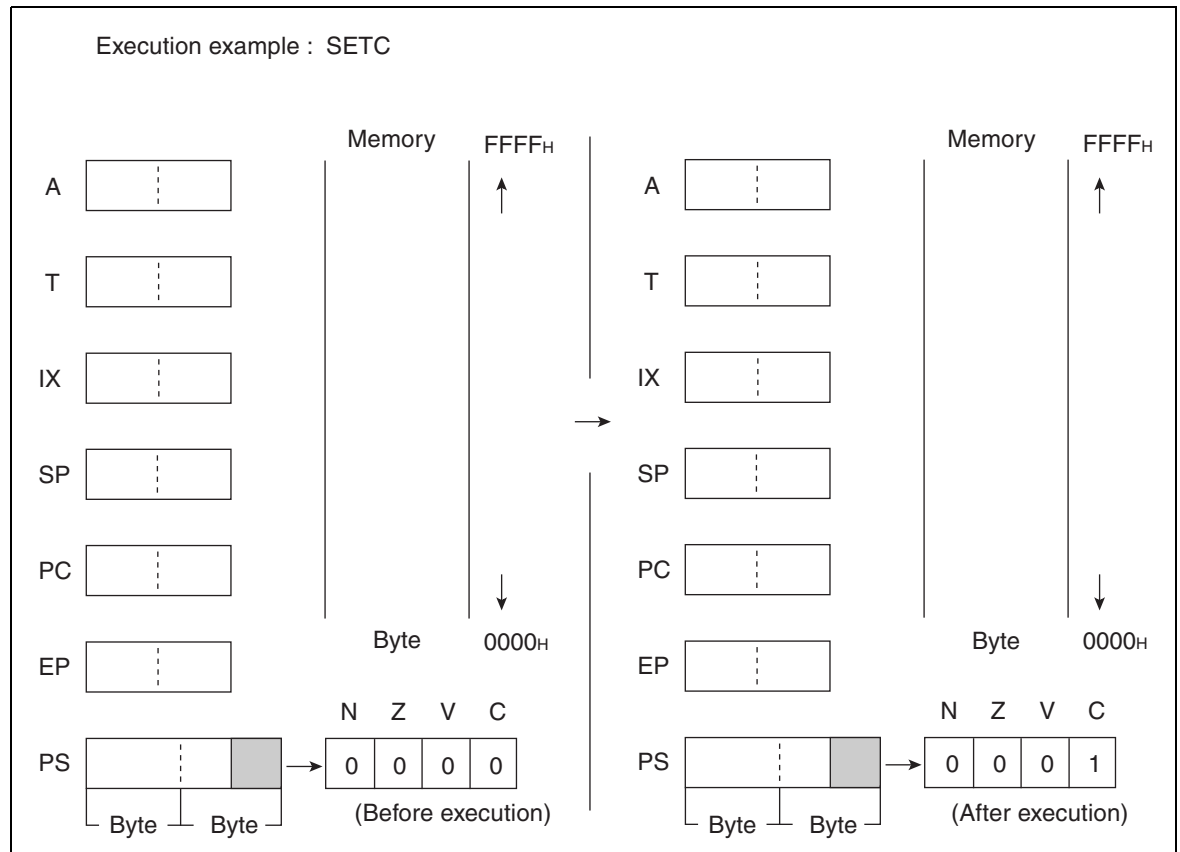
V: Not changed

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

$(I) \leftarrow 1$

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

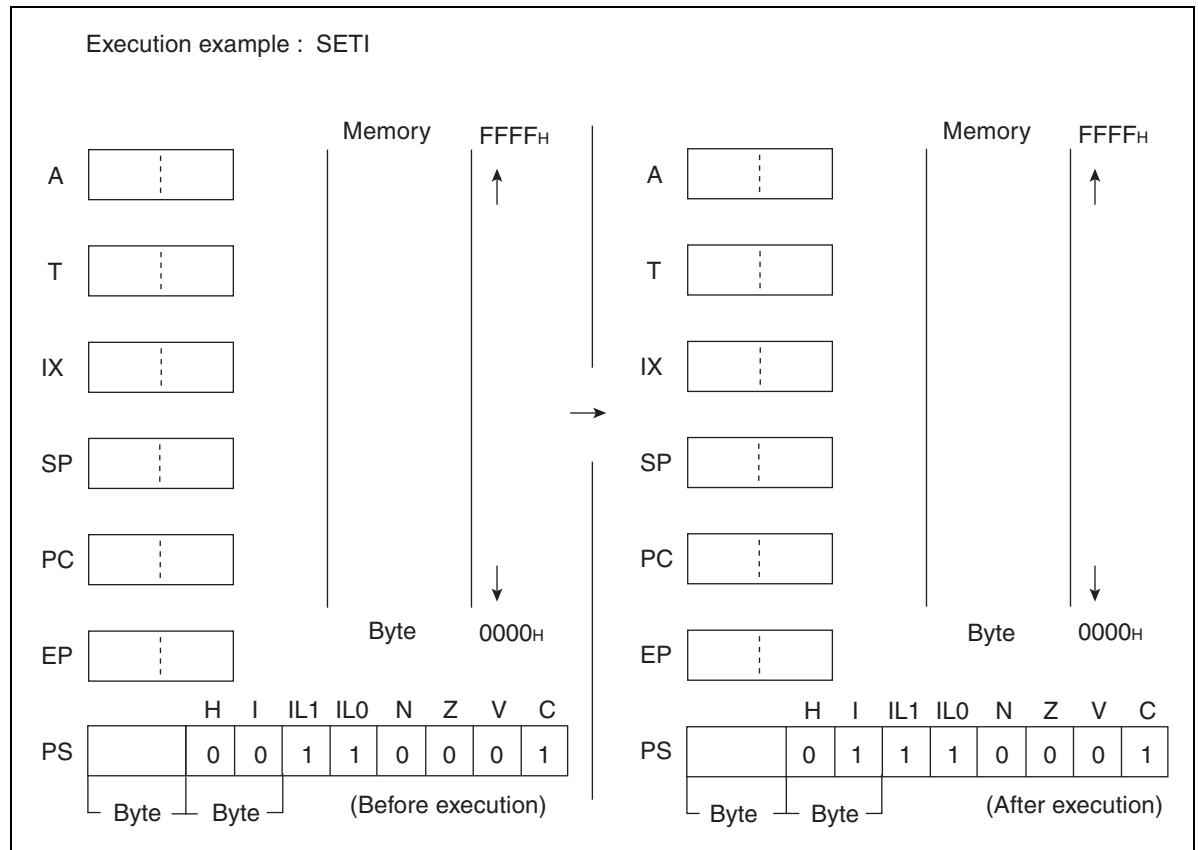
V: Not changed

C: Not changed

Number of execution cycle: 1

Byte count: 1

OP code: 90



## 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

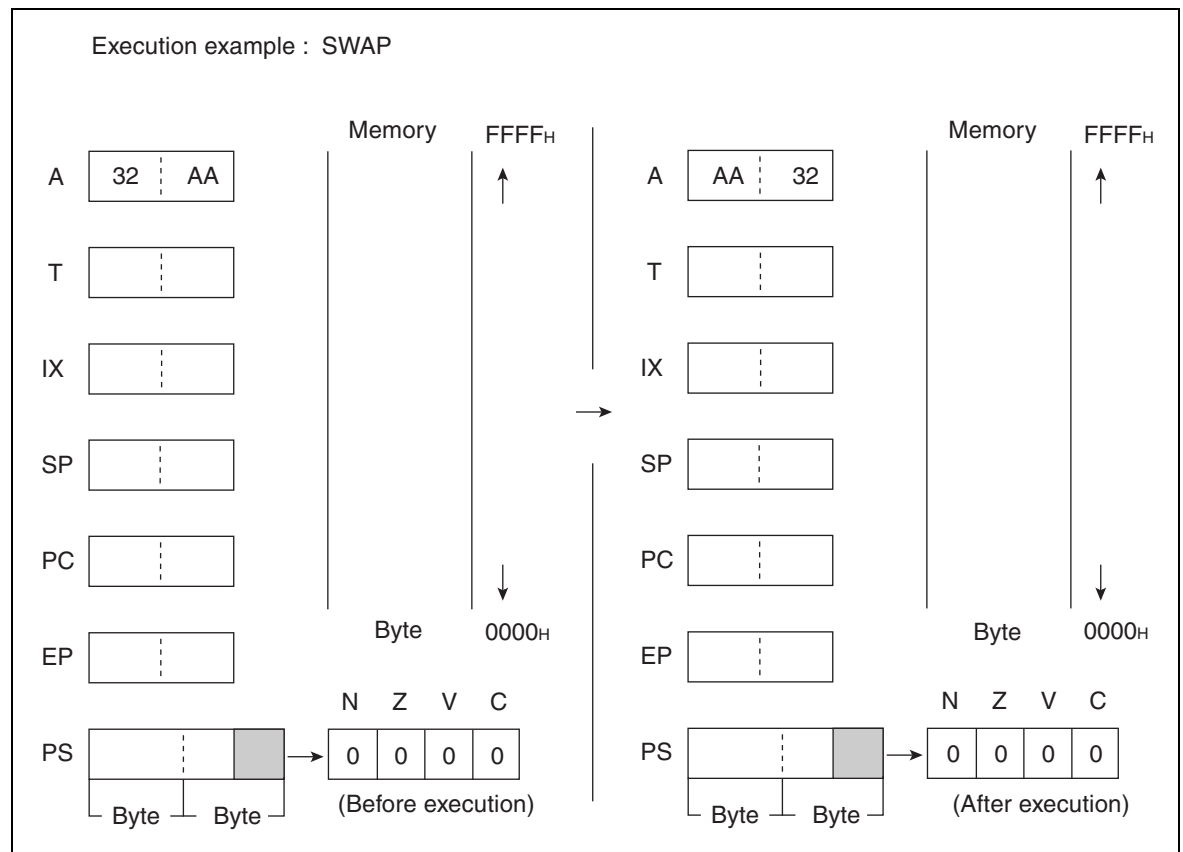
V: Not changed

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

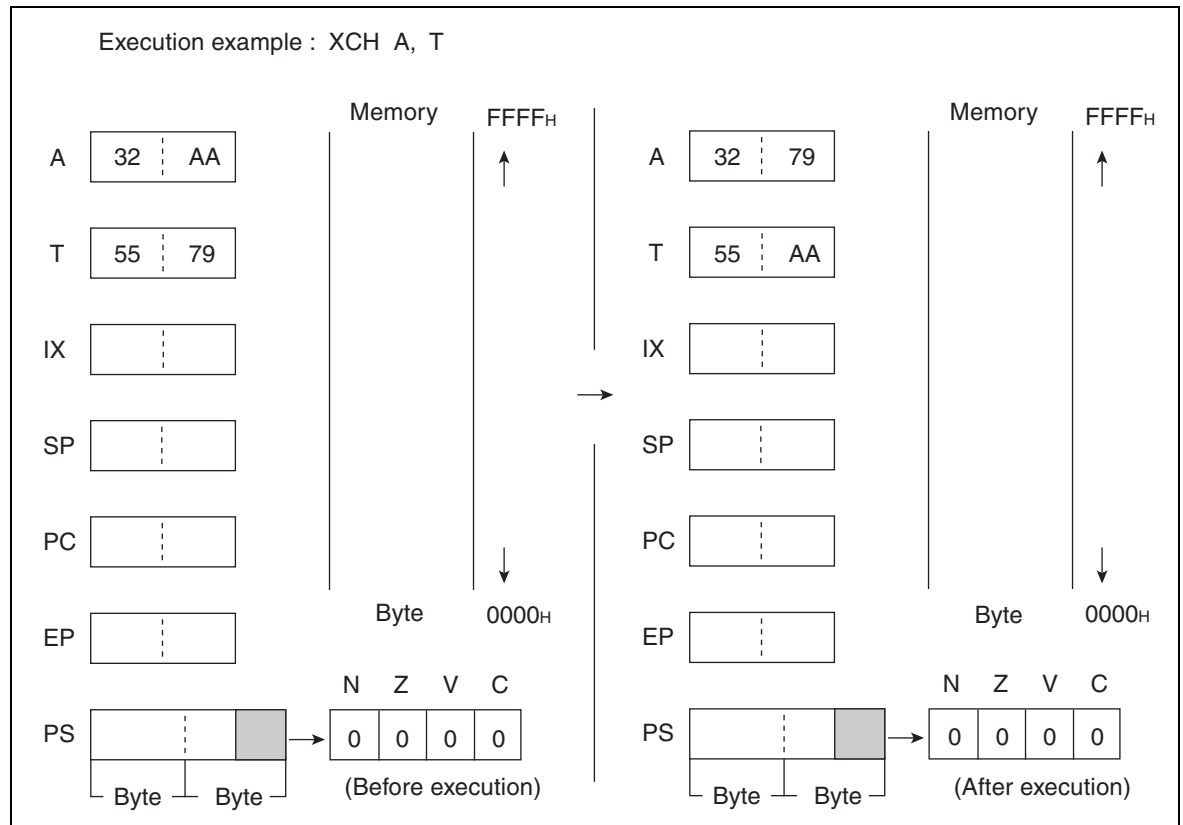
V: Not changed

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

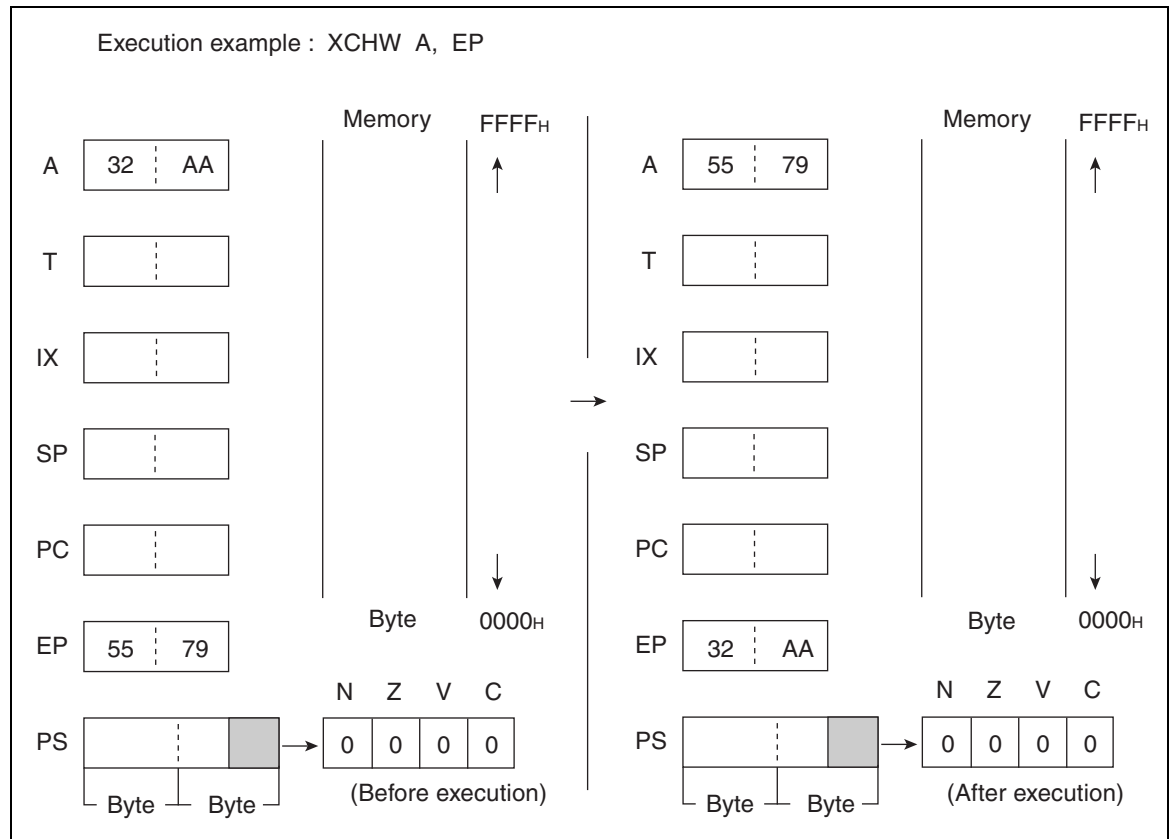
V: Not changed

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

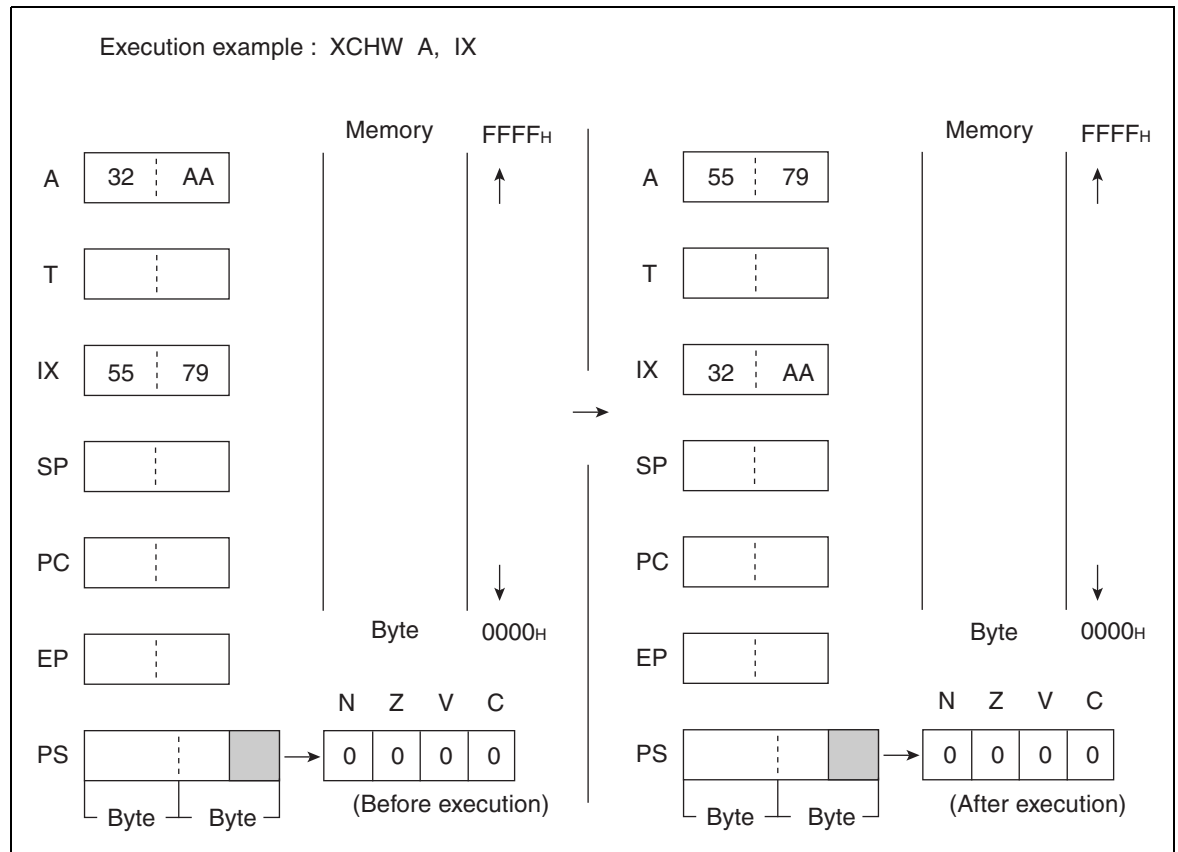
V: Not changed

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

$(PC) \leftarrow (A)$  (word transfer)

$(A) \leftarrow (PC) + 1$  (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

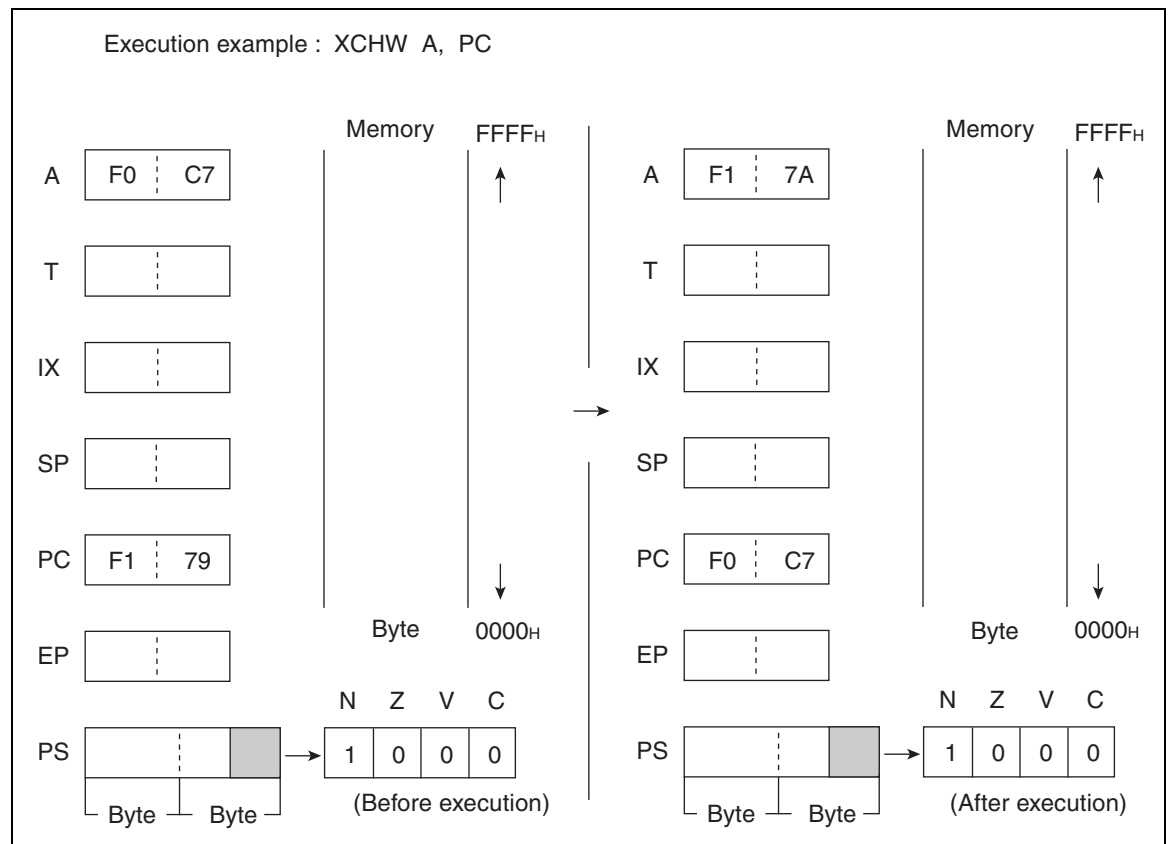
V: Not changed

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

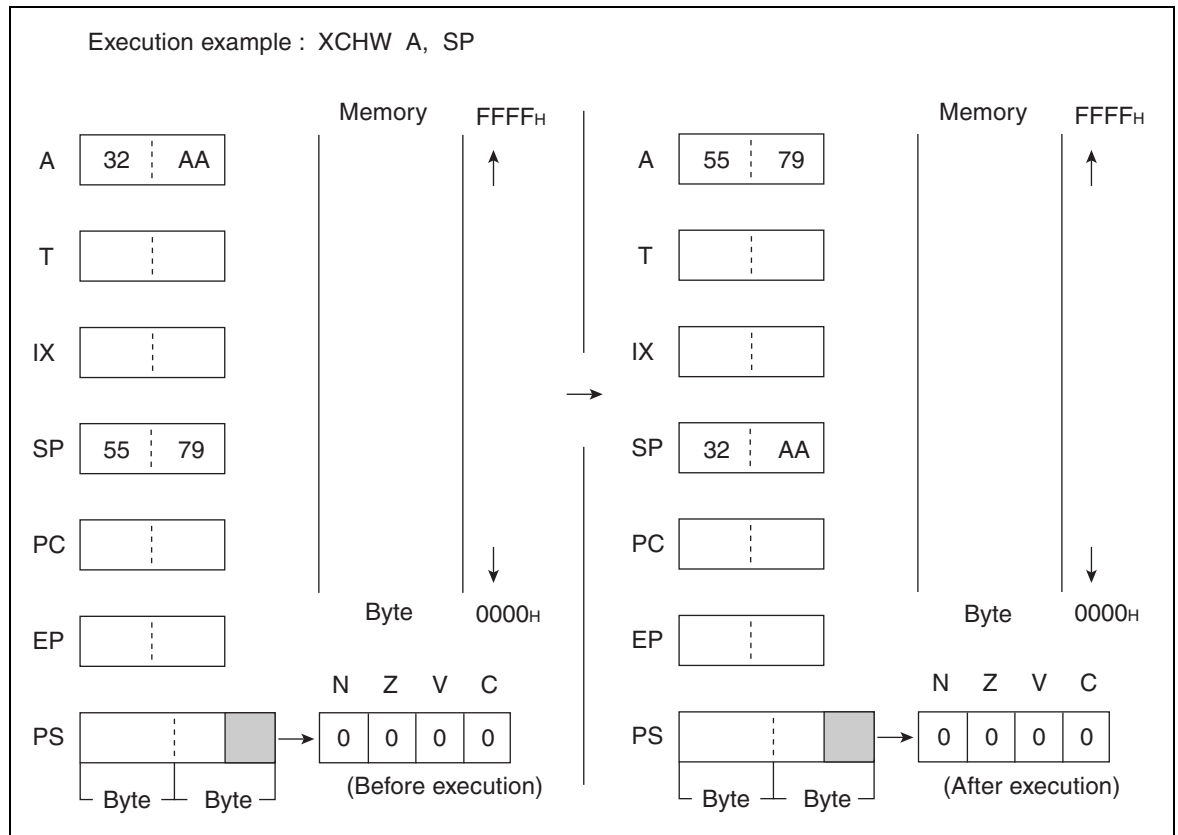
V: Not changed

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

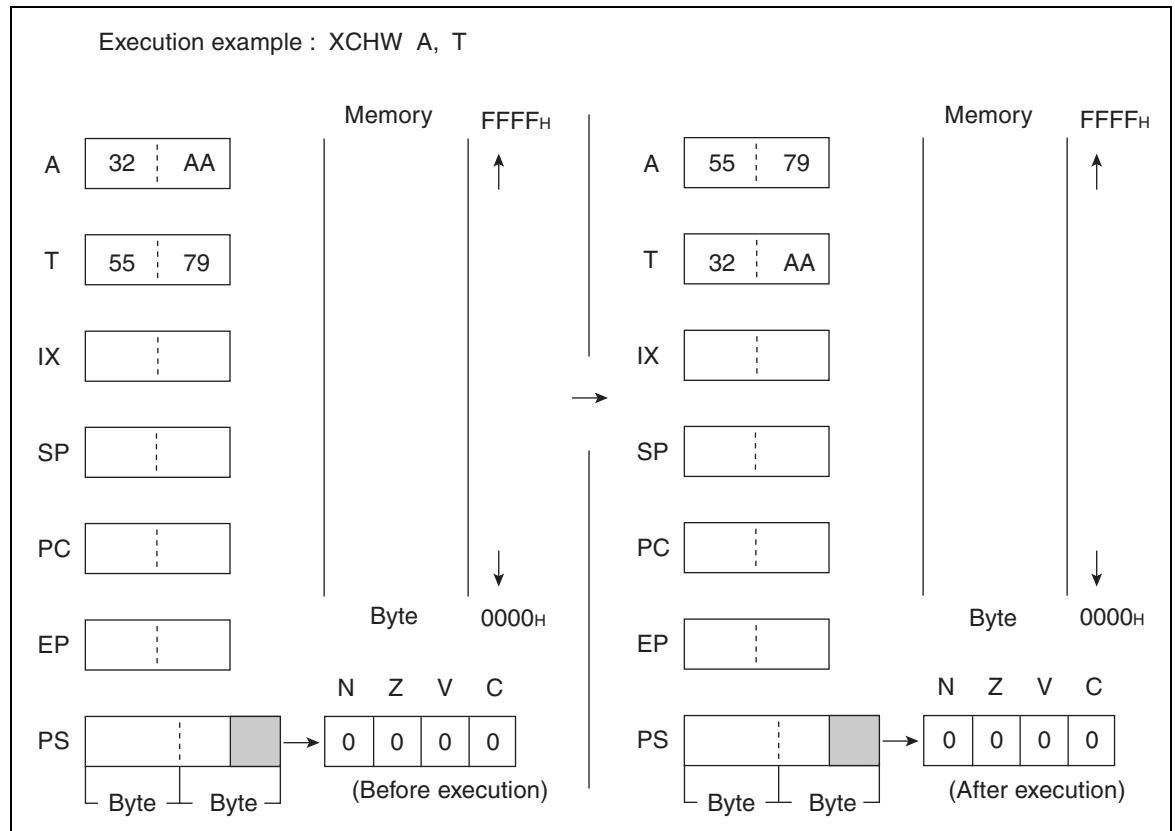
V: Not changed

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

$(AL) \leftarrow (AL) \vee (TL)$  (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<sub>H</sub> and set to 0 in other cases.

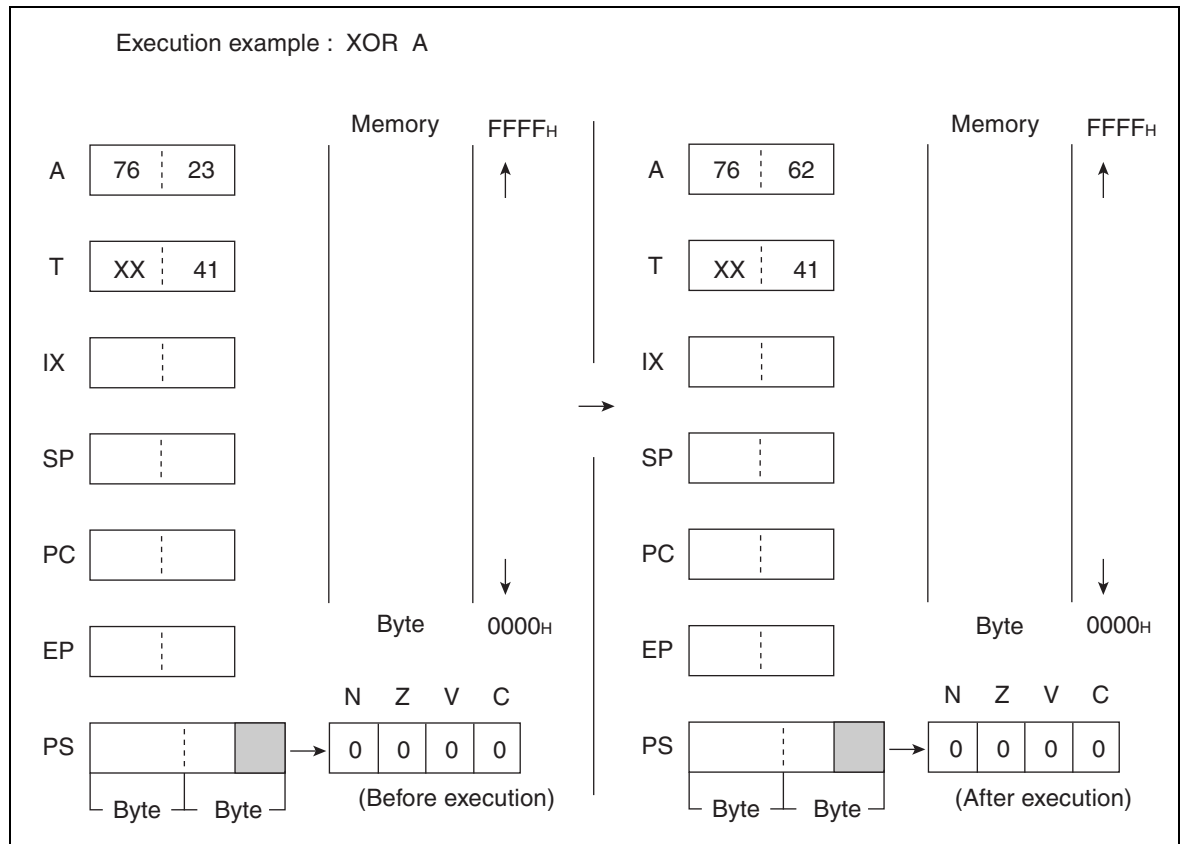
V: Always set to 0

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

$(AL) \leftarrow (AL) \vee (EA)$  (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.

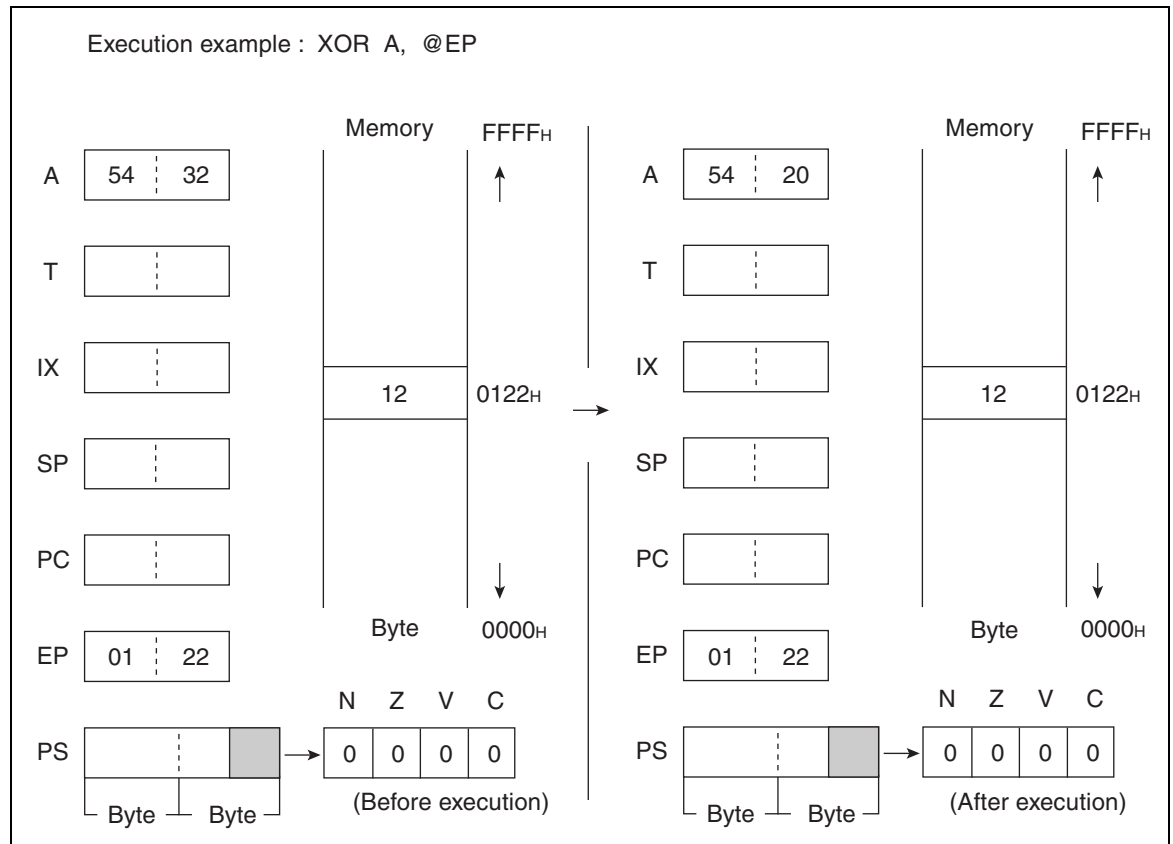
Z: Set to 1 if the result of operation is 00<sub>H</sub> and set to 0 in other cases.

V: Always set to 0

C: Not changed

Table 6-25. 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

$(A) \leftarrow (A) \vee (T)$  (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<sub>H</sub> 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



# Major Changes



Spansion Publication Number: CM26-00301-2E

| Page | Changes  |
|------|--|
| 11   | 2.2.2 Program Area<br>Table 2.2-2 CALLV Jump Address Table<br>( " FFC8 <sub>H</sub> " → " FFC9 <sub>H</sub> " )  |
| 53   | Execution example : ADDCW A<br>( NZVC = "1010" → NZVC = "0000" )   |
| 147  | Execution example : MOVW A, PC<br>( A = "F0 63" → A = "F0 62" )<br>( PC = "F0 63" → PC = "F0 62" )   |
| 176  | 6.65 PUSHW (PUSH Word Data of Inherent Register to Stack Memory)<br>( " Transfer the word value from the memory indicated by SP to dr. Then, subtract 2 from the value of SP. " →<br>" 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)<br>■ PUSHW (PUSH Word Data of Inherent Register to Stack Memory)<br>( "(SP) <-- (dr) (Word transfer) " → " (SP) ← (SP) - 2 (Word subtraction) " )<br>( " (SP) <-- (SP) - 2 (Word subtraction) " → " ((SP)) ← (dr) (Word transfer) " ) |
| 226  | A.2 Operation List<br>( "(IX)+off) <-- d8 " → " ((IX)+off) ← d8 " )  |
| 232  | Table A.2-4 Operation List (for Other Instructions)<br>( "(SP) ← (SP)-2, ((SP)) ← (A)<br>(A) ← ((SP)),<br>(SP) ← (SP)+2<br>(SP) ← (SP)-2,<br>((SP)) ← (IX)<br>(IX) ← ((SP)),<br>(SP) ← (SP)+2<br>No operation<br>(C) ← 0<br>(C) ← 1<br>(I) ← 0<br>(I) ← 1 " ) is added.                                |

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

# A. Instruction List



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

- A. Instruction List
- B. Bus Operation List
- C. Instruction Map



## APPENDIX A Instruction List

Appendix A contains lists of instructions used in the assembler.

A.1 F<sup>2</sup>MC-8FX CPU Instruction Overview

A.2 Operation List

A.3 Flag Change Table

## A.1 F<sup>2</sup>MC-8FX CPU Instruction Overview

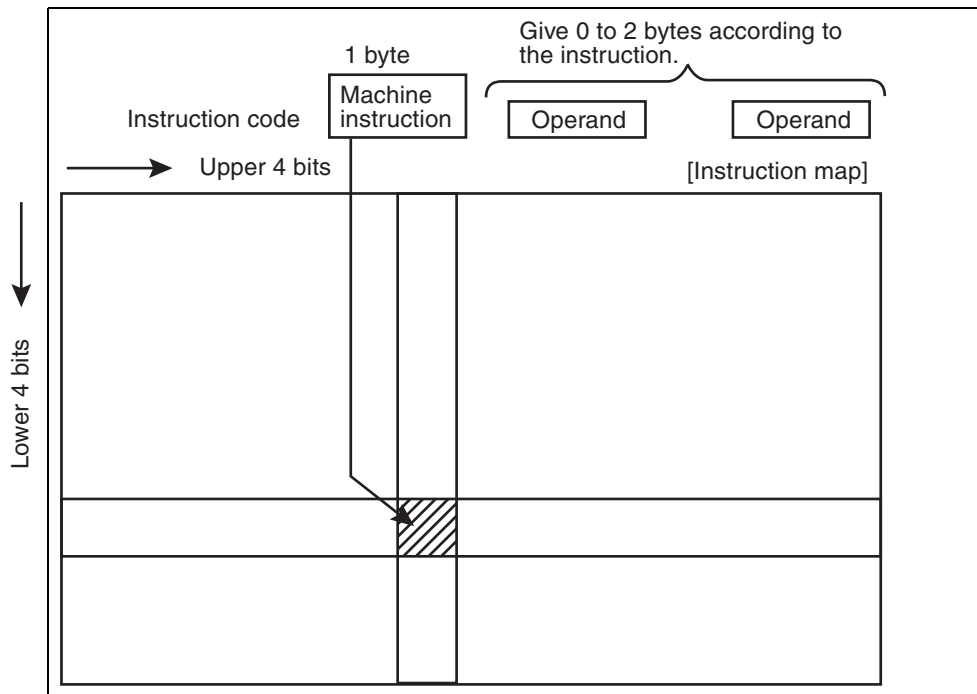
This section explains the F<sup>2</sup>MC-8FX CPU instructions.

### F<sup>2</sup>MC-8FX CPU Instruction Overview

In the F<sup>2</sup>MC-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-33. shows the instruction code and the correspondence of the instruction map.

Figure A-33. Correspondence between the Instruction Code and the Instruction Map



The following are enumerated as a feature of F<sup>2</sup>MC-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-7. explains the sign used by describing the instruction code in the table.

Table A-7. 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-8. explains the item of instruction table.

Table A-8. 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.  |
| ~          | <p>Cycle of the instruction number is shown. One instruction cycle is one machine cycle.</p> <p>Note:</p> <p>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.</p>   |
| #          | The number of bytes for the instruction is shown.  |
| Operation  | The operation of the instruction is shown.   |
| TL, TH, AH | <p>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.</p> <ul style="list-style-type: none"> <li>- : Do not change.</li> <li>dH : Upper 8 bits of the data written in operation</li> <li>AL, AH : Become the contents of AL or AH immediately before instruction.</li> <li>00 : Become 00.</li> </ul> |
| N, Z, V, C | <p>The flag changed when each instruction is executed is shown. The sign in the column shows the following respectively.</p> <ul style="list-style-type: none"> <li>- : Do not change.</li> <li>+ : Change.</li> <li>R : Become 0.</li> <li>S : Become 1.</li> </ul>   |
| OP CODE    | <p>The code of the instruction is shown. When a pertinent instruction occupies two or more codes, it follows the following description rules.</p> <p>48 to 4F: 48, 49, ..., 4F are shown.</p>  |

## A.2 Operation List

Table A-9. is the operation list for transfer instructions. Table A-10. is the operation list for operation instructions. Table A-11. is the operation list for branch instructions. Table A-12. is the operation list for other instructions.

### Operation List

Table A-9. Operation List (for Transfer Instructions) (Sheet 1 of 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),<br>(dir+1) ← (AL)           | —  | —  | —  | — — — — | D5       |
| 18 | MOVW @IX+off, A  | 4 | 2 | ((IX)+off) ← (AH),<br>((IX)+off+1) ← (AL) | —  | —  | —  | — — — — | D6       |

Table A-9. Operation List (for Transfer Instructions) (Sheet 2 of 3)

| No | MNEMONIC        | ~ | # | OPERATION                                 | TL | TH | AH | N Z V C | OP CODE |
|----|-----------------|---|---|---|----|----|----|---------|---------|
| 19 | MOVW ext, A     | 5 | 3 | (ext) ← (AH),<br>(ext+1) ← (AL)           | —  | —  | —  | — — — — | D4      |
| 20 | MOVW @EP, A     | 3 | 1 | ((EP)) ← (AH),<br>((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),<br>(AL) ← (dir+1)           | AL | AH | dH | + + — — | C5      |
| 24 | MOVW A, @IX+off | 4 | 2 | (AH) ← ((IX)+off),<br>(AL) ← ((IX)+off+1) | AL | AH | dH | + + — — | C6      |
| 25 | MOVW A, ext     | 5 | 3 | (AH) ← (ext),<br>(AL) ← (ext+1)           | AL | AH | dH | + + — — | C4      |
| 26 | MOVW A, @A      | 3 | 1 | (AH) ← ((A)),<br>(AL) ← ((A)+1)           | AL | AH | dH | + + — — | 93      |
| 27 | MOVW A, @EP     | 3 | 1 | (AH) ← ((EP)),<br>(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),<br>((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      |

Table A-9. Operation List (for Transfer Instructions) (Sheet 3 of 3)

| No | MNEMONIC      | ~ | # | OPERATION   | TL | TH | AH | N Z V C | OP CODE  |
|----|---------------|---|---|-------------|----|----|----|---------|----------|
| 39 | MOVW SP, #d16 | 3 | 3 | (SP) ← d16  | —  | —  | —  | — — — — | E5       |
| 40 | SWAP          | 1 | 1 | (AH) ↔ (AL) | —  | —  | AL | — — — — | 10       |
| 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:

In byte transfer to A, T ← A is only low bytes.

The operands of an instruction with two or more operands should be stored in the order designated in MNEMONIC.

Table A-10. Operation List (for Operation Instructions) (Sheet 1 of 4)

| 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       |

Table A-10. Operation List (for Operation Instructions) (Sheet 2 of 4)

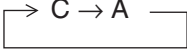
| No | MNEMONIC        | ~  | # | OPERATION   | TL | TH | AH | NZVC | OP CODE  |
|----|-----------------|----|---|---|----|----|----|------|----------|
| 10 | SUBC A, dir     | 3  | 2 | $(A) \leftarrow (A) - (\text{dir}) - C$   | —  | —  | —  | ++++ | 35       |
| 11 | SUBC A, @IX+off | 3  | 2 | $(A) \leftarrow (A) - ((IX) + \text{off}) - C$                                      | —  | —  | —  | ++++ | 36       |
| 12 | SUBC A, @EP     | 2  | 1 | $(A) \leftarrow (A) - (EP) - C$   | —  | —  | —  | ++++ | 37       |
| 13 | SUBCW A         | 1  | 1 | $(A) \leftarrow (T) - (A) - C$  | —  | —  | dH | ++++ | 33       |
| 14 | SUBC A          | 1  | 1 | $(AL) \leftarrow (TL) - (AL) - C$   | —  | —  | —  | ++++ | 32       |
| 15 | IINC Ri         | 3  | 1 | $(Ri) \leftarrow (Ri) + 1$  | —  | —  | —  | +++- | C8 to CF |
| 16 | INCW EP         | 1  | 1 | $(EP) \leftarrow (EP) + 1$  | —  | —  | —  | ---- | C3       |
| 17 | INCW IX         | 1  | 1 | $(IX) \leftarrow (IX) + 1$  | —  | —  | —  | ---- | C2       |
| 18 | INCW A          | 1  | 1 | $(A) \leftarrow (A) + 1$  | —  | —  | dH | ++-- | C0       |
| 19 | DEC Ri          | 3  | 1 | $(Ri) \leftarrow (Ri) - 1$  | —  | —  | —  | +++- | D8 to DF |
| 20 | DECW EP         | 1  | 1 | $(EP) \leftarrow (EP) - 1$  | —  | —  | —  | ---- | D3       |
| 21 | DECW IX         | 1  | 1 | $(IX) \leftarrow (IX) - 1$  | —  | —  | —  | ---- | D2       |
| 22 | DECW A          | 1  | 1 | $(A) \leftarrow (A) - 1$  | —  | —  | dH | ++-- | D0       |
| 23 | MULU A          | 8  | 1 | $(A) \leftarrow (AL) * (TL)$  | —  | —  | dH | ---- | 01       |
| 24 | DIVU A          | 17 | 1 | $(A) \leftarrow (T) / (A),$<br>$MOD \rightarrow (T)$                                | dL | dH | dH | -+-- | 11       |
| 25 | ANDW A          | 1  | 1 | $(A) \leftarrow (A) \wedge (T)$   | —  | —  | dH | ++R- | 63       |
| 26 | ORW A           | 1  | 1 | $(A) \leftarrow (A) \vee (T)$   | —  | —  | dH | ++R- | 73       |
| 27 | XORW A          | 1  | 1 | $(A) \leftarrow (A) \vee (T)$   | —  | —  | 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       |



Table A-10. Operation List (for Operation Instructions) (Sheet 3 of 4)

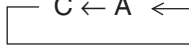
| No | MNEMONIC       | ~ | # | OPERATION   | TL | TH | AH | NZVC | OP CODE  |
|----|----------------|---|---|---|----|----|----|------|----------|
| 31 | ROLCA          | 1 | 1 |  | —  | —  | —  | ++—+ | 02       |
| 32 | CMP A, #d8     | 2 | 2 | (A)- d8   | —  | —  | —  | ++++ | 14       |
| 33 | CMP 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       |
| 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     | 2 | 1 | (A) ← (AL) ∨ ((EP))   | —  | —  | —  | ++R— | 57       |
| 43 | XOR A, @IX+off | 3 | 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       |

Table A-10. Operation List (for Operation Instructions) (Sheet 4 of 4)

| No | MNEMONIC            | ~ | # | OPERATION                                    | TL | TH | AH | NZVC | OP CODE  |
|----|---------------------|---|---|--|----|----|----|------|----------|
| 53 | OR A, dir           | 3 | 2 | $(A) \leftarrow (AL) \vee (\text{dir})$      | —  | —  | —  | ++R— | 75       |
| 54 | OR A,@EP            | 2 | 1 | $(A) \leftarrow (AL) \vee ((EP))$            | —  | —  | —  | ++R— | 77       |
| 55 | OR A,<br>@IX,off    | 3 | 2 | $(A) \leftarrow (AL) \vee ((IX)+\text{off})$ | —  | —  | —  | ++R— | 76       |
| 56 | OR A, Ri            | 2 | 1 | $(A) \leftarrow (AL) \vee (Ri)$              | —  | —  | —  | ++R— | 78 to 7F |
| 57 | CMP dir, #d8        | 4 | 3 | $(\text{dir}) - d8$                          | —  | —  | —  | ++++ | 95       |
| 58 | CMP @EP,<br>#d8     | 3 | 2 | $((EP)) - d8$                                | —  | —  | —  | ++++ | 97       |
| 59 | CMP<br>@IX+off, #d8 | 4 | 3 | $((IX)+\text{off}) - d8$                     | —  | —  | —  | ++++ | 96       |
| 60 | CMP Ri, #d8         | 3 | 2 | $(Ri) - d8$                                  | —  | —  | —  | ++++ | 98 to 9F |
| 61 | INCW SP             | 1 | 1 | $(SP) \leftarrow (SP) + 1$                   | —  | —  | —  | ---- | C1       |
| 62 | DECW SP             | 1 | 1 | $(SP) \leftarrow (SP) - 1$                   | —  | —  | —  | ---- | D1       |

Table A-11. Operation List (for Branch Instructions)

| No | MNEMONIC          |                      | ~ | # | OPERATION                        | TL | TH | A<br>H | NZVC    | OP<br>CODE  |
|----|-------------------|----------------------|---|---|----------------------------------|----|----|--------|---------|-------------|
| 1  | BZ/BEQ<br>rel     | (diver-<br>gence)    | 4 | 2 | if Z=1 then PC ←<br>PC+rel       | -  | -  | -      | -----   | FD          |
|    |                   | (no diver-<br>gence) | 2 |   |                                  |    |    |        |         |             |
| 2  | BNZ/<br>BNE rel   | (diver-<br>gence)    | 4 | 2 | if Z=0 then PC ←<br>PC+rel       | -  | -  | -      | -----   | FC          |
|    |                   | (no diver-<br>gence) | 2 |   |                                  |    |    |        |         |             |
| 3  | BC/BLO<br>rel     | (diver-<br>gence)    | 4 | 2 | if C=1 then PC ←<br>PC+rel       | -  | -  | -      | -----   | F9          |
|    |                   | (no diver-<br>gence) | 2 |   |                                  |    |    |        |         |             |
| 4  | BNC/<br>BHS rel   | (diver-<br>gence)    | 4 | 2 | if C=0 then PC ←<br>PC+rel       | -  | -  | -      | -----   | F8          |
|    |                   | (no diver-<br>gence) | 2 |   |                                  |    |    |        |         |             |
| 5  | BN rel            | (diver-<br>gence)    | 4 | 2 | if N=1 then PC ←<br>PC+rel       | -  | -  | -      | -----   | FB          |
|    |                   | (no diver-<br>gence) | 2 |   |                                  |    |    |        |         |             |
| 6  | BP rel            | (diver-<br>gence)    | 4 | 2 | if N=0 then PC ←<br>PC+rel       | -  | -  | -      | -----   | FA          |
|    |                   | (no diver-<br>gence) | 2 |   |                                  |    |    |        |         |             |
| 7  | BLT rel           | (diver-<br>gence)    | 4 | 2 | if V ∨ N=1 then<br>PC ← PC+rel   | -  | -  | -      | -----   | FF          |
|    |                   | (no diver-<br>gence) | 2 |   |                                  |    |    |        |         |             |
| 8  | BGE rel           | (diver-<br>gence)    | 4 | 2 | if V ∨ N=0 then<br>PC ← PC+rel   | -  | -  | -      | -----   | FE          |
|    |                   | (no diver-<br>gence) | 2 |   |                                  |    |    |        |         |             |
| 9  | BBC<br>dir:b, rel |                      | 5 | 3 | if (dir:b)=0 then<br>PC ← PC+rel | -  | -  | -      | - + - - | B0 to<br>B7 |
| 10 | BBS<br>dir:b, rel |                      | 5 | 3 | if (dir:b)=1 then<br>PC ← PC+rel | -  | -  | -      | - + - - | B8 to<br>BF |
| 11 | JMP @A            |                      | 3 | 1 | (PC) ← (A)                       | -  | -  | -      | -----   | E0          |
| 12 | JMP ext           |                      | 4 | 3 | (PC) ← ext                       | -  | -  | -      | -----   | 21          |
| 13 | CALLV<br>#vct     |                      | 7 | 1 | vector call                      | -  | -  | -      | -----   | E8 to<br>EF |
| 14 | CALL ext          |                      | 6 | 3 | subroutine call                  | -  | -  | -      | -----   | 31          |

Table A-11. Operation List (for Branch Instructions) (*continued*)

| No | MNEMONIC |       | ~ | # | OPERATION   | TL | TH | A<br>H | NZVC    | OP<br>CODE |
|----|----------|-------|---|---|---|----|----|--------|---------|------------|
| 15 | XCHW     | A, PC | 3 | 1 | $(PC) \leftarrow (A),$<br>$(A) \leftarrow (PC)+1$ | —  | —  | dH     | — — — — | F4         |
| 16 | RET      |       | 6 | 1 | return from subroutine                            | —  | —  | —      | — — — — | 20         |
| 17 | RETI     |       | 8 | 1 | return from interrupt                             | —  | —  | —      | restore | 30         |

Table A-12. Operation List (for Other Instructions)

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

## A.3 Flag Change Table

Table A-13. is the flag change table for transfer instructions. Table A-14. is the flag change table for operation instructions. Table A-15. is the flag change table for branch instructions. Table A-16. is the flag change table for other instructions.

Flag Change Table

Table A-13. Flag Change Table (for Transfer Instructions) (Sheet 1 of 3)

| Instruction      | Flag change  |
|------------------|--|
| MOV dir, A       | N: Not changed   |
| MOV @IX+off, A   | Z: Not changed   |
| MOV ext, A       | V: Not changed   |
| MOV @EP, A       | C: Not changed   |
| MOV Ri, A        |  |
| MOV , #d8        | N: Set to 1 if the transferred data is negative and set to 0 in other cases. |
| MOV A, dir       | Z: Set to 1 if the transferred data is 0 and set to 0 in other cases         |
| MOV A, @IX+off   | V: Not changed   |
| MOV A, ext       | C: Not changed   |
| MOV A, @A        |  |
| MOV A, @EP       |  |
| MOV A, Ri        |  |
| MOV dir, #d8     | N: Not changed   |
| MOV @IX+off, #d8 | Z: Not changed   |
| MOV @EP, #d8     | V: Not changed   |
| MOV Ri, #d8      | C: Not changed   |
| MOVW dir, A      | N: Not changed   |
| MOVW @IX+off, A  | Z: Not changed   |
| MOVW ext, A      | V: Not changed   |
| MOVW @EP, A      | C: Not changed   |
| MOVW A, #d16     | N: Set to 1 if the transferred data is negative and set to 0 in other cases. |
| MOVW A, dir      | Z: Set to 1 if the transferred data is 0 and set to 0 in other cases         |
| MOVW A, @IX+off  | V: Not changed   |
| MOVW A, ext      | C: Not changed   |
| MOVW A, @A       |  |
| MOVW A, @EP      |  |

Table A-13. Flag Change Table (for Transfer Instructions) (Sheet 2 of 3)

| Instruction  | Flag change  |
|--|--|
| MOVW A, EP<br>MOVW EP, #d16<br>MOVW IX, A<br>MOVW A, IX<br>MOVW SP, A<br>MOVW A, SP<br>MOVW SP, #d16 | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed   |
| MOV @A, T<br>MOVW @A, T  | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed   |
| MOVW IX, #d16<br>MOVW A, PS<br>MOVW A, PC<br>JMP @A  | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed   |
| MOVW PS, A   | N: Set to 1 if bit 3 of A is 1 and set to 0 if 0.<br>Z: Set to 1 if bit 2 of A is 1 and set to 0 if 0.<br>V: Set to 1 if bit 1 of A is 1 and set to 0 if 0.<br>C: Set to 1 if bit 0 of A is 1 and set to 0 if 0. |
| SETB dir:b<br>CLRB dir:b   | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed   |
| SWAP<br>XCH A, T   | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed   |
| XCHW A, T<br>XCHW A, EP<br>XCHW A, IX<br>XCHW A, SP  | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed   |

Table A-13. Flag Change Table (for Transfer Instructions) (Sheet 3 of 3)

| Instruction | Flag change |
|-------------|-------------|
| XCHW A, PC  |             |

Table A-14. Flag Change Table (for Operation Instructions) (Sheet 1 of 4)

| Instruction  | Flag change  |
|--|--|
| ADDC A, Ri<br>ADDC A, #d8<br>ADDC A, dir<br>ADDC A, @IX+off<br>ADDC A, @EP | N: Set to 1 if the result of operation is negative and set to 0 in other cases.<br>Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.<br>V: Set to 1 if an overflow occurs and set to 0 in other cases.<br>C: Set to 1 if a carry occurs and set to 0 in other cases.  |
| ADDC A<br>ADDCW A  | N: Set to 1 if the result of operation is negative and set to 0 in other cases.<br>Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.<br>V: Set to 1 if an overflow occurs and set to 0 in other cases.<br>C: Set to 1 if a carry occurs and set to 0 in other cases.  |
| SUBC A, Ri<br>SUBC A, #d8<br>SUBC A, dir<br>SUBC A, @IX+off<br>SUBC A, @EP | N: Set to 1 if the result of operation is negative and set to 0 in other cases.<br>Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.<br>V: Set to 1 if an overflow occurs and set to 0 in other cases.<br>C: Set to 1 if a borrow occurs and set to 0 in other cases. |
| SUBC A<br>SUBCW A  | N: Set to 1 if the result of operation is negative and set to 0 in other cases.<br>Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.<br>V: Set to 1 if an overflow occurs and set to 0 in other cases.<br>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.<br>Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.<br>V: Set to 1 if an overflow occurs and set to 0 in other cases.<br>C: Not changed  |
| INCW EP<br>INCW IX<br>INCW SP  | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed   |
| INCW A   | N: Set to 1 if the result of operation is negative and set to 0 in other cases.<br>Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.  |



Table A-14. Flag Change Table (for Operation Instructions) (Sheet 2 of 4)

| Instruction   | Flag change   |
|---|---|
|   | V: Not changed<br>C: Not changed  |
| DEC Ri  | N: Set to 1 if the result of operation is negative and set to 0 in other cases.<br>Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.<br>V: Set to 1 if an overflow occurs and set to 0 in other cases.<br>C: Not changed |
| DECW EP<br>DECW IX<br>DECW SP   | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed  |
| DECW A  | N: Set to 1 if the result of operation is negative and set to 0 in other cases.<br>Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.<br>V: Not changed<br>C: Not changed   |
| MULU A  | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed  |
| DIVU A  | N: Not changed<br>Z: Set to 1 if A before operation is 0000 <sub>H</sub> and set to 0 in other cases.<br>V: Not changed<br>C: Not changed   |
| ANDW A  | N: Set to 1 if the result of operation is negative and set to 0 in other cases.<br>Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.<br>V: Always Set to 0<br>C: Not changed   |
| AND A, #d8<br>AND A, dir<br>AND A, @EP<br>AND A, @IX+off<br>AND A, Ri | N: Set to 1 if the result of operation is negative and set to 0 in other cases.<br>Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.<br>V: Always set to 0<br>C: Not changed   |

Table A-14. Flag Change Table (for Operation Instructions) (Sheet 3 of 4)

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

Table A-14. Flag Change Table (for Operation Instructions) (Sheet 4 of 4)

| Instruction  | Flag change  |
|--|--|
| CMP dir, #d8<br>CMP @EP #d8<br>CMP @IX+off, #d8<br>CMP Ri, #d8 | N: Set to 1 if the result of operation is negative and set to 0 in other cases.<br>Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.<br>V: Set to 1 if an overflow occurs and set to 0 in other cases.<br>C: Set to 1 if a borrow occurs and set to 0 in other cases.         |
| RORC A<br>ROLC A   | N: Set to 1 if the result of operation is negative and set to 0 in other cases.<br>Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.<br>V: Not changed<br>C: Enter bit 0 (when RORA) or bit 7 (when ROLA) of A before the operation.  |
| DAA<br>DAS   | N: Set to 1 if the result of operation is negative and set to 0 in other cases.<br>Z: Set to 1 if the result of operation is 0 and set to 0 in other cases.<br>V: Set to 1 if an overflow occurs and set to 0 in other cases.<br>C: Set to 1 if a carry (borrow) occurs and set to 0 in other cases. |

Table A-15. Flag Change Table (for Branch Instructions)

| Instruction  | Flag change  |
|--|--|
| BZ rel/BEQ rel<br>BNZ rel/BNE rel<br>BC rel/BLO rel<br>BNC rel/BHS rel<br>BN rel<br>BP rel<br>BLT rel<br>BGE rel | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed                               |
| JMP addr16   | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed                               |
| BBC dir:b, rel<br>BBS dir:b, rel   | N: Not changed<br>Z: Set to 1 if bit b is 0 and set to 0 if 1.<br>V: Not changed<br>C: Not changed |

Table A-15. Flag Change Table (for Branch Instructions) (*continued*)

| Instruction                      | Flag change  |
|----------------------------------|--|
| CALL addr16<br>CALLV #vct<br>RET | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed   |
| RETI                             | N: N value of saved CCR is entered.<br>Z: Z value of saved CCR is entered.<br>V: V value of saved CCR is entered.<br>C: C value of saved CCR is entered. |

Table A-16. Flag Change Table (for Other Instructions)

| Instruction         | Flag change  |
|---------------------|--|
| PUSHW A<br>PUSHW IX | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed |
| POPW A<br>POPW IX   | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed |
| NOP                 | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed |
| CLRC                | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Become to 0 |
| SETC                | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Become to 1 |

Table A-16. Flag Change Table (for Other Instructions) (*continued*)

| Instruction | Flag change  |
|-------------|--|
| CLRI        | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed<br>I: Become to 0 |
| SETI        | N: Not changed<br>Z: Not changed<br>V: Not changed<br>C: Not changed<br>I: Become to 1 |

## B. Bus Operation List



Table B-1. is a bus operation list.

### Bus Operation ListK

Table B-1. Bus Operation List (Sheet 1 of 15)

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

Table B-1. Bus Operation List (Sheet 2 of 15)

| CODE | MNEMONIC    | ~ | Cycle | Address bus | Data bus                            | RD | WR | RMW |
|------|-------------|---|-------|-------------|-------------------------------------|----|----|-----|
| 04   | MOV A, #d8  | 2 | 1     | N +2        | The following instruction           | 1  | 0  | 0   |
| 14   | CMP A, #d8  |   | 2     | N +3        | The following following instruction | 1  | 0  | 0   |
| 24   | ADDC A, #d8 |   |       |             |                                     |    |    |     |
| 34   | SUBC A, #d8 |   |       |             |                                     |    |    |     |
| 54   | XOR A, #d8  |   |       |             |                                     |    |    |     |
| 64   | AND A, #d8  |   |       |             |                                     |    |    |     |
| 74   | OR A, #d8   |   |       |             |                                     |    |    |     |

Table B-1. Bus Operation List (Sheet 3 of 15)

| 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 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   |



Table B-1. Bus Operation List (Sheet 4 of 15)

| CODE    | MNEMONIC   | ~ | Cycle | Address bus  | Data bus                            | RD | WR | RMW |
|---------|------------|---|-------|--------------|-------------------------------------|----|----|-----|
| 07      | MOV A, @EP | 2 | 1     | N +2         | The following following instruction | 1  | 0  | 0   |
| 17      | CMP A,     |   | 2     | (EP)         | Data                                | 1  | 0  | 0   |
| 27      | ADDC A,    |   |       |              |                                     |    |    |     |
| 37      | SUBC A,    |   |       |              |                                     |    |    |     |
| 57      | XOR A,     |   |       |              |                                     |    |    |     |
| 67      | AND A,     |   |       |              |                                     |    |    |     |
| 77      | OR A,      |   |       |              |                                     |    |    |     |
| 47      | MOV @EP, A | 2 | 1     | N +2         | The following following instruction | 1  | 0  | 0   |
|         |            |   | 2     | (EP) address | Data                                | 0  | 1  | 0   |
| 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    |   |       |              |                                     |    |    |     |

Table B-1. Bus Operation List (Sheet 5 of 15)

| CODE | MNEMONIC   | ~ | Cycle | Address bus           | Data bus                            | RD | WR | RMW |
|------|------------|---|-------|-----------------------|-------------------------------------|----|----|-----|
| 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 (Sheet 6 of 15)

| 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  |    |     |

Table B-1. Bus Operation List (Sheet 7 of 15)

| CODE | MNEMONIC    | ~ | Cycle | Address bus   | Data bus                            | RD | WR | RMW |
|------|-------------|---|-------|---------------|-------------------------------------|----|----|-----|
| 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   |
| 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   |

Table B-1. Bus Operation List (Sheet 8 of 15)

| CODE | MNEMONIC        | ~ | Cycle | Address bus        | Data bus                            | RD | WR | RMW |
|------|-----------------|---|-------|--------------------|-------------------------------------|----|----|-----|
| 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   |
| 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   |

Table B-1. Bus Operation List (Sheet 9 of 15)

| CODE | MNEMONIC         | ~ | Cycle | Address bus      | Data bus                            | RD | WR | RMW |
|------|------------------|---|-------|------------------|-------------------------------------|----|----|-----|
| 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   |
| 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   |

Table B-1. Bus Operation List (Sheet 10 of 15)

| CODE    | MNEMONIC      | ~ | Cycle | Address bus    | Data bus                            | RD | WR | RMW |
|---------|---------------|---|-------|----------------|-------------------------------------|----|----|-----|
| 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   |
| 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 |   |       |                |                                     |    |    |     |

Table B-1. Bus Operation List (Sheet 11 of 15)

| CODE    | MNEMONIC   | ~ | Cycle | Address bus                 | Data bus                            | RD | WR | RMW |
|---------|------------|---|-------|-----------------------------|-------------------------------------|----|----|-----|
| 84      | DAA        | 1 | 1     | N +2                        | The following following instruction | 1  | 0  | 0   |
| 94      | DAS        |   |       |                             |                                     |    |    |     |
| 02      | ROLCA      |   |       |                             |                                     |    |    |     |
| 03      | RORCA      |   |       |                             |                                     |    |    |     |
| 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 (Sheet 12 of 15)

| CODE | MNEMONIC | ~              | Cycle | Address bus                 | Data bus                            | RD | WR | RMW |
|------|----------|----------------|-------|-----------------------------|-------------------------------------|----|----|-----|
| F8   | BNC rel  | 4              |       | Diver-gence                 |                                     |    |    |     |
| F9   | BC rel   |                | 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  | No diver-gence | 4     | Address divergence ahead +1 | The following following instruction | 1  | 0  | 0   |
| FD   | BZ rel   |                |       |                             |                                     |    |    |     |
| FE   | BGE rel  |                | 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   |

Table B-1. Bus Operation List (Sheet 13 of 15)

| CODE | MNEMONIC | ~ | Cycle | Address bus                 | Data bus                            | RD | WR | RMW |
|------|----------|---|-------|-----------------------------|-------------------------------------|----|----|-----|
| 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   |
| 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   |

Table B-1. Bus Operation List (Sheet 14 of 15)

| CODE | MNEMONIC | ~  | Cycle | Address bus        | Data bus                            | RD | WR | RMW |
|------|----------|----|-------|--------------------|-------------------------------------|----|----|-----|
| 01   | MULU A   | 8  | 1     | N +2               | The following following instruction | 1  | 0  | 0   |
|      |          |    | 2     | –                  | –                                   | 0  | 0  | 0   |
|      |          |    | to    |                    |                                     |    |    |     |
|      |          |    | 8     | –                  | –                                   | 0  | 0  | 0   |
| 11   | DIVU A   | 17 | 1     | N +2               | The following following instruction | 1  | 0  | 0   |
|      |          |    | 2     | –                  | –                                   | 0  | 0  | 0   |
|      |          |    | to    |                    |                                     |    |    |     |
|      |          |    | 17    | –                  | –                                   | 0  | 0  | 0   |
| –    | RESET    | 7  | 1     | –                  | –                                   | 0  | 0  | 0   |
|      |          |    | 2     | 0FFFD <sub>H</sub> | Mode data                           | 1  | 0  | 0   |
|      |          |    | 3     | 0FFFE <sub>H</sub> | Reset vector (H)                    | 1  | 0  | 0   |
|      |          |    | 4     | 0FFFF <sub>H</sub> | 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 (Sheet 15 of 15)

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

## C. Instruction Map



Table C-1. is an instruction map.

Instruction Map

Table C-1. Instruction Map

| LH | O    | 1    | 2     | 3     | 4           | 5         | 6         | 7        | 8          | 9          | A    | B               | C    | D    | E       | F          |
|----|------|------|-------|-------|-------------|-----------|-----------|----------|------------|------------|------|-----------------|------|------|---------|------------|
| 0  | NOP  | SWAP | RET   | RET1  | PUSHW       | POPW      | MOV       | MOVW     | CLRI       | SETI       | CLRB | BBC dir :0. rel | INCW | DECW | JMP @A  | MOVW A. PC |
| 1  | MULU | DIVU | JMP   | CALL  | PUSHW       | POPW      | MOV       | MOVW     | CLRC       | SETC       | CLRB | BBC dir :1. rel | INCW | DECW | MOVW SP | MOVW A. SP |
| 2  | ROLU | CMP  | ADDC  | SUBC  | XCH         | XOR       | AND       | OR       | MOV @A. T  | MOV A. @A  | CLRB | BBC dir :2. rel | INCW | DECW | MOVW IX | MOVW A. IX |
| 3  | RORC | CMPW | ADDCW | SUBCW | XCHW        | XORW      | ANDW      | ORW      | MOVW @A. T | MOVW A. @A | CLRB | BBC dir :3. rel | INCW | DECW | MOVW EP | MOVW A. EP |
| 4  | MOV  | CMP  | ADDC  | SUBC  | <div></div> | XOR       | AND       | OR       | DAA        | DAS        | CLRB | BBC dir :4. rel | MOVW | MOVW | MOVW    | MOVW A. PC |
| 5  | MOV  | CMP  | ADDC  | SUBC  | MOV         | XOR       | AND       | OR       | MOV        | CMP        | CLRB | BBC dir :5. rel | MOVW | MOVW | MOVW    | MOVW A. SP |
| 6  | MOV  | CMP  | ADDC  | SUBC  | MOV @IX+d   | XOR @IX+d | AND @IX+d | OR @IX+d | MOV IX+d   | CMP @IX+d  | CLRB | BBC dir :6. rel | MOVW | MOVW | MOVW    | MOVW A. IX |
| 7  | MOV  | CMP  | ADDC  | SUBC  | MOV @EP     | XOR @EP   | AND @EP   | OR @EP   | MOV @EP    | CMP @EP    | CLRB | BBC dir :7. rel | MOVW | MOVW | MOVW    | MOVW A. EP |
| 8  | MOV  | CMP  | ADDC  | SUBC  | MOV         | XOR       | AND       | OR       | MOV        | CMP        | SETB | BBS dir :0. rel | INC  | DEC  | CALLV   | BNC        |
| 9  | MOV  | CMP  | ADDC  | SUBC  | MOV         | XOR       | AND       | OR       | MOV        | CMP        | SETB | BBS dir :1. rel | INC  | DEC  | CALLV   | BC         |
| A  | MOV  | CMP  | ADDC  | SUBC  | MOV         | XOR       | AND       | OR       | MOV        | CMP        | SETB | BBS dir :2. rel | INC  | DEC  | CALLV   | BP         |
| B  | MOV  | CMP  | ADDC  | SUBC  | MOV         | XOR       | AND       | OR       | MOV        | CMP        | SETB | BBS dir :3. rel | INC  | DEC  | CALLV   | BN         |
| C  | MOV  | CMP  | ADDC  | SUBC  | MOV         | XOR       | AND       | OR       | MOV        | CMP        | SETB | BBS dir :4. rel | INC  | DEC  | CALLV   | BNZ        |
| D  | MOV  | CMP  | ADDC  | SUBC  | MOV         | XOR       | AND       | OR       | MOV        | CMP        | SETB | BBS dir :5. rel | INC  | DEC  | CALLV   | BZ         |
| E  | MOV  | CMP  | ADDC  | SUBC  | MOV         | XOR       | AND       | OR       | MOV        | CMP        | SETB | BBS dir :6. rel | INC  | DEC  | CALLV   | BGE        |
| F  | MOV  | CMP  | ADDC  | SUBC  | MOV         | XOR       | AND       | OR       | MOV        | CMP        | SETB | BBS dir :7. rel | INC  | DEC  | CALLV   | BLT        |

# Revision History



## Document Revision History

| Document Title: F <sup>2</sup> MC-8FX Programming Specifications |         |            |                  |  |
|--|---------|------------|------------------|--|
| Document Number: 002-07004                                       |         |            |                  |  |
| Revision   | ECN#    | Issue Date | Origin of Change | Description of Change  |
| **   |         | 02/06/2008 | GERR             | Initial Release  |
| *A   | 5661013 | 03/15/2017 | GERR             | Updated to Cypress template.                                     |
| *B   | 6533826 | 04/24/2019 | HTER             | Changed title and category<br>Updated Table A-10 (No.42, 43, 60) |