



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

Document Number: 002-07003 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. 本書面は、Cypress Semiconductor Corporation 及び Spansion LLC を含むその子会社（以下「Cypress」という。）に帰属する財産である。本書面（本書面に含まれ又は言及されているあらゆるソフトウェア若しくはファームウェア（以下「本ソフトウェア」という。）を含む）は、アメリカ合衆国及び世界のその他の国における知的財産法令及び条約に基づき Cypress が所有する。Cypress はこれらの法令及び条約に基づく全ての権利を留保し、本段落で特に記載されているものを除き、その特許権、著作権、商標権又はその他の知的財産権のライセンスを一切許諾しない。本ソフトウェアにライセンス契約書が伴っておらず、かつ Cypress との間で別途本ソフトウェアの使用方法を定める書面による合意がない場合、Cypress は、(1) 本ソフトウェアの著作権に基づき、(a) ソースコード形式で提供されている本ソフトウェアについて、Cypress ハードウェア製品と共に用いるためにのみ、かつ組織内部でのみ、本ソフトウェアの修正及び複製を行うこと、並びに (b) Cypress のハードウェア製品ユニットに用いるためにのみ、（直接又は再販売者及び販売代理店を介して間接のいずれかで）本ソフトウェアをバイナリーコード形式で外部エンドユーザーに配布すること、並びに (2) 本ソフトウェア (Cypress により提供され、修正がなされていないもの) が抵触する Cypress の特許権のクレームに基づき、Cypress ハードウェア製品と共に用いるためにのみ、本ソフトウェアの作成、利用、配布及び輸入を行うことについての非独占的で譲渡不能な一身専属的ライセンス（サブライセンスの権利を除く）を付与する。本ソフトウェアのその他の使用、複製、修正、変換又はコンパイルを禁止する。

**適用される法律により許される範囲内で、Cypress は、本書面又はいかなる本ソフトウェア若しくはこれに伴うハードウェアに関しても、明示又は黙示をとわず、いかなる保証（商品性及び特定の目的への適合性の黙示の保証を含むがこれらに限られない）も行わない。**いかなるコンピューティングデバイスも絶対に安全ということはない。従って、Cypress のハードウェアまたはソフトウェア製品に講じられたセキュリティ対策にもかかわらず、Cypress は、Cypress 製品への権限のないアクセスまたは使用といったセキュリティ違反から生じる一切の責任を負わない。加えて、本書面に記載された製品には、エラーと呼ばれる設計上の欠陥またはエラーが含まれている可能性があり、公表された仕様とは異なる動作をする場合がある。適用される法律により許される範囲内で、Cypress は、別途通知することなく、本書面を変更する権利を留保する。Cypress は、本書面に記載のある、いかなる製品若しくは回路の適用又は使用から生じる一切の責任を負わない。本書面で提供されたあらゆる情報（あらゆるサンプルデザイン情報又はプログラムコードを含む）は、参照目的のためのみに提供されたものである。この情報で構成するあらゆるアプリケーション及びその結果としてのあらゆる製品の機能性及び安全性を適切に設計、プログラム、かつテストすることは、本書面のユーザーの責任において行われるものとする。Cypress 製品は、兵器、兵器システム、原子力施設、生命維持装置若しくは生命維持システム、蘇生用の設備及び外科的移植を含むその他の医療機器若しくは医療システム、汚染管理若しくは有害物質管理の運用のために設計され若しくは意図されたシステムの重要な構成部分としての使用、又は装置若しくはシステムの不具合が人身傷害、死亡若しくは物的損害を生じさせるようなその他の使用（以下「本目的外使用」という。）のためには設計、意図又は承認されていない。重要な構成部分とは、その不具合が装置若しくはシステムの不具合を生じさせるか又はその安全性若しくは実効性に影響すると合理的に予想できるような装置若しくはシステムのあらゆる構成部分をいう。Cypress 製品のあらゆる本目的外使用から生じ、若しくは本目的外使用に関連するいかなる請求、損害又はその他の責任についても、Cypress はその全部又は一部をとわず一切の責任を負わず、かつ Cypress はそれら一切から本書により免除される。Cypress は Cypress 製品の本来目的外使用から生じ又は本目的外使用に関連するあらゆる請求、費用、損害及びその他の責任（人身傷害又は死亡に基づく請求を含む）から免責補償される。

Cypress, Cypress のロゴ, Spansion, Spansion のロゴ及びこれらの組み合わせ, WICED, PSoC, CapSense, EZ-USB, F-RAM, 及び Traveo は、米国及びその他の国における Cypress の商標又は登録商標である。Cypress のより完全な商標のリストは、cypress.com を参照すること。その他の名称及びブランドは、それぞれの権利者の財産として権利主張がなされている可能性がある。

# Contents



<b>1. F<sup>2</sup>MC-8FX CPUの概要と構成例</b>	<b>6</b>
1.1 F <sup>2</sup> MC-8FX CPUの概要 .....	7
1.2 F <sup>2</sup> MC-8FX CPUを用いたデバイスの構成例 .....	8
<b>2. メモリ空間</b>	<b>9</b>
2.1 CPUのメモリ空間 .....	10
2.2 メモリ空間とアドレッシング .....	11
<b>3. レジスタ</b>	<b>18</b>
3.1 F <sup>2</sup> MC-8FXのレジスタ .....	19
3.2 プログラムカウンタ(PC)/スタックポインタ(SP) .....	20
3.3 アキュムレータ(A)/テンポラリアキュムレータ(T) .....	21
3.4 プログラムステータス(PS) .....	26
3.5 インデックスレジスタ(IX)/エクストラポインタ(EP) .....	29
3.6 レジスタバンク .....	30
3.7 ダイレクトバンク .....	31
<b>4. 割込み処理</b>	<b>32</b>
4.1 割込み動作概要 .....	33
4.2 割込み許可/禁止/優先順位機構 .....	34
4.3 割込み処理プログラムの作成方法 .....	35
4.4 多重割込み .....	36
4.5 リセット動作 .....	37
<b>5. CPUソフトウェアアーキテクチャ</b>	<b>38</b>
5.1 アドレッシングの種類 .....	39
5.2 特殊な命令 .....	42
<b>6. 実行命令細則</b>	<b>46</b>
6.1 ADDC (ADD Byte Data of Accumulator and Temporary Accumulator with Carry to Accumulator) .....	47
6.2 ADDC (ADD Byte Data of Accumulator and Memory with Carry to Accumulator) .....	49
6.3 ADDCW (ADD Word Data of Accumulator and Temporary Accumulator with Carry to Accumulator) .....	51
6.4 AND (AND Byte Data of Accumulator and Temporary Accumulator to Accumulator) .....	53
6.5 AND (AND Byte Data of Accumulator and Memory to Accumulator) .....	55
6.6 ANDW (AND Word Data of Accumulator and Temporary Accumulator to Accumulator) .....	57
6.7 BBC (Branch if Bit is Clear) .....	59

6.8	BBS (Branch if Bit is Set).....	61
6.9	BC (Branch relative if C=1)/BLO (Branch if LOWER).....	63
6.10	BGE (Branch Great or Equal: relative if $\geq$ Zero).....	65
6.11	BLT (Branch Less Than zero: relative if $<$ Zero).....	67
6.12	BN (Branch relative if N = 1).....	69
6.13	BNZ (Branch relative if Z = 0)/BNE (Branch if Not Equal) .....	71
6.14	BNC (Branch relative if C = 0)/BHS (Branch if Higher or Same) .....	73
6.15	BP (Branch relative if N = 0: PLUS).....	75
6.16	BZ (Branch relative if Z = 1)/BEQ (Branch if Equal) .....	77
6.17	CALL (CALL subroutine).....	79
6.18	CALLV (CALL Vectored subroutine).....	81
6.19	CLRB (Clear direct Memory Bit) .....	83
6.20	CLRC (Clear Carry flag) .....	85
6.21	CLRI (CLear Interrupt flag).....	87
6.22	CMP (CoMPare Byte Data of Accumulator and Temporary Accumulator) .....	89
6.23	CMP (CoMPare Byte Data of Accumulator and Memory) .....	91
6.24	CMP (CoMPare Byte Data of Immediate Data and Memory) .....	93
6.25	CMPW (CoMPare Word Data of Accumulator and Temporary Accumulator) .....	95
6.26	DAA (Decimal Adjust for Addition).....	97
6.27	DAS (Decimal Adjust for Subtraction).....	99
6.28	DEC (DECrement Byte Data of General-purpose Register).....	101
6.29	DECW (DECrement Word Data of Accumulator) .....	103
6.30	DECW (DECrement Word Data of Extra Pointer).....	105
6.31	DECW (DECrement Word Data of Index Pointer) .....	107
6.32	DECW (DECrement Word Data of Stack Pointer) .....	109
6.33	DIVU (DIVide Unsigned).....	111
6.34	INC (INCrement Byte Data of General-purpose Register).....	113
6.35	INCW (INCrement Word Data of Accumulator) .....	115
6.36	INCW (INCrement Word Data of Extra Pointer) .....	117
6.37	INCW (INCrement Word Data of Index Register).....	119
6.38	INCW (INCrement Word Data of Stack Pointer).....	121
6.39	JMP (JuMP to address pointed by Accumulator).....	123
6.40	JMP (JuMP to effective Address) .....	125
6.41	MOV (MOVE Byte Data from Temporary Accumulator to Address Pointed by Accumulator) .....	127
6.42	MOV (MOVE Byte Data from Memory to Accumulator).....	129
6.43	MOV (MOVE Immediate Byte Data to Memory).....	131
6.44	MOV (MOVE Byte Data from Accumulator to Memory).....	133
6.45	MOVW (MOVE Word Data from Temporary Accumulator to Address Pointed by Accumulator) .....	135
6.46	MOVW (MOVE Word Data from Memory to Accumulator).....	137
6.47	MOVW (MOVE Word Data from Extra Pointer to Accumulator) .....	139
6.48	MOVW (MOVE Word Data from Index Register to Accumulator).....	141
6.49	MOVW (MOVE Word Data from Program Status Register to Accumulator) .....	143
6.50	MOVW (MOVE Word Data from Program Counter to Accumulator) .....	145
6.51	MOVW (MOVE Word Data from Stack Pointer to Accumulator) .....	147
6.52	MOVW (MOVE Word Data from Accumulator to Memory).....	149
6.53	MOVW (MOVE Word Data from Accumulator to Extra Pointer) .....	151
6.54	MOVW (MOVE Immediate Word Data to Extra Pointer) .....	153
6.55	MOVW (MOVE Word Data from Accumulator to Index Register).....	155
6.56	MOVW (MOVE Immediate Word Data to Index Register).....	157
6.57	MOVW (MOVE Word data from Accumulator to Program Status Register) .....	159
6.58	MOVW (MOVE Immediate Word Data to Stack Pointer).....	161

6.59	MOVW (MOVE Word data from Accumulator to Stack Pointer) .....	163
6.60	MULU (MULTIply Unsigned) .....	165
6.61	NOP (NoOperation) .....	167
6.62	OR (OR Byte Data of Accumulator and Temporary Accumulator to Accumulator) ..	169
6.63	OR (OR Byte Data of Accumulator and Memory to Accumulator) .....	171
6.64	ORW (OR Word Data of Accumulator and Temporary Accumulator to Accumulator) .....	173
6.65	PUSHW (PUSH Word Data of Inherent Register to Stack Memory) .....	175
6.66	POPW (POP Word Data of Inherent Register from Stack Memory) .....	177
6.67	RET (RETurn from subroutine) .....	179
6.68	RETI (RETurn from Interrupt) .....	181
6.69	ROL (Rotate Byte Data of Accumulator with Carry to Left) .....	183
6.70	ROR (Rotate Byte Data of Accumulator with Carry to Right) .....	185
6.71	SUBC (SUBtract Byte Data of Accumulator from Temporary Accumulator with Carry to Accumulator) .....	187
6.72	SUBC (SUBtract Byte Data of Memory from Accumulator with Carry to Accumulator) .....	189
6.73	SUBCW (SUBtract Word Data of Accumulator from Temporary Accumulator with Carry to Accumulator) .....	191
6.74	SETB (Set Direct Memory Bit) .....	193
6.75	SETC (SET Carry flag) .....	195
6.76	SETI (SET Interrupt flag) .....	197
6.77	SWAP (SWAP Byte Data Accumulator "H" and Accumulator "L") .....	199
6.78	XCH (eXCHange Byte Data Accumulator "L" and Temporary Accumulator "L") .....	201
6.79	XCHW (eXCHange Word Data Accumulator and Extrapointer) .....	203
6.80	XCHW (eXCHange Word Data Accumulator and Index Register) .....	205
6.81	XCHW (eXCHange Word Data Accumulator and Program Counter) .....	207
6.82	XCHW (eXCHange Word Data Accumulator and Stack Pointer) .....	209
6.83	XCHW (eXCHange Word Data Accumulator and Temporary Accumulator) .....	211
6.84	XOR (eXclusive OR Byte Data of Accumulator and Temporary Accumulator to Accumulator) .....	213
6.85	XOR (eXclusive OR Byte Data of Accumulator and Memory to Accumulator) .....	215
6.86	XORW (eXclusive OR Word Data of Accumulator and Temporary Accumulator to Accumulator) .....	217

## **本版での主な変更内容** **219**

### **A. 命令一覧表** **221**

A.1	F <sup>2</sup> MC-8FX CPUの命令概要 .....	223
A.2	動作一覧表 .....	226
A.3	フラグ変化表 .....	233

### **B. バス動作一覧表** **241**

### **C. 命令マップ** **252**

### **改訂履歴.** **254**

# 1. F<sup>2</sup>MC-8FX CPU の概要と構成例



この章では、F<sup>2</sup>MC-8FX CPU の概要と構成例について説明します。

1.1 F<sup>2</sup>MC-8FX CPU の概要

1.2 F<sup>2</sup>MC-8FX CPU を用いたデバイスの構成例

## 1.1 F<sup>2</sup>MC-8FX CPU の概要

F<sup>2</sup>MC-8FX CPU は、各種産業用、OA 用機器などの組込み制御向けに設計された高性能 8 ビット CPU です。

### F<sup>2</sup>MC-8FX CPU の概要

F<sup>2</sup>MC-8FX CPU は、各種産業用、OA 用機器などの組込み制御向けに設計された高性能 8 ビット CPU です。特に低電圧、低消費電力を必要とする分野を対象としています。8 ビット CPU でありながら 16 ビット転送・演算を行うことができ、制御を行う上で 16 ビットデータを必要とする用途にも適しています。また、F<sup>2</sup>MC-8FX CPU は F<sup>2</sup>MC-8L CPU の上位互換 CPU であり、命令サイクル数の短縮、除算命令の強化、ダイレクト領域の拡張を行っています。

### F<sup>2</sup>MC-8FX CPU の特長

以下に F<sup>2</sup>MC-8FX CPU の特長を示します。

- 最小命令実行時間 : 100ns
- メモリ空間 : 64K バイト
- コントローラ用途に適した命令体系
  - データタイプ : ビット、バイト、ワード
  - アドレッシングモード : 9 種類
  - 高いコード効率
  - 16 ビットデータ演算 : A, T 間の演算
  - ビット命令の強化 : セット、リセット、チェック
  - 乗除算命令 :  $8 \times 8 = 16$  ビット,  $16 \div 16 = 16$  ビット
- 割込みプライオリティレベル : 4 種

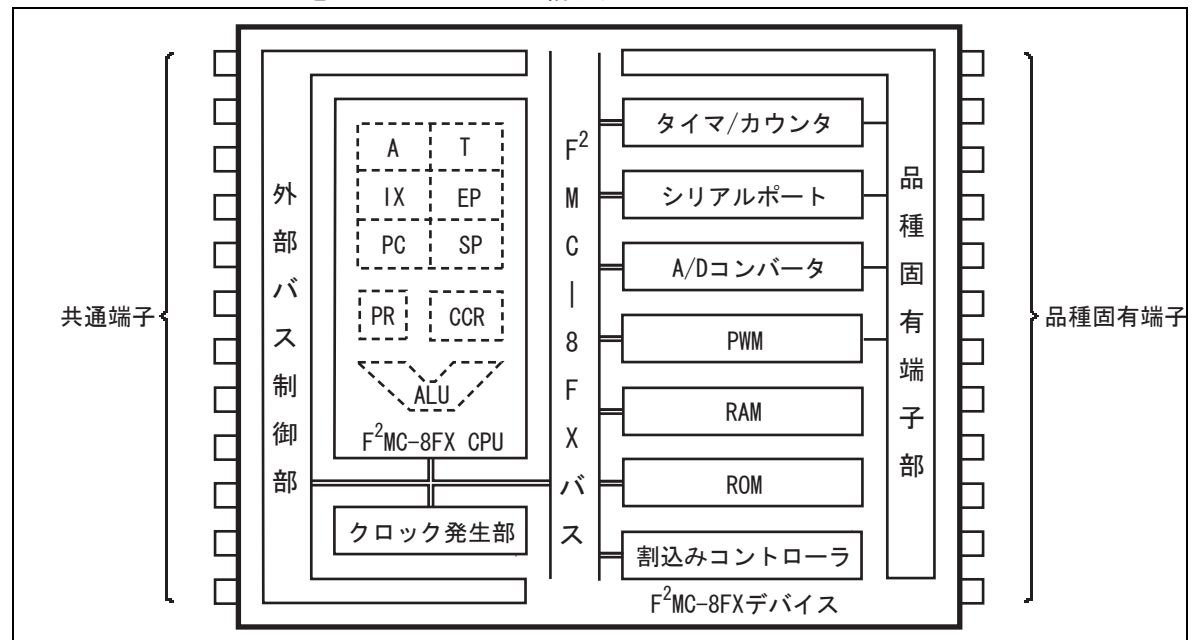
## 1.2 F<sup>2</sup>MC-8FX CPU を用いたデバイスの構成例

F<sup>2</sup>MC-8FX デバイスは、CPU、ROM、RAM および各種周辺をモジュール単位で設計してありますので、メモリサイズの変更や、周辺の入替えなどの操作により、アプリケーションに適合した品種の作成が行いやすくなっています。

### F<sup>2</sup>MC-8FX CPU を用いたデバイスの構成例

図 1-1. に F<sup>2</sup>MC-8FX CPU を用いたデバイスの構成例を示します。

図 1-1. F<sup>2</sup>MC-8FX CPU を用いたデバイスの構成例





## 2. メモリ空間



この章では、F<sup>2</sup>MC-8FX CPU のメモリ空間について説明します。

### 2.1 CPU のメモリ空間

### 2.2 メモリ空間とアドレッシング

## 2.1 CPU のメモリ空間

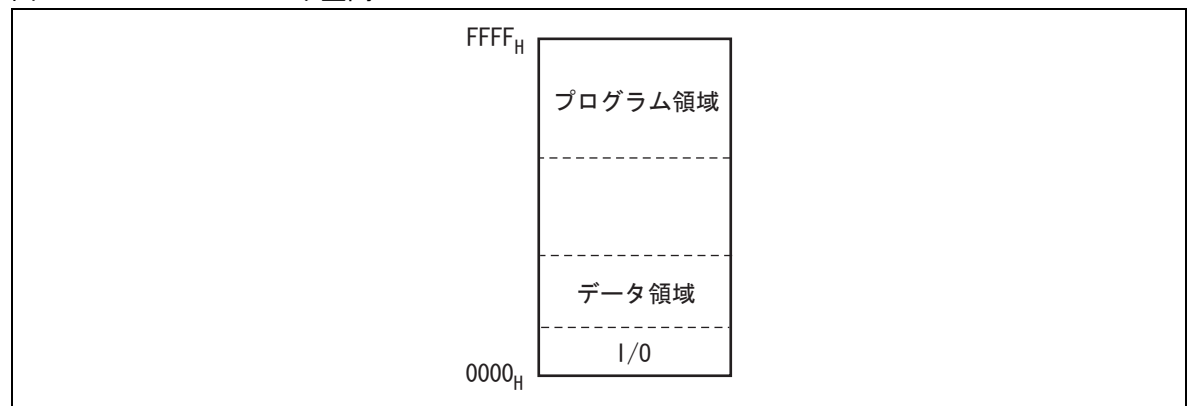
F<sup>2</sup>MC-8FX CPU の管理するデータ、プログラム、I/O はすべて F<sup>2</sup>MC-8FX CPU の持つ 64K バイトのメモリ空間に配置されます。CPU は、16 ビットのアдресバスでこれらのアドレスを示すことにより各リソースをアクセスできます。

### CPU のメモリ空間

図 2-1. に F<sup>2</sup>MC-8FX のメモリ空間のアドレス構成を示します。

I/O はアドレスの最下位近くにあり、データ領域はそのすぐ上に配置します。データ領域は、用途別にレジスタバンク領域、スタック領域、ダイレクト領域などに分割することができます。プログラム領域は正反対にアドレス空間の最上位付近にあり、その中でも最も上位に割込みリセットベクタやベクタコール命令のテーブルなどを配置します。

図 2-1. F<sup>2</sup>MC-8FX メモリ空間



## 2.2 メモリ空間とアドレッシング

F<sup>2</sup>MC-8FX の持つアドレッシングにおいて、メモリのアクセスに関係するもののなかにはアドレスに依存して最適なものが変わるものもありますので、適切な命令を使用することで命令のコード効率を上げることができます。

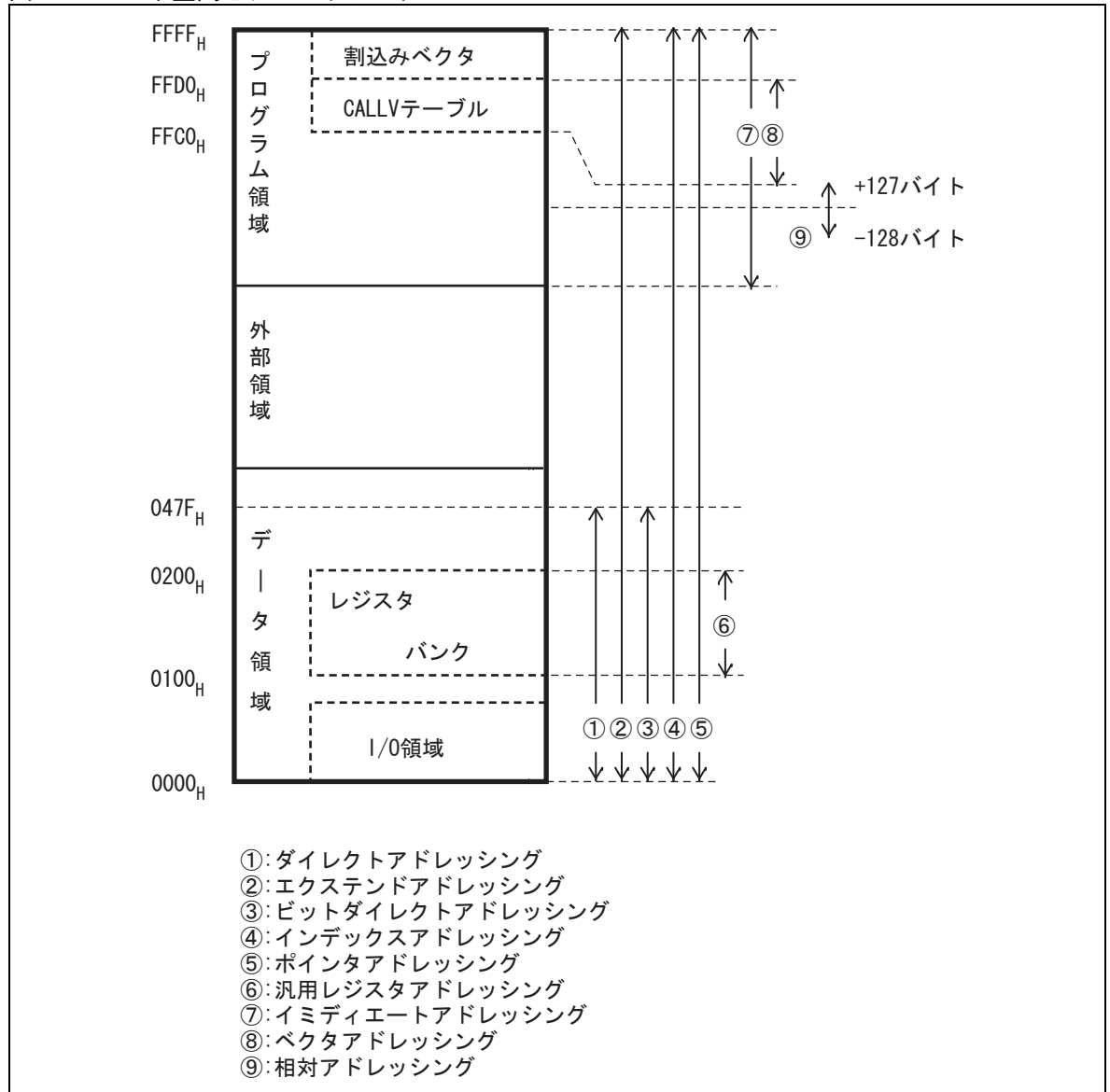
### メモリ空間とアドレッシング

F<sup>2</sup>MC-8FX CPU の持つメモリのアクセスに関係するもののアドレッシングを以下に示します ( [ ] は 1 バイトを示す )。

- **ダイレクトアドレッシング**: アドレスの下位 8 ビットをオペランドで指定するアドレッシング。オペランドアドレス 00<sub>H</sub> ~ 7F<sub>H</sub> のアクセスは常に 0000<sub>H</sub> ~ 007F<sub>H</sub> となる。オペランドアドレス 80<sub>H</sub> ~ FF<sub>H</sub> のアクセスはダイレクトバンクポインタ (DP) の設定により、0080<sub>H</sub> ~ 047F<sub>H</sub> へマッピングされる。  
 [構造] [← OP コード→] [←下位 8 ビット→] ( [←オペランドがあれば→] )
- **エクステンドアドレッシング**: 16 ビット全部をオペランドで指定するアドレッシング  
 [構造] [← OP コード→] [←上位 8 ビット→] [←下位 8 ビット→]
- **ビットダイレクトアドレッシング**: アドレスの下位 8 ビットをオペランドで指定するアドレッシング。オペランドアドレス 00<sub>H</sub> ~ 7F<sub>H</sub> のアクセスは常に 0000<sub>H</sub> ~ 007F<sub>H</sub> となる。オペランドアドレス 80<sub>H</sub> ~ FF<sub>H</sub> のアクセスはダイレクトバンクポインタ (DP) の設定により、0080<sub>H</sub> ~ 047F<sub>H</sub> へマッピングされる。ビット位置は OP コード内に含まれる  
 [構造] [← OP コード: ビット→] [←下位 8 ビット→]
- **インデックスアドレッシング**: オペランドの 8 ビットをインデックスレジスタ IX に符号付きで加算し、その結果をアドレスとするアドレッシング  
 [構造] [← OP コード→] [←オフセット 8 ビット→] ( [←オペランドがあれば→] )
- **ポインタアドレッシング**: ポインタ EP の内容をそのままアドレスとするアドレッシング  
 [構造] [← OP コード→]
- **汎用レジスタアドレッシング**: 汎用レジスタを指定するアドレッシング。レジスタ番号は OP コードに含まれる  
 [構造] [← OP コード: レジスタ→]
- **イミディエートアドレッシング**: OP コードに続く 1 バイトをデータとするアドレッシング  
 [構造] [← OP コード→] [←イミディエートデータ→]
- **ベクタアドレッシング**: テーブル番号に対応したテーブルからデータを引き出すアドレッシング。テーブル番号は OP コードに含まれる  
 [構造] [← OP コード: テーブル→]
- **相対アドレッシング**: 現在の PC の内容に対して相対的にアドレスを算出するアドレッシング。相対ジャンプ命令とビットチェック命令で使用  
 [構造] [← OP コード→] [←相対値 8 ビット→]

図 2-2. に各アドレッシングとそれがアクセスするメモリ空間の対応を示します。

図 2-2. メモリ空間とアドレッシング



## 2.2.1 データ領域

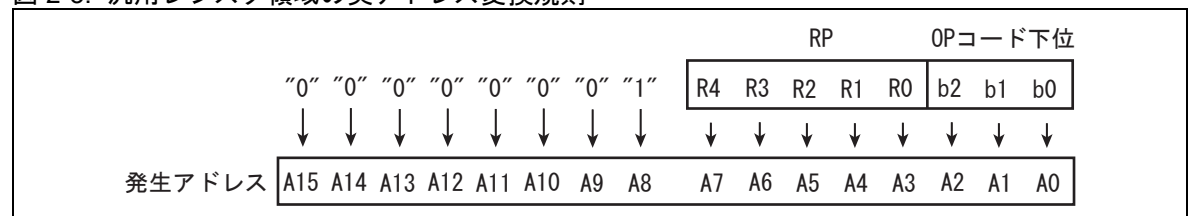
F<sup>2</sup>MC-8FX CPU のデータ領域は用途別に、以下の 3 つの領域に分割できます。

- 汎用レジスタバンク領域
- スタック領域
- ダイレクト領域

### 汎用レジスタバンク領域

F<sup>2</sup>MC-8FX の汎用レジスタバンク領域は 0100<sub>H</sub> ~ 01FF<sub>H</sub> に割り当ててあり、汎用レジスタのレジスタ番号から実際のアドレスへの変換はレジスタバンクポインタ (RP) と OP コードの下位 3 ビットを用いて、図 2-3. に示す変換規則で行います。

図 2-3. 汎用レジスタ領域の実アドレス変換規則



### スタック領域

F<sup>2</sup>MC-8FX のスタック領域はサブルーチンコール命令の実行時、割込み発生時の帰り番地や専用レジスタの退避用領域として使用します。スタック領域へデータを退避する際には、退避する前に 16 ビット長のスタックポインタ (SP) の内容を先に 2 つだけ減らし、その後 SP が示すアドレスへ退避データを書き込みます。また、スタック領域からデータを復帰する際には、まず SP が示すアドレスからデータを復帰し、その後 SP の内容を後で 2 つだけ増やします。このような理由のため、SP が示しているアドレスにはスタックに退避した最も新しいデータを格納してあることになります。これらの様子を図 2-4.、図 2-5. に示します。

図 2-4. スタック領域への退避の例

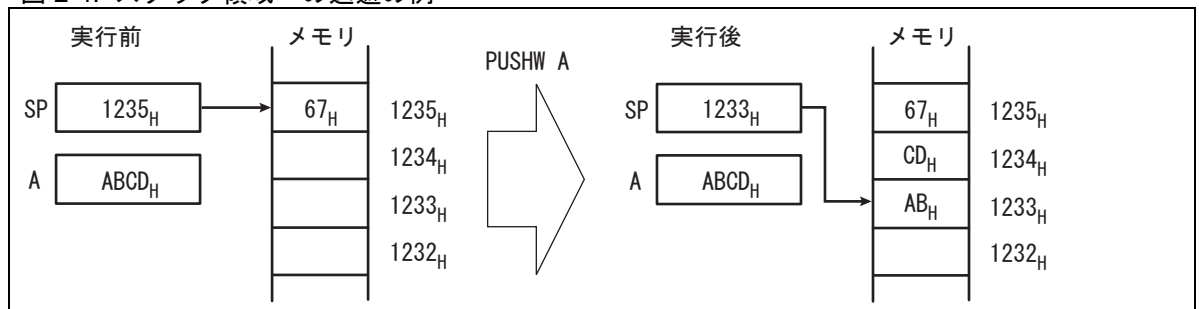
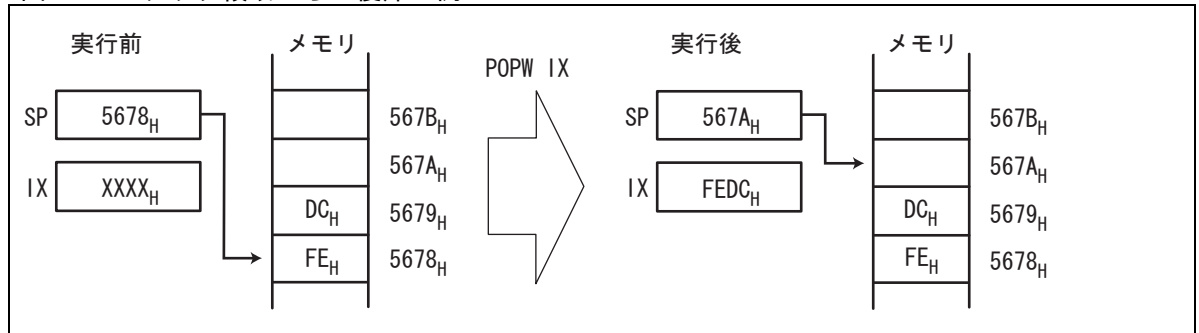


図 2-5. スタック領域からの復帰の例



## ダイレクト領域

F<sup>2</sup>MC-8FX のダイレクト領域は、メモリ空間の下位側、すなわち 0000<sub>H</sub> ~ 047F<sub>H</sub> の 1152 バイトにあり、ダイレクトアドレッシングおよびビットダイレクトアドレッシングでアクセスできます。ダイレクトアドレッシングおよびビットダイレクトアドレッシングを用いたアクセスで、一度に使用できるダイレクト領域は、256 バイトです。0000<sub>H</sub> ~ 007F<sub>H</sub> の 128 バイトは常にダイレクト領域として使用可能です。0080<sub>H</sub> ~ 047F<sub>H</sub> は 128 バイト×8 のダイレクトバンクであり、ダイレクトバンクポインタ (DP) を設定することにより 1 つのダイレクトバンクをダイレクト領域として使用できます。ダイレクトアドレッシングおよびビットダイレクトアドレッシングのオペランドアドレスから実アドレスへの変換は、DP を用いて表 2-1. に示す変換規則で行います。

ダイレクトアドレッシングおよびビットダイレクトアドレッシングでダイレクト領域へアクセスすると、使用するコードが 2 バイトで済みます。このため頻繁にアクセスする I/O の制御レジスタや、RAM の一部をここに配置しています。

表 2-1. ダイレクトアドレッシングおよびビットダイレクトアドレッシングの実アドレス変換規則

オペランドアドレス	ダイレクトバンクポインタ (DP)	実アドレス
00 <sub>H</sub> ~ 7F <sub>H</sub>		0000 <sub>H</sub> ~ 007F <sub>H</sub>
80 <sub>H</sub> ~ FF <sub>H</sub>	000	0080 <sub>H</sub> ~ 00FF <sub>H</sub>
	001	0100 <sub>H</sub> ~ 017F <sub>H</sub>
	010	0180 <sub>H</sub> ~ 01FF <sub>H</sub>
	011	0200 <sub>H</sub> ~ 027F <sub>H</sub>
	100	0280 <sub>H</sub> ~ 02FF <sub>H</sub>
	101	0300 <sub>H</sub> ~ 037F <sub>H</sub>
	110	0380 <sub>H</sub> ~ 03FF <sub>H</sub>
	111	0400 <sub>H</sub> ~ 047F <sub>H</sub>

## 2.2.2 プログラム領域

F<sup>2</sup>MC-8FX CPU のプログラム領域には、以下の 2 つがあります。

- ベクタコール命令テーブル
- リセット / 割込みベクタテーブル

### ベクタコール命令テーブル

メモリ空間の FFC0<sub>H</sub> ~ FFCF<sub>H</sub> はベクタコール命令のテーブルとして使用されます。F<sup>2</sup>MC-8FX のベクタコール命令は、OP コード内に含まれたベクタ番号に対応してこの領域をアクセスし、そこに書かれた内容をジャンプ先アドレスとしたサブルーチンコールを行います。表 2-2. にベクタ番号とジャンプ先アドレステーブルの対応を示します。

表 2-2. CALLV ジャンプ先アドレステーブル

CALLV #k	ジャンプ先アドレステーブル	
	アドレス上位側	アドレス下位側
#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>

## リセット / 割込みベクタテーブル

メモリ空間の FFCC<sub>H</sub> ~ FFFF<sub>H</sub> は割込みあるいはリセットの開始アドレスを示すテーブルとして使用されます。各割込み番号あるいはリセットと参照されるテーブルの対応を表 2-3. に示します。

表 2-3. リセットと割込みのベクタテーブル

割込み番号	テーブルのアドレス			割込み番号	テーブルのアドレス	
	上位データ	下位データ			上位データ	下位データ
リセット	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>
#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	FFEE <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>

FFFC<sub>H</sub>: 予約

FFFD<sub>H</sub>: モード

( 注意事項 ) 実際の番号数は各品種に依存します。

割込み番号 #22 と #23 はベクタコール命令 CALLV #6, CALLV #7 と排他的に使用してください。



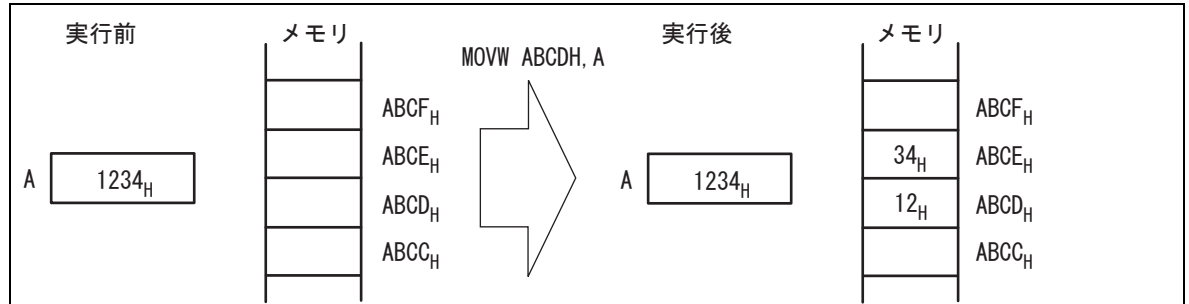
### 2.2.3 16 ビットデータにおけるメモリ空間配置

F<sup>2</sup>MC-8FX CPU は、8 ビット CPU でありながら 16 ビット転送・演算を行うことができます。以下に、その 16 ビットデータにおけるメモリ空間の配置を示します。

#### 16 ビットデータにおけるメモリ空間配置

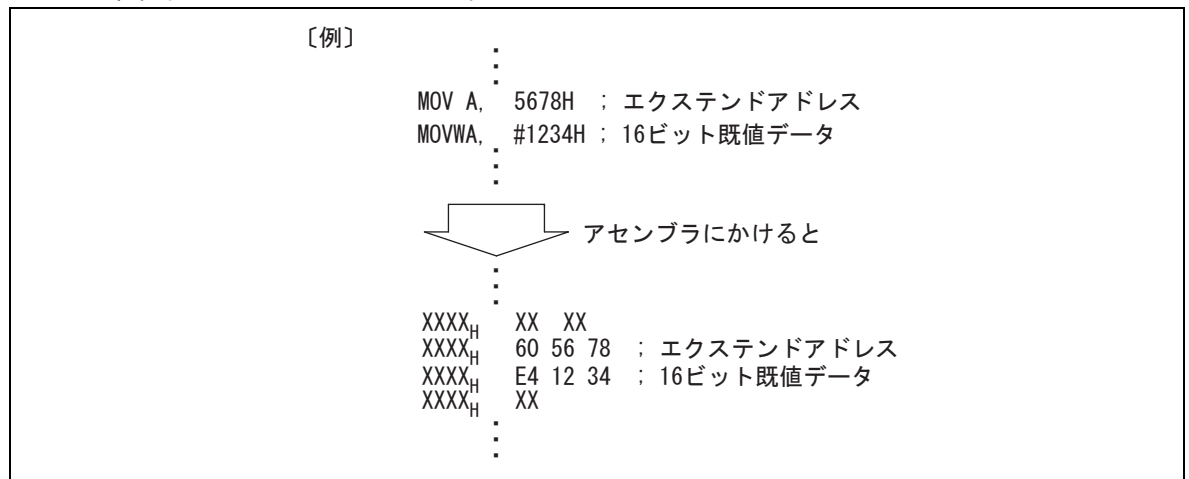
F<sup>2</sup>MC-8FX はメモリ上の 16 ビットデータを扱うときには、図 2-6. に示すようにアドレスの値の少ない方に書かれたデータを上位として、その次のアドレスに書かれたデータを下位として扱います。

図 2-6. メモリ上の 16 ビットデータの配置



また、命令中のオペランドで 16 ビット指定をする場合も同様に、OP コードに近い方から上位バイト、その次に下位バイトの順番になります。これはオペランドが図 2-7. に示すようにメモリアドレスを示す場合でも、16 ビットの即値データの場合でも同じです。

図 2-7. 命令中の 16 ビットデータの配置



さらに、割込みなどでスタックに退避したデータも同様です。

## 3. レジスタ



この章では、F<sup>2</sup>MC-8FX の専用レジスタと汎用レジスタについて説明します。

- 3.1 F<sup>2</sup>MC-8FX のレジスタ
- 3.2 プログラムカウンタ (PC)/ スタックポインタ (SP)
- 3.3 アキュムレータ (A)/ テンポラリアキュムレータ (T)
- 3.4 プログラムステータス (PS)
- 3.5 インデックスレジスタ (IX)/ エクストラポインタ (EP)
- 3.6 レジスタバンク
- 3.7 ダイレクトバンク

### 3.1 F<sup>2</sup>MC-8FX のレジスタ

F<sup>2</sup>MC-8FX には CPU 内にある用途専用のレジスタとメモリ上にある汎用レジスタの 2 種類のレジスタがあります。

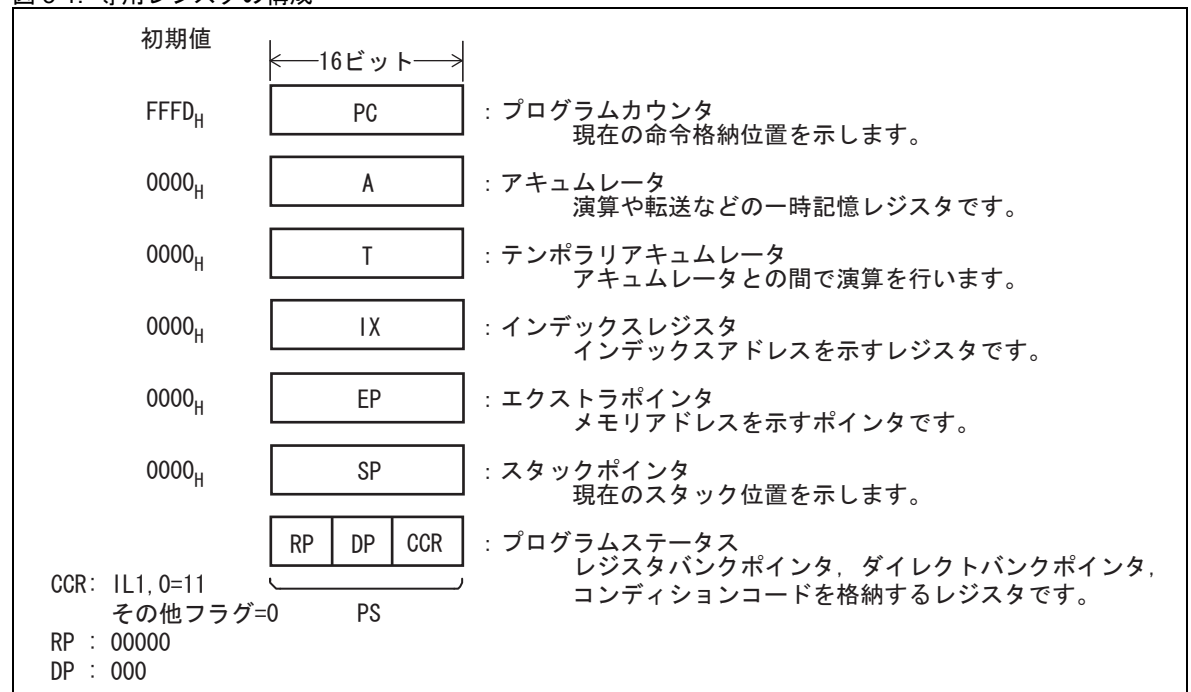
#### F<sup>2</sup>MC-8FX の専用レジスタ

専用レジスタは、CPU の内部に専用ハードウェアとして存在し、使用する用途が CPU のアーキテクチャ上に限定されているものです。

専用レジスタは 7 種類の 16 ビットレジスタによって構成されています。その中で、一部のレジスタについては下位 8 ビットのみを使用もできます。

図 3-1. に 7 種類の専用レジスタの構成を示します。

図 3-1. 専用レジスタの構成



#### F<sup>2</sup>MC-8FX の汎用レジスタ

汎用レジスタは以下のものが該当します。

- レジスタバンク : 8 ビット長、データを格納するレジスタ

## 3.2 プログラムカウンタ (PC)/ スタックポインタ (SP)

プログラムカウンタ (PC), スタックポインタ (SP) は用途専用レジスタで CPU 内にあります。

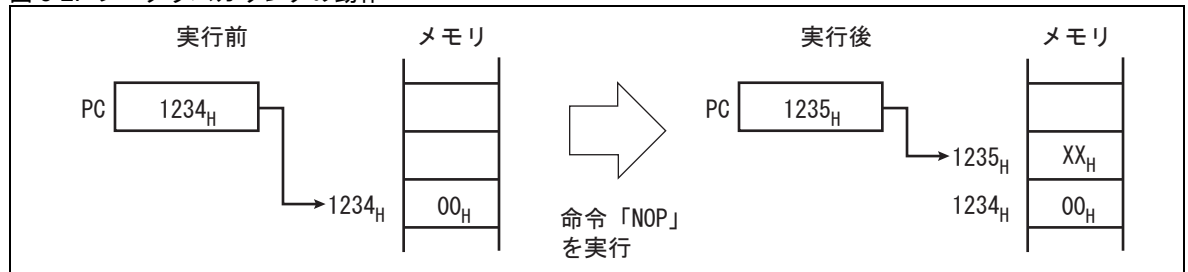
プログラムカウンタ (PC) は, 現在実行中の命令を格納しているアドレスを示しています。

スタックポインタ (SP) は, 割込みやスタック退避 / 復帰命令などで参照するデータのアドレスを保持します。現在のスタックポインタ (SP) の値は, スタック内に退避した最新のデータが格納されているアドレスになっています。

### プログラムカウンタ (PC)

図 3-2. にプログラムカウンタ (PC) の動作を示します。

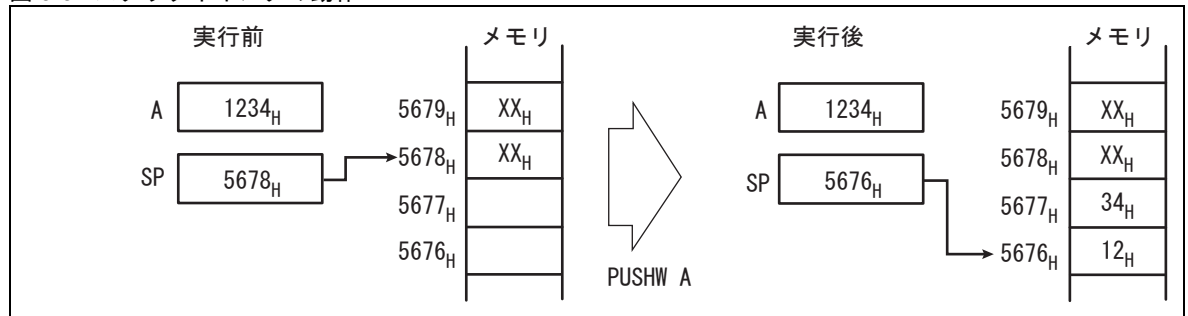
図 3-2. プログラムカウンタの動作



### スタックポインタ (SP)

図 3-3. にスタックポインタ (SP) の動作を示します。

図 3-3. スタックポインタの動作



### 3.3 アキュムレータ (A)/ テンポラリアキュムレータ (T)

アキュムレータ (A), テンポラリアキュムレータ (T) は用途専用レジスタで CPU 内にあります。

アキュムレータ (A) は、演算などを行ったときの結果の一時記憶領域として使用します。

テンポラリアキュムレータ (T) は、アキュムレータ (A) へのデータ転送の際の旧データの一時退避領域として、また、演算などのオペランドとして使用します。

#### アキュムレータ (A)

図 3-4. に示すように、16 ビット長の演算では 16 ビットすべてを使用します。8 ビット長の演算では図 3-5. に示すように、下位 8 ビットのみを使用します。

図 3-4. アキュムレータ (A) の動作 (16 ビット演算の場合)

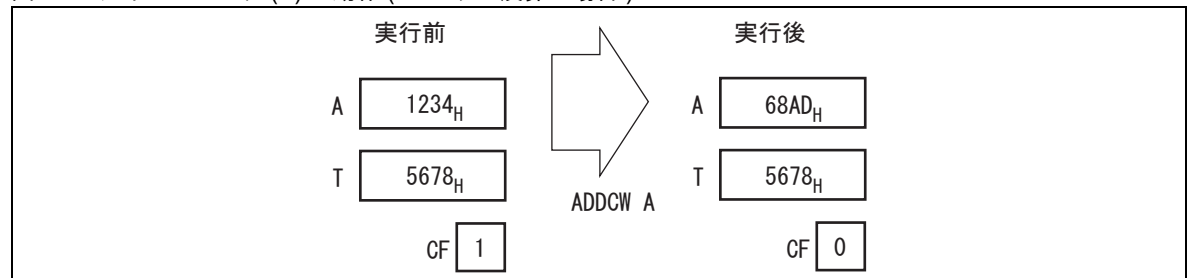
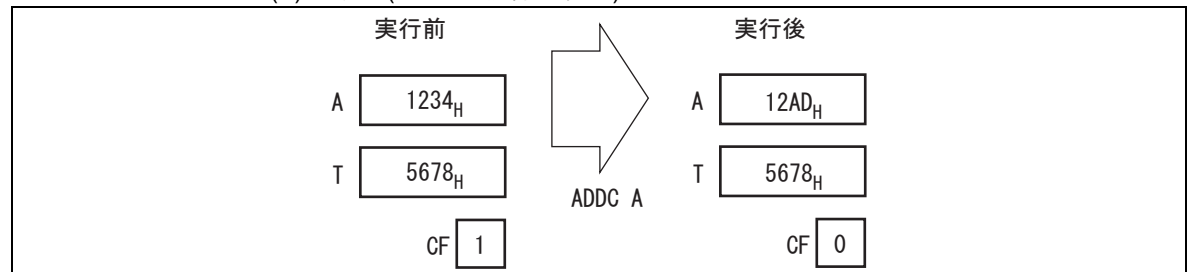


図 3-5. アキュムレータ (A) の動作 (8 ビット演算の場合)



## テンポラリアキュムレータ (T)

図 3-6. に示すように、アキュムレータ (A) へ 16 ビット長のデータ転送を行ったときは、A 内の旧 16 ビットデータすべてをテンポラリアキュムレータ (T) へ転送します。そして、図 3-7. に示すように、8 ビット長のデータ転送を行ったときは、A の下位 8 ビットに格納してあった旧 8 ビットデータを T の下位 8 ビットへ転送します。16 ビット長の演算では、図 3-8. に示すように、16 ビットすべてをオペランドとして使用します。そして、図 3-9. に示すように、8 ビット長の演算では下位 8 ビットのみを使用します。

図 3-6. アキュムレータ (A) とテンポラリアキュムレータ (T) 間の転送の動作 (16 ビット転送の場合)

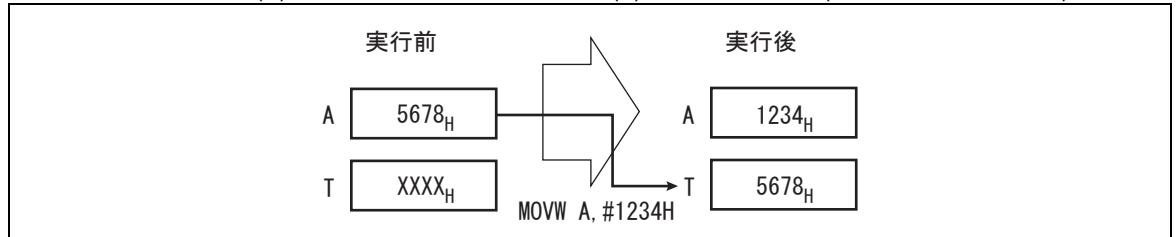


図 3-7. アキュムレータ (A) とテンポラリアキュムレータ (T) 間の転送の動作 (8 ビット転送の場合)

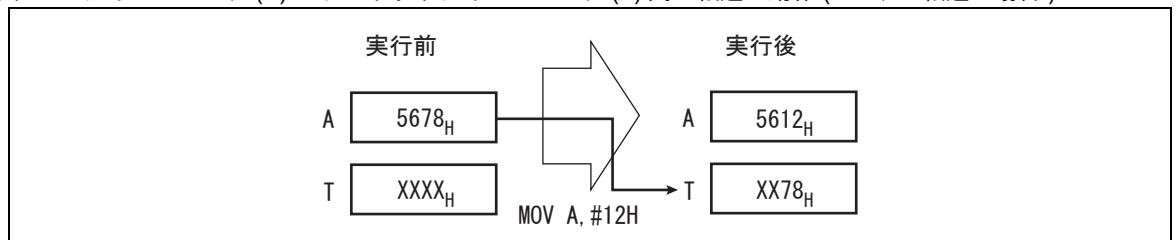


図 3-8. アキュムレータ (A) とテンポラリアキュムレータ (T) 間の演算の動作 (16 ビット演算の場合)

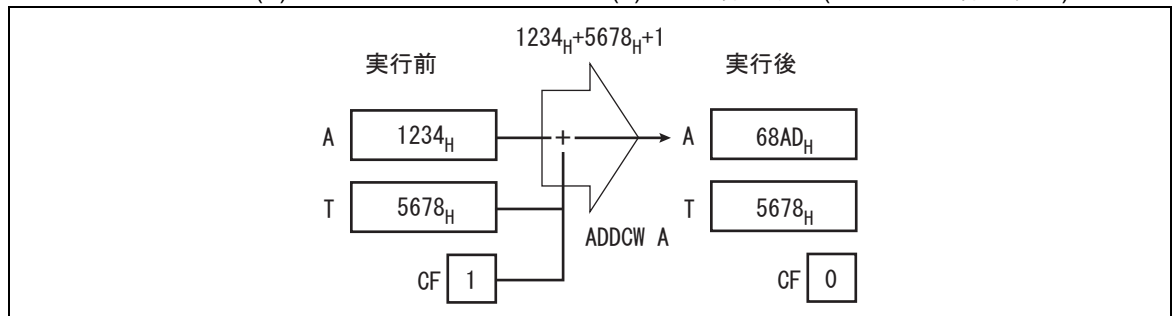
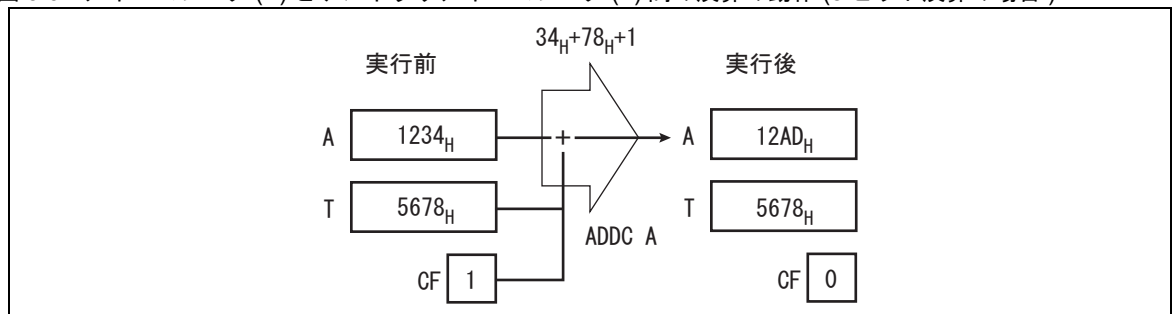


図 3-9. アキュムレータ (A) とテンポラリアキュムレータ (T) 間の演算の動作 (8 ビット演算の場合)



### 3.3.1 テンポラリアキュムレータ (T) の使い方

F<sup>2</sup>MC-8FX はテンポラリアキュムレータと呼ばれる特殊用途のレジスタがあります。ここではこのレジスタの動作について解説します。

### テンポラリアキュムレータ (T) の使い方

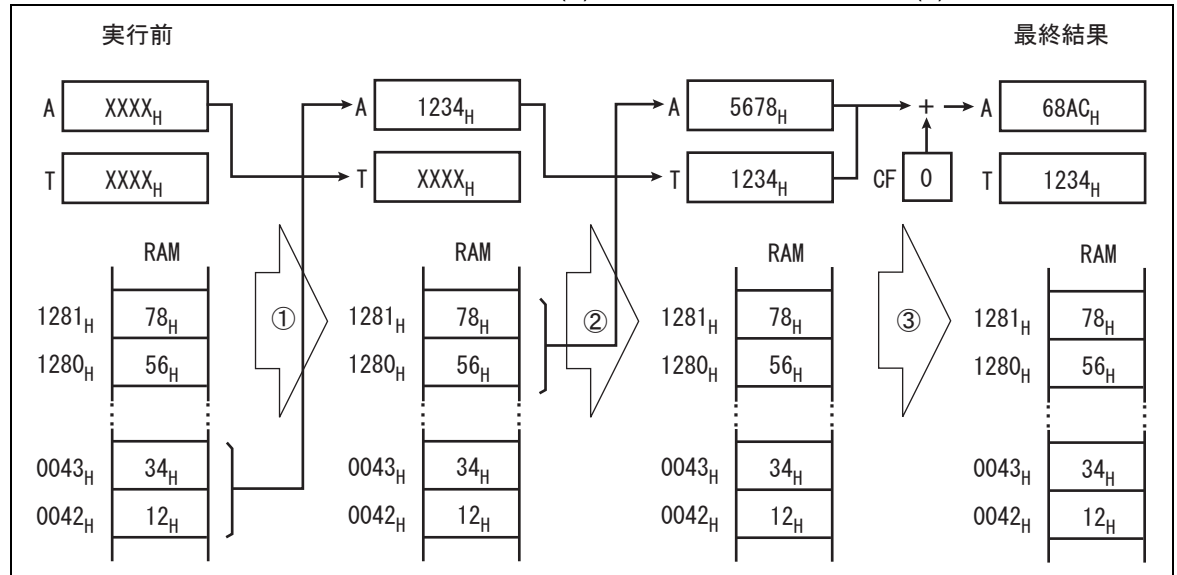
F<sup>2</sup>MC-8FX は各種の 2 項演算命令、一部の転送命令、16 ビットデータ演算用にテンポラリアキュムレータ (T) を備えています。T へ直接データを転送する命令はありませんが、アキュムレータ (A) へのデータ転送命令の実行に先立ち元の A の値を T へ転送しますので、A-T 間の演算を行う場合には 2 回 A への転送命令を行ったあとに実行すればよいことになります。すべての命令で T へ自動的に転送が行われるわけではないので、実際に転送を行う命令の詳細は命令一覧表の TL, TH 欄を参考にしてください。

以下の例は 1280<sub>H</sub> 番地に格納した 16 ビットデータと 0042<sub>H</sub> 番地に格納した 16 ビットデータのキャリー付きの加算を行うものです。

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

上の例を実行したときの A と T の動作を図 3-10. に示します。

図 3-10. ワードデータ処理におけるアキュムレータ (A) とテンポラリアキュムレータ (T) の動作例



### 3.3.2 アキュムレータ (A) とテンポラリアキュムレータ (T) のバイトデータの転送・演算

アキュムレータ (A) への転送がバイト単位であるときは転送データを AL に格納しますが、テンポラリデータ (T) への自動転送もバイト単位で行われ、元の AL の内容のみを TL へ格納します。A も T も上位 8 ビットは転送の影響を受けません。また、A-T 間のバイト演算は下位 8 ビットのみを使用し、A も T も上位 8 ビットは演算の影響を受けません。

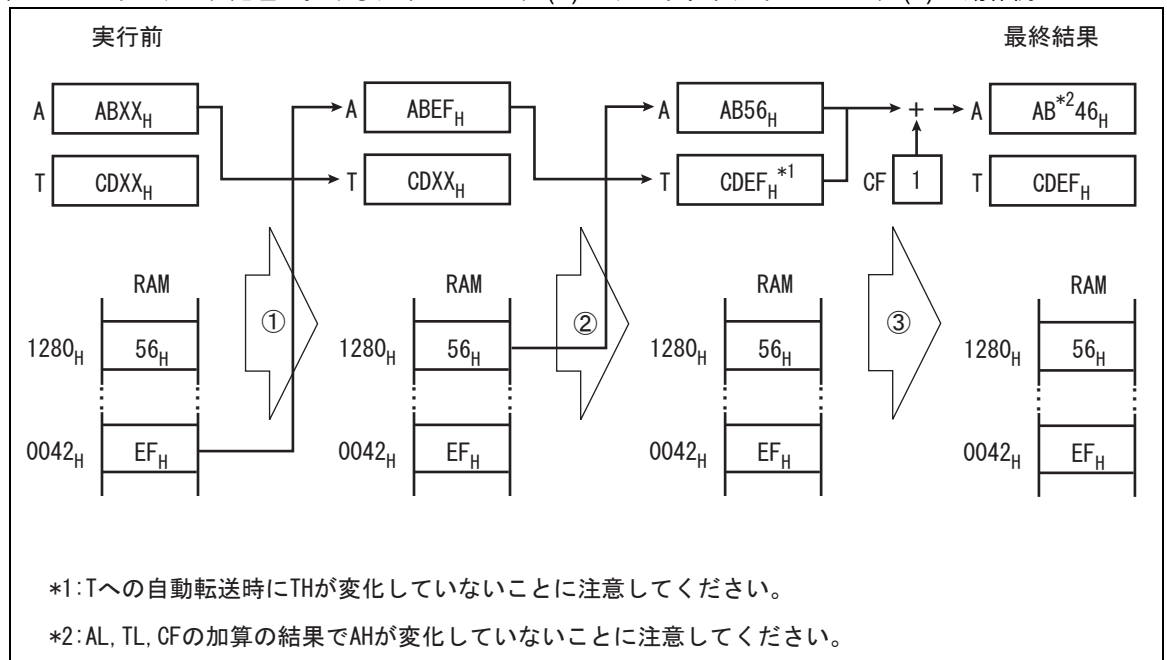
### バイトデータ処理におけるアキュムレータ (A) とテンポラリアキュムレータ (T) の動作例

以下の例は 1280<sub>H</sub> 番地に格納した 8 ビットデータと 0042<sub>H</sub> 番地に格納した 8 ビットデータのキャリー付きの加算を行うものです。

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

上の例を実行したときの A と T の動作を図 3-11. に示します。

図 3-11. バイトデータ処理におけるアキュムレータ (A) とテンポラリアキュムレータ (T) の動作例

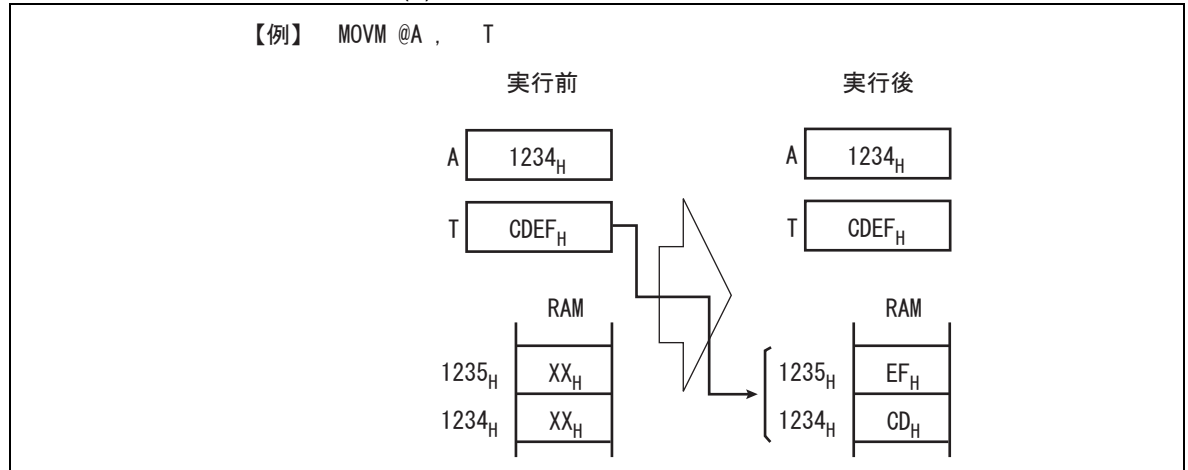




## テンポラリアキュムレータ (T) から直接転送する場合

基本的に、テンポラリアキュムレータ (T) はアキュムレータ (A) の一時記憶用ですので、ここから直接メモリへ転送することはできませんが、例外として A をポインタとして使用することで T の内容をメモリへ退避することができます。図 3-12. にその例を示します。

図 3-12. テンポラリアキュムレータ (T) から直接転送する場合の例



### 3.4 プログラムステータス (PS)

プログラムステータス (PS) は 16 ビット長の用途専用レジスタで、CPU 内にあります。上位バイトのうち、上位 5 ビットはレジスタバンクポインタ、下位 3 ビットはダイレクトバンクポインタです。下位バイトはコンディションコードレジスタとなっています。プログラムステータス (PS) 上位バイト (RP, DP) は、アドレス 0078<sub>H</sub> へのアクセスによりリード・ライトが可能です。

#### プログラムステータス (PS) の構造

図 3-13. に、プログラムステータスの構造を示します。

RP は現在使用しているレジスタバンクのアドレスを示すもので、RP の内容と実アドレスの関係は図 3-14. に示す変換規則になっています。

DP はダイレクトアドレッシングおよびビットダイレクトアドレッシングに使用するメモリ領域 (ダイレクトバンク) を示します。ダイレクトアドレッシングおよびビットダイレクトアドレッシングのオペランドアドレスから実アドレスへの変換は、DP を用いて表 3-1. に示す変換規則で行います。

CCR は演算の結果や転送データの内容を示すビットと、割込み時の CPU の動作を制御するビットがあります。

図 3-13. プログラムステータス (PS) の構造

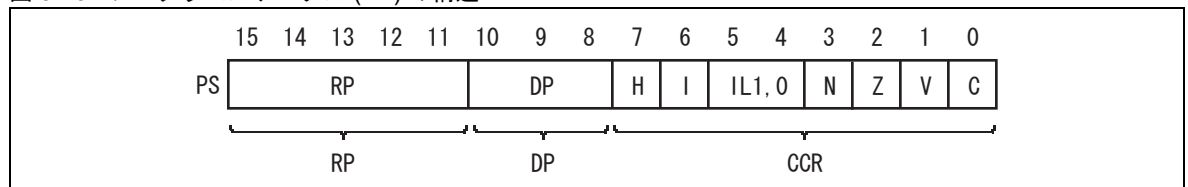


図 3-14. 汎用レジスタ領域の実アドレス変換規則

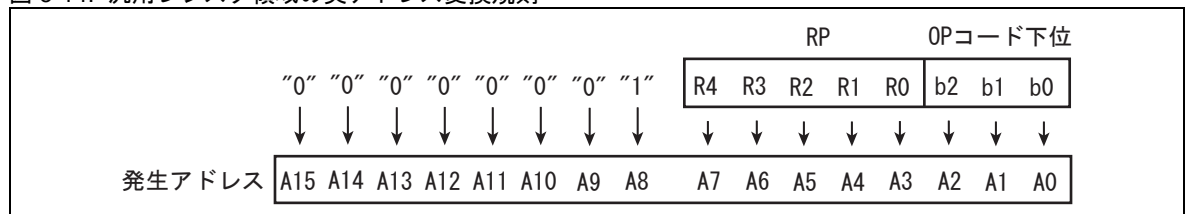


表 3-1. ダイレクトアドレッシングおよびビットダイレクトアドレッシングの実アドレス変換規則

オペランドアドレス	ダイレクトバンクポインタ (DP)	実アドレス
00 <sub>H</sub> ~ 7F <sub>H</sub>		0000 <sub>H</sub> ~ 007F <sub>H</sub>
80 <sub>H</sub> ~ FF <sub>H</sub>	000	0080 <sub>H</sub> ~ 00FF <sub>H</sub>
	001	0100 <sub>H</sub> ~ 017F <sub>H</sub>
	010	0180 <sub>H</sub> ~ 01FF <sub>H</sub>
	011	0200 <sub>H</sub> ~ 027F <sub>H</sub>
	100	0280 <sub>H</sub> ~ 02FF <sub>H</sub>
	101	0300 <sub>H</sub> ~ 037F <sub>H</sub>
	110	0380 <sub>H</sub> ~ 03FF <sub>H</sub>
	111	0400 <sub>H</sub> ~ 047F <sub>H</sub>

## プログラムステータス (PS) のフラグ

以下に、プログラムステータスのフラグについて説明します。

### H フラグ

演算の結果、ビット 3 からビット 4 へのキャリーやボローが発生した場合に 1 になり、それ以外は 0 になります。このフラグは十進補正命令用ですので、加減算以外の用途に使用した場合の動作は保証できません。

### I フラグ

このフラグが 1 のとき割込みを許可し、0 のとき割込みを禁止します。リセット時は 0 となり、割込み禁止状態になります。

### IL1, IL0

現在許可している割込みのレベルを示します。このビットが示す値より少ない値の割込み要求があった場合のみ、割込み処理を行います。

IL1	IL0	割込みレベル	強弱
0	0	0	最強 ↑ ↓ 最弱
0	1	1	
1	0	2	
1	1	3	

## N フラグ

演算の結果、最上位ビットが 1 のときに 1 になり、0 のとき 0 になります。

## Z フラグ

演算の結果、0 であれば 1 になり、それ以外のとき 0 になります。

## V フラグ

演算の結果、2 の補数のオーバーフローが発生したときに 1 になり、発生しなかったとき 0 になります。

## C フラグ

演算の結果、バイト時にはビット 7 から、ワード時にはビット 15 からキャリーやボローが発生した場合に 1 になり、それ以外は 0 になります。また、シフト命令ではシフトアウトした値になります。

## レジスタバンクポインタ・ダイレクトバンクポインタへのアクセス

アドレス 0078<sub>H</sub> は、プログラムステータス (PS) の上位バイト (PR, DP) のミラーアドレスです。レジスタバンクポインタおよびダイレクトバンクポインタは、PS を操作する命令 (MOVW A,PS または MOVW PS,A) 以外に 0078<sub>H</sub> 番地へのアクセスによりリード・ライトが可能です。

### 3.5 インデックスレジスタ (IX)/ エクストラポインタ (EP)

インデックスレジスタ (IX), エクストラポインタ (EP) は 16 ビット長の用途専用レジスタで CPU 内にあります。

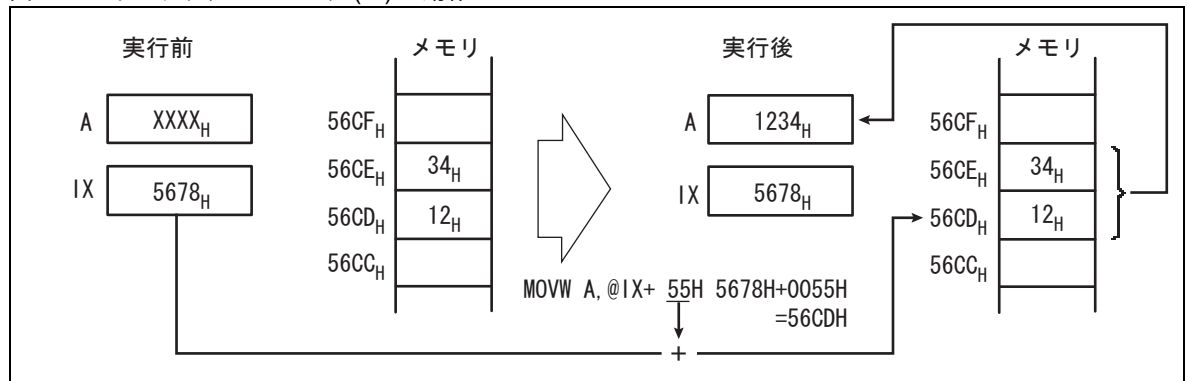
インデックスレジスタ (IX) は, 8 ビット長のオフセット値を符号付き加算することで, オペランドの格納しているアドレスを生成します。

エクストラポインタ (EP) は, オペランドの格納しているアドレスを示します。

#### インデックスレジスタ (IX)

図 3-15. にインデックスレジスタの動作について示します。

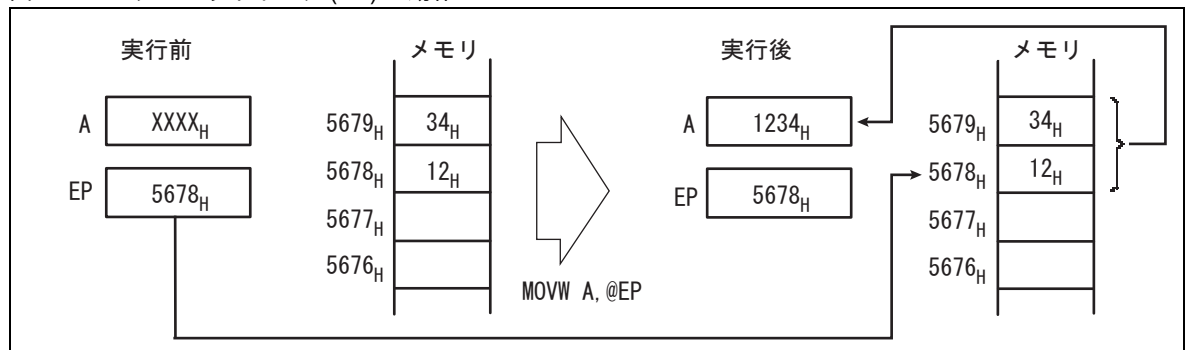
図 3-15. インデックスレジスタ (IX) の動作



#### エクストラポインタ (EP)

図 3-16. にエクストラポインタの動作について示します。

図 3-16. エクストラポインタ (EP) の動作



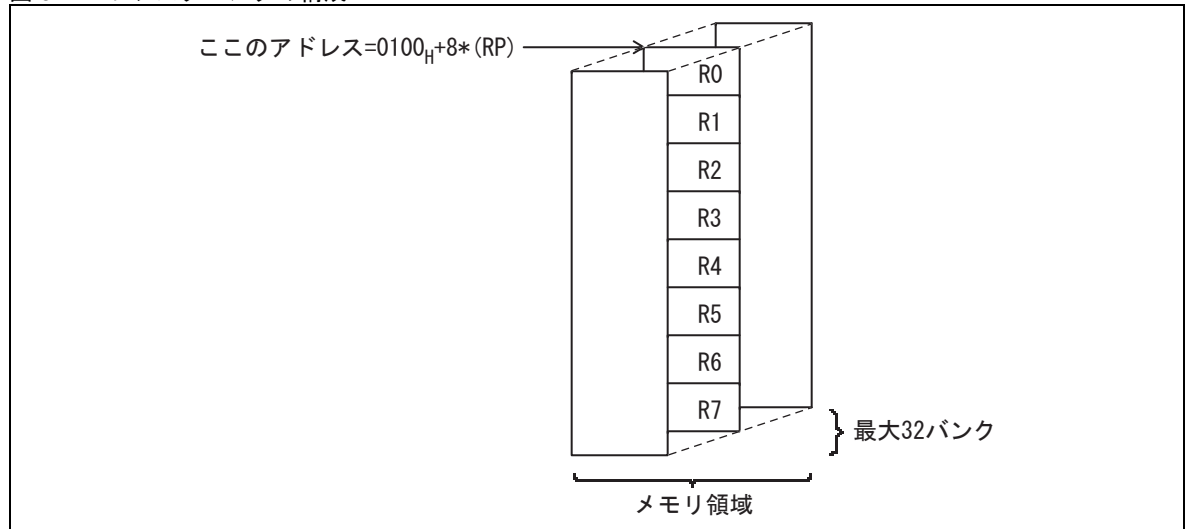
## 3.6 レジスタバンク

レジスタバンクは 8 ビット長の汎用レジスタで、メモリ上のレジスタバンク内にあります。1 バンクあたり 8 個のレジスタがあり、全部で 32 バンクまで拡張することができます。現在使用しているバンクはレジスタバンクポインタ (RP) で示します。

### レジスタバンクレジスタ

図 3-17. にレジスタバンクの構成を示します。

図 3-17. レジスタバンクの構成



### 3.7 ダイレクトバンク

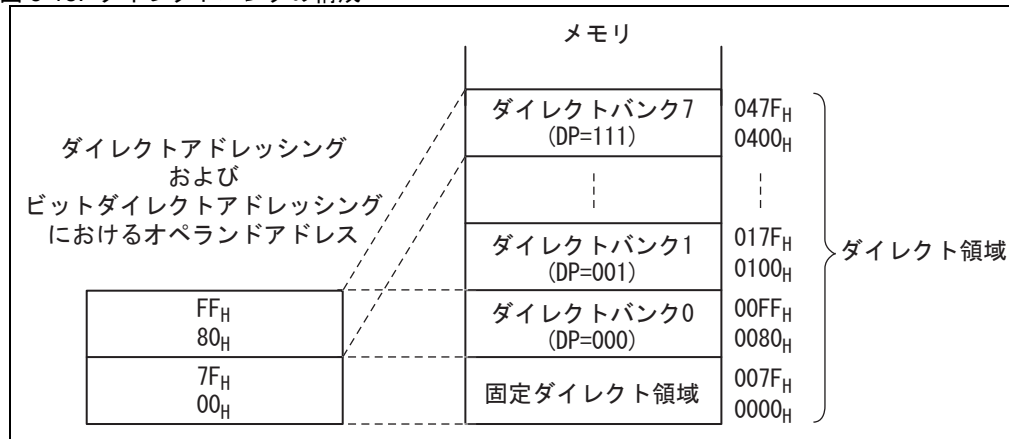
ダイレクトバンクはダイレクト領域のうち 0080<sub>H</sub> ~ 047F<sub>H</sub> にあり, 128 バイト × 8 のバンクで構成されます。オペランドアドレス 80<sub>H</sub> ~ FF<sub>H</sub> のダイレクトアドレッシングおよびビットダイレクトアドレッシングを用いたアクセスは, ダイレクトバンクポインタ (DP) の値により 8 つのダイレクトバンクへ拡張することができます。現在使用しているダイレクトバンクはダイレクトバンクポインタ (DP) で示します。

#### ダイレクトバンク

図 3-18. にダイレクトバンクの構成を示します。

オペランドアドレス 80<sub>H</sub> ~ FF<sub>H</sub> のダイレクトアドレッシングおよびビットダイレクトアドレッシングを用いたアクセスは, ダイレクトバンクポインタ (DP) の値により 8 つのダイレクトバンクへ拡張することができます。オペランドアドレス 00<sub>H</sub> ~ 7F<sub>H</sub> のダイレクトアドレッシングおよびビットダイレクトアドレッシングを用いたアクセスはダイレクトバンクポインタ (DP) の影響を受けず, 0000<sub>H</sub> ~ 007F<sub>H</sub> の固定ダイレクト領域へアクセスします。

図 3-18. ダイレクトバンクの構成



## 4. 割込み処理



この章では、F<sup>2</sup>MC-8FX の割込み処理の機能と動作について説明します。

- 4.1 割込み動作概要
- 4.2 割込み許可 / 禁止 / 優先順位機構
- 4.3 割込み処理プログラムの作成方法
- 4.4 多重割込み
- 4.5 リセット動作





## 4.2 割込み許可 / 禁止 / 優先順位機構

F<sup>2</sup>MC-8FX では割込み要求を以下の 3 種類の許可 / 禁止機構を経由して CPU へ伝達します。

- 周辺の割込み許可フラグによる要求許可チェック
- 割込みレベル判定部によるレベルチェック
- CPU の I フラグによる割込み起動のチェック

周辺で発生した割込みは、割込み優先順位機構により優先するレベルを決定し CPU へ伝達します。

### 割込み許可 / 禁止 / 優先順位機構

- 周辺の割込み許可フラグによる要求許可チェック

割込み発生元からの要求の許可 / 禁止を行う機構です。周辺の割込み許可フラグが許可になっていれば周辺から割込みコントローラへ割込み要求信号を伝達します。

この機能は周辺単位で割込みのありなしを制御するためのもので、周辺の動作の単位でソフトウェアを記述した場合に、ほかの周辺割込みの許可 / 禁止に注意する必要がないため特に有効です。

- 割込みレベル判定部によるレベルチェック

この部分は割込みのレベル判定を行う部分です。周辺が発生した割込みに対応した割込みレベルを CPU 内の IL ビットと比較し、値が IL ビットより少なければ割込み要求を起こす資格があると判定します。複数ある割込みの間に優先順位を付けたい場合に有効です。

- CPU の I フラグによる割込み起動のチェック

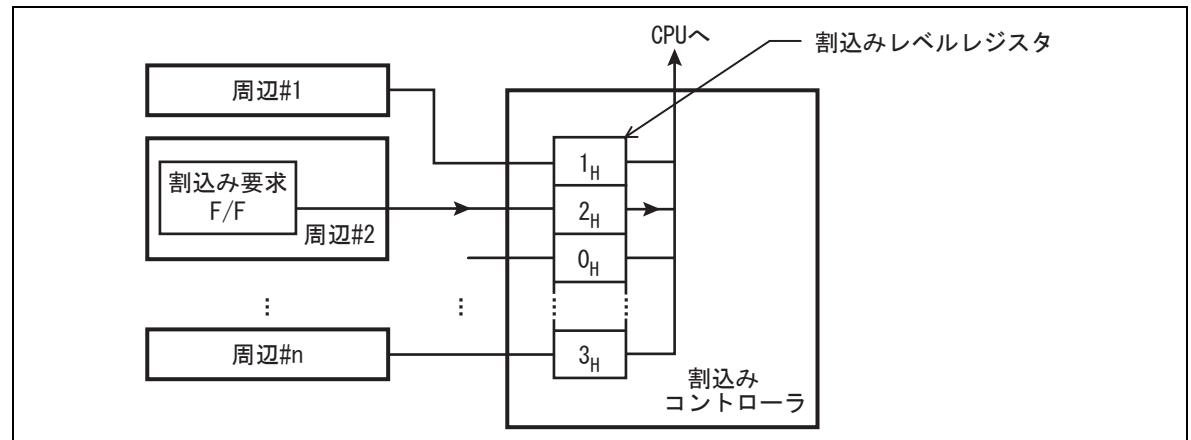
I フラグは割込み全体に関する許可 / 禁止を行う機構です。割込み要求があつて、かつ CPU の I フラグが割込み許可に設定してあれば、CPU は命令実行の流れを一時中断して割込み処理を行います。割込み全体を一時禁止する際に有効です。

### 周辺の割込み要求

周辺で発生した割込みは、図 4-2. で示すように割込みコントローラの内部でその割込みに対応する割込みレベルレジスタにソフトウェアで設定した値に変換されて CPU へ伝達します。

割込みのレベルは数値の低い方が高いレベル、数値の高い方が低いレベルと定義されています。

図 4-2. 周辺の割込み要求と割込みレベルの関係



### 4.3 割込み処理プログラムの作成方法

F<sup>2</sup>MC-8FX で用いる周辺の割込み要求は、要求の発生はハードウェアで、要求のクリアはソフトウェアで行うことが基本となっています。

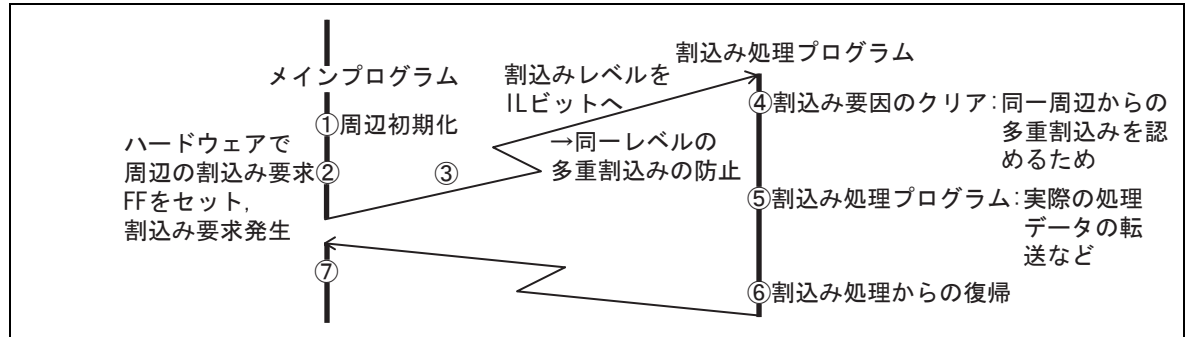
#### 割込み処理プログラムの作成方法

割込み処理のための制御の流れは以下のようになります。

1. 周辺の初期化を行って周辺を動作させます。
2. 割込みが発生するのを待ちます。
3. 割込みが発生すると、割込みを受け付ける状態になっていれば割込み処理を行い、割込み処理ルーチンへ分岐します。
4. 割込み処理ルーチンの先頭で割込み要因をクリアするようにソフトウェアを組みます。これは割込みを起こした周辺が割込み処理プログラム中で再割込みを発生するような用途のためです。
5. 割込み処理を行って必要なデータ転送を行います。
6. 割込み復帰命令を使用して割込み処理から復帰します。
7. 再び割込みが発生するまでメインプログラムを実行し続けます。

標準的な割込み処理の流れは図 4-3. のようになります。番号は上で示したものに对应します。

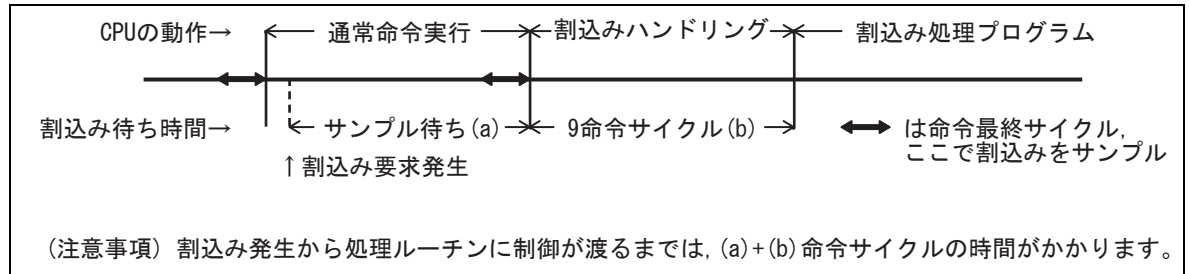
図 4-3. 割込み処理の流れ



割込み発生を受付けから処理プログラムへ制御が渡るまでの時間 ( 図 4-3. の例ですと③の部分 ) は 9 命令サイクルです。さらに、割込みを受け付ける点は各命令の最終サイクルだけですので、割込みが発生してからその割込み処理ルーチンに制御が渡るまでは図 4-4. に示す時間がかかります。

最も長いものは DIVU 命令実行開始直後の割込み要求発生の場合で、このときは 17+9=26 命令サイクルかかることになります。

図 4-4. 割込み応答時間



## 4.4 多重割込み

F<sup>2</sup>MC-8FX はマスク可能な割込みに最大 4 種のレベルをもつことができます。これを利用することで周辺などの割込みに優先順位を付けることができます。

### 多重割込み

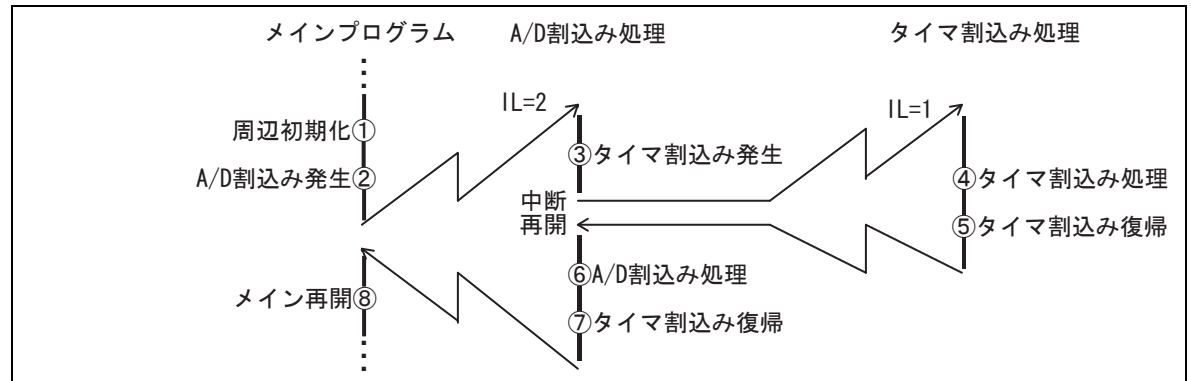
多重割込みの具体例を以下に示します。

□ A/D コンバータよりタイマの割込みを優先させたいとき

START	MOV	ADIL,	#2	A/D コンバータの割込みレベルを 2 にします
	MOV	TMIL,	#1	タイマの割込みレベルを 1 にします
				ADIL も TMIL も割込みコントローラ内の IL ビットです
	CALL	STAD		A/D コンバータを起動させます
	CALL	STTM		タイマを起動させます
	.			
	.			
	.			

というプログラムを起動すると、時間が経つと A/D コンバータとタイマから割込みが発生します。このとき、A/D コンバータの割込み処理中にタイマ割込みが発生すると、図 4-5. のようなシーケンスで処理を行います。

図 4-5. 多重割込み例



A/D 割込み処理開始時に CPU の PS レジスタ内の IL ビットが自動的に要求と同じ値（ここでは 2）になりますので、A/D の割込み要求を落とさなくとも A/D 割込み処理中はレベル 1 あるいはレベル 0 の割込み要求が発生すると、この割込み処理を優先して行います。また、A/D 割込みの一部で自分以上の優先順位の割込みさえも一時的に禁止したいときは、CPU の PS レジスタ内の I フラグを割込み禁止に設定するか、IL ビットを 0 にすることもできます。

各割込み処理が終了し、復帰命令で割り込まれた処理へ制御を返すと、PS レジスタはスタック内に退避しておいた値になりますので、IL ビットの値は中断前の値となります。

実際のコーディングの際には各品種のハードウェアマニュアルを参照し、割込みコントローラや各種周辺のアドレスやサポートする割込みの種類などを確認してください。

## 4.5 リセット動作

F<sup>2</sup>MC-8FX はリセットが発生するとプログラムステータスのフラグを 0, IL ビットを 11 にし, 解除するとリセットベクタ : FFFE<sub>H</sub>, FFFF<sub>H</sub> に書かれた開始アドレスからリセット処理を実行します。

### リセット動作

リセットによる影響は以下のとおりです。

- アキュムレータ, テンポラリアキュムレータ : 0000<sub>H</sub> に初期化
- スタックポインタ : 0000<sub>H</sub> に初期化
- エクストラポインタ, インデックスレジスタ : 0000<sub>H</sub> に初期化
- プログラムステータス : フラグは 0, IL ビットは 11, RP ビットは 00000, DP ビットは 000 に初期化
- プログラムカウンタ : リセットベクタの値
- RAM (汎用レジスタを含む) : リセット前の値を保存
- 周辺 : 基本的に停止
- そのほか : 各端子の状況は各品種のマニュアルを参照してください。

そのほか, 特殊なリセット発生条件に対する各レジスタの値, 動作の継続などの詳細は各品種のマニュアルを参照してください。

## 5. CPU ソフトウェアアーキテクチャ



この章では、F<sup>2</sup>MC-8FX CPU の命令について説明します。

5.1 アドレッシングの種類

5.2 特殊な命令

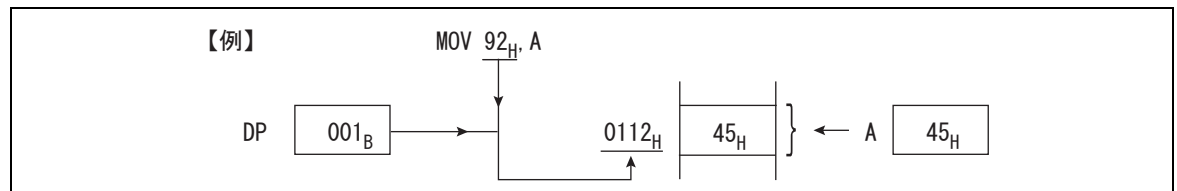
## 5.1 アドレッシングの種類

F<sup>2</sup>MC-8FX には以下に述べる 10 種のアドレッシングがあります。

- ダイレクトアドレッシング (dir)
- エクステンダドレッシング (ext)
- ビットダイレクトアドレッシング (dir:b)
- インデックスアドレッシング (@IX+off)
- ポインタアドレッシング (@EP)
- 汎用レジスタアドレッシング (Ri)
- イミディエートアドレッシング (#imm)
- ベクタアドレッシング (#k)
- 相対アドレッシング (rel)
- インヘレントアドレッシング

### ダイレクトアドレッシング (dir)

命令表中で "dir" と示したアドレッシングで、ダイレクト領域 0000<sub>H</sub> ~ 047F<sub>H</sub> をアクセスする際に使用します。このアドレッシングでは、オペランドアドレスが 00<sub>H</sub> ~ 7F<sub>H</sub> の場合、0000<sub>H</sub> ~ 007F<sub>H</sub> にアクセスします。また、オペランドアドレスが 80<sub>H</sub> ~ FF<sub>H</sub> の場合、ダイレクトバンクポインタ DP の設定により 0080<sub>H</sub> ~ 047F<sub>H</sub> にアクセスがマッピングできます。

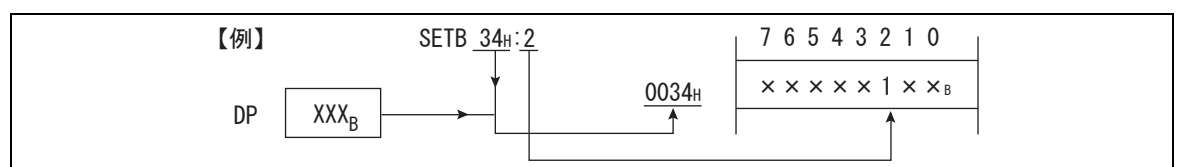


### エクステンダドレッシング (ext)

命令表中で "ext" と示したアドレッシングで、64K バイト全体の領域をアクセスする際に使用します。このアドレッシングでは第 1 オペランドでアドレスの上位 1 バイトを、第 2 オペランドでアドレスの下位 1 バイトを指定します。

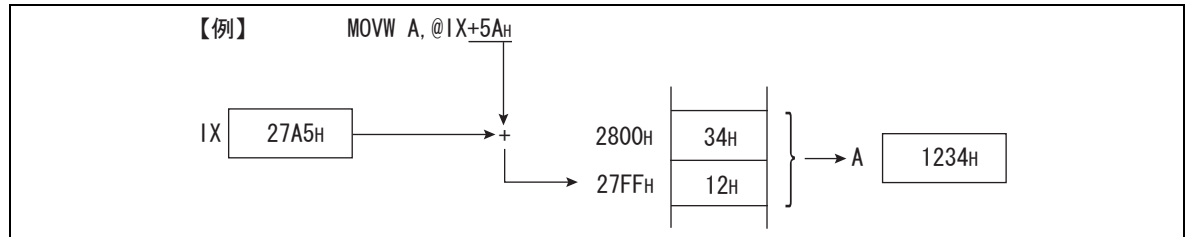
### ビットダイレクトアドレッシング (dir:b)

命令表中で "dir:b" と示したアドレッシングで、ダイレクト領域 0000<sub>H</sub> ~ 047F<sub>H</sub> をビット単位でアクセスする際に使用します。このアドレッシングでは、オペランドアドレスが 00<sub>H</sub> ~ 7F<sub>H</sub> の場合、0000<sub>H</sub> ~ 007F<sub>H</sub> にアクセスします。また、オペランドアドレスが 80<sub>H</sub> ~ FF<sub>H</sub> の場合、ダイレクトバンクポインタ DP の設定により 0080<sub>H</sub> ~ 047F<sub>H</sub> にアクセスがマッピングできます。指定したアドレス内のビットの位置は命令コードの下位 3 ビットの値で指定します。



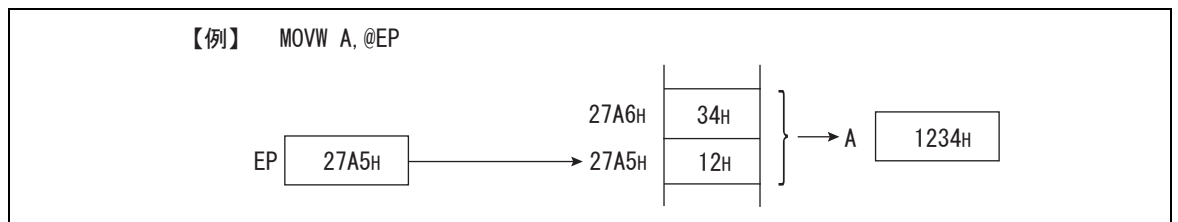
## インデックスアドレッシング (@IX+off)

命令表中で "@IX+off" と示したアドレッシングで、64K バイト全体の領域をアクセスする際に使用します。このアドレッシングでは第 1 オペランドの内容を符号拡張した上でインデックスレジスタ IX に加算してその結果をアドレスとします。



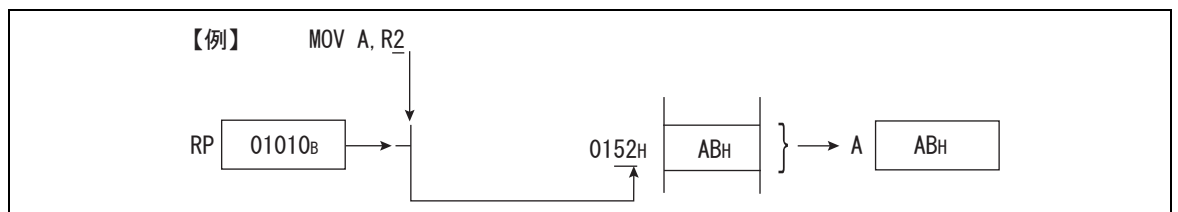
## ポインタアドレッシング (@EP)

命令表中で "@EP" と示したアドレッシングで、64K バイト全体の領域をアクセスする際に使用します。このアドレッシングではエクストラポインタ EP の内容をアドレスとします。



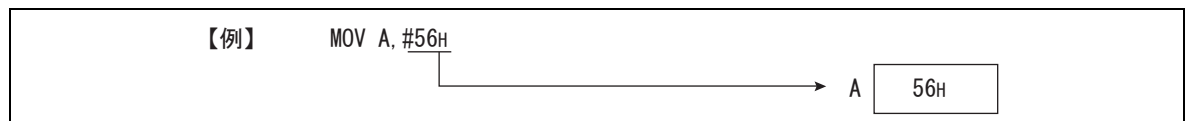
## 汎用レジスタアドレッシング (Ri)

命令表中で "Ri" と示したアドレッシングで、レジスタバンク領域をアクセスする際に使用します。このアドレッシングではアドレスの上位 1 バイトは "01" に固定し、下位 1 バイトをレジスタバンクポインタ RP の内容と命令の下位 3 ビットから作成し、このアドレスに対してアクセスを行います。



## イミディエートアドレッシング (#imm)

命令表中で "#imm" と示したアドレッシングで、即値データを必要とする際に使用します。このアドレッシングではオペランドがそのまま即値データになります。バイト / ワードの指定は命令コードにより決まります。



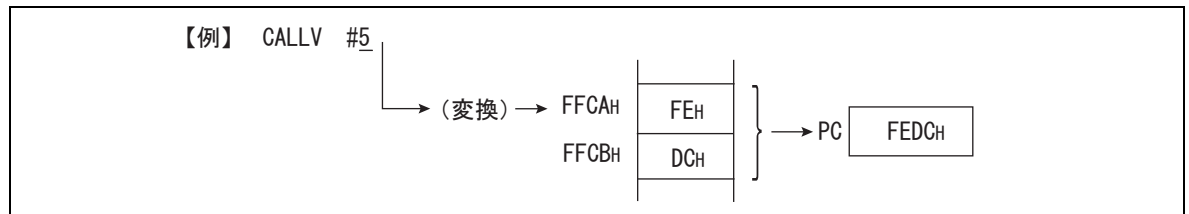


## ベクタアドレッシング (#k)

命令表中で "#k" と示したアドレッシングで、テーブル内に登録したサブルーチンアドレスに分岐するときに使用します。このアドレッシングでは命令コード内に "#k" の情報を含み、表 5-1. に示す対応でテーブルのアドレスを作成します。

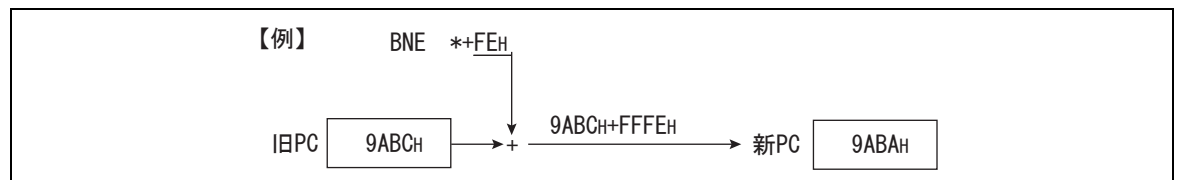
表 5-1. ジャンプ先アドレステーブル

#k	アドレステーブル (ジャンプ先上位アドレス : 下位アドレス)
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>



## 相対アドレッシング (rel)

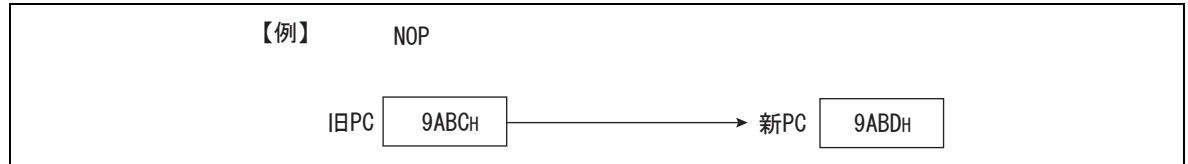
命令表の中で "rel" と示したアドレッシングで、プログラムカウンタ PC の前後 128 バイトの領域に分岐する際に使用します。このアドレッシングではオペランドの内容を PC に符号付きで加算し、その結果を PC に格納します。



この例では結局 BNE の命令コードが格納されているアドレスへジャンプするので、結果として無限ループになる。

## インヘレントアドレッシング

命令表の中でオペランドを持たないアドレッシングで、命令コードで決まる動作を行う際に使用します。このアドレッシングでは動作が命令ごとに異なります。



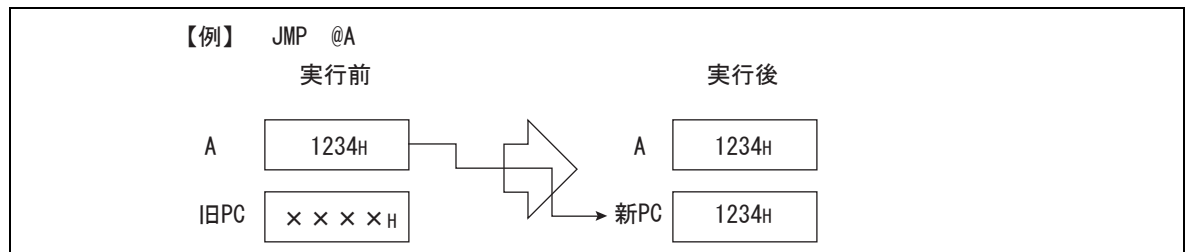
## 5.2 特殊な命令

F<sup>2</sup>MC-8FX には、以下の 6 つの特殊な命令があります。

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

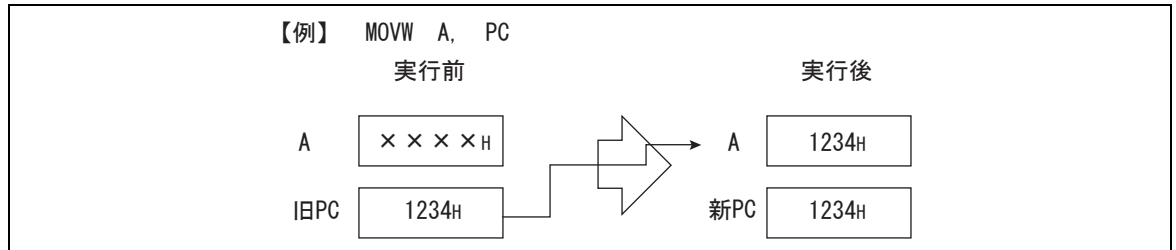
### JMP @A

この命令はアキュムレータ A の内容をアドレスとしてそこへ分岐するというものです。N 個のジャンプ先をテーブル状に並べておき、その内容のいずれか 1 つを選択して A に転送し、この命令を実行することで N 分岐処理が行えます。



## MOVW A, PC

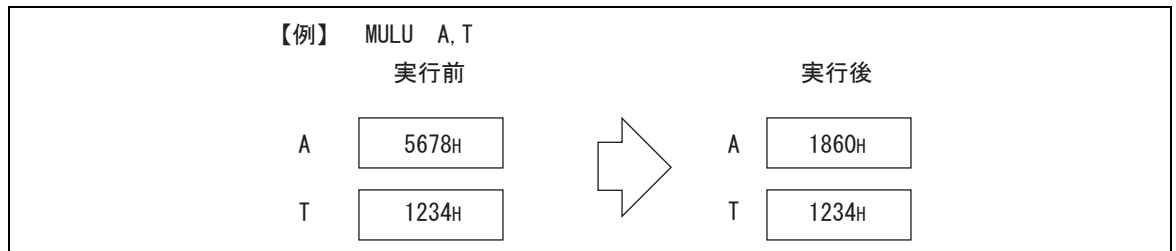
この命令は「JMP @A」と反対の動作を行うものです。すなわち、PCの内容をアキュムレータ A に格納するものです。メインルーチン内でこの命令を実行しておき、特定のサブルーチン呼び出すような設定において、そのサブルーチン内で A の内容が決められた値になっていることを確認することで、予想以外の部分からの分岐でないことが識別でき、暴走判断に使用することができます。



この命令を実行したときの A の内容はこの命令のオペコードが格納されているアドレスではなく、次の命令が格納されているアドレスと同じ値になります。したがって、上の例では A に格納した値「1234<sub>H</sub>」は「MOVW A, PC」の次のオペコードが格納されているアドレスに一致します。

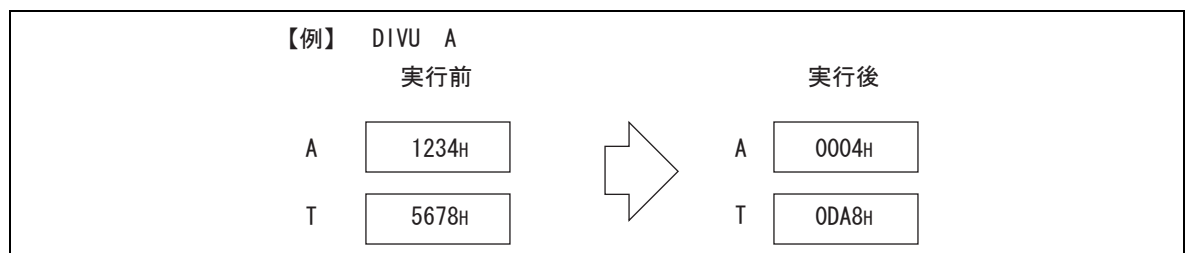
## MULU A

この命令は AL の 8 ビットと TL の 8 ビットを符号なしで掛け合わせ、16 ビット長の結果を A に格納します。T の内容は変化しません。演算に関して、元の AH, TH の内容は使用していません。フラグは変化しませんので、乗算の結果によって分岐するなどのときには注意が必要です。



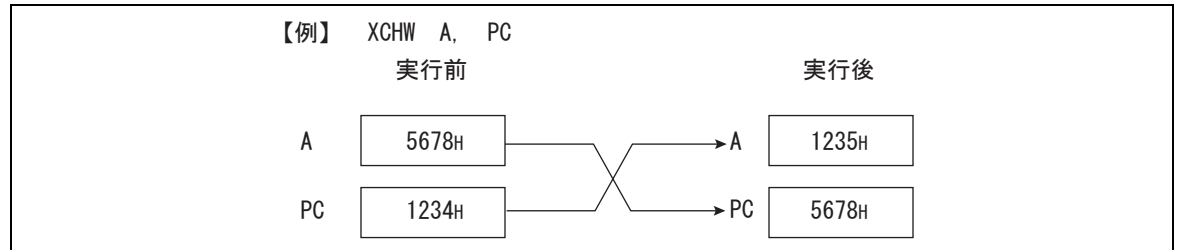
## DIVU A

この命令は T の 16 ビットを A の 16 ビットで符号なしデータとして割り、結果を 16 ビットとして A に、余りも 16 ビットとして T に格納するものです。A が 0000<sub>H</sub> の場合、0 除算として Z フラグが 1 になります。このときの演算結果は、保証されません。



## XCHW A, PC

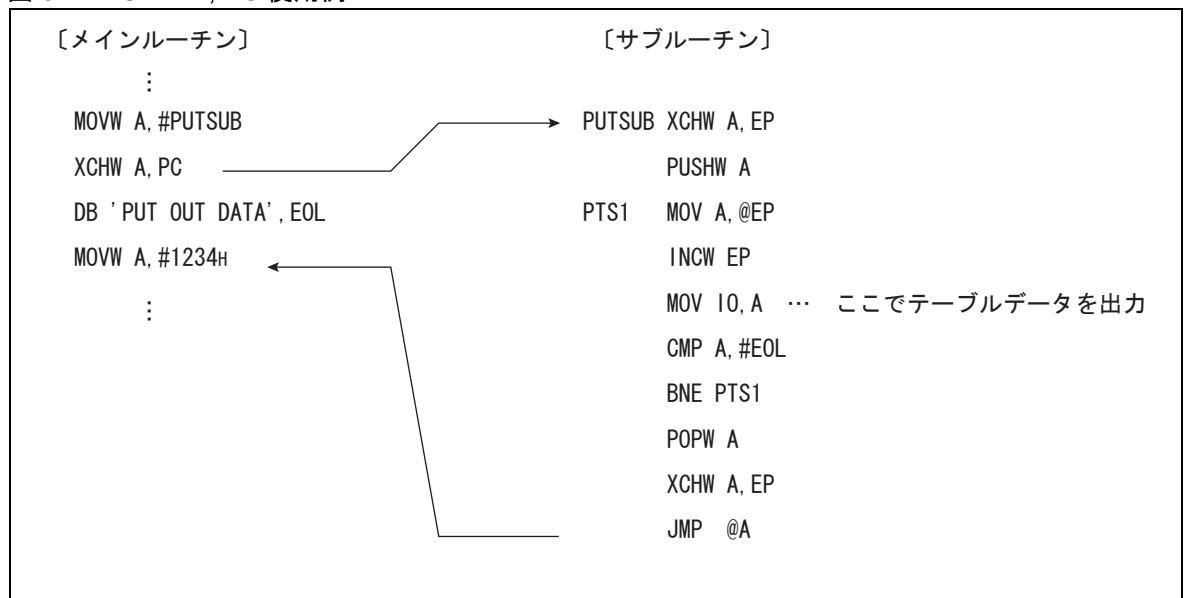
この命令はアキュムレータ A とプログラムカウンタ PC の内容を交換するもので、結果として元の A の内容が示す番地へ分岐し、現 A は「XCHW A, PC」の命令コードが格納されているアドレスの次のアドレスの値になります。この命令は特にメインルーチンでテーブルを指定し、サブルーチンでそれを使用する場合に有効です。



この命令を実行したときの A の内容はこの命令の命令コードが格納されているアドレスではなく、次の命令が格納されているアドレスと同じ値になります。したがって、上の例では A の値「1235<sub>H</sub>」は「XCHW A, PC」の次の命令のコードが格納されているアドレスに一致します。そのため、「1234<sub>H</sub>」でなく、「1235<sub>H</sub>」となっています。

アセンブラ表記上ではこの命令は図 5-1. のような使い方をします。

図 5-1. XCHW A, PC 使用例

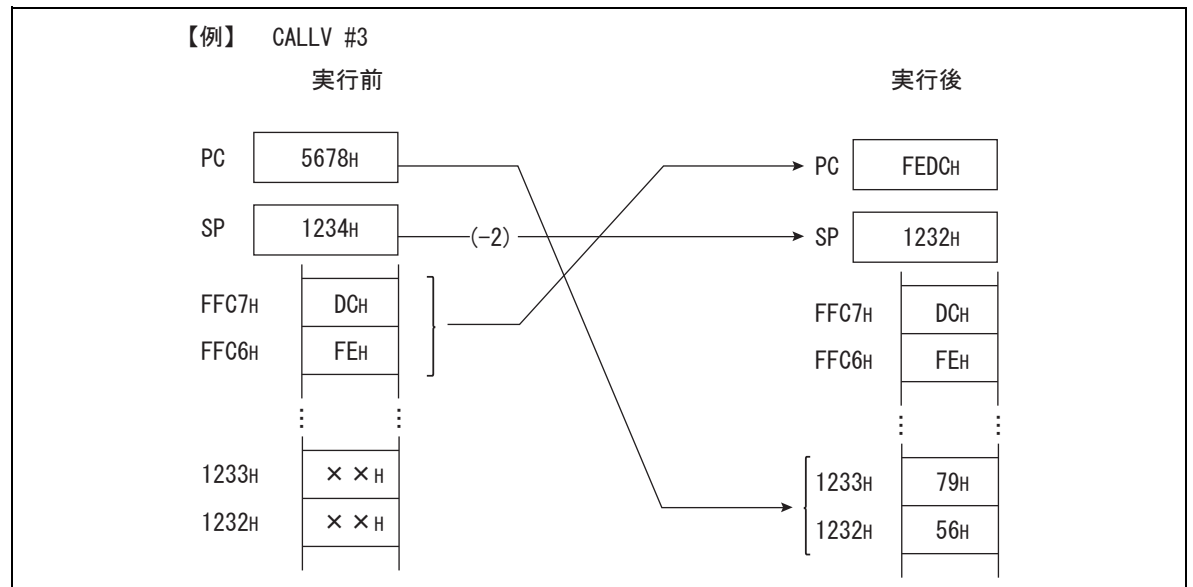


## CALLV #k

テーブル内に登録したサブルーチンアドレスに分岐するときに使用します。このアドレッシングでは命令コード内に "#k" の情報を含み、表 5-2. に示す対応でテーブルのアドレスを作成します。現在のプログラムカウンタの内容をスタックへ退避したのち、テーブルに記載したアドレスへ分岐します。1 バイトの命令ですので、頻繁に使用するサブルーチンに対してこの命令を使用することで、プログラム全体のサイズを縮小することができます。

表 5-2. ジャンプ先アドレステーブル

#k	アドレステーブル (ジャンプ先上位アドレス : 下位アドレス)
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. 実行命令細則



この章では、アセンブラで使用する各実行命令について、リファレンス形式で説明します。

なお、各実行命令は、アルファベット順に掲載します。

実行命令ごとに説明されている各項目の概要や記号（略称）の意味などについては、「5. CPU ソフトウェアアーキテクチャ」を参照してください。

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

AL と TL のバイトデータを加算して、さらに最下位ビットにキャリーを加算し、その結果を AL に戻します。また、AH は変化しません。

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

オペレーション

$(AL) \leftarrow (AL) + (TL) + (C)$  ( バイト加算 , キャリー付き )

アセンブラ形式

ADDC A

コンディションコード (CCR)

N	Z	V	C
+	+	+	+

+: 命令実行により変化します

-: 変化しません

N: 演算結果の AL が MSB=1 ならば 1 になり、それ以外は 0 になります

Z: 演算結果が 00<sub>H</sub> ならば 1 になり、それ以外は 0 になります

V: 演算の結果オーバーフローが発生したときに 1 になり、それ以外は 0 になります

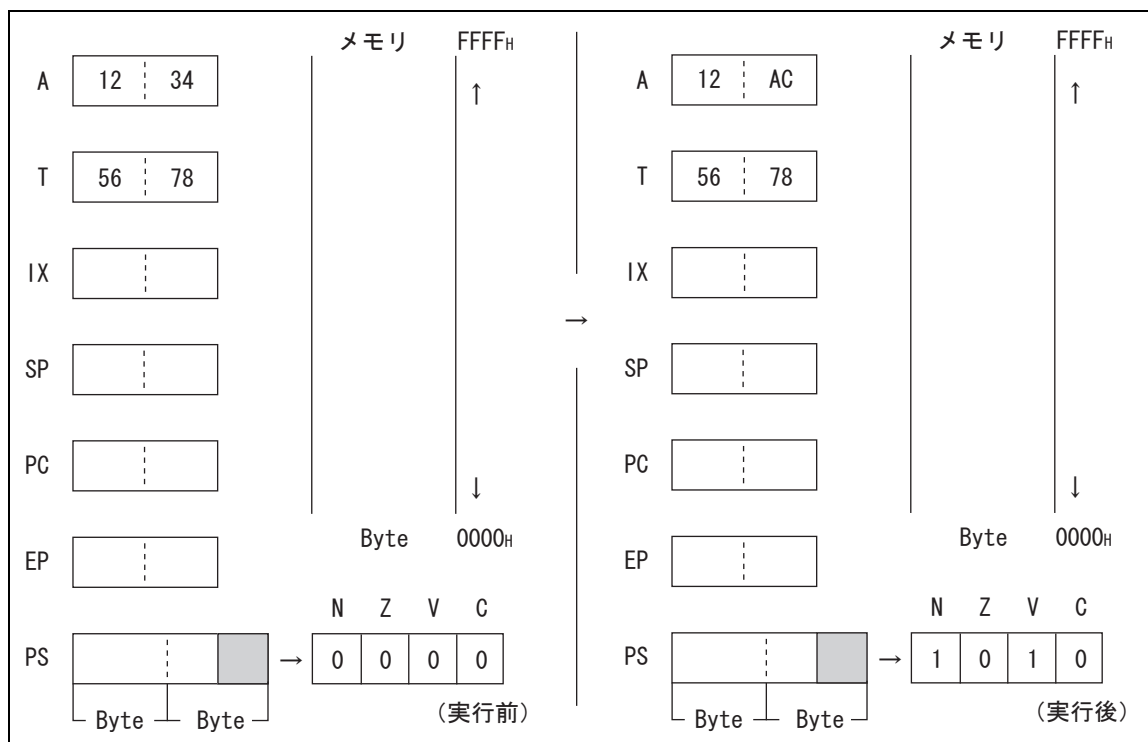
C: 演算の結果キャリーが発生したときに 1 になり、それ以外は 0 になります

実行サイクル数: 1

バイト数: 1

オペコード: 22

実行例 : ADDC A





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

AL と EA メモリ ( 各種アドレッシングで表現されるメモリ ) のバイトデータを加算して , さらに最下位ビットにキャリーを加算し , その結果を AL に戻します。また , AH は変化しません。

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

オペレーション

$(AL) \leftarrow (AL) + (EA) + (C)$  ( バイト加算 , キャリー付き )

アセンブラ形式

ADDC A, EA

コンディションコード (CCR)

N	Z	V	C
+	+	+	+

+: 命令実行により変化します

-: 変化しません

N: 演算結果の AL が MSB=1 ならば 1 になり , それ以外は 0 になります

Z: 演算結果が 00<sub>H</sub> ならば 1 になり , それ以外は 0 になります

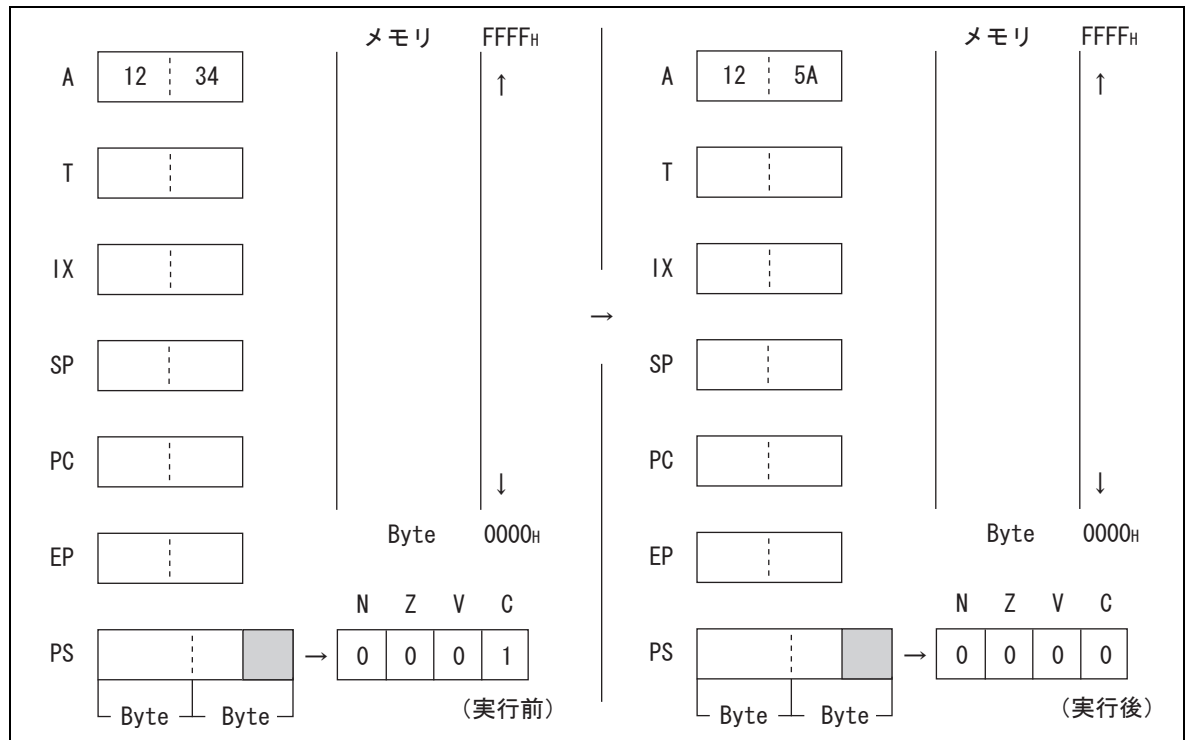
V: 演算の結果オーバーフローが発生したときに 1 になり , それ以外は 0 になります

C: 演算の結果キャリーが発生したときに 1 になり , それ以外は 0 になります

表 6-1. 実行サイクル数 / バイト数 / オペコード

EA	#d8	dir	@IX+off	@EP	Ri
実行サイクル数	2	3	3	2	2
バイト数	2	2	2	1	1
オペコード	24	25	26	27	28 ~ 2F

実行例 : ADDC A, #25H



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

A と T のワードデータを加算して、さらに最下位ビットにキャリーを加算し、その結果を A に戻します。

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

オペレーション

$(A) \leftarrow (A) + (T) + (C)$  (ワード加算, キャリー付き)

アセンブラ形式

ADDCW A

コンディションコード (CCR)

N	Z	V	C
+	+	+	+

+: 命令実行により変化します

-: 変化しません

N: 演算結果の A が MSB=1 ならば 1 になり、それ以外は 0 になります

Z: 演算結果が 0000<sub>H</sub> ならば 1 になり、それ以外は 0 になります

V: 演算の結果オーバーフローが発生したときに 1 になり、それ以外は 0 になります

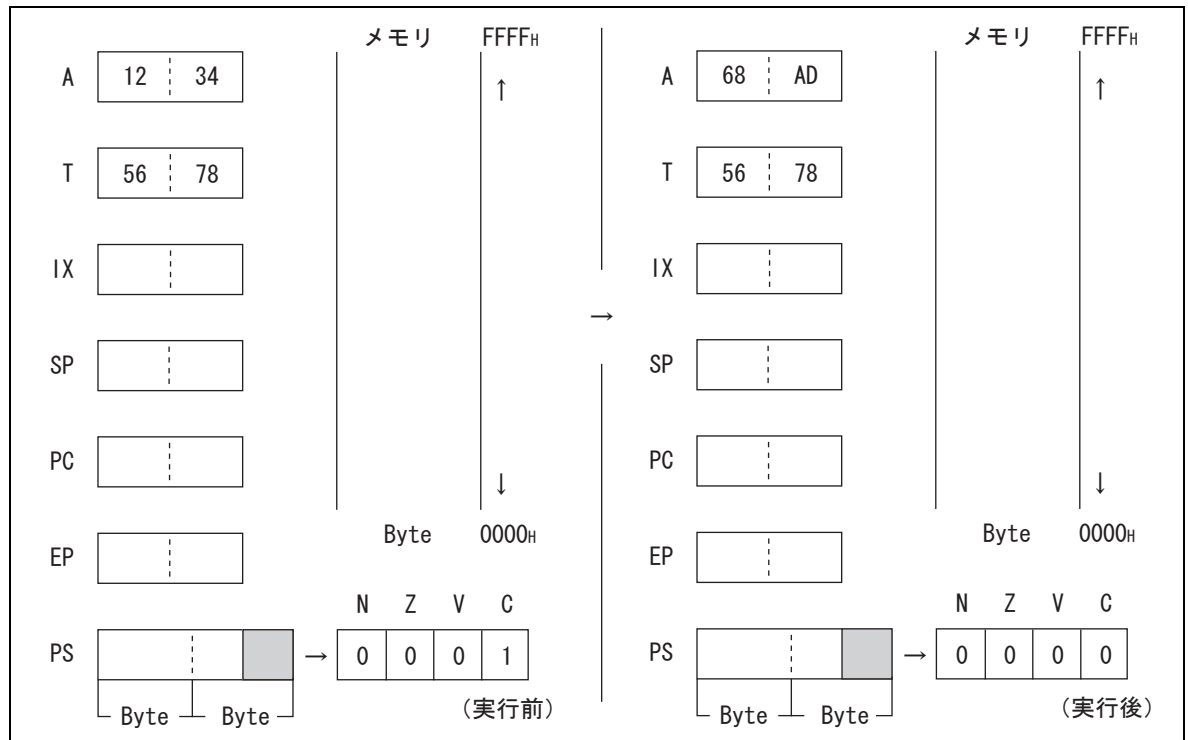
C: 演算の結果キャリーが発生したときに 1 になり、それ以外は 0 になります

実行サイクル数: 1

バイト数: 1

オペコード: 23

実行例 : ADDCW A



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

AL のバイトデータと TL のバイトデータをビットごとに論理積をとり、結果を AL に戻します。  
 AH は、変化しません。

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

オペレーション

$(AL) \leftarrow (AL) \wedge (TL)$  (バイト論理積)

アセンブラ形式

AND A

コンディションコード (CCR)

N	Z	V	C
+	+	R	-

+: 命令実行により変化します

-: 変化しません

R: 命令実行により 0 になります

N: 演算結果の AL が MSB=1 ならば 1 になり、それ以外は 0 になります

Z: 演算結果が 00<sub>H</sub> ならば 1 になり、それ以外は 0 になります

V: 常に 0 になります

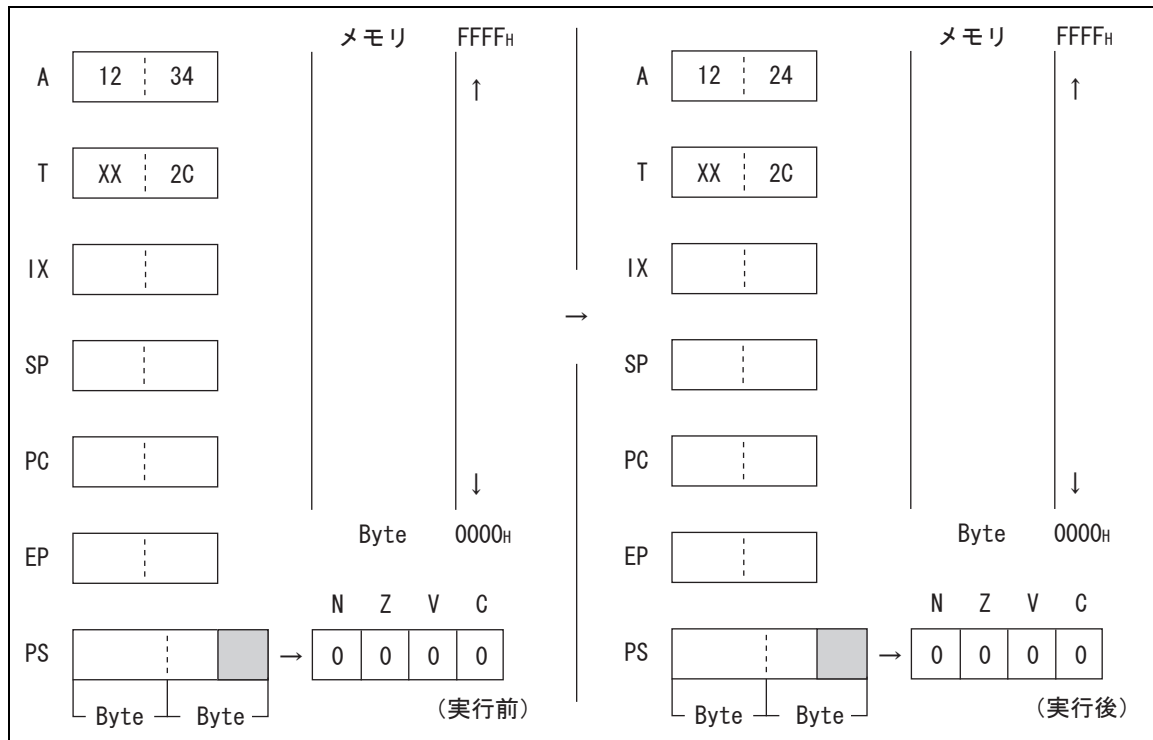
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: 62

実行例 : AND A



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

AL のバイトデータと EA メモリ ( 各種アドレッシングで表現されるメモリ ) のバイトデータをビットごとに論理積をとり、結果を AL に戻します。AH は、変化しません。

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

オペレーション

$(AL) \leftarrow (AL) \wedge (EA)$  ( バイト論理積 )

アセンブラ形式

AND A, EA

コンディションコード (CCR)

N	Z	V	C
+	+	R	-

+: 命令実行により変化します

-: 変化しません

R: 命令実行により 0 になります

N: 演算結果の AL が MSB=1 ならば 1 になり、それ以外は 0 になります

Z: 演算結果が  $00_H$  ならば 1 になり、それ以外は 0 になります

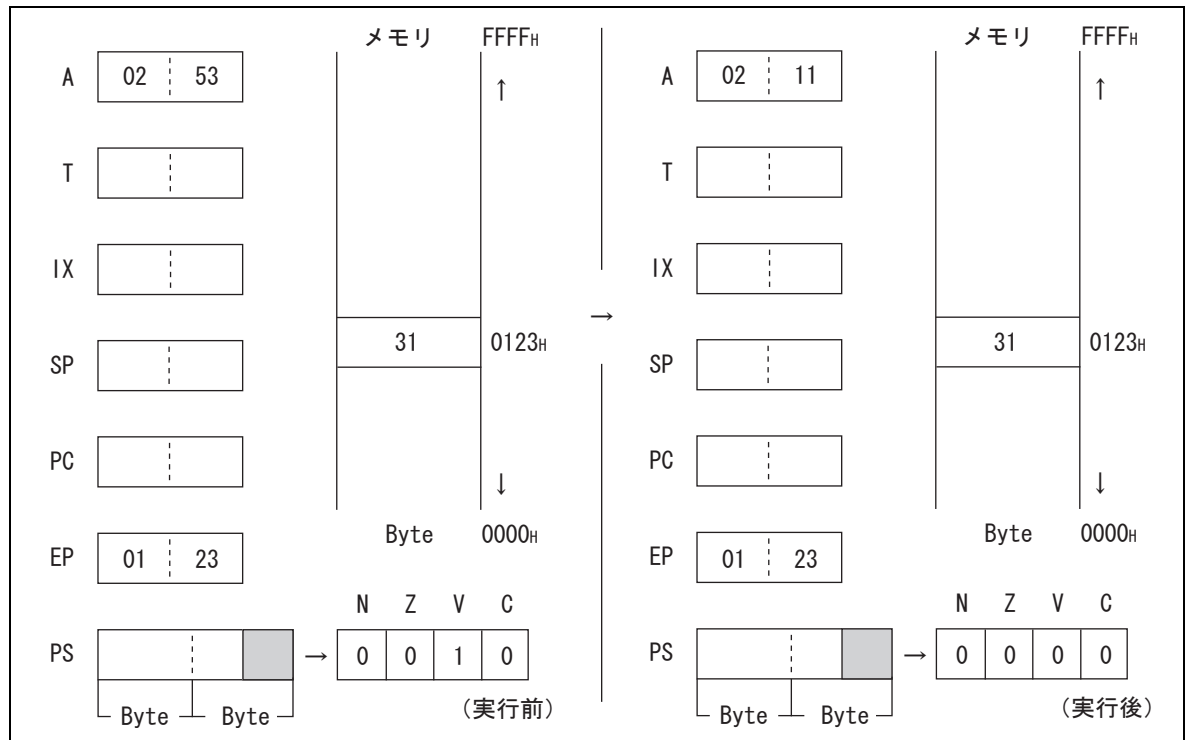
V: 常に 0 になります

C: 変化しません

表 6-2. 実行サイクル数 / バイト数 / オペコード

EA	#d8	dir	@IX+off	@EP	Ri
実行サイクル数	2	3	3	2	2
バイト数	2	2	2	1	1
オペコード	64	65	66	67	68 ~ 6F

実行例 : AND, @EP





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

A のワードデータと T のワードデータをビットごとに論理積をとり、結果を A に戻します。

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

オペレーション

$(A) \leftarrow (A) \wedge (T)$  (ワード論理積)

アセンブラ形式

ANDW A

コンディションコード (CCR)

N	Z	V	C
+	+	R	-

+: 命令実行により変化します

-: 変化しません

R: 命令実行により 0 になります

N: 演算結果の A が MSB=1 ならば 1 になり、それ以外は 0 になります

Z: 演算結果が 0000<sub>H</sub> ならば 1 になり、それ以外は 0 になります

V: 常に 0 になります

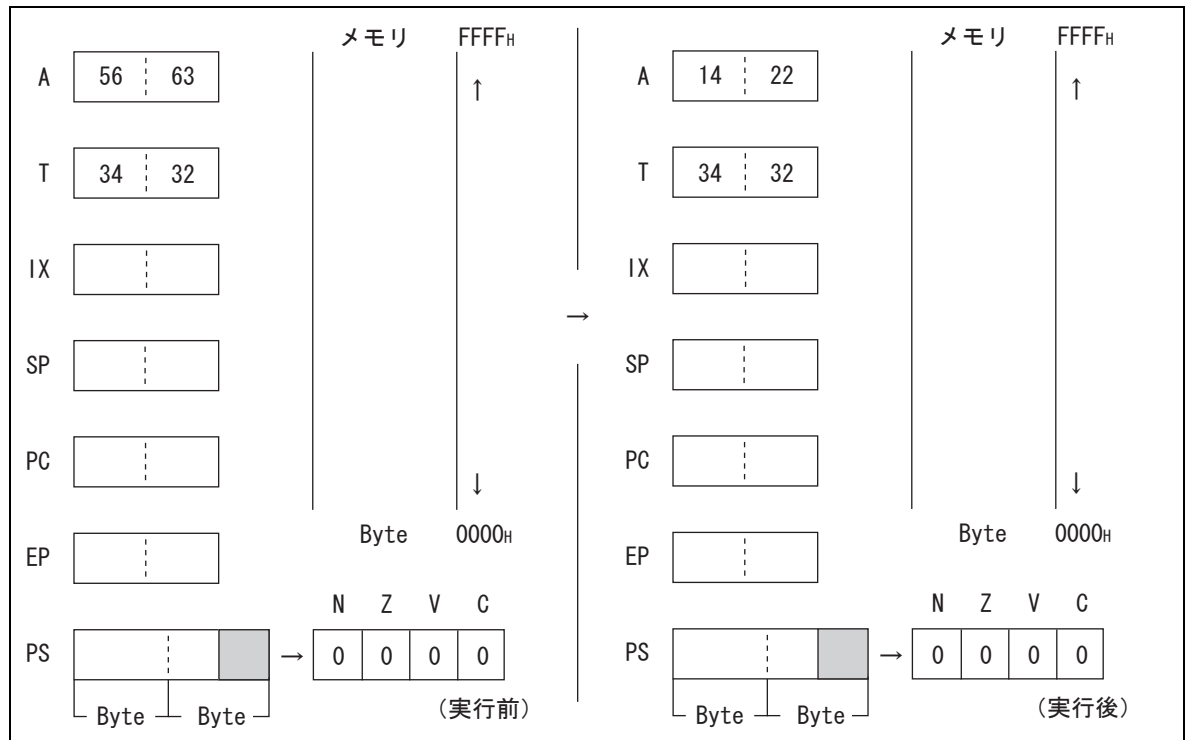
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: 63

実行例 : ANDW A



## 6.7 BBC (Branch if Bit is Clear)

dir メモリ上のビット b の値が 0 のとき分岐します。分岐先アドレスは、次の命令の PC 値 (ワード値) と rel を符号拡張した値 (ワード値) のワード加算となります。

### BBC (Branch if Bit is Clear)

オペレーション

(bit)b = 0 : (PC) ← (PC) + 3 + rel (ワード加算)

(bit)b = 1 : (PC) ← (PC) + 3 (ワード加算)

アセンブラ形式

BBC dir:b, rel

コンディションコード (CCR)

N	Z	V	C
-	+	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: dir:b の値が 0 のとき 1 になり, 1 のとき 0 になります

V: 変化しません

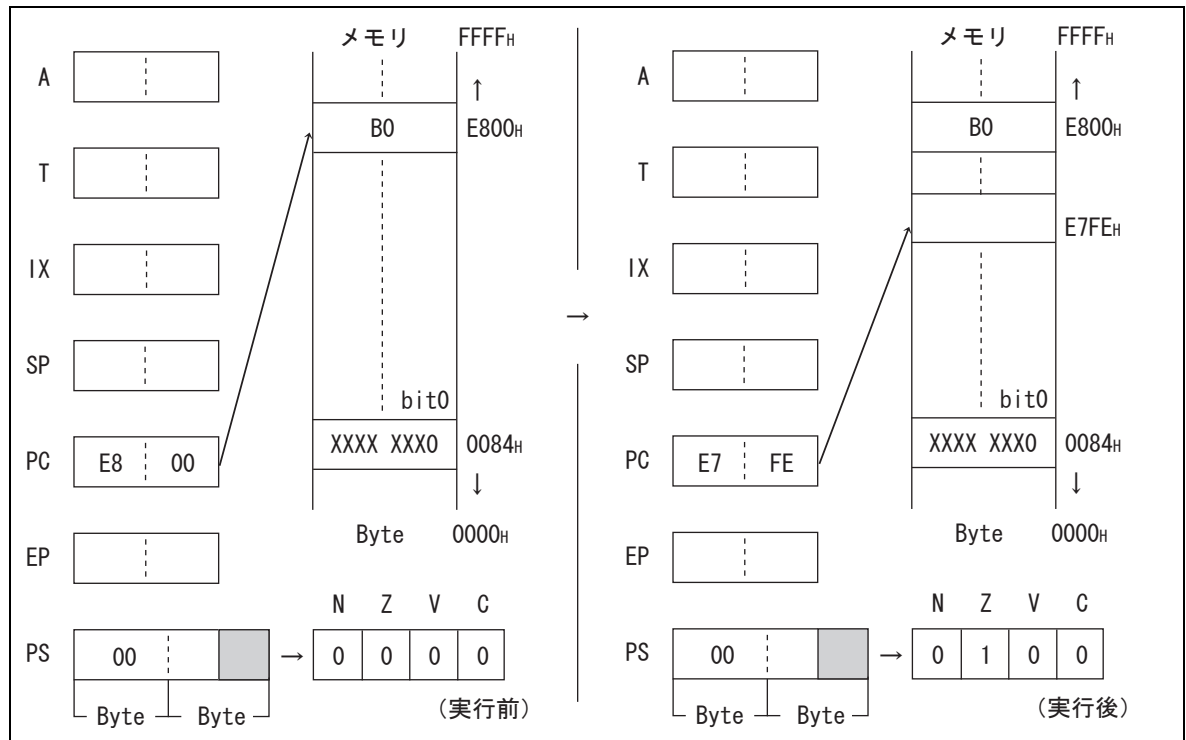
C: 変化しません

実行サイクル数: 5

バイト数: 3

オペコード: B0 ~ B7

実行例 : BBC 84H:0, 0FBH



## 6.8 BBS (Branch if Bit is Set)

dir メモリ上のビット b の値が 1 のとき分岐します。分岐先アドレスは、次の命令の PC 値 (ワード値) と rel を符号拡張した値 (ワード値) のワード加算となります。

### BBS (Branch if Bit is Set)

オペレーション

(bit)b = 0 : (PC) ← (PC) + 3 (ワード加算)

(bit)b = 1 : (PC) ← (PC) + 3 + rel (ワード加算)

アセンブラ形式

BBS dir:b, rel

コンディションコード (CCR)

N	Z	V	C
-	+	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: dir:b の値が 0 のとき 1 になり, 1 のとき 0 となります

V: 変化しません

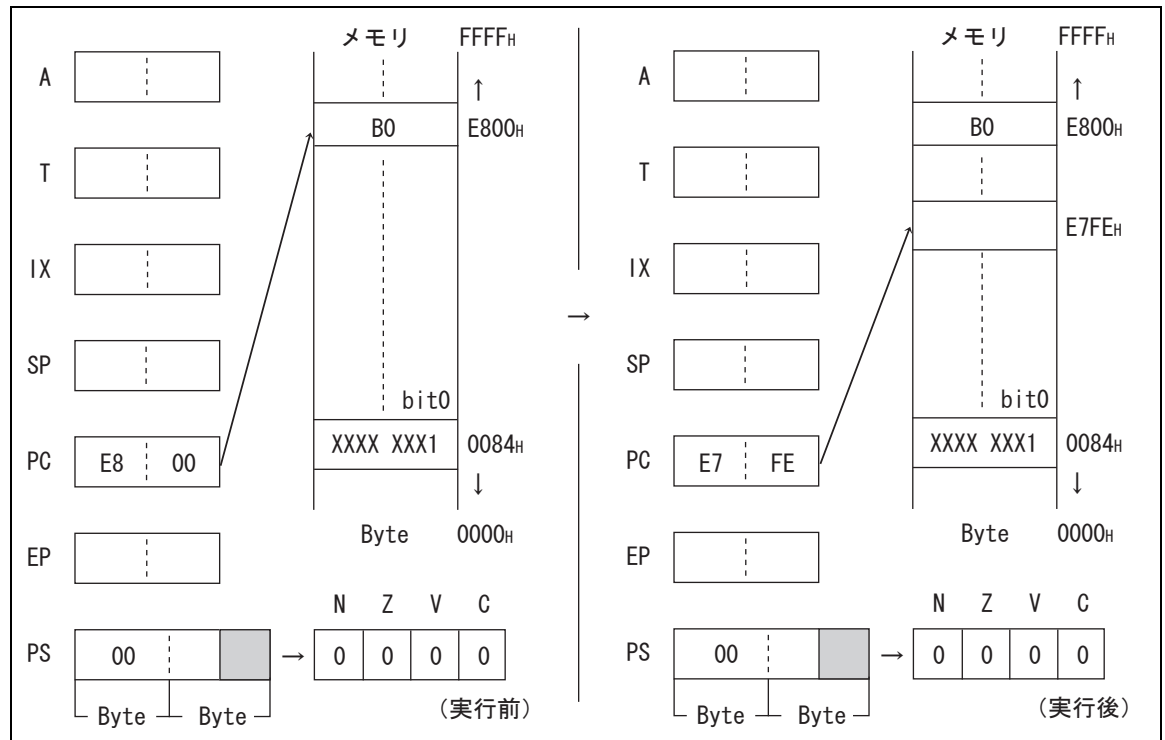
C: 変化しません

実行サイクル数: 5

バイト数: 3

オペコード: B8 ~ BF

実行例 : BBS 84H:0, 0FBH



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

C フラグ =0 ならば次の命令を実行します。

C フラグ =1 ならば分岐を実行します。分岐先アドレスは、次の命令の PC 値 (ワード値) と rel を符号拡張した値 (ワード値) のワード加算値となります。

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

オペレーション

(C) = 0 : (PC)  $\leftarrow$  (PC) + 2 (ワード加算)

(C) = 1 : (PC)  $\leftarrow$  (PC) + 2 + rel (ワード加算)

アセンブラ形式

BC rel/BLO rel

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

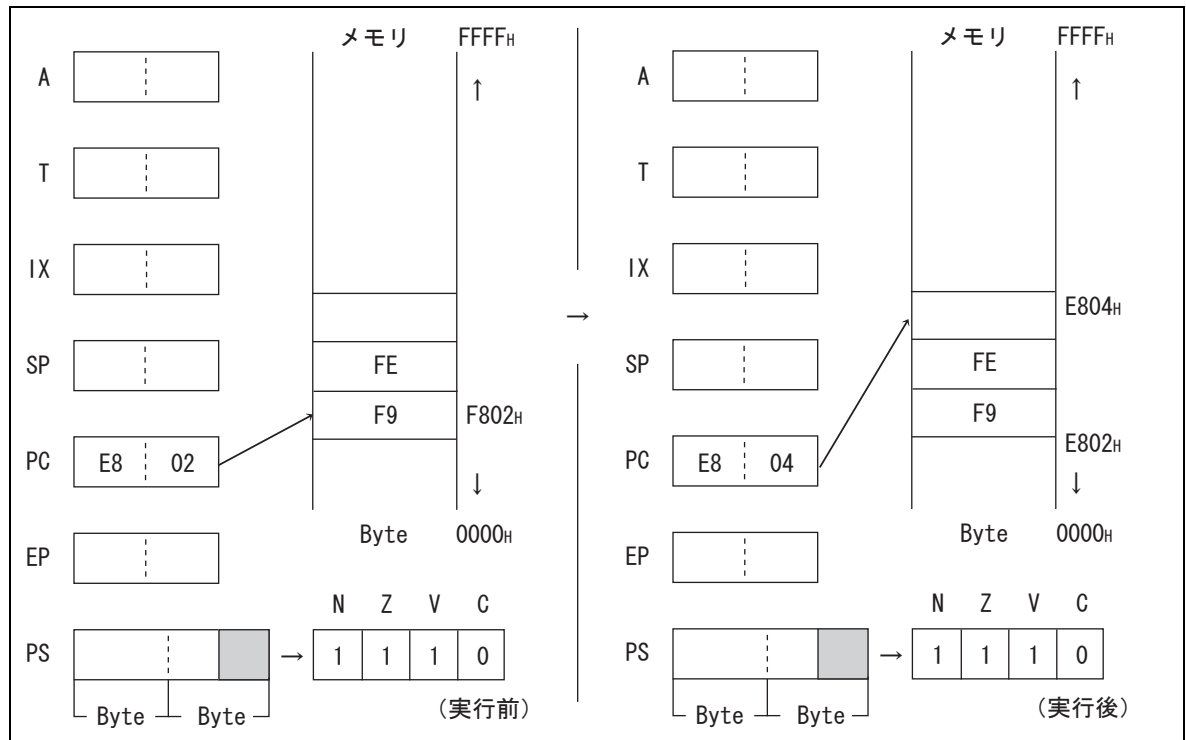
C: 変化しません

実行サイクル数: 4 (分岐時) / 2 (非分岐時)

バイト数: 2

オペコード: F9

実行例 : BC 0FEH





## 6.10 BGE (Branch Great or Equal: relative if $\geq$ Zero)

V フラグと N フラグの排他的論理和が 1 ならば次の命令を実行します。  
 V フラグと N フラグの排他的論理和が 0 ならば分岐を実行します。分岐先アドレスは、次の命令の PC 値 (ワード値) と rel を符号拡張した値 (ワード値) のワード加算値となります。

### BGE (Branch Great or Equal: relative if $\geq$ Zero)

オペレーション

(V)  $\vee$  (N) = 1 : (PC)  $\leftarrow$  (PC) + 2 (ワード加算)

(V)  $\vee$  (N) = 0 : (PC)  $\leftarrow$  (PC) + 2 + rel (ワード加算)

アセンブラ形式

BGE rel

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

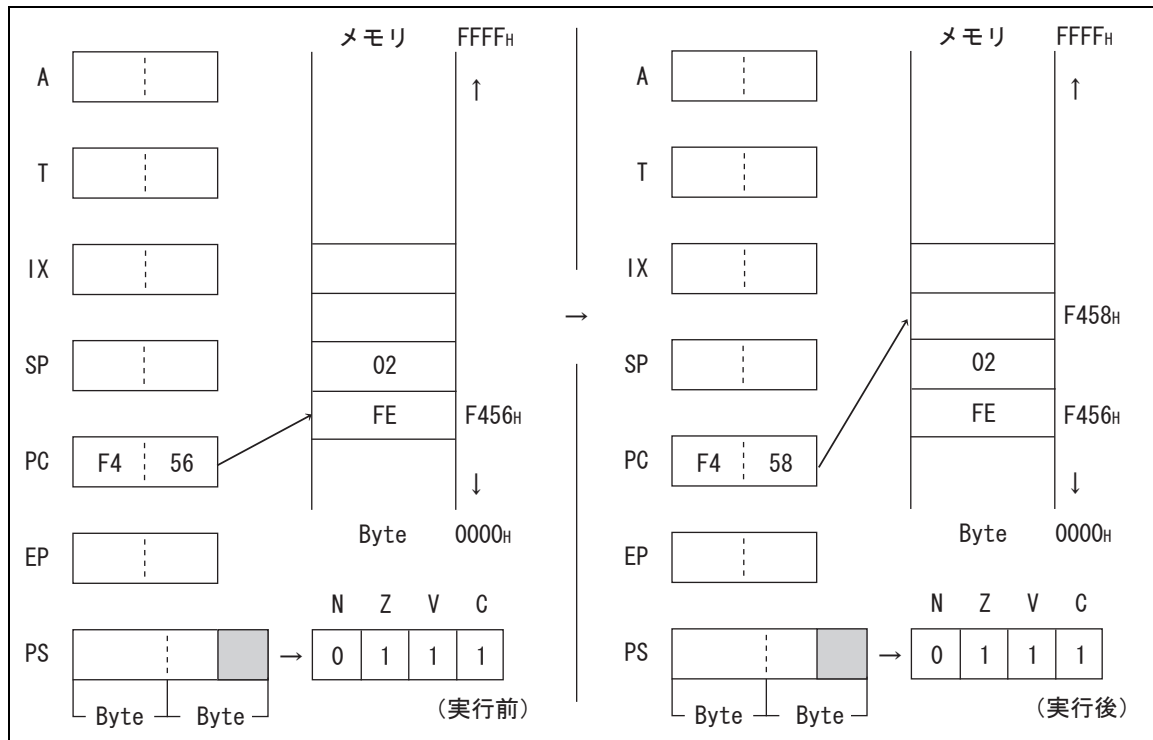
C: 変化しません

実行サイクル数: 4 (分岐時) / 2 (非分岐時)

バイト数: 2

オペコード: FE

実行例 : BGE 02H



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

V フラグと N フラグの排他的論理和が 0 ならば次の命令を実行します。  
 V フラグと N フラグの排他的論理和が 1 ならば分岐を実行します。分岐先アドレスは、次の命令の PC 値 (ワード値) と rel を符号拡張した値 (ワード値) のワード加算値となります。

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

オペレーション

(V)  $\vee$  (N) = 0 : (PC)  $\leftarrow$  (PC) + 2 (ワード加算)

(V)  $\vee$  (N) = 1 : (PC)  $\leftarrow$  (PC) + 2 + rel (ワード加算)

アセンブラ形式

BLT rel

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

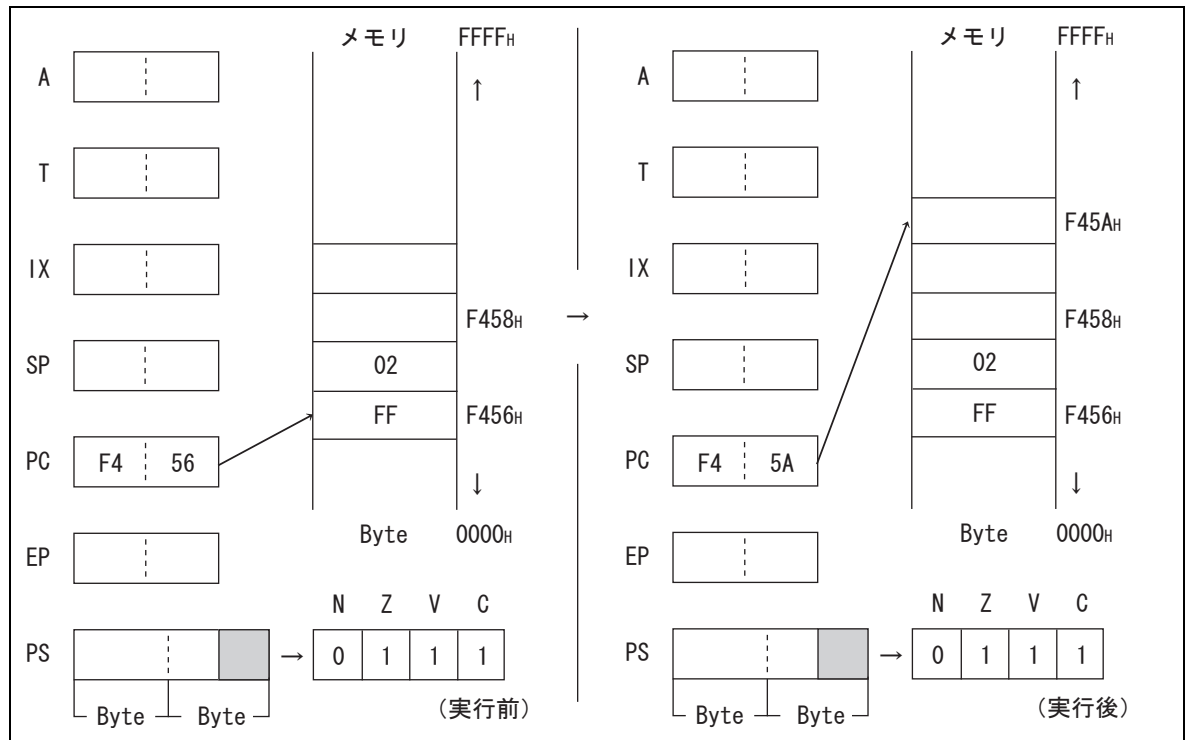
C: 変化しません

実行サイクル数: 4 (分岐時) / 2 (非分岐時)

バイト数: 2

オペコード: FF

実行例 : BLT 02H



## 6.12 BN (Branch relative if N = 1)

N フラグ =0 ならば次の命令を実行します。

N フラグ =1 ならば分岐を実行します。分岐先アドレスは、次の命令の PC 値 (ワード値) と rel を符号拡張した値 (ワード値) のワード加算値となります。

### BN (Branch relative if N = 1)

オペレーション

N = 0 : (PC) ← (PC) + 2 (ワード加算)

N = 1 : (PC) ← (PC) + 2 + rel (ワード加算)

アセンブラ形式

BN rel

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

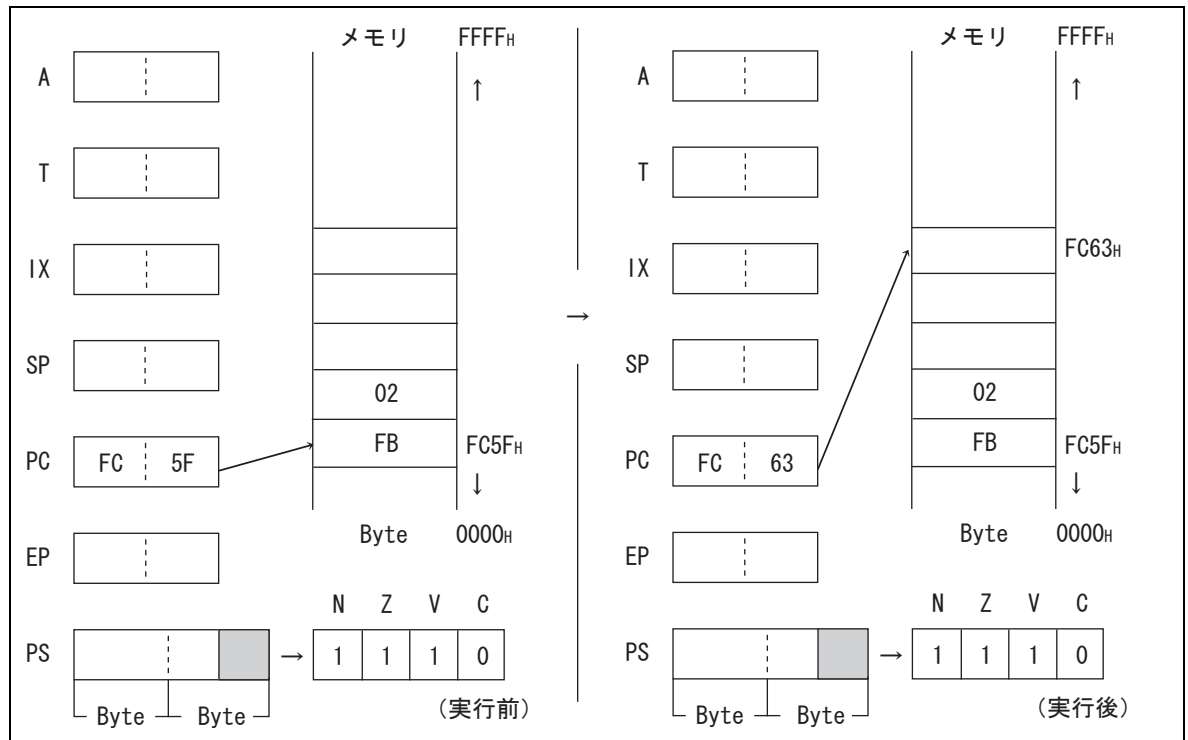
C: 変化しません

実行サイクル数: 4 (分岐時) / 2 (非分岐時)

バイト数: 2

オペコード: FB

実行例 : BN 02H



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

Z フラグ =1 ならば次の命令を実行します。

Z フラグ =0 ならば分岐を実行します。分岐先アドレスは、次の命令の PC 値 (ワード値) と rel を符号拡張した値 (ワード値) のワード加算値となります。

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

オペレーション

$(Z) = 1 : (PC) \leftarrow (PC) + 2$  (ワード加算)

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

アセンブラ形式

BNZ rel/BNE rel

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

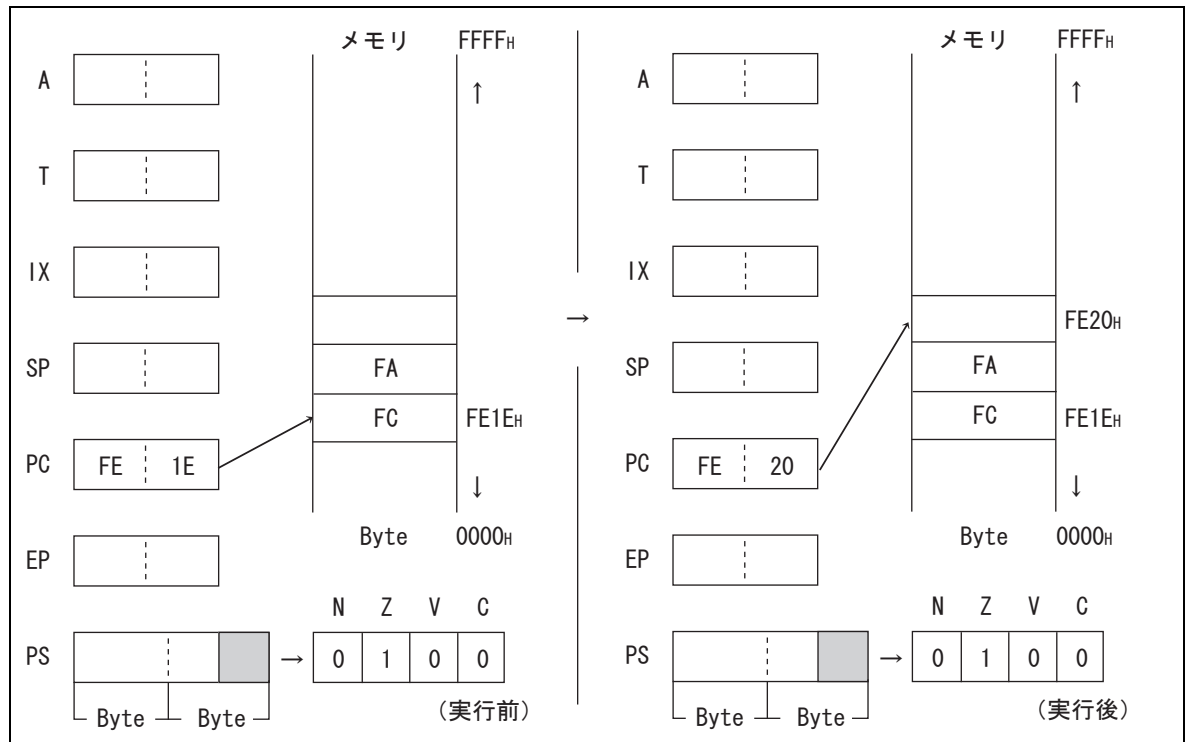
C: 変化しません

実行サイクル数: 4 (分岐時) / 2 (非分岐時)

バイト数: 2

オペコード: FC

実行例 : BNZ 0FAH





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

C フラグ =1 ならば次の命令を実行します。

C フラグ =0 ならば分岐を実行します。分岐先アドレスは、次の命令の PC 値 (ワード値) と rel を符号拡張した値 (ワード値) のワード加算値となります。

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

オペレーション

(C) = 1 : (PC) ← (PC) + 2 (ワード加算)

(C) = 0 : (PC) ← (PC) + 2 + rel (ワード加算)

アセンブラ形式

BNC rel/BHS rel

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

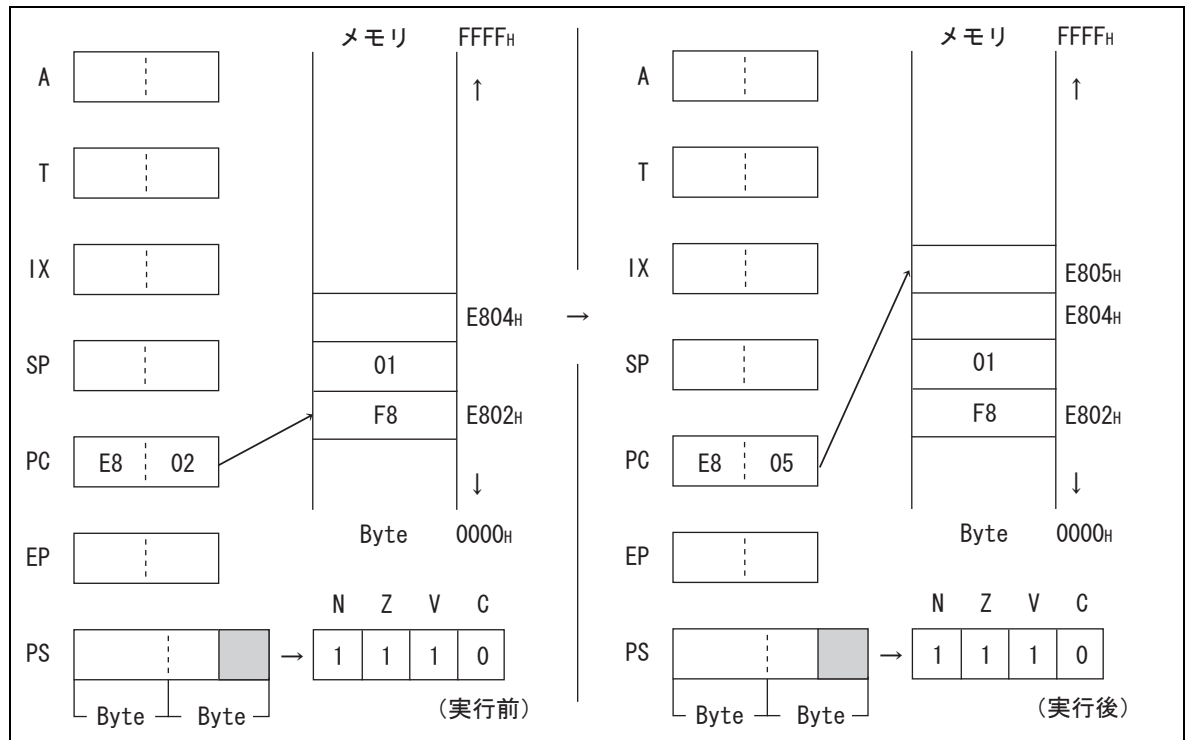
C: 変化しません

実行サイクル数: 4 (分岐時) / 2 (非分岐時)

バイト数: 2

オペコード: F8

実行例 : BNC 01H



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

N フラグ =1 ならば次の命令を実行します。

N フラグ =0 ならば分岐を実行します。分岐先アドレスは、次の命令の PC 値 (ワード値) と rel を符号拡張した値 (ワード値) のワード加算値となります。

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

オペレーション

(N) = 1 : (PC) ← (PC) + 2 (ワード加算)

(N) = 0 : (PC) ← (PC) + 2 + rel (ワード加算)

アセンブラ形式

BP rel

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

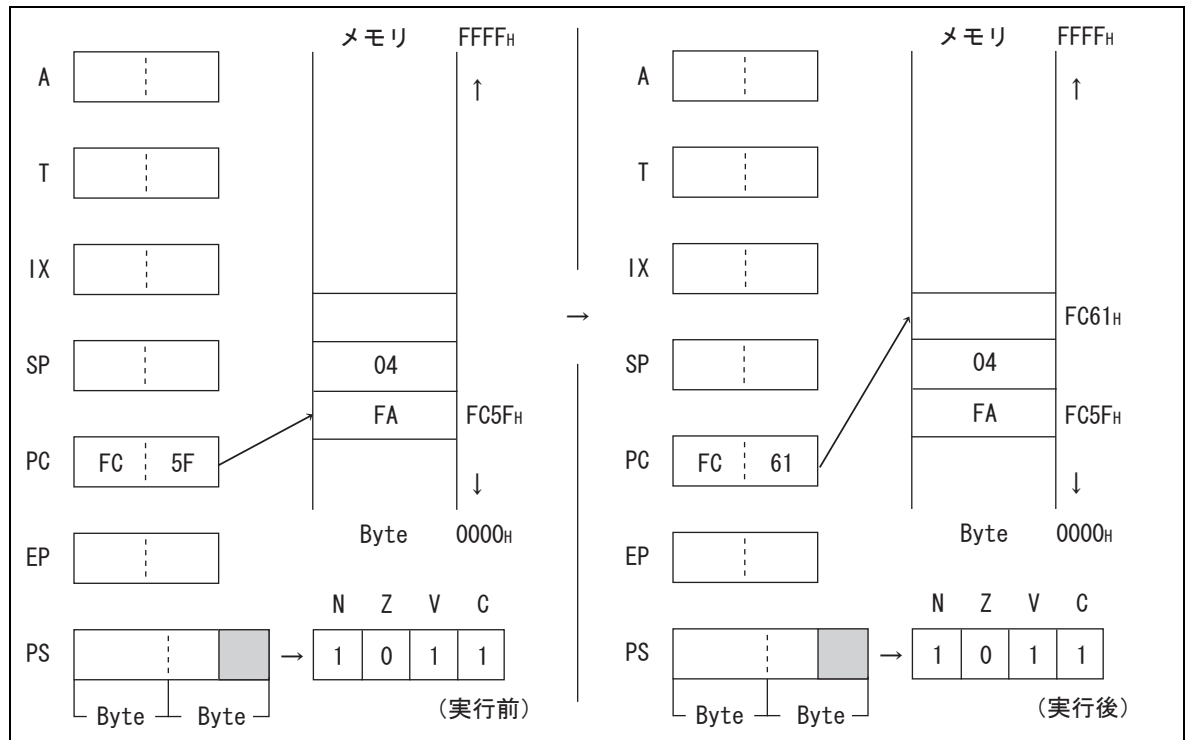
C: 変化しません

実行サイクル数: 4 (分岐時) / 2 (非分岐時)

バイト数: 2

オペコード: FA

実行例 : BP 04H



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

Z フラグ =0 ならば次の命令を実行します。

Z フラグ =1 ならば分岐を実行します。分岐先アドレスは、次の命令の PC 値 (ワード値) と rel を符号拡張した値 (ワード値) のワード加算値となります。

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

オペレーション

(Z) = 0 : (PC)  $\leftarrow$  (PC) + 2 (ワード加算)

(Z) = 1 : (PC)  $\leftarrow$  (PC) + 2 + rel (ワード加算)

アセンブラ形式

BZ rel/BEQ rel

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

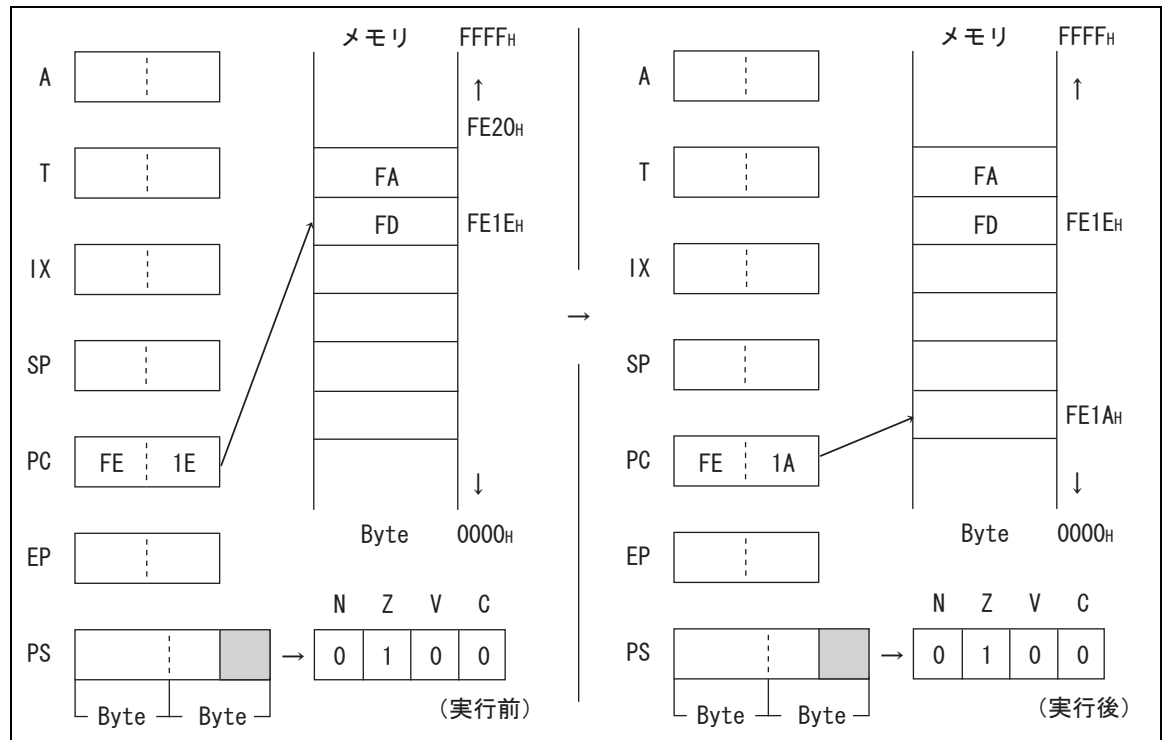
C: 変化しません

実行サイクル数: 4 (分岐時) / 2 (非分岐時)

バイト数: 2

オペコード: FD

実行例 : BZ 0FAH



## 6.17 CALL (CALL subroutine)

ext のアドレスへ分岐します。  
 分岐サブルーチンの RET 命令により、この命令の次の命令に戻ります。

### CALL (CALL subroutine)

オペレーション

$(SP) \leftarrow (SP) - 2$  (ワード減算),  $((SP)) \leftarrow (PC)$  (ワード転送)

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

アセンブラ形式

CALL ext

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

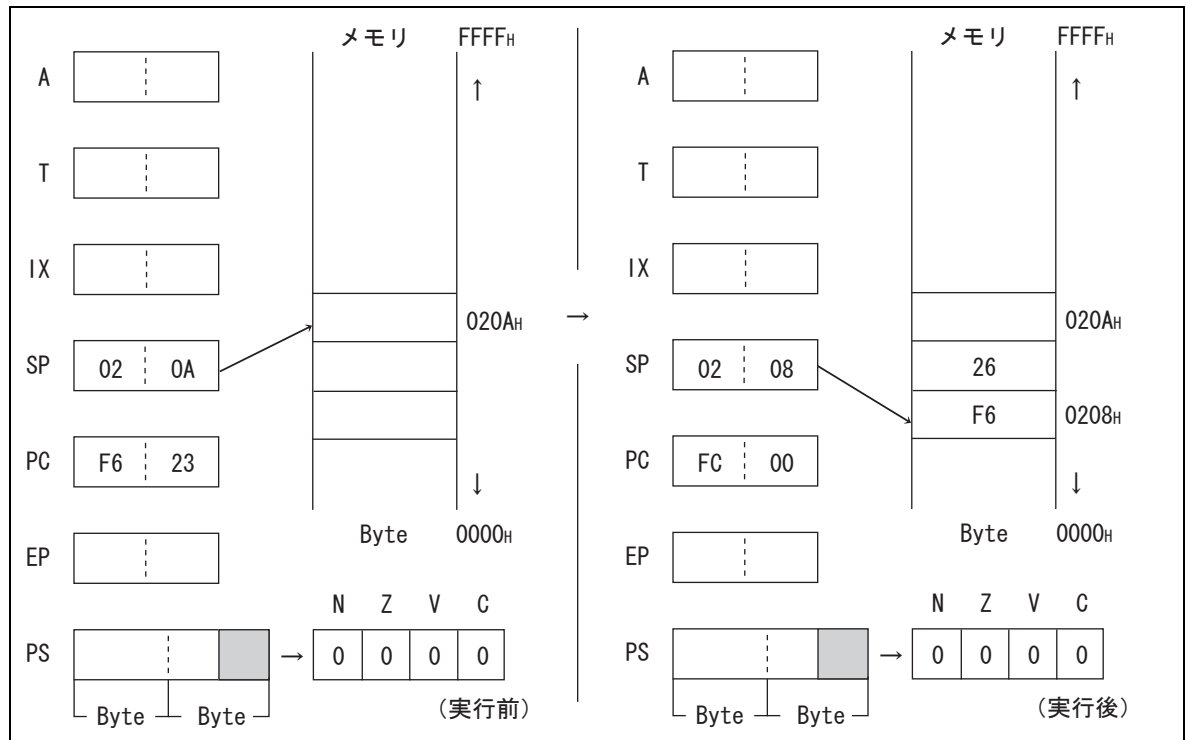
C: 変化しません

実行サイクル数: 6

バイト数: 3

オペコード: 31

実行例 : CALL 0FC00H





## 6.18 CALLV (CALL Vectored subroutine)

vct のベクタアドレス (VA) が示すアドレスへ分岐します。  
 分岐サブルーチンの RET 命令により、この命令の次の命令に戻ります。  
 vct により示されるベクタアドレス (VA) を次のページに示します。

### CALLV (CALL Vectored subroutine)

オペレーション

$(SP) \leftarrow (SP) - 2$  (ワード減算),  $((SP)) \leftarrow (PC)$  (ワード転送)

$(PC) \leftarrow (VA)$

アセンブラ形式

CALLV #vct

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

C: 変化しません

実行サイクル数: 7

バイト数: 1

オペコード: E8 ~ EF

実行例 : CALLV #02H

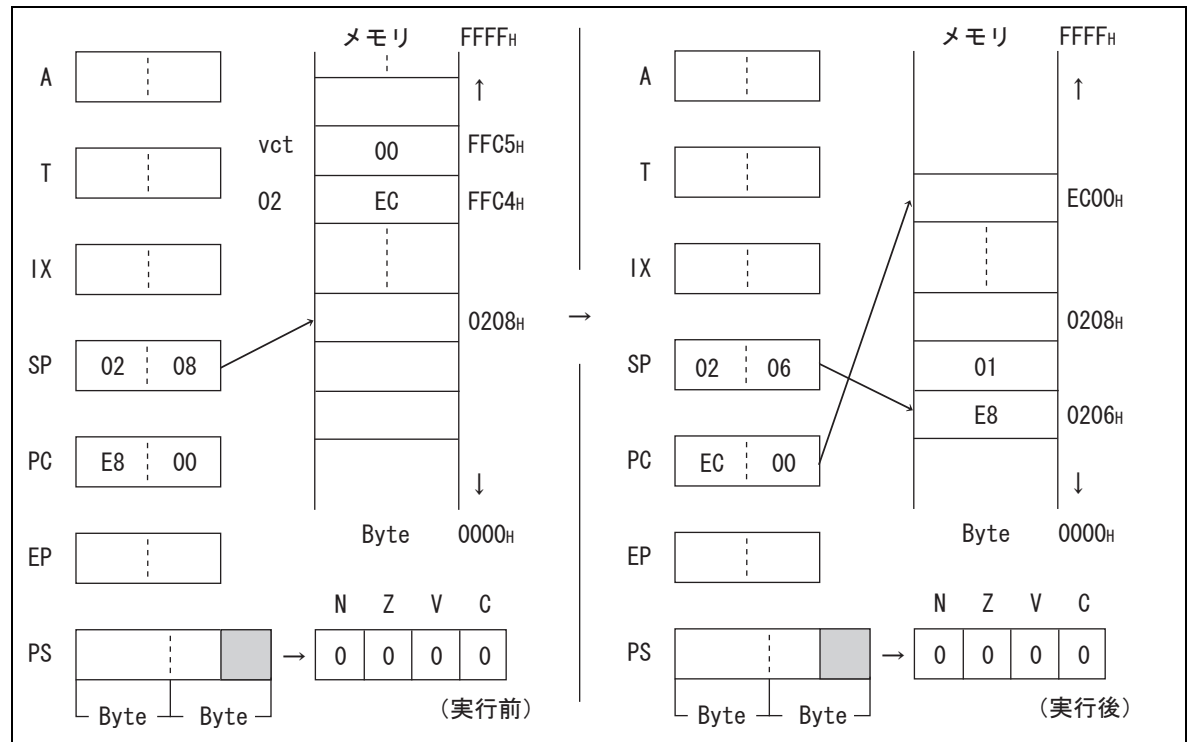


表 6-3. ベクタコール命令のコール先アドレス格納番地

ベクタ (VA)		命令
下位アドレス	上位アドレス	
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 CLRB (Clear direct Memory Bit)

ダイレクト領域の 1 ビット ( ニーモニックの下位 3 ビット (b) で示される ) の内容を 0 にします。

### CLRB (Clear direct Memory Bit)

オペレーション

(dir:b) ← 0

アセンブラ形式

CLRB dir:b

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

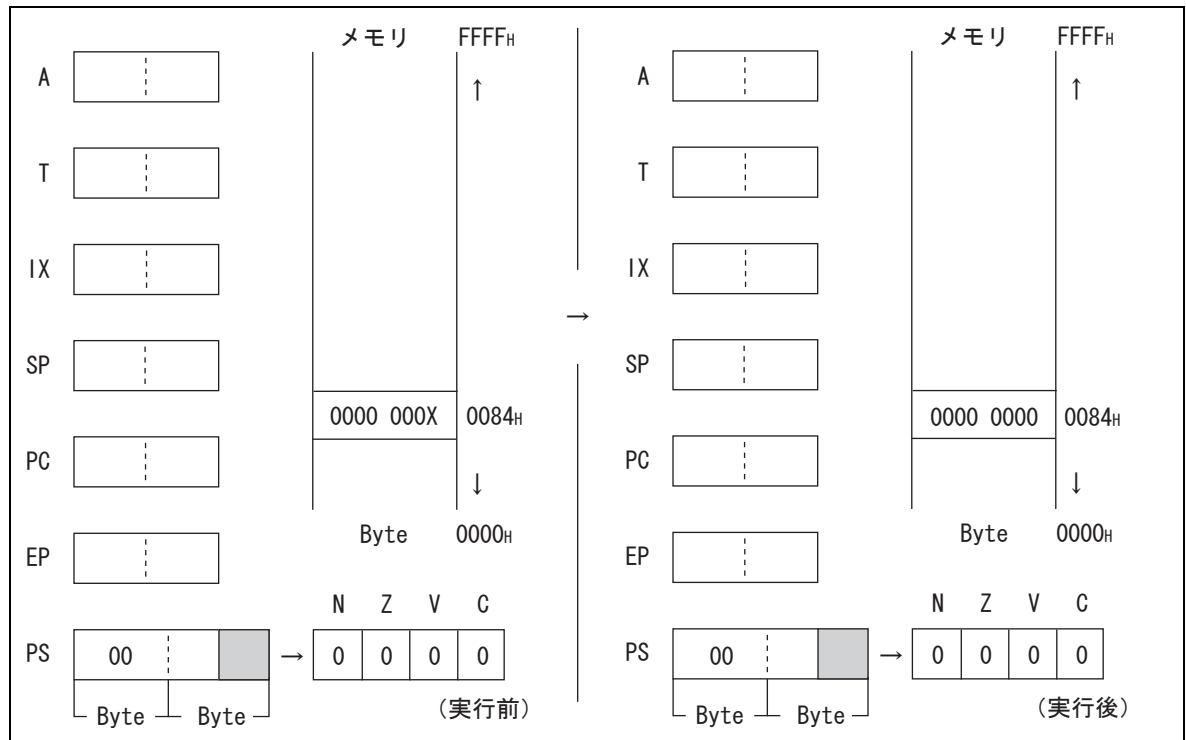
C: 変化しません

実行サイクル数: 4

バイト数: 2

オペコード: A0 ~ A7

実行例 : CLRB 84H:0



## 6.20 CLRC (Clear Carry flag)

C フラグを 0 にします。

### CLRC (Clear Carry flag)

オペレーション

$(C) \leftarrow 0$

アセンブラ形式

CLRC

コンディションコード (CCR)

N	Z	V	C
-	-	-	R

+: 命令実行により変化します

-: 変化しません

R: 命令実行により 0 になります

N: 変化しません

Z: 変化しません

V: 変化しません

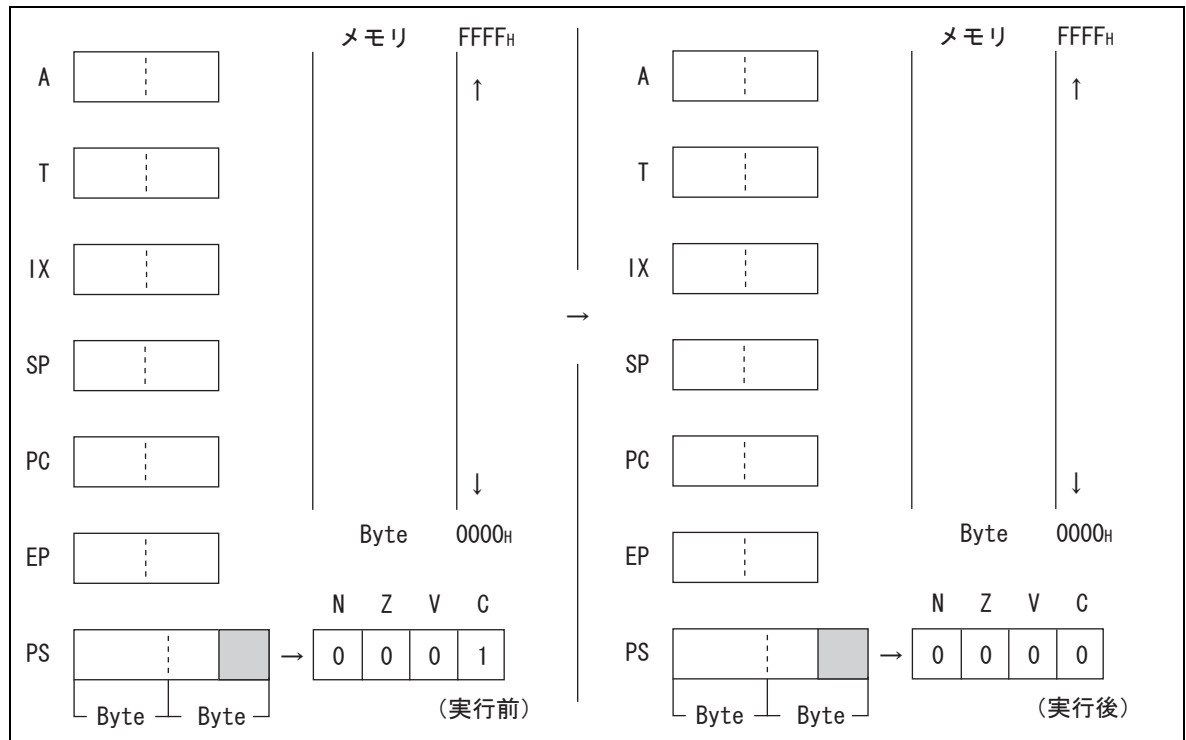
C: 0 になります

実行サイクル数: 1

バイト数: 1

オペコード: 81

実行例 : CLRC



## 6.21 CLRI (CLear Interrupt flag)

I フラグを 0 にします。

### CLRI (CLear Interrupt flag)

オペレーション

(I) ← 0

アセンブラ形式

CLRI

コンディションコード (CCR)

I	N	Z	V	C
R	-	-	-	-

+: 命令実行により変化します

-: 変化しません

R: 命令実行により 0 になります

I: 0 になります

N: 変化しません

Z: 変化しません

V: 変化しません

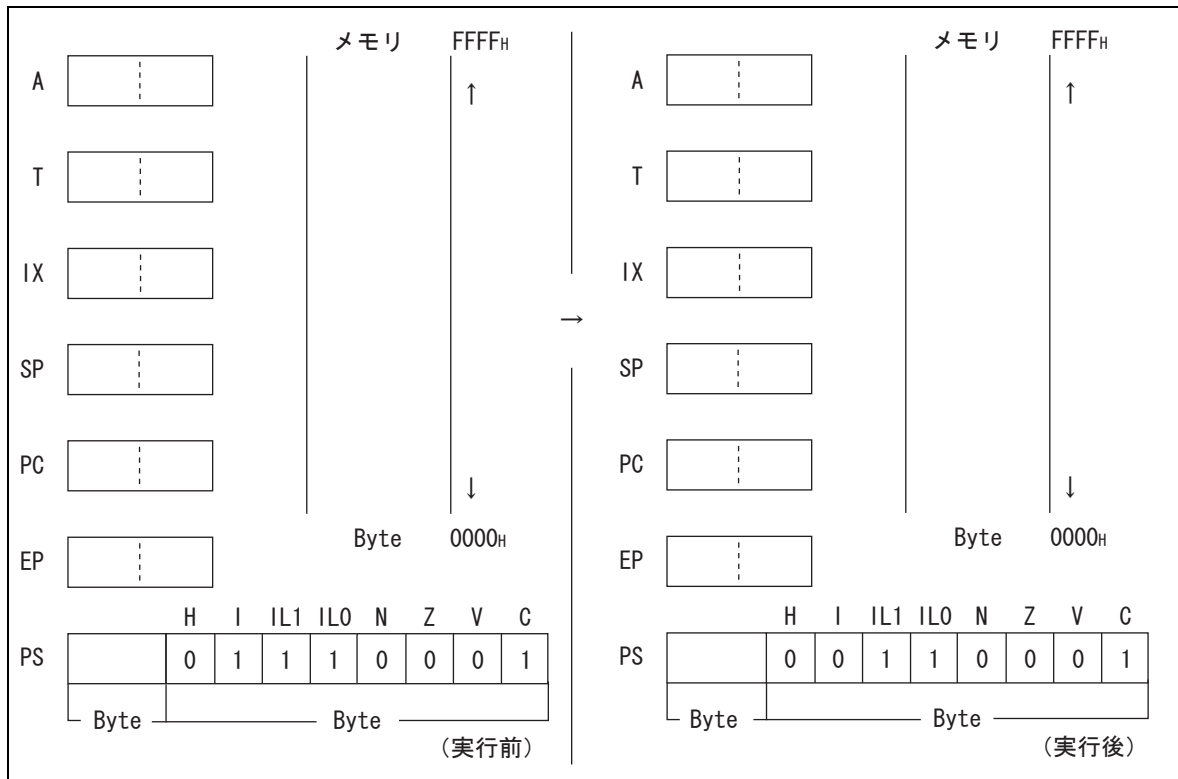
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: 80

実行例 : CLRI





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

AL のバイトデータと TL のバイトデータとを比較し, 結果を CCR にセットします。  
 AL と TL は, 変わりません。

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

オペレーション

(TL) - (AL)

アセンブラ形式

CMP A

コンディションコード (CCR)

N	Z	V	C
+	+	+	+

+: 命令実行により変化します

-: 変化しません

N: 演算結果が MSB=1 ならば 1 になり, それ以外は 0 になります

Z: 演算結果が 00<sub>H</sub> ならば 1 になり, それ以外は 0 になります

V: 演算の結果オーバーフローが発生したときに 1 になり, それ以外は 0 になります

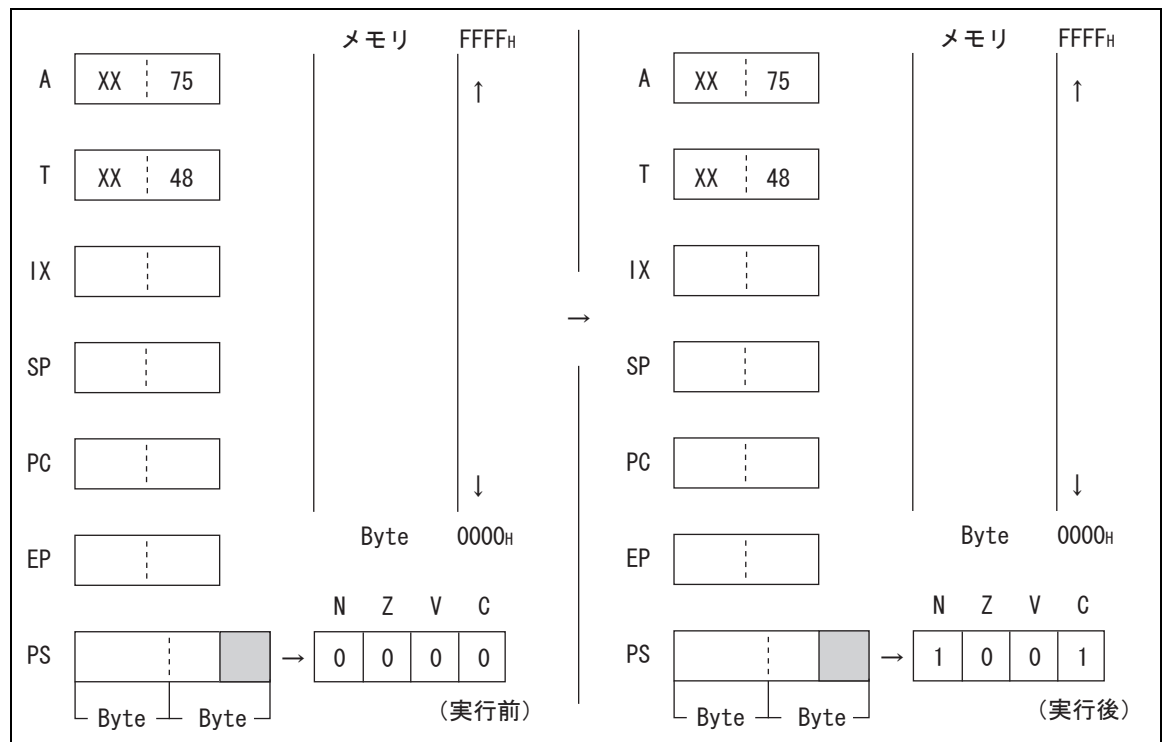
C: 演算の結果キャリーが発生したときに 1 になり, それ以外は 0 になります

実行サイクル数: 1

バイト数: 1

オペコード: 12

実行例 : CMP A



## 6.23 CMP (CoMPare Byte Data of Accumulator and Memory)

AL のバイトデータと EA メモリ ( 各種アドレッシングで表現されるメモリ ) のバイトデータを比較し、結果を CCR にセットします。  
 AL と EA メモリは、変わりません。

### CMP (CoMPare Byte Data of Accumulator and Memory)

オペレーション

(AL) - (EA)

アセンブラ形式

CMP A, EA

コンディションコード (CCR)

N	Z	V	C
+	+	+	+

+: 命令実行により変化します

-: 変化しません

N: 演算結果が MSB=1 ならば 1 となり、それ以外は 0 となります

Z: 演算結果が 00<sub>H</sub> ならば 1 となり、それ以外は 0 となります

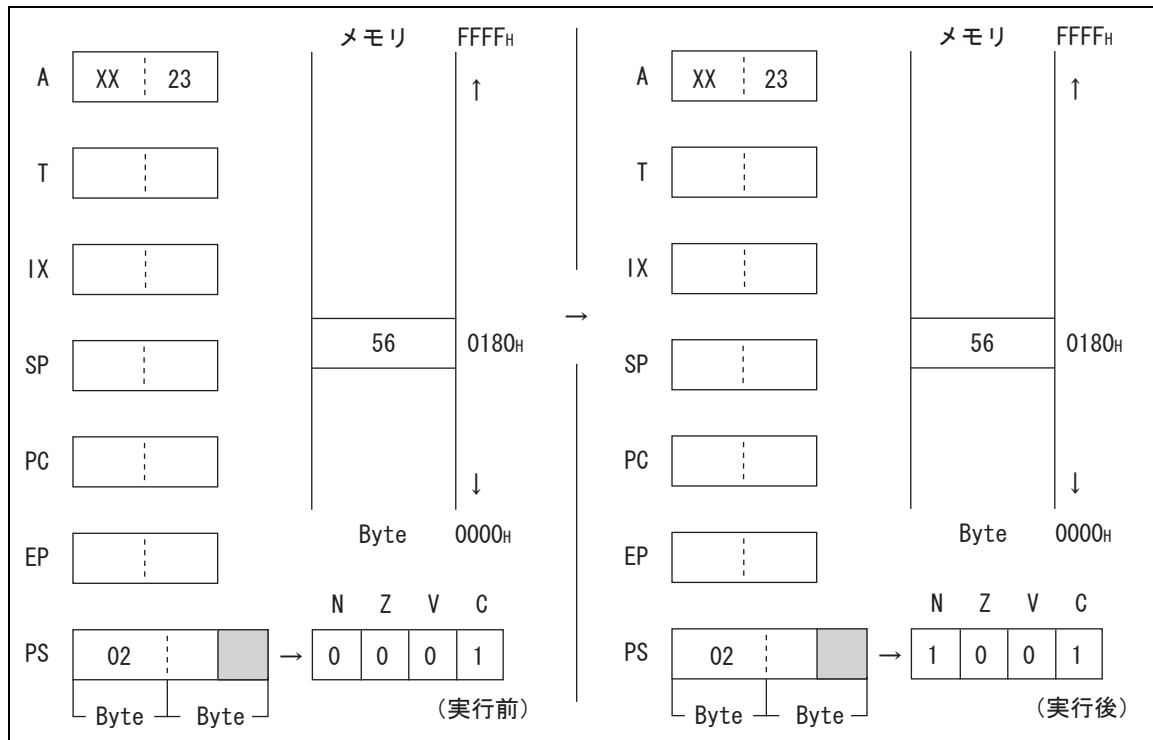
V: 演算の結果オーバフローが発生したときに 1 となり、それ以外は 0 となります

C: 演算の結果キャリーが発生したときに 1 となり、それ以外は 0 となります

表 6-4. 実行サイクル数 / バイト数 / オペコード

EA	#d8	dir	@IX+off	@EP	Ri
実行サイクル数	2	3	3	2	2
バイト数	2	2	2	1	1
オペコード	14	15	16	17	18 ~ 1F

実行例 : CMP A, 80H



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

EA メモリ ( 各種アドレッシングで表現されるメモリ ) のバイトデータと即値データとを比較し, 結果を CCR にセットします。

EA メモリは, 変わりません。

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

オペレーション

(EA) - d8

アセンブラ形式

CMP EA, #d8

コンディションコード (CCR)

N	Z	V	C
+	+	+	+

+: 命令実行により変化します

-: 変化しません

N: 演算結果が MSB=1 ならば 1 となり, それ以外は 0 となります

Z: 演算結果が 00<sub>H</sub> ならば 1 となり, それ以外は 0 となります

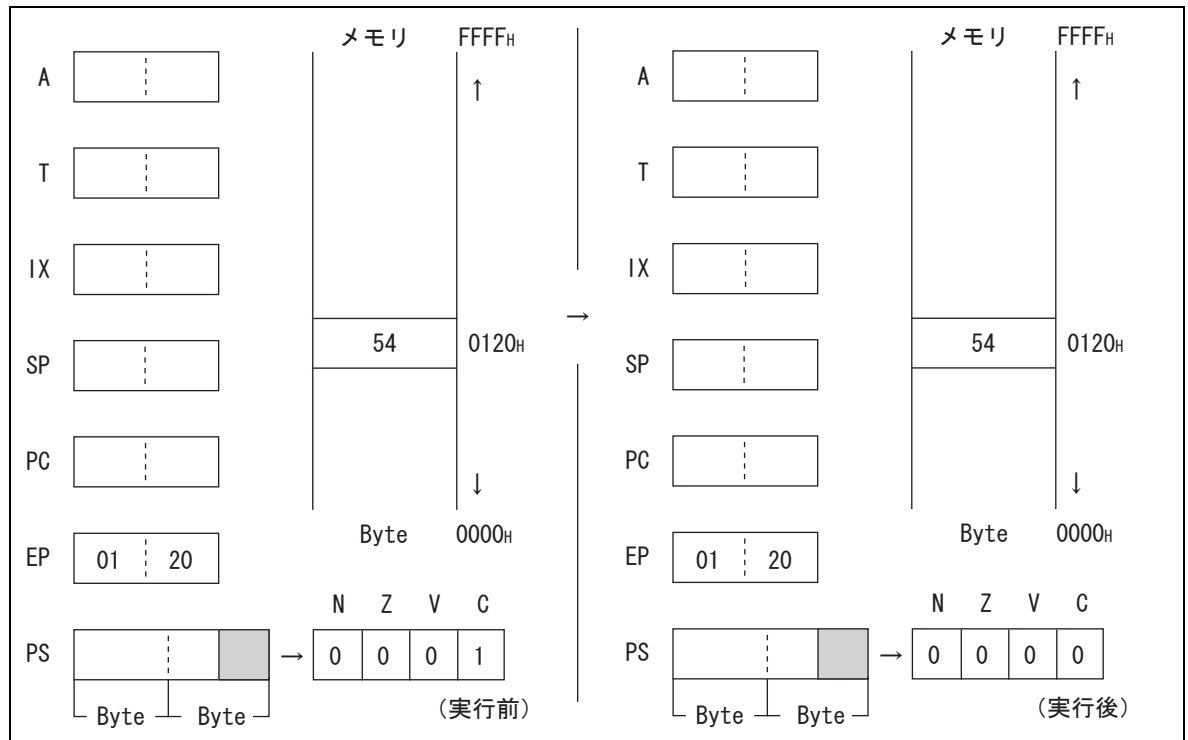
V: 演算の結果オーバフローが発生したときに 1 となり, それ以外は 0 となります

C: 演算の結果キャリーが発生したときに 1 となり, それ以外は 0 となります

表 6-5. 実行サイクル数 / バイト数 / オペコード

EA	dir	@IX+off	@EP	Ri
実行サイクル数	4	4	3	3
バイト数	3	3	2	2
オペコード	95	96	97	98 ~ 9F

実行例 : CMP @EP, #33H



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

A のワードデータと T のワードデータとを比較し, 結果を CCR にセットします。  
 A と T は, 変わりません。

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

オペレーション

(T) - (A)

アセンブラ形式

CMPW A

コンディションコード (CCR)

N	Z	V	C
+	+	+	+

+: 命令実行により変化します

-: 変化しません

N: 演算結果が MSB=1 ならば 1 となり, それ以外は 0 となります

Z: 演算結果が 0000<sub>H</sub> ならば 1 となり, それ以外は 0 となります

V: 演算の結果オーバーフローが発生したときに 1 となり, それ以外は 0 となります

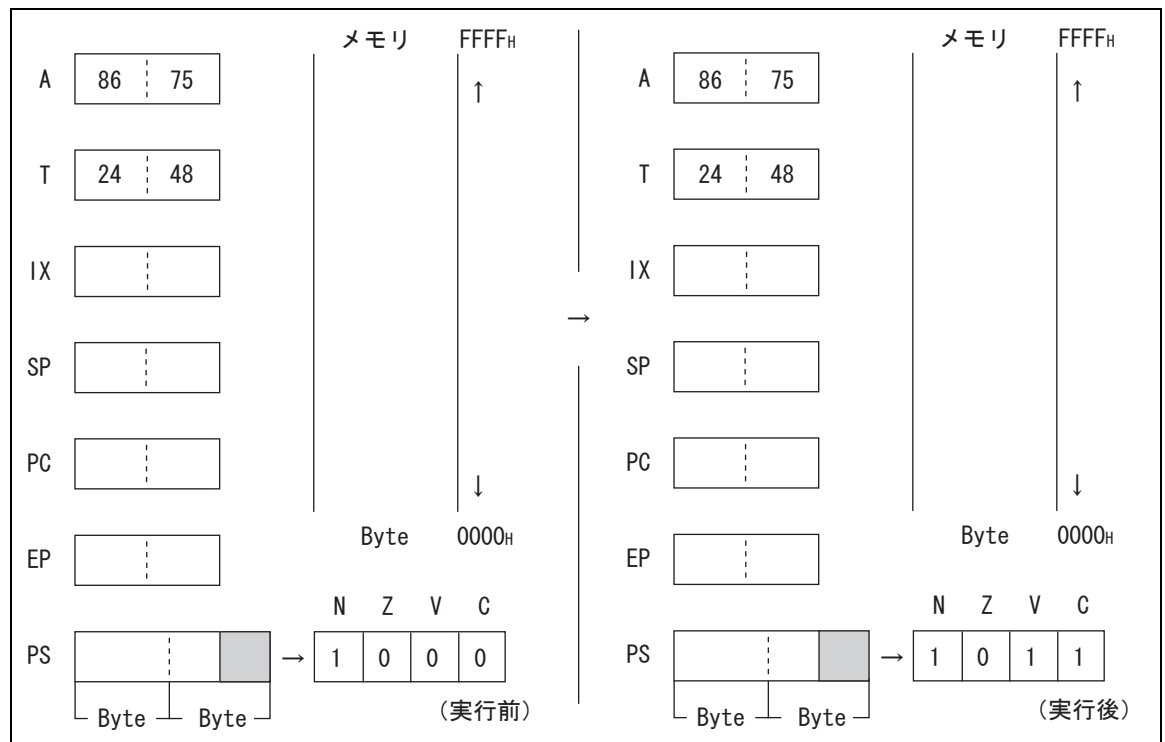
C: 演算の結果キャリーが発生したときに 1 となり, それ以外は 0 となります

実行サイクル数: 2

バイト数: 1

オペコード: 13

実行例 : CMPW A





## 6.26 DAA (Decimal Adjust for Addition)

命令実行前のキャリーおよびハーフキャリーの状態によって AL に補正値を加算して 10 進演算の補正をします。

### DAA (Decimal Adjust for Addition)

オペレーション

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

(AL の値と, C, H フラグにより次ページに示す補正値を AL に加算する)

アセンブラ形式

DAA

コンディションコード (CCR)

N	Z	V	C
+	+	+	+

+: 命令実行により変化します

-: 変化しません

N: 演算結果が MSB=1 ならば 1 となり, それ以外は 0 となります

Z: 演算結果が 00<sub>H</sub> ならば 1 となり, それ以外は 0 となります

V: 演算の結果オーバフローが発生したときに 1 となり, それ以外は 0 となります

C: 次ページに示すとおりに変化します

実行サイクル数: 1

バイト数: 1

オペコード: 84

## 実行例 : DAA

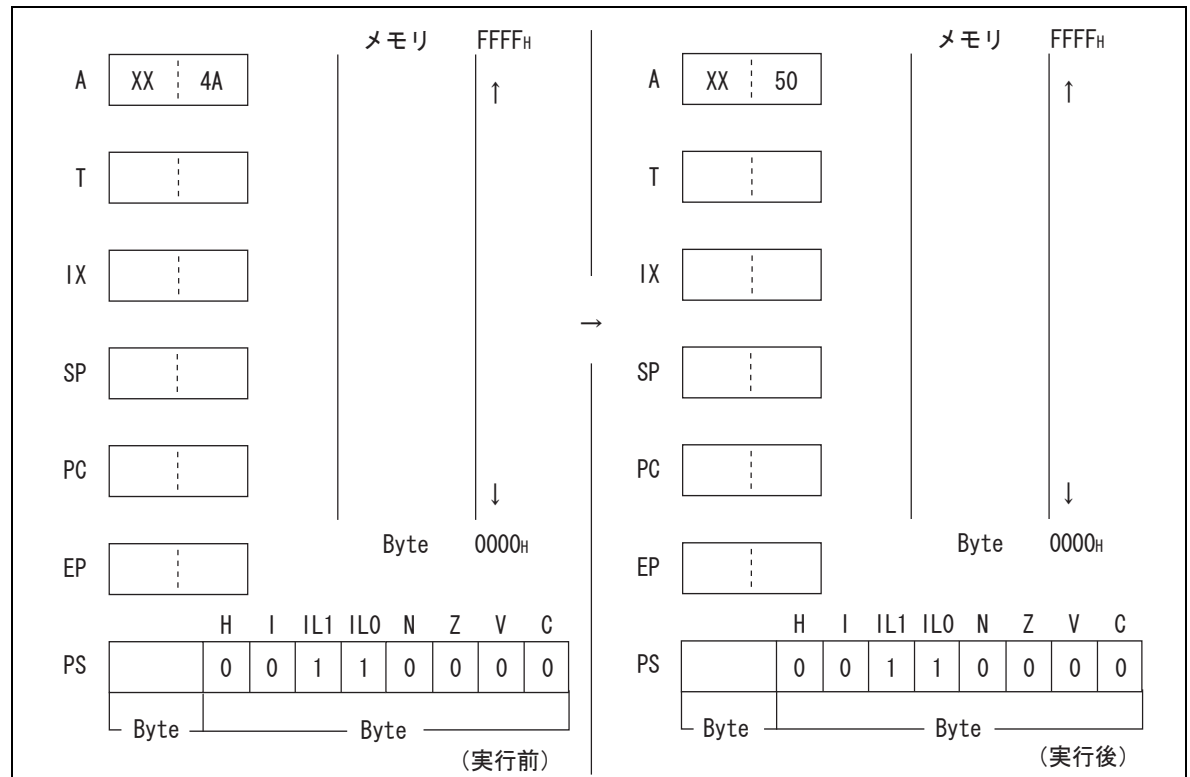


表 6-6. 10 進補正表 (DAA)

C フラグ	AL (ビット 7 ~ 4)	H フラグ	AL (ビット 3 ~ 0)	補正值	実行後の C フラグ
0	0 - 9	0	0 - 9	00	0
0	0 - 8	0	A - F	06	0
0	0 - 9	1	0 - 3	06	0
0	A - F	0	0 - 9	60	1
0	9 - F	0	A - F	66	1
0	A - F	1	0 - 3	66	1
1	0 - 2	0	0 - 9	60	1
1	0 - 2	0	A - F	66	1
1	0 - 3	1	0 - 3	66	1

表 6-7. 実行例

ニーモニック	AL	C	H
MOV A, #75H	75	0	×
ADDC A, #25H	9A	0	0
DAA	00	1	0

## 6.27 DAS (Decimal Adjust for Subtraction)

命令実行前のキャリーおよびハーフキャリーの状態によって AL から補正値を減算して 10 進演算の補正をします。

### DAS (Decimal Adjust for Subtraction)

オペレーション

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

AL の値と, C, H フラグにより次ページに示す補正値を AL から減算する。

アセンブラ形式

DAS

コンディションコード (CCR)

N	Z	V	C
+	+	+	+

+: 命令実行により変化します

-: 変化しません

N: 演算結果が MSB=1 ならば 1 となり, それ以外は 0 となります

Z: 演算結果が 00<sub>H</sub> ならば 1 となり, それ以外は 0 となります

V: 演算の結果オーバフローが発生したときに 1 となり, それ以外は 0 となります

C: 次ページに示すとおりに変化します

実行サイクル数: 1

バイト数: 1

オペコード: 94

実行例 : DAS

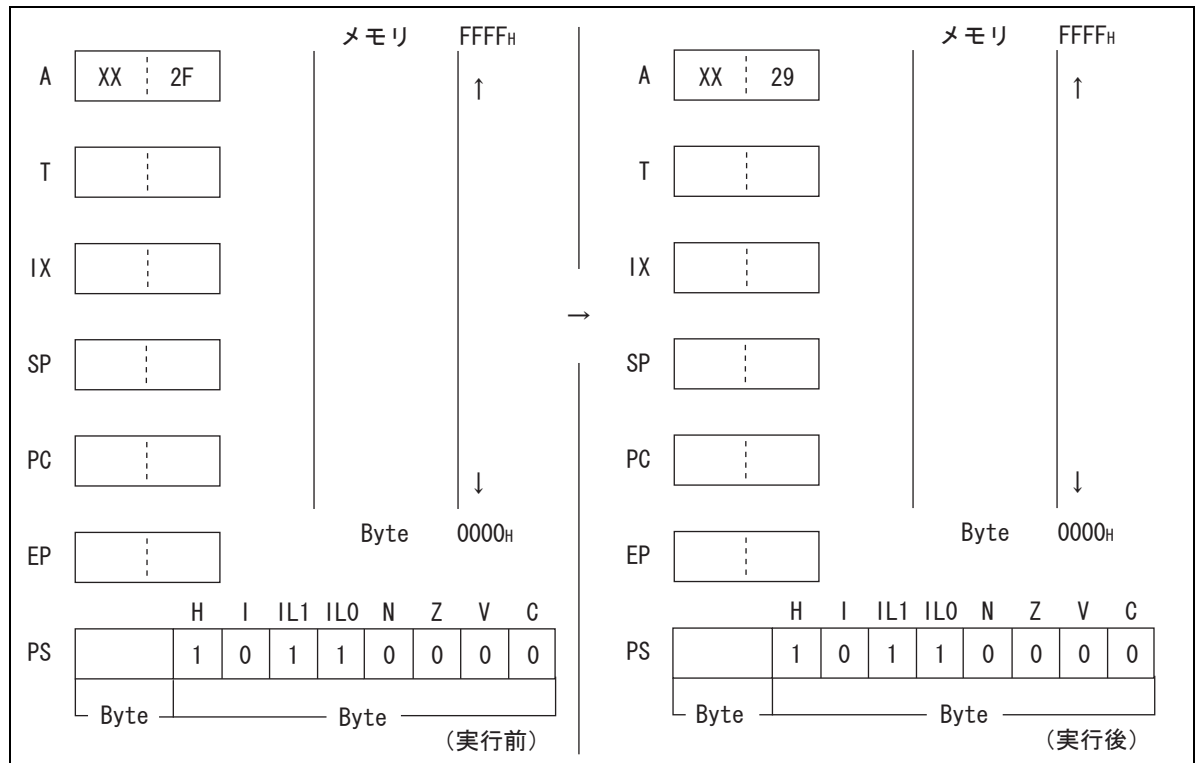


表 6-8. 10 進補正表 (DAS)

C フラグ	H フラグ	補正值	実行後の C フラグ
0	0	00	0
1	1	66	1
0	1	06	0
1	0	60	1

表 6-9. 実行例

ニーモニック	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)

Ri のバイトデータを 1 減算します。

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

オペレーション

$(Ri) \leftarrow (Ri) - 1$  (バイト減算)

アセンブラ形式

DEC Ri

コンディションコード (CCR)

N	Z	V	C
+	+	+	-

+: 命令実行により変化します

-: 変化しません

N: 演算結果が MSB=1 ならば 1 となり, それ以外は 0 となります

Z: 演算結果が 00<sub>H</sub> ならば 1 となり, それ以外は 0 となります

V: 演算の結果オーバーフローが発生したときに 1 となり, それ以外は 0 となります

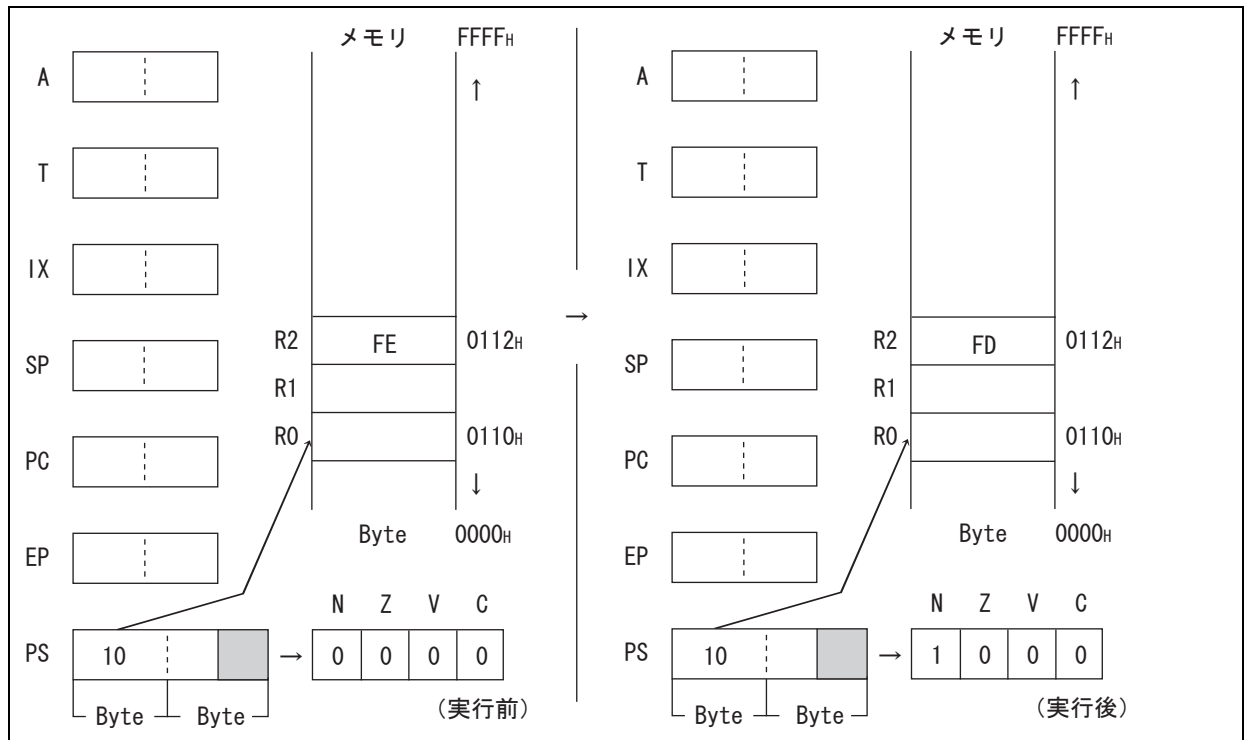
C: 変化しません

実行サイクル数: 3

バイト数: 1

オペコード: D8 ~ DF

実行例 : DEC R2



## 6.29 DECW (DECrement Word Data of Accumulator)

A のワードデータを 1 減算します。

### DECW (DECrement Word Data of Accumulator)

オペレーション

$(A) \leftarrow (A) - 1$  (ワード減算)

アセンブラ形式

DECW A

コンディションコード (CCR)

N	Z	V	C
+	+	-	-

+: 命令実行により変化します

-: 変化しません

N: 演算結果が MSB=1 ならば 1 となり, それ以外は 0 となります

Z: 演算結果が 0000<sub>H</sub> ならば 1 となり, それ以外は 0 となります

V: 変化しません

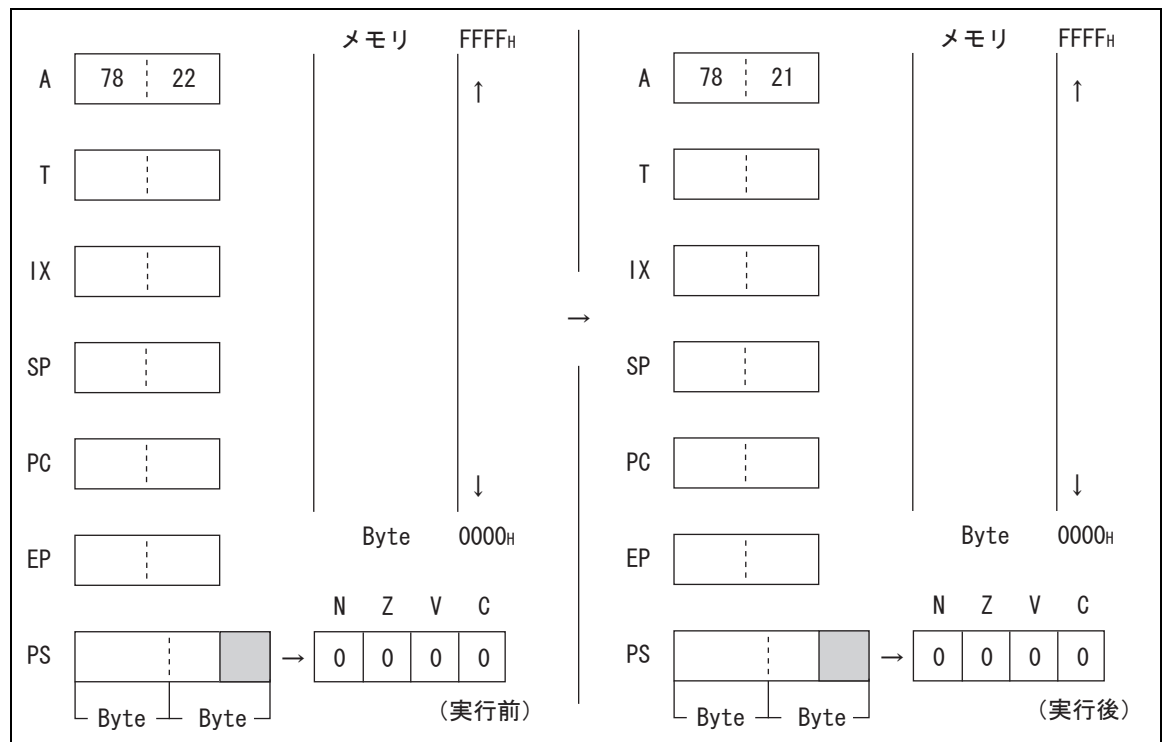
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: D0

実行例 : DECW A





## 6.30 DECW (DECrement Word Data of Extra Pointer)

EP のワードデータを 1 減算します。

### DECW (DECrement Word Data of Extra Pointer)

オペレーション

$(EP) \leftarrow (EP) - 1$  (ワード減算)

アセンブラ形式

DECW EP

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

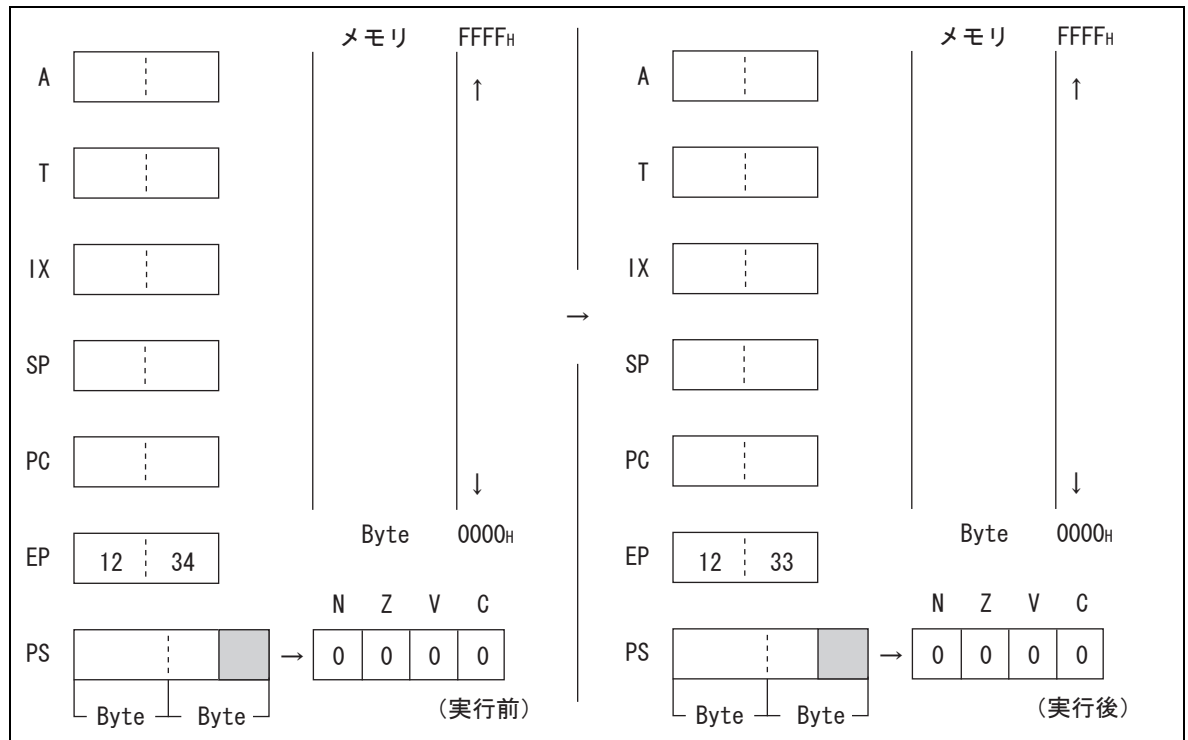
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: D3

実行例 : DECW EP



## 6.31 DECW (DECrement Word Data of Index Pointer)

IX のワードデータを 1 減算します。

### DECW (DECrement Word Data of Index Pointer)

オペレーション

$(IX) \leftarrow (IX) - 1$  (ワード減算)

アセンブラ形式

DECW IX

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

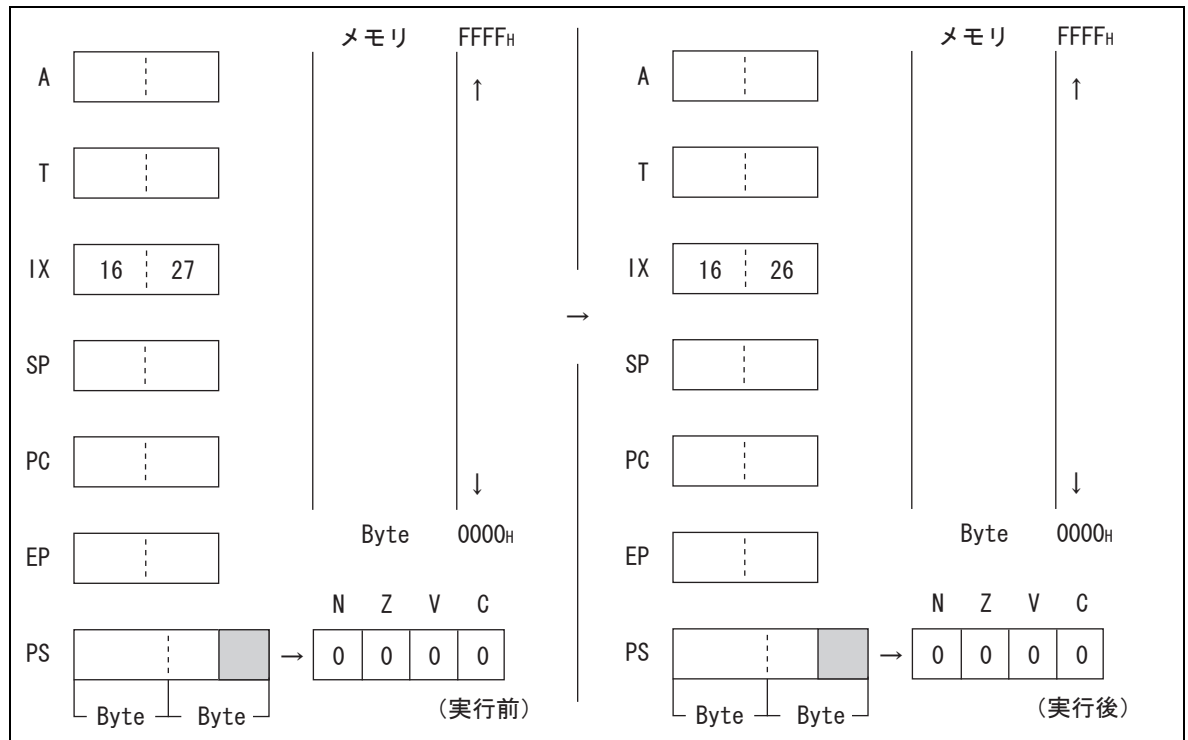
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: D2

実行例 : DECW IX



## 6.32 DECW (DECrement Word Data of Stack Pointer)

SP のワードデータを 1 減算します。

### DECW (DECrement Word Data of Stack Pointer)

オペレーション

$(SP) \leftarrow (SP) - 1$  (ワード減算)

アセンブラ形式

DECW SP

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

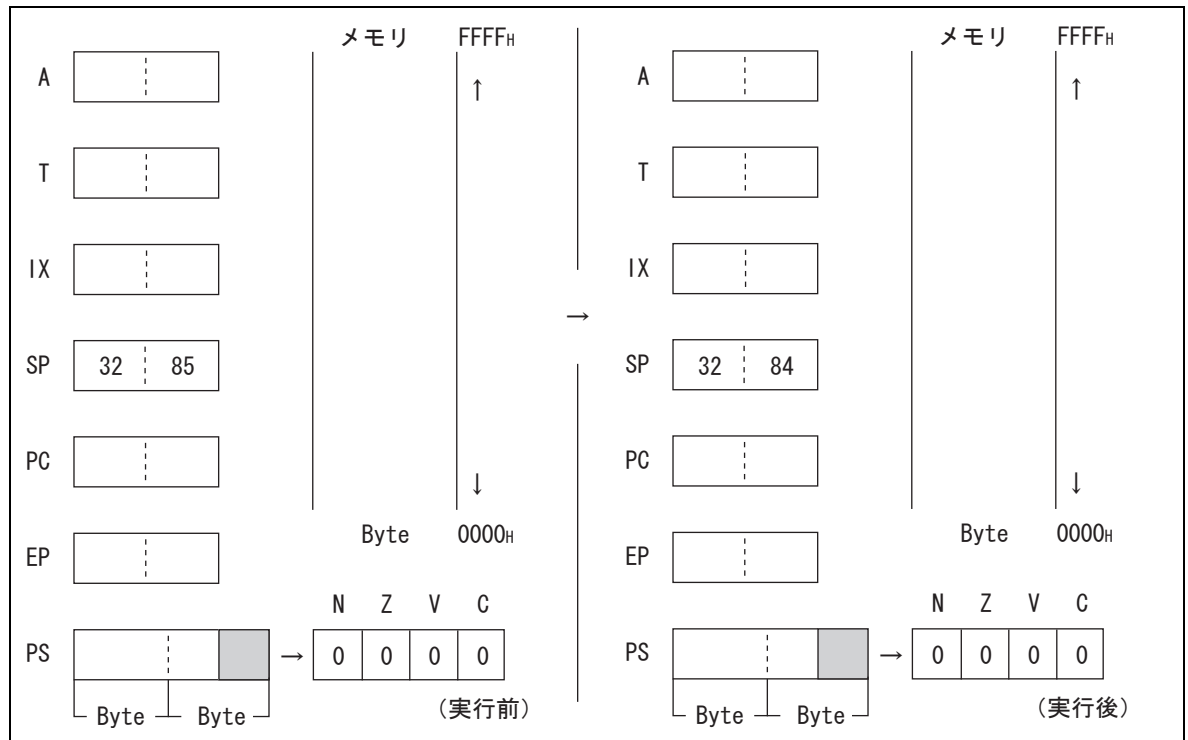
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: D1

実行例 : DECW SP



### 6.33 DIVU (DIVide Unsigned)

T のワードデータを A のワードデータで、符号なし 2 進数として除算します。  
 結果の商を A に余りを T に戻します。  
 また、A が 0 のときは、結果は不定で、0 除算を示すために Z フラグが 1 になります。

#### DIVU (DIVide Unsigned)

オペレーション

商 (A)  $\leftarrow (T) / (A)$

余 (T)  $\leftarrow (T) \text{ MOD } (A)$

アセンブラ形式

DIVU A

コンディションコード (CCR)

N	Z	V	C
-	+	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 命令実行前の A が 0000<sub>H</sub> ならば 1 になり、それ以外では 0 になります。

V: 変化しません

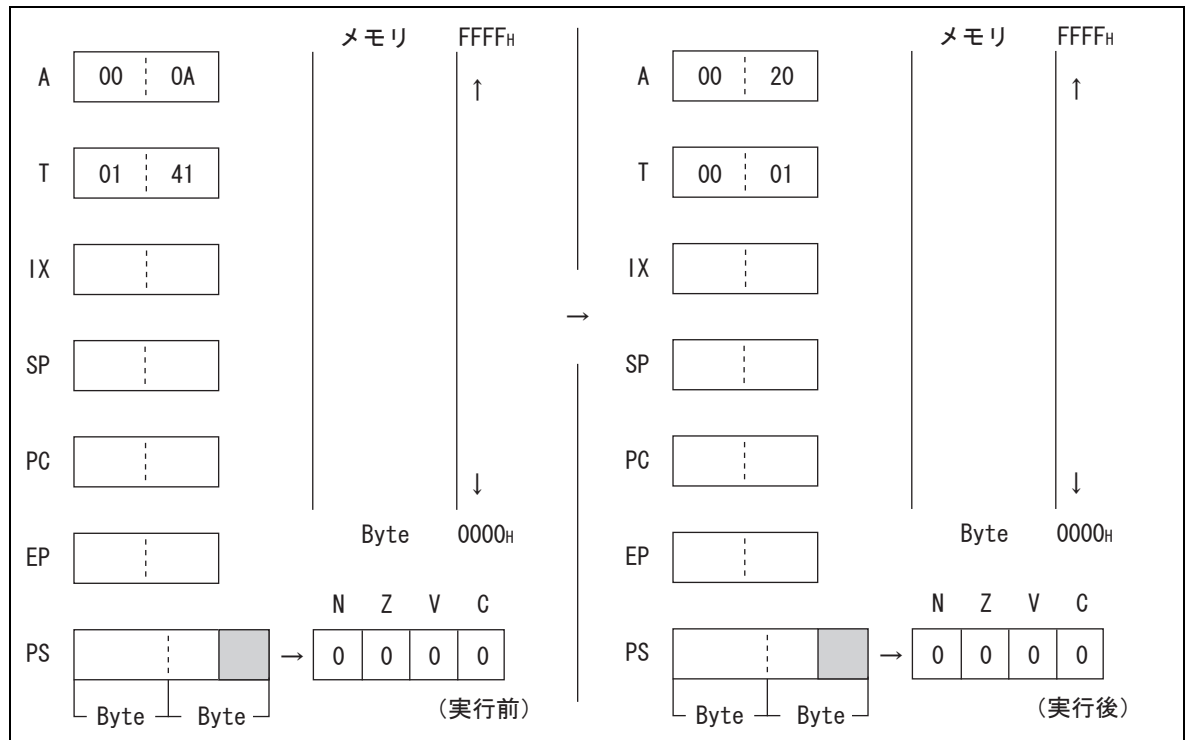
C: 変化しません

実行サイクル数: 17

バイト数: 1

オペコード: 11

実行例 : DIVU A





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

Ri のバイトデータに 1 を加算します。

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

オペレーション

$(Ri) \leftarrow (Ri) + 1$  (ワード加算)

アセンブラ形式

INC Ri

コンディションコード (CCR)

N	Z	V	C
+	+	+	-

+: 命令実行により変化します

-: 変化しません

N: 演算結果が MSB=1 ならば 1 となり, それ以外は 0 となります

Z: 演算結果が 00<sub>H</sub> ならば 1 となり, それ以外は 0 となります

V: 演算の結果オーバーフローが発生したときに 1 となり, それ以外は 0 となります

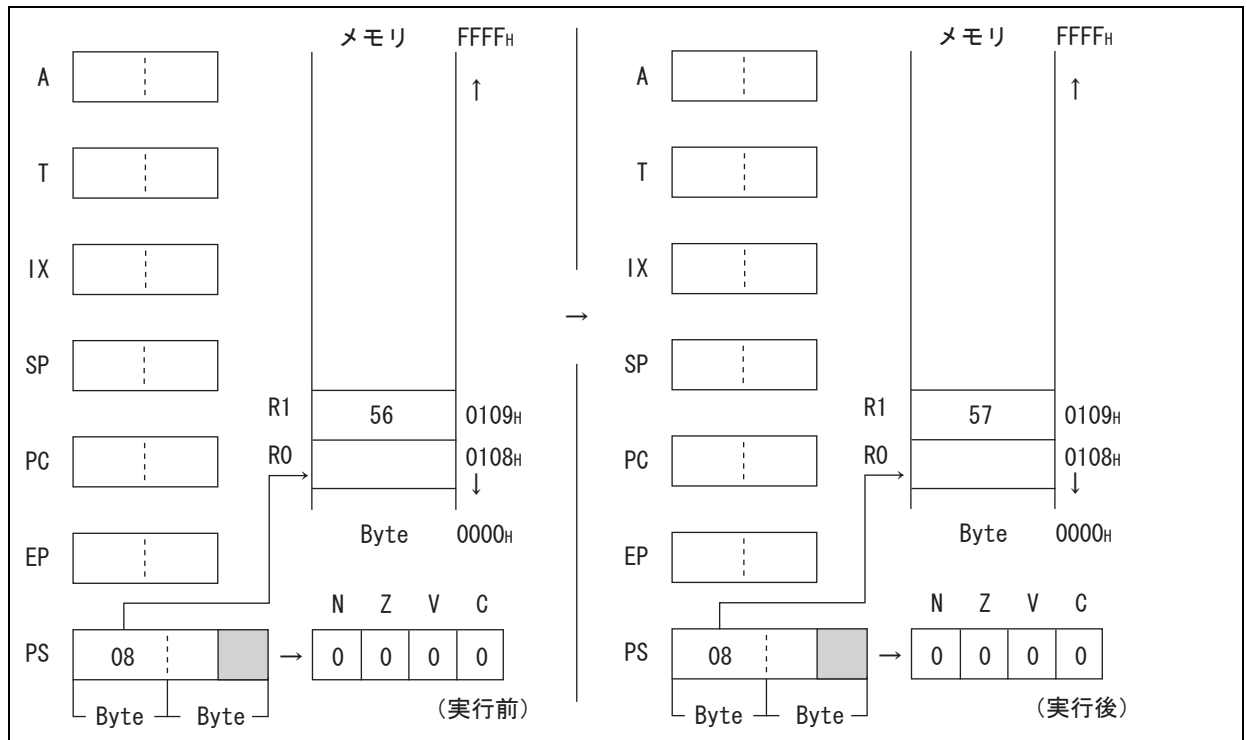
C: 変化しません

実行サイクル数: 3

バイト数: 1

オペコード: C8 ~ CF

実行例 : INC R1



## 6.35 INCW (INCrement Word Data of Accumulator)

A のワードデータに 1 を加算します。

### INCW (INCrement Word Data of Accumulator)

オペレーション

$(A) \leftarrow (A) + 1$  (ワード加算)

アセンブラ形式

INCW A

コンディションコード (CCR)

N	Z	V	C
+	+	-	-

+: 命令実行により変化します

-: 変化しません

N: 演算結果の A が MSB=1 ならば 1 となり, それ以外は 0 となります

Z: 演算結果が 0000<sub>H</sub> ならば 1 となり, それ以外は 0 となります

V: 変化しません

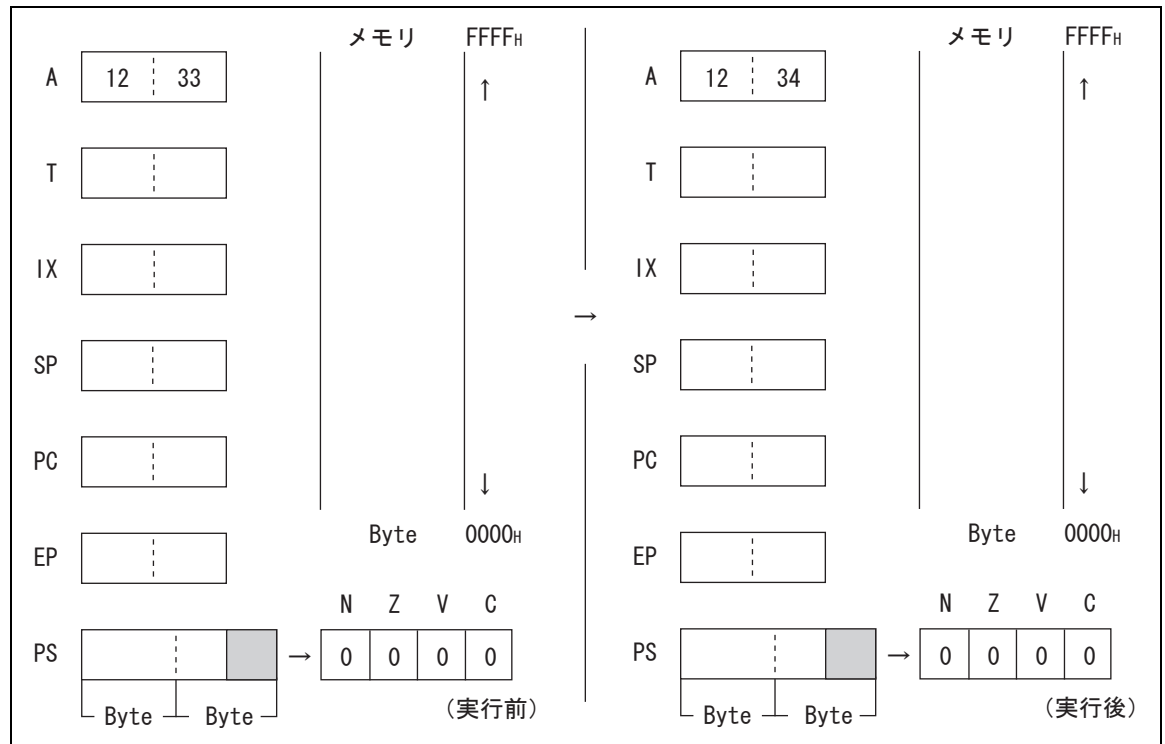
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: C0

実行例 : INCW A



## 6.36 INCW (INCrement Word Data of Extra Pointer)

EP のワードデータに 1 を加算します。

### INCW (INCrement Word Data of Extra Pointer)

オペレーション

$(EP) \leftarrow (EP) + 1$  (ワード加算)

アセンブラ形式

INCW EP

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

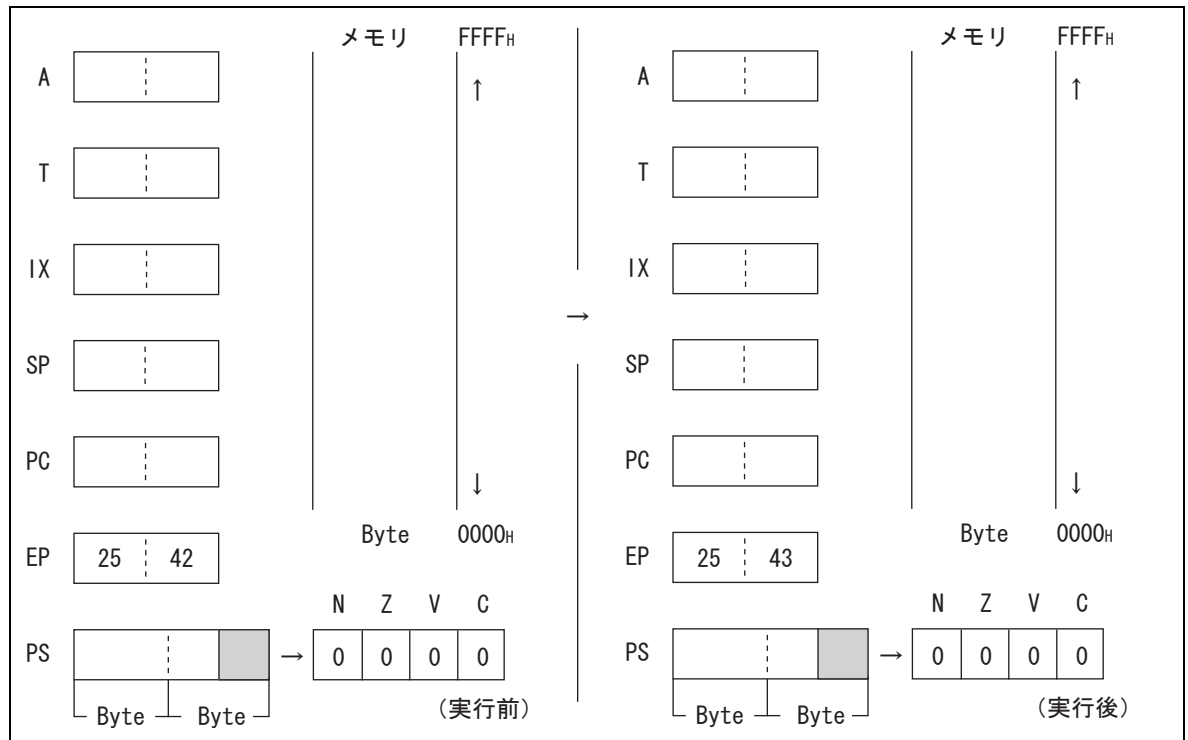
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: C3

実行例 : INCW EP



## 6.37 INCW (INCrement Word Data of Index Register)

IX のワードデータに 1 を加算します。

### INCW (INCrement Word Data of Index Register)

オペレーション

$(IX) \leftarrow (IX) + 1$  (ワード加算)

アセンブラ形式

INCW IX

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

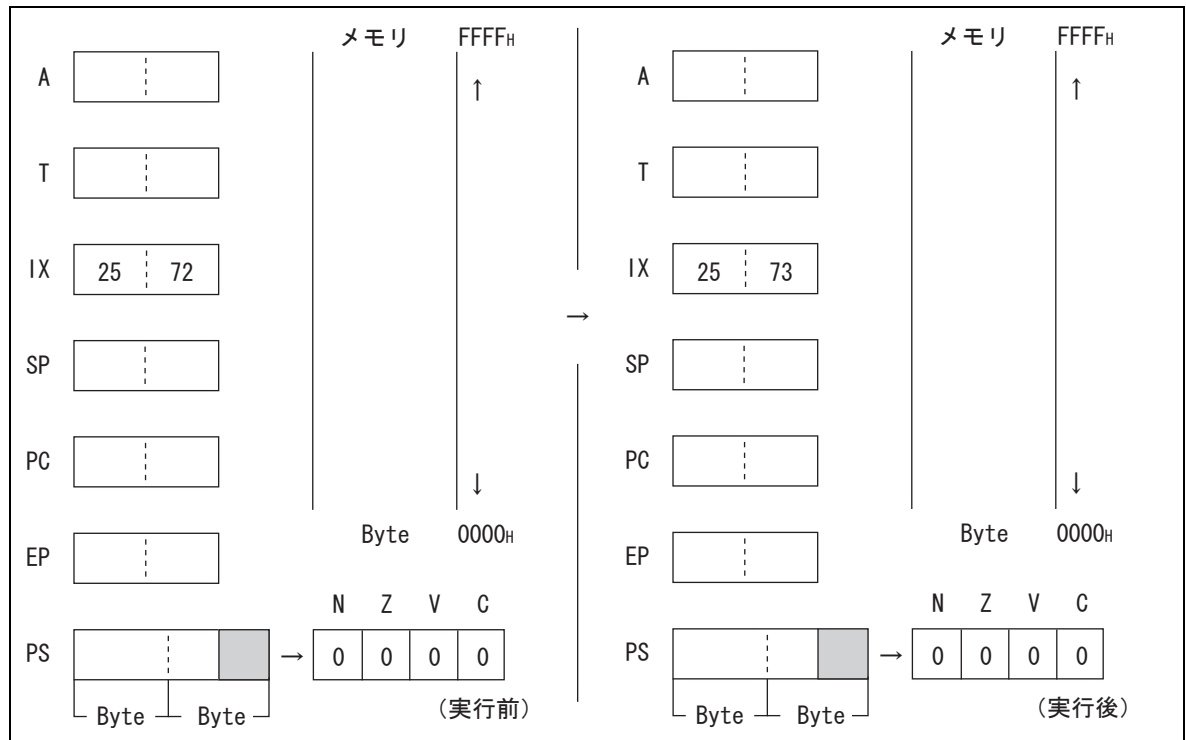
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: C2

実行例 : INCW IX





## 6.38 INCW (INCrement Word Data of Stack Pointer)

SP のワードデータに 1 を加算します。

### INCW (INCrement Word Data of Stack Pointer)

オペレーション

$(SP) \leftarrow (SP) + 1$  (ワード加算)

アセンブラ形式

INCW SP

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

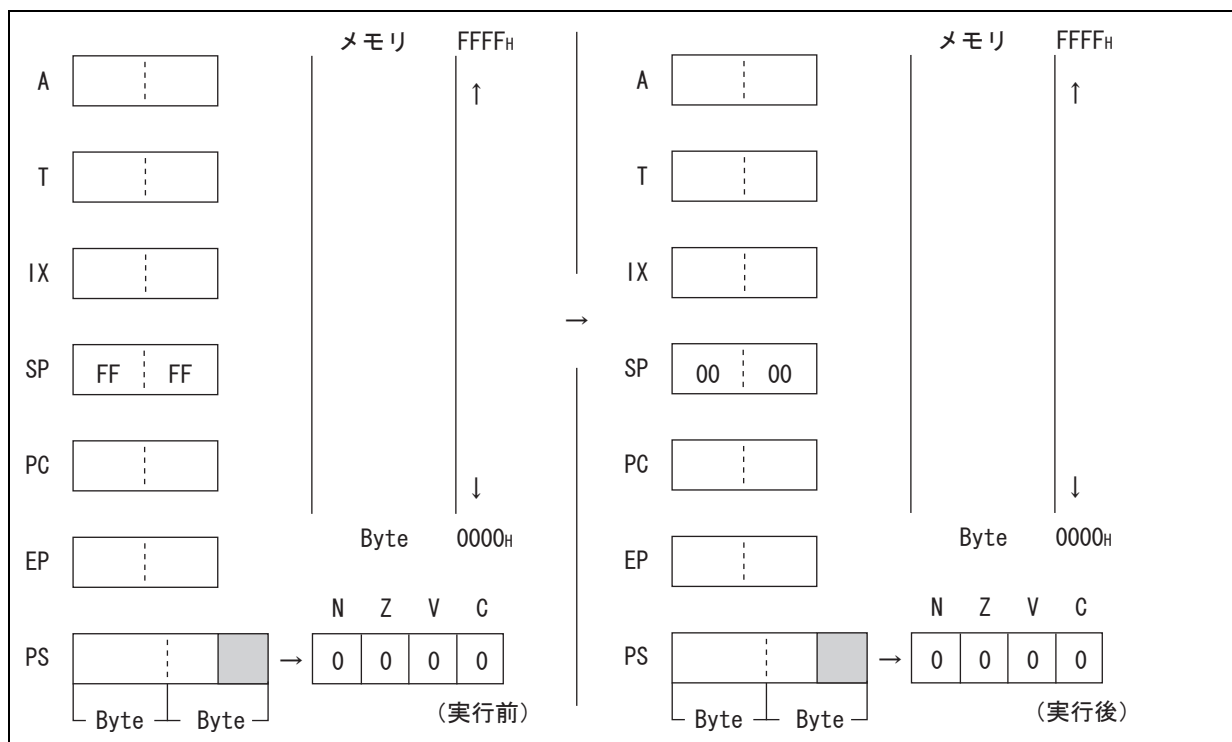
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: C1

実行例 : INCW SP



## 6.39 JMP (JuMP to address pointed by Accumulator)

A のワードデータを PC へ転送します。

### JMP (JuMP to address pointed by Accumulator)

オペレーション

(PC) ← (A) (ワード転送)

アセンブラ形式

JMP @A

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

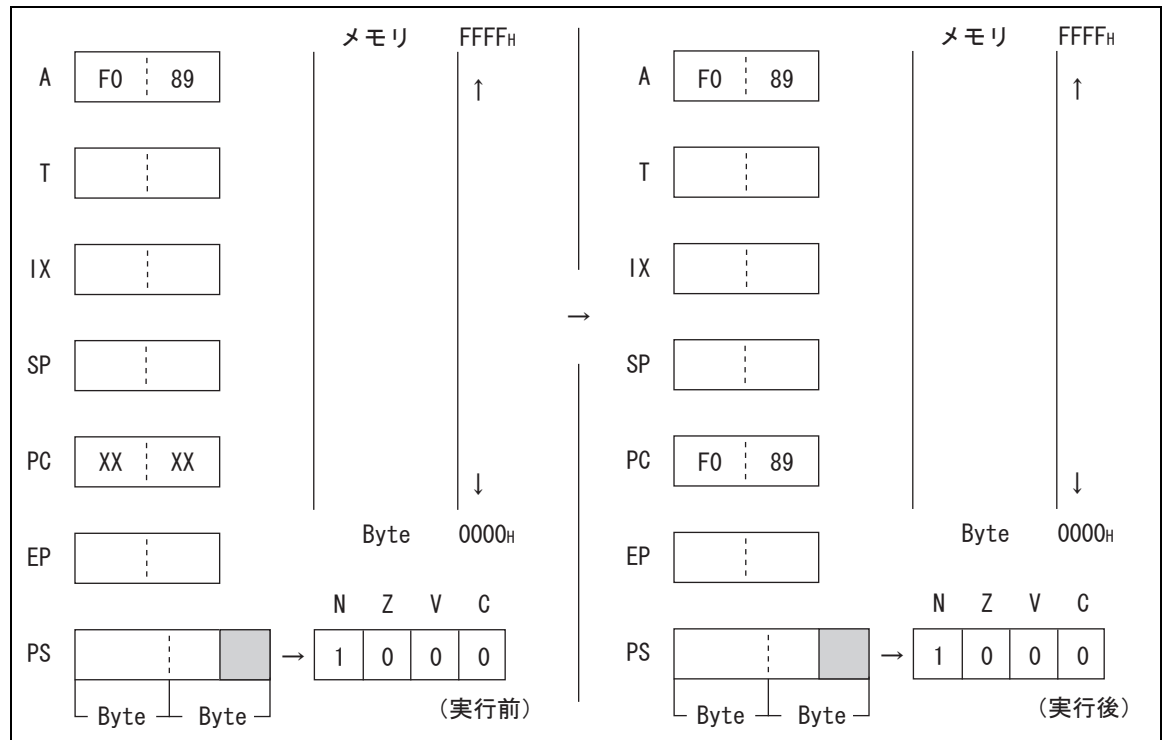
C: 変化しません

実行サイクル数: 3

バイト数: 1

オペコード: E0

実行例 : JMP @A



## 6.40 JMP (JuMP to effective Address)

ext で示される PC 値へ分岐します。

### JMP (JuMP to effective Address)

オペレーション

(PC) ← ext (ワード転送)

アセンブラ形式

JMP ext

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

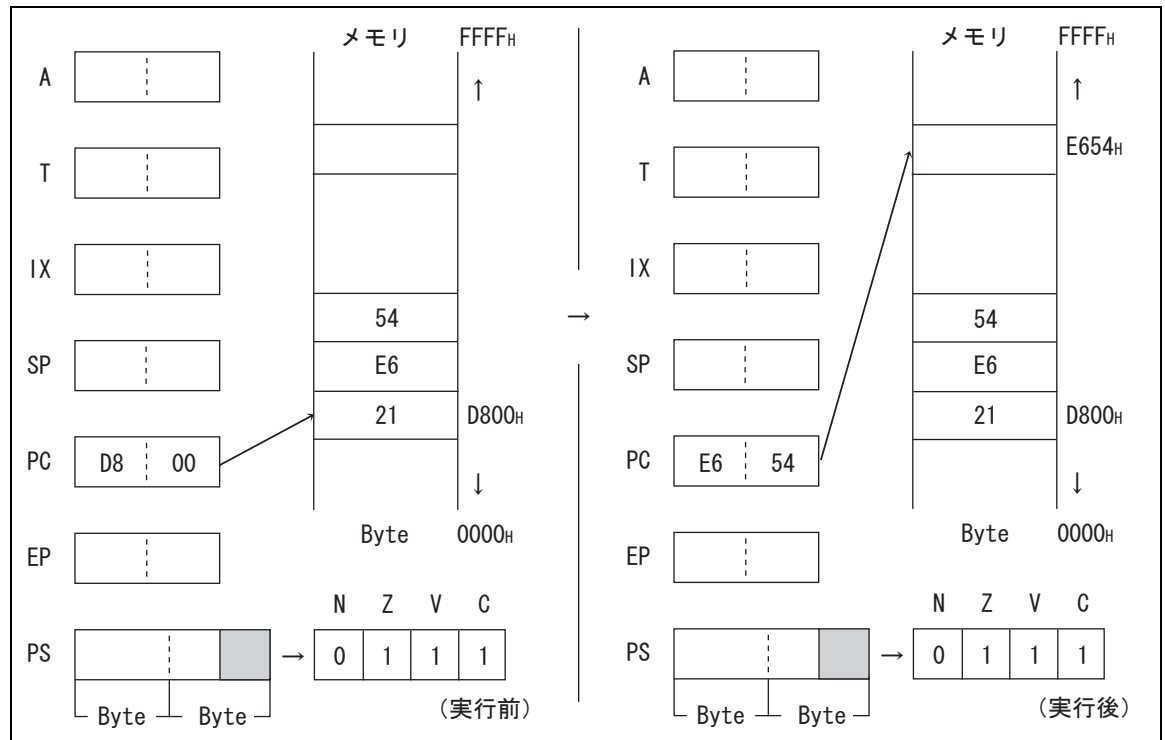
C: 変化しません

実行サイクル数: 4

バイト数: 3

オペコード: 21

実行例 : JMP 0E654H



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

T のバイトデータを A によって間接アドレスされるメモリへ転送します。

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

オペレーション

((A)) ← T (ワード転送)

アセンブラ形式

MOV @A, T

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

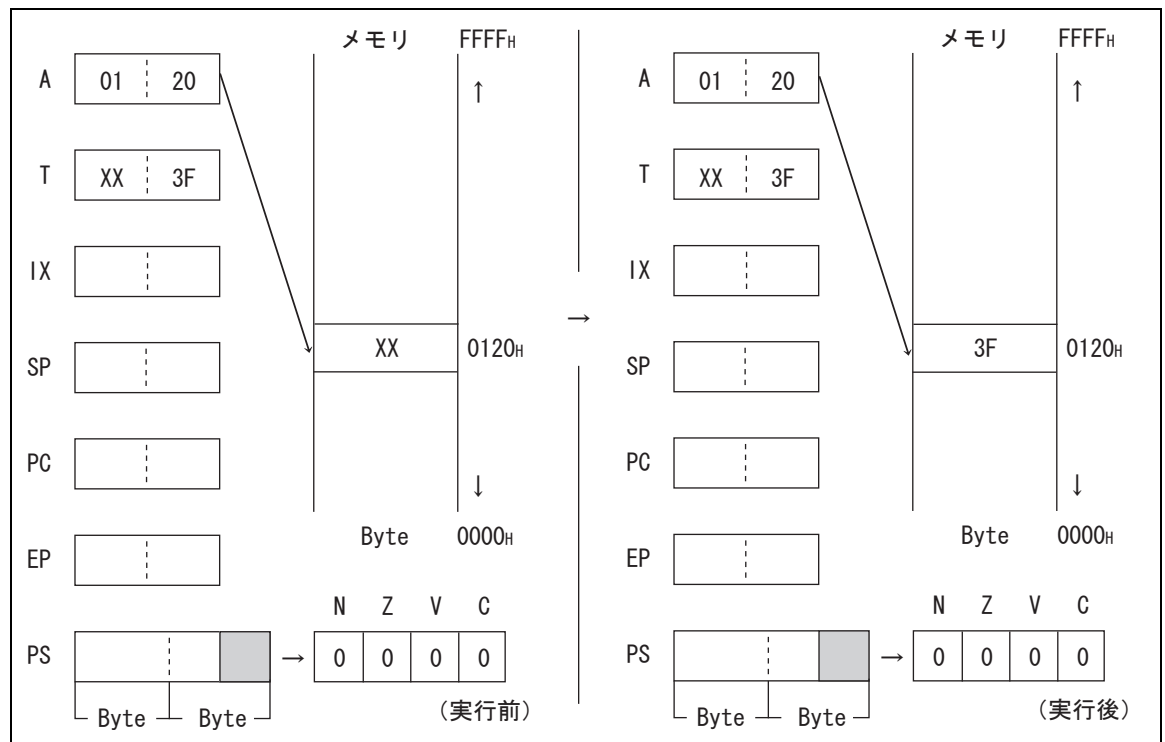
C: 変化しません

実行サイクル数: 2

バイト数: 1

オペコード: 82

実行例 : MOV @A, T





## 6.42 MOV (MOVE Byte Data from Memory to Accumulator)

EA メモリ ( 各種アドレッシングで表現されるメモリ ) のバイトデータを A へ転送します。  
 AL にあったバイトデータは、TL に転送されます。  
 AH は変化しません。

### MOV (MOVE Byte Data from Memory to Accumulator)

オペレーション

(AL) ← (EA) ( バイト転送 )

アセンブラ形式

MOV A, EA

コンディションコード (CCR)

N	Z	V	C
+	+	-	-

+: 命令実行により変化します

-: 変化しません

N: 転送したデータが MSB=1 ならば 1 となり、それ以外は 0 となります

Z: 転送したデータが 00<sub>H</sub> ならば 1 となり、それ以外は 0 となります

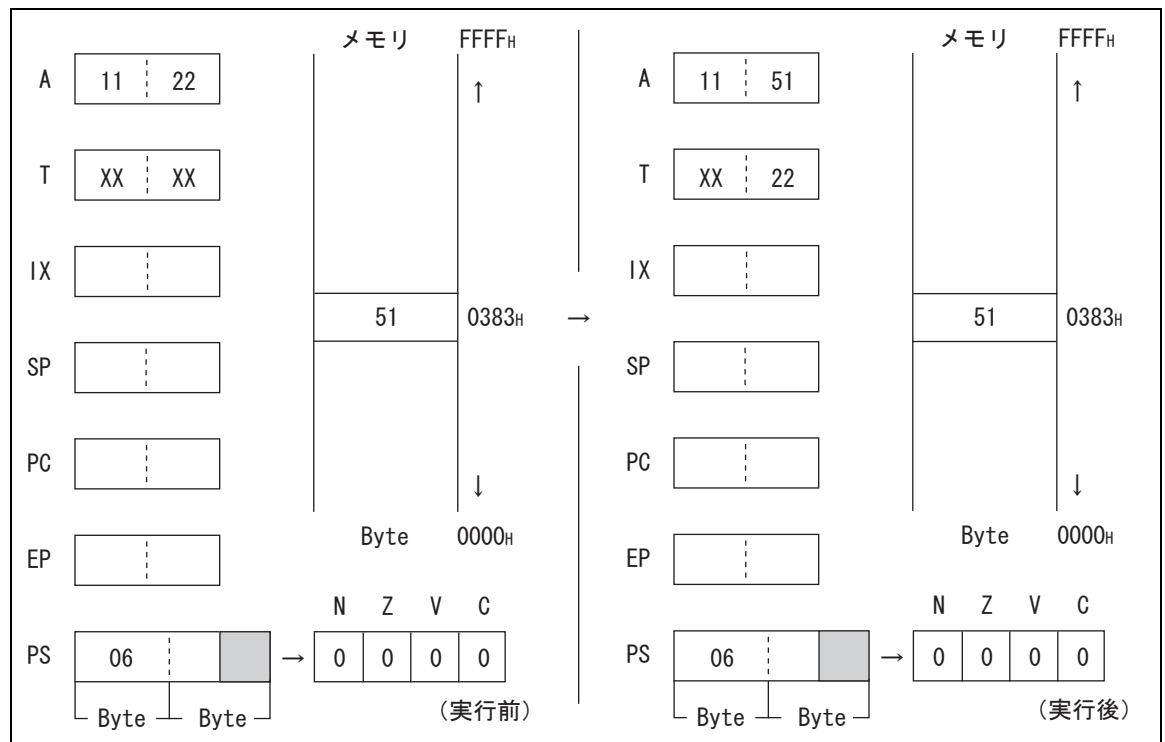
V: 変化しません

C: 変化しません

表 6-10. 実行サイクル数 / バイト数 / オペコード

EA	#d8	dir	@lX+off	ext	@A	@EP	Ri
実行サイクル数	2	3	3	4	2	2	2
バイト数	2	2	2	3	1	1	1
オペコード	04	05	06	60	92	07	08 ~ 0F

実行例 : MOV A, 83H



## 6.43 MOV (MOVE Immediate Byte Data to Memory)

バイトの即値データを EA メモリ ( 各種アドレッシングで表現されるメモリ ) へ転送します。

### MOV (MOVE Immediate Byte Data to Memory)

オペレーション

(EA) ← d8 ( バイト転送 )

アセンブラ形式

MOV EA, #d8

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

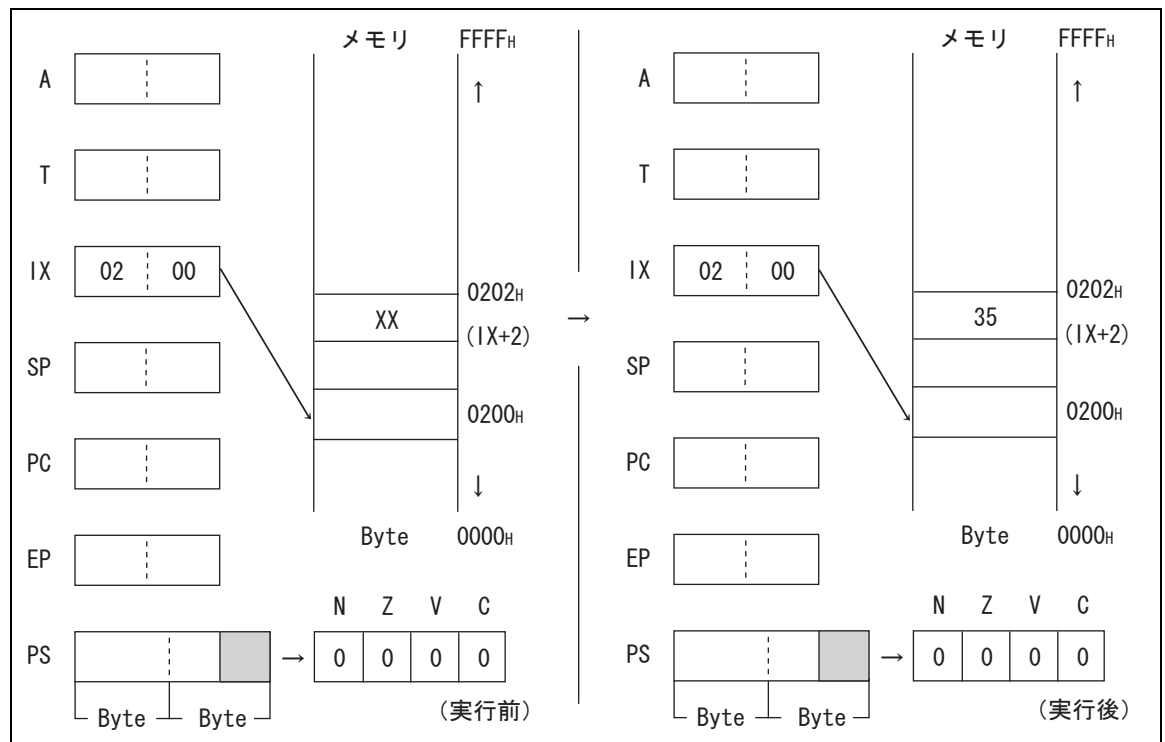
V: 変化しません

C: 変化しません

表 6-11. 実行サイクル数 / バイト数 / オペコード

EA	dir	@IX+off	@EP	Ri
実行サイクル数	4	4	3	3
バイト数	3	3	2	2
オペコード	85	86	87	88 ~ 8F

実行例 : MOV @IX+02, #35H



## 6.44 MOV (MOVE Byte Data from Accumulator to Memory)

AL のバイトデータを EA メモリ ( 各種アドレッシングで表現されるメモリ ) へ転送します。

### MOV (MOVE Byte Data from Accumulator to Memory)

オペレーション

(EA) ← (AL) ( バイト転送 )

アセンブラ形式

MOV EA, A

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

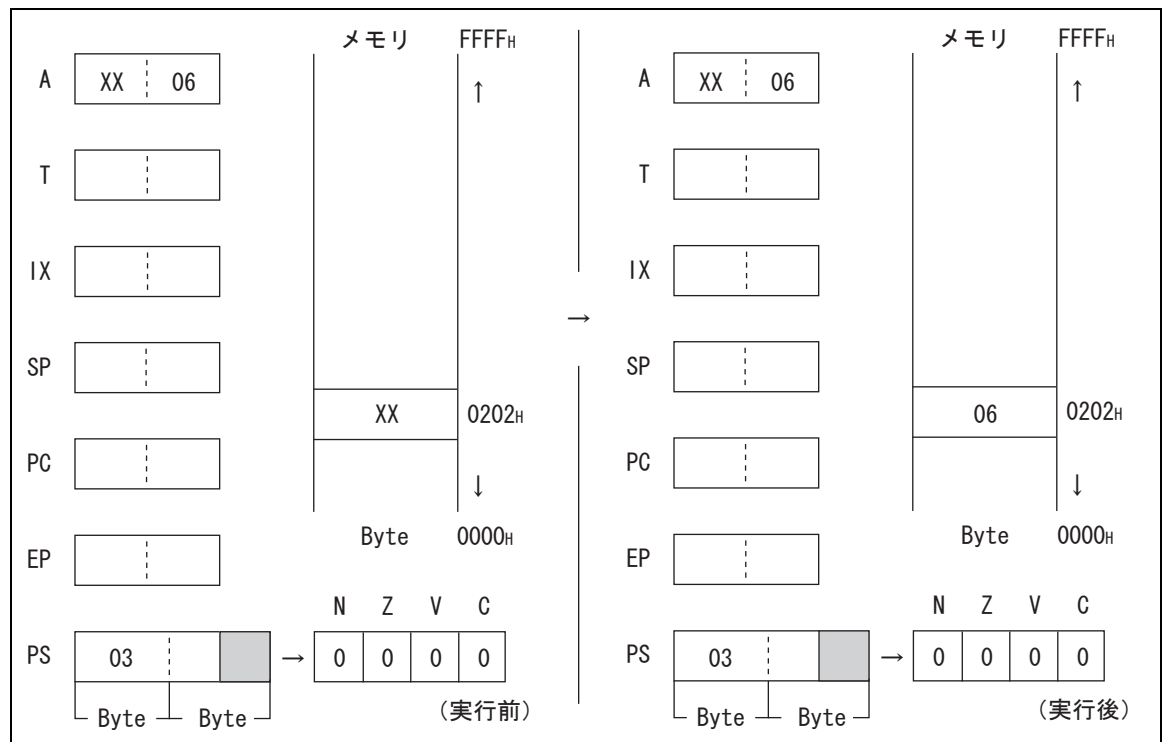
V: 変化しません

C: 変化しません

表 6-12. 実行サイクル数 / バイト数 / オペコード

EA	dir	@IX+off	ext	@EP	Ri
実行サイクル数	3	3	4	2	2
バイト数	2	2	3	1	1
オペコード	45	46	61	47	48 ~ 4F

実行例 : MOV 82H, A



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

T のワードデータを A によって間接アドレスされるメモリへ転送します。

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

オペレーション

((A)) ← (T) (ワード転送)

アセンブラ形式

MOVW @A, T

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

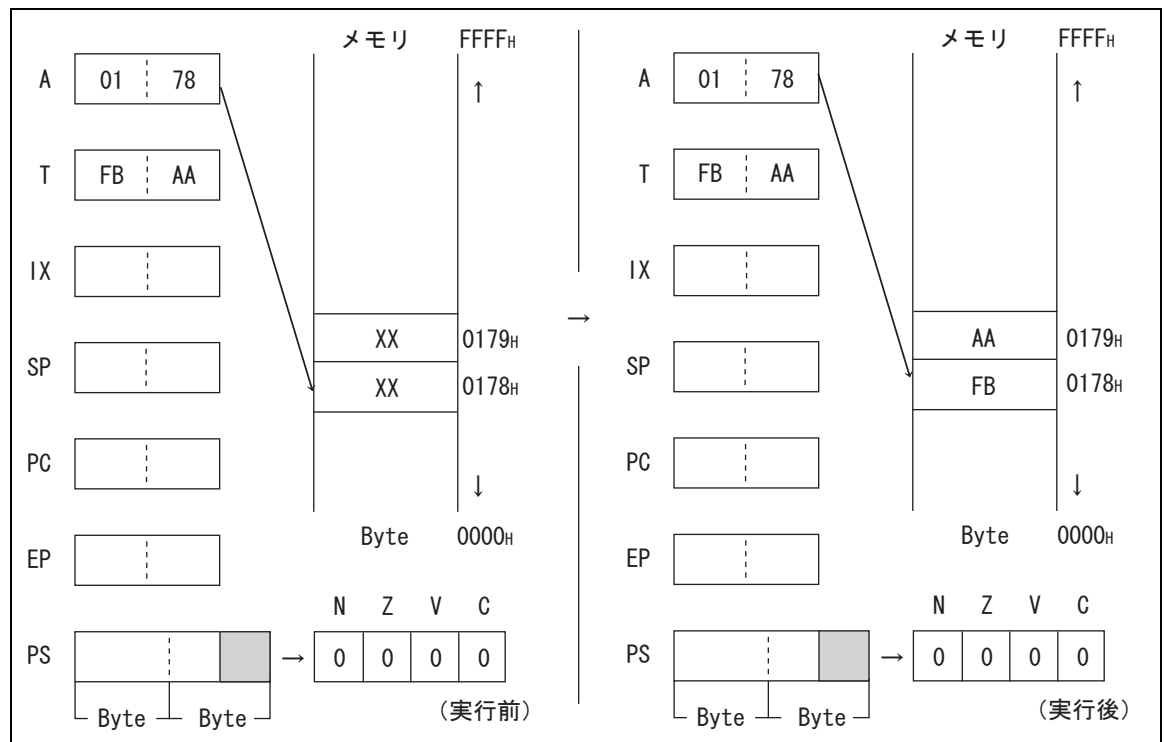
C: 変化しません

実行サイクル数: 3

バイト数: 1

オペコード: 83

実行例 : MOVW @A, T





## 6.46 MOVW (MOVE Word Data from Memory to Accumulator)

EA,EA+1 (EA は各種アドレッシングで表現されるアドレス) のメモリ内容のワードデータを A へ転送します。

また, A にあったワードデータは, T に転送されます。

### MOVW (MOVE Word Data from Memory to Accumulator)

オペレーション

(A) ← (EA) (ワード転送)

アセンブラ形式

MOVW A, EA

コンディションコード (CCR)

N	Z	V	C
+	+	-	-

+: 命令実行により変化します

-: 変化しません

N: 転送したデータが MSB=1 ならば 1 となり, それ以外は 0 となります

Z: 転送したデータが 0000<sub>H</sub> ならば 1 となり, それ以外は 0 となります

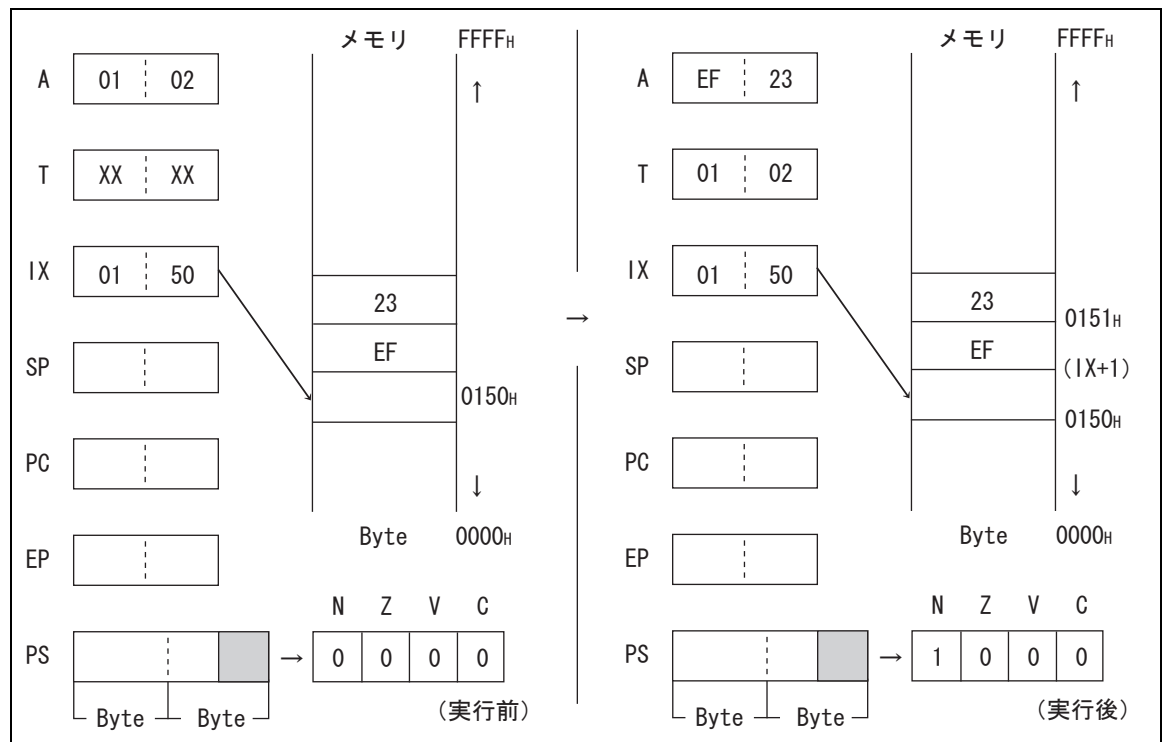
V: 変化しません

C: 変化しません

表 6-13. 実行サイクル数 / バイト数 / オペコード

EA	#d16	dir	@IX+off	ext	@A	@EP
実行サイクル数	3	4	4	5	3	3
バイト数	3	2	2	3	1	1
オペコード	E4	C5	C6	C4	93	C7

実行例 : MOVW A, @IX+01H



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

EP のワードデータを A へ転送します。

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

オペレーション

(A) ← (EP) (ワード転送)

アセンブラ形式

MOVW A, EP

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

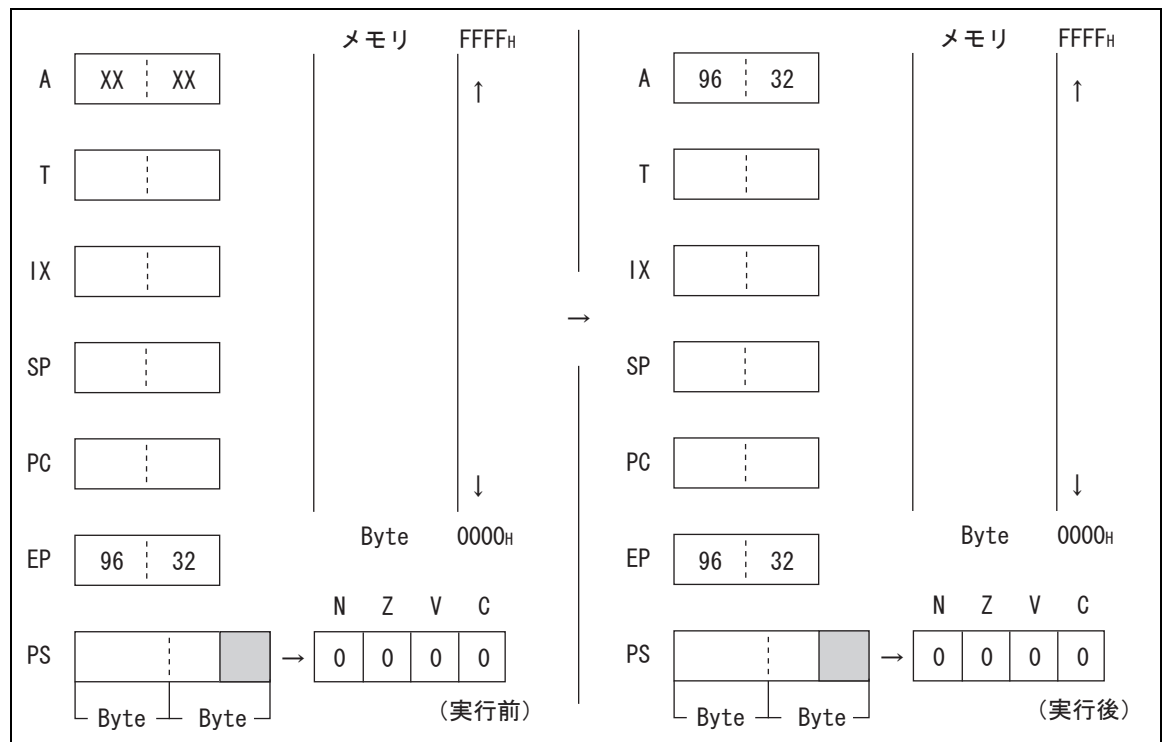
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: F3

実行例 : MOVW A, EP



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

IX のワードデータを A へ転送します。

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

オペレーション

(A) ← (IX) (ワード転送)

アセンブラ形式

MOVW A, IX

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

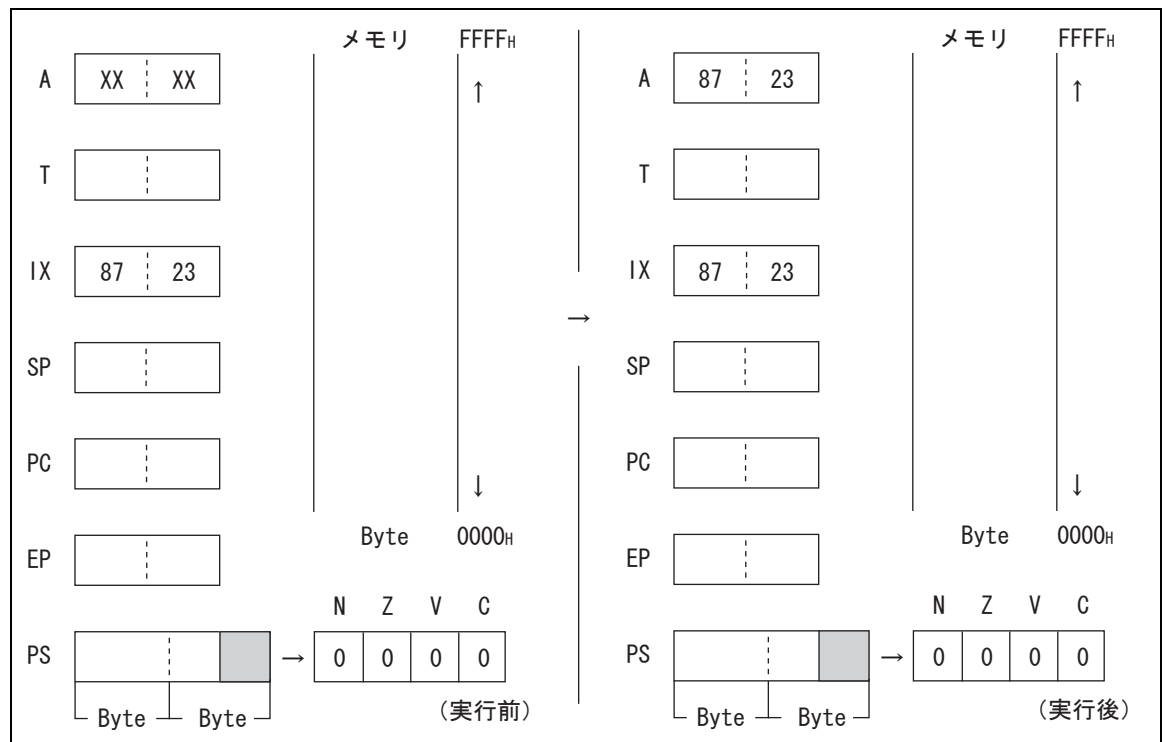
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: F2

実行例 : MOVW A, IX



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

PS のワードデータを A へ転送します。

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

オペレーション

(A) ← (PS) (ワード転送)

アセンブラ形式

MOVW A, PS

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

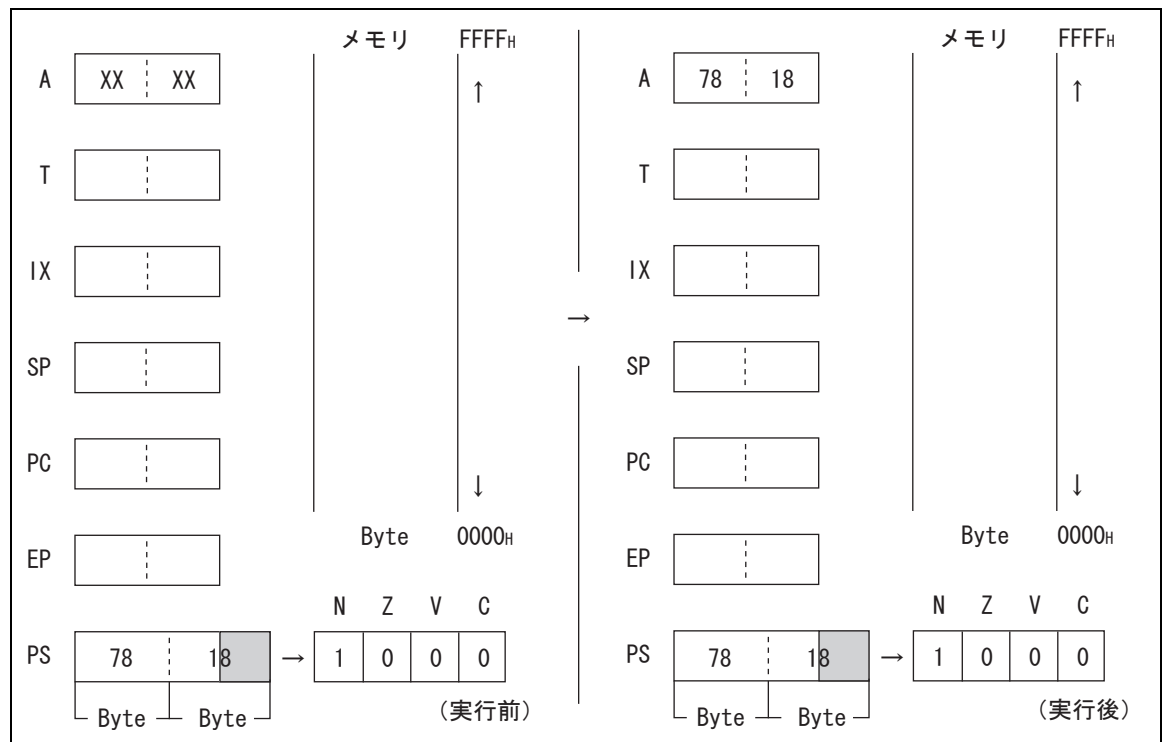
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: 70

実行例 : MOVW A, PS





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

PC のワードデータを A へ転送します。

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

オペレーション

(A) ← (PC) (ワード転送)

アセンブラ形式

MOVW A, PC

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

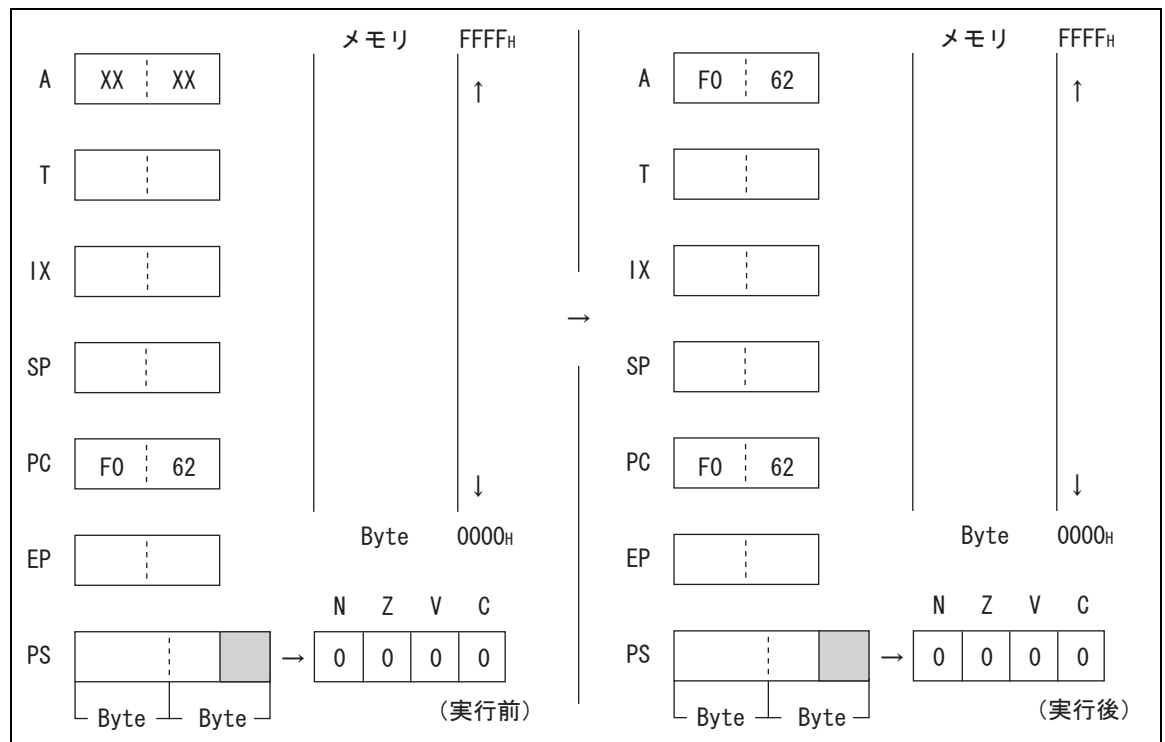
C: 変化しません

実行サイクル数: 2

バイト数: 1

オペコード: F0

実行例 : MOVW A, PC



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

SP のワードデータを A へ転送します。

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

オペレーション

(A) ← (SP) (ワード転送)

アセンブラ形式

MOVW A, SP

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

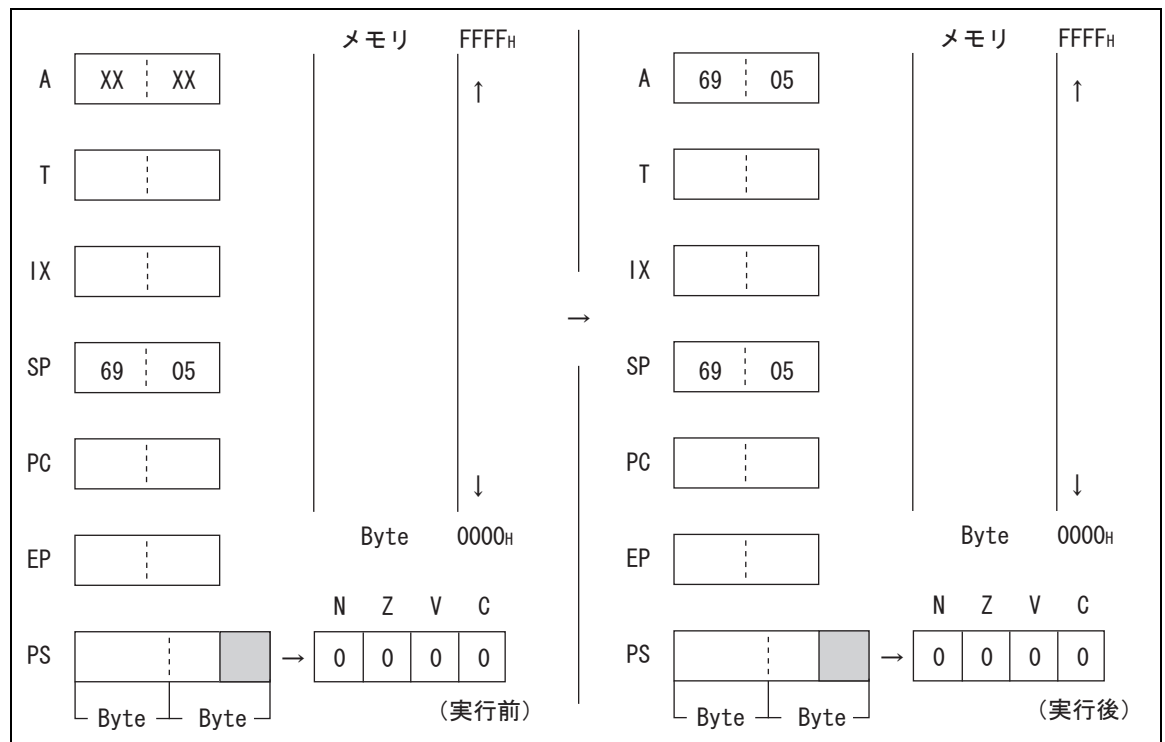
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: F1

実行例 : MOVW A, SP



## 6.52 MOVW (MOVE Word Data from Accumulator to Memory)

A のワードデータを EA,EA+1 (EA は各種アドレッシングで表現されるアドレス) のメモリへ転送します。

### MOVW (MOVE Word Data from Accumulator to Memory)

オペレーション

(EA) ← (A) (ワード転送)

アセンブラ形式

MOVW EA, A

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

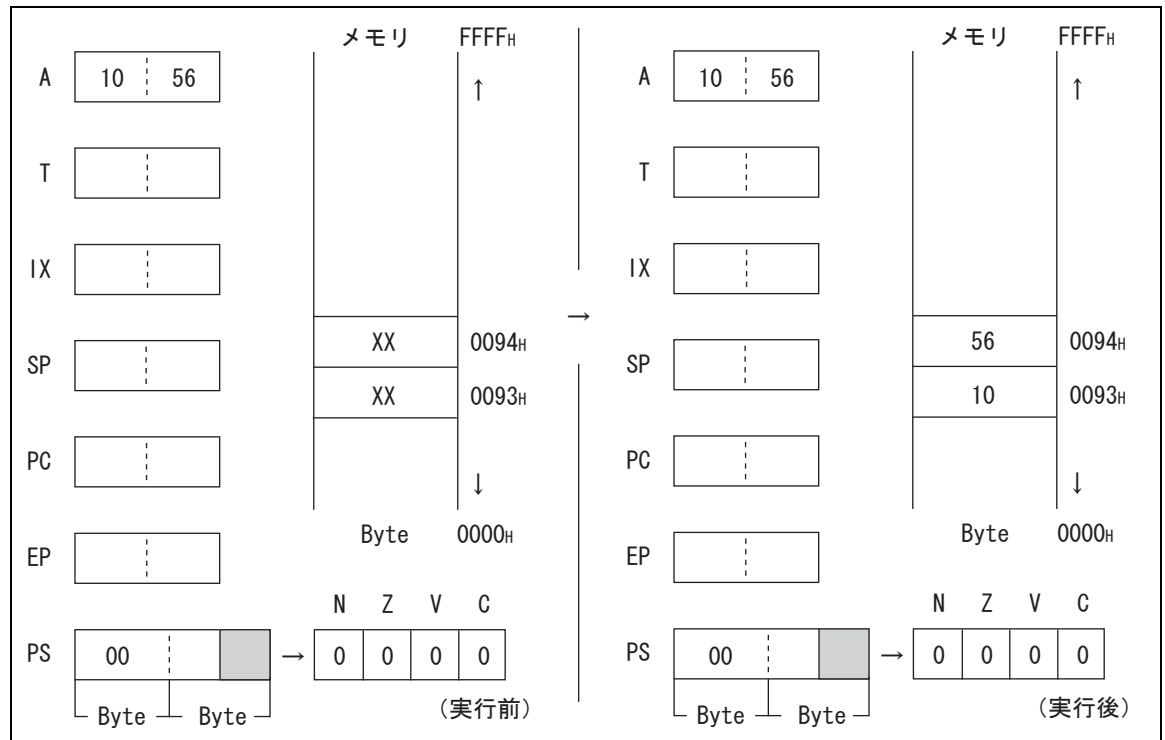
V: 変化しません

C: 変化しません

表 6-14. 実行サイクル数 / バイト数 / オペコード

EA	dir	@IX+off	ext	@EP
実行サイクル数	4	4	5	3
バイト数	2	2	3	1
オペコード	D5	D6	D4	D7

実行例 : MOVW 93H, A



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

A のワードデータを EP へ転送します。

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

オペレーション

(EP) ← (A) (ワード転送)

アセンブラ形式

MOVW EP, A

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

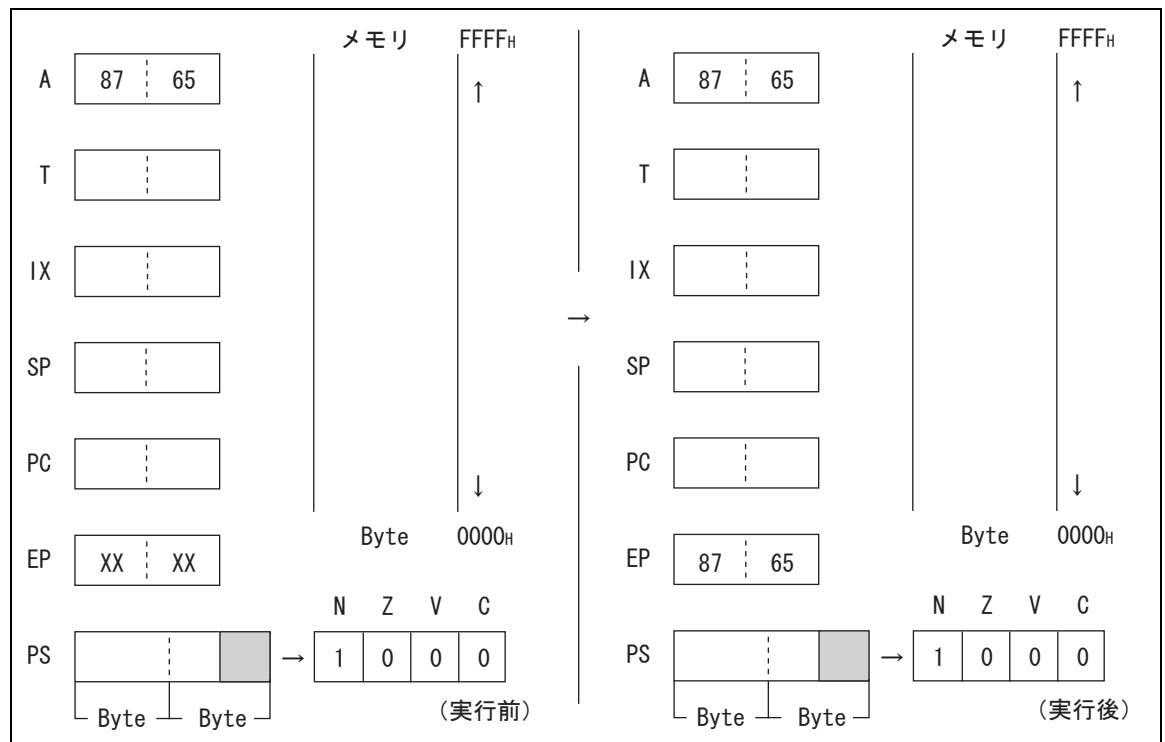
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: E3

実行例 : MOVW EP, A





## 6.54 MOVW (MOVE Immediate Word Data to Extra Pointer)

ワードの即値データを EP へ転送します。

### MOVW (MOVE Immediate Word Data to Extra Pointer)

オペレーション

(EP) ← d16 (ワード転送)

アセンブラ形式

MOVW EP, #d16

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

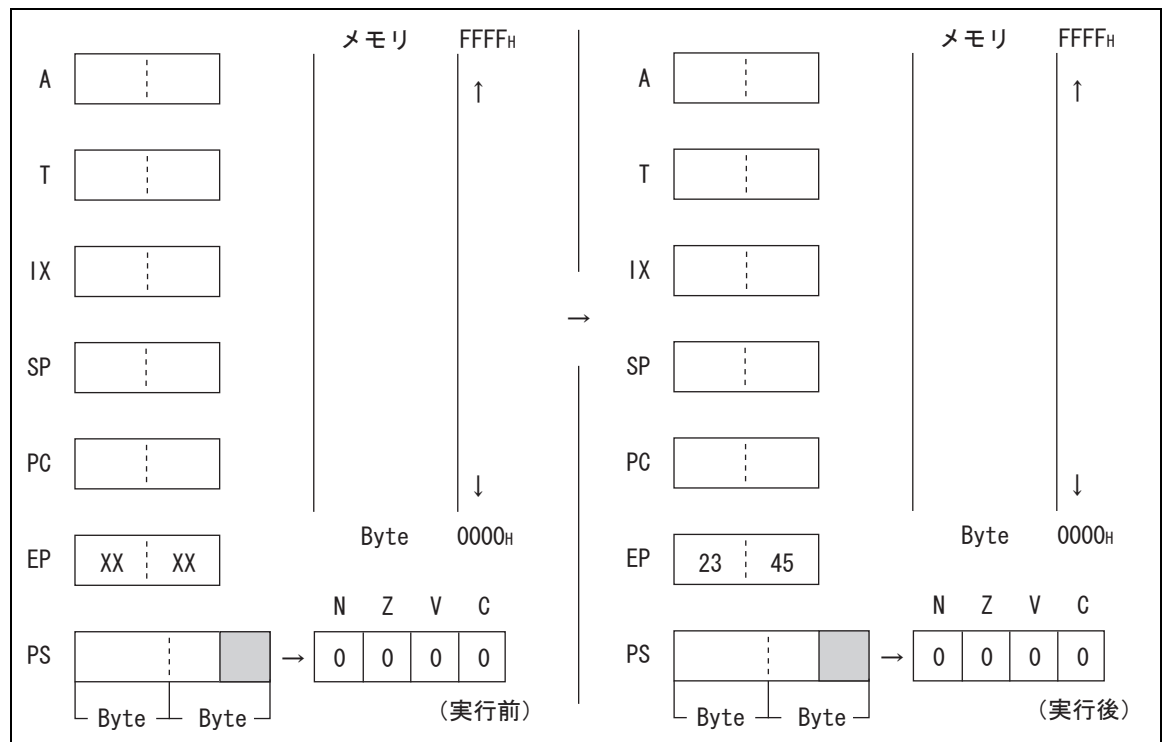
C: 変化しません

実行サイクル数: 3

バイト数: 3

オペコード: E7

実行例 : MOVW EP, #2345H



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

A のワードデータを IX へ転送します。

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

オペレーション

(IX) ← (A) (ワード転送)

アセンブラ形式

MOVW IX, A

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

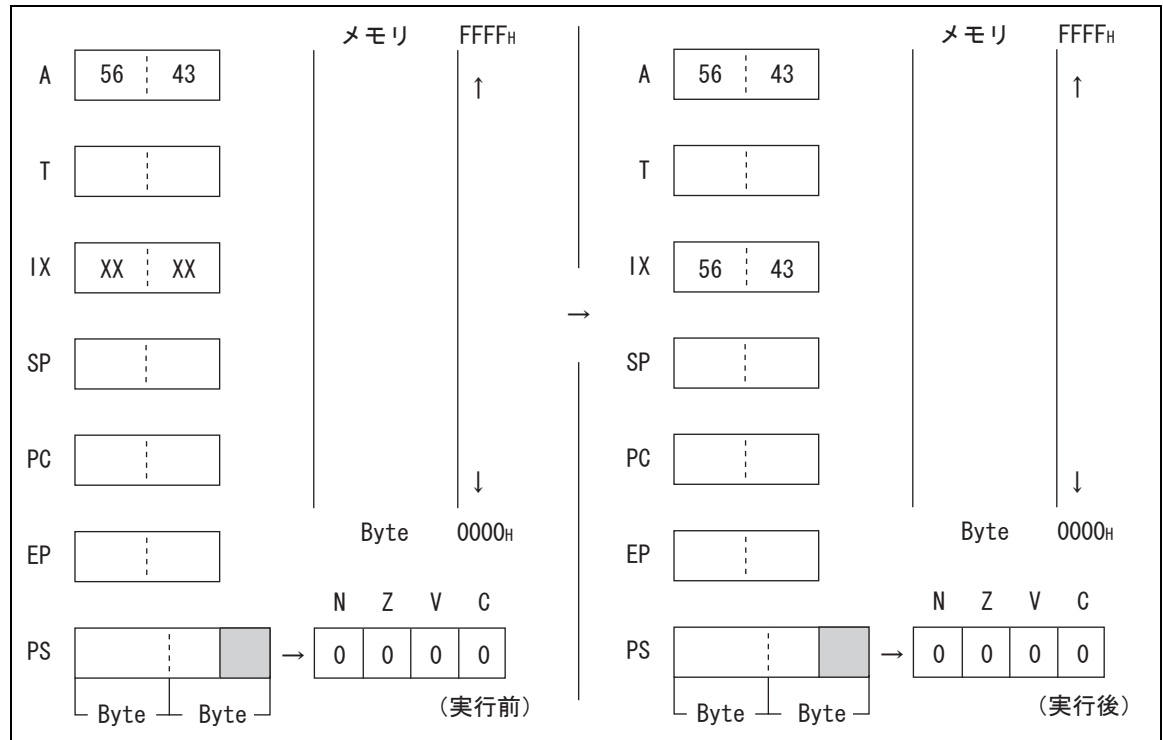
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: E2

実行例 : MOVW IX, A



## 6.56 MOVW (MOVE Immediate Word Data to Index Register)

ワードの即値データを IX へ転送します。

### MOVW (MOVE Immediate Word Data to Index Register)

オペレーション

(IX) ← d16 (ワード転送)

アセンブラ形式

MOVW IX, #d16

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

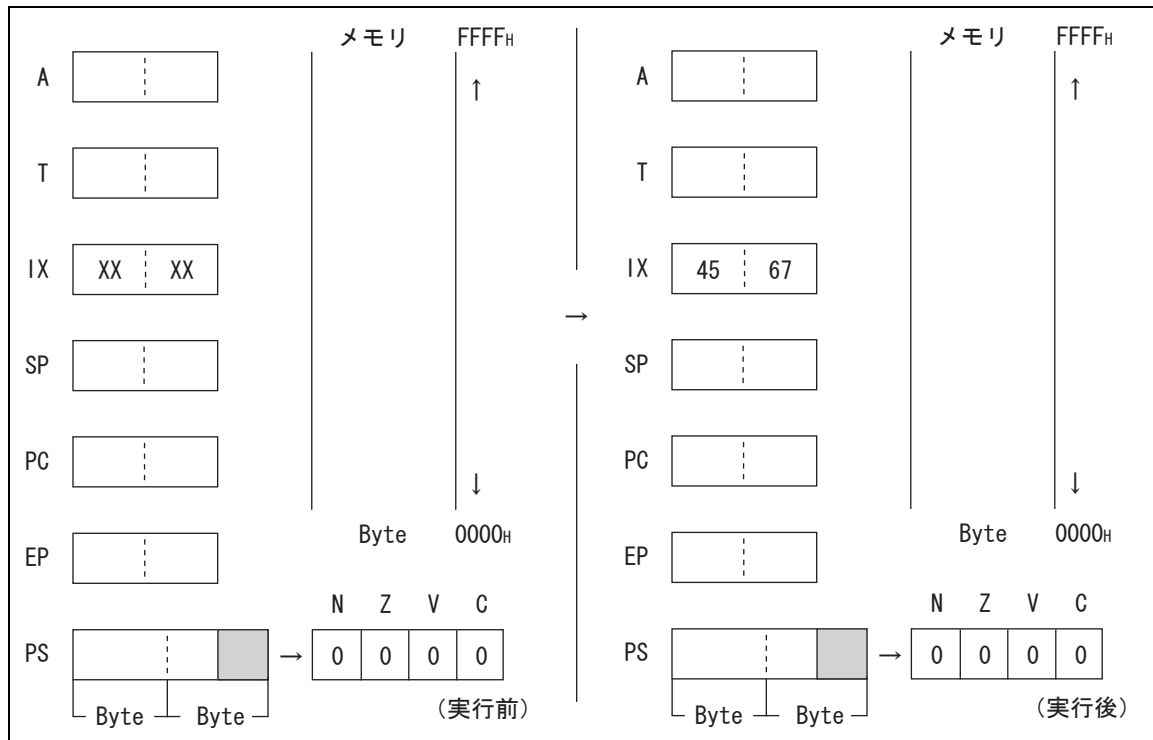
C: 変化しません

実行サイクル数: 3

バイト数: 3

オペコード: E6

実行例 : MOVW IX, #4567H



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

A のワードデータを PS へ転送します。

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

オペレーション

(PS) ← (A) (ワード転送)

アセンブラ形式

MOVW PS, A

コンディションコード (CCR)

N	Z	V	C
+	+	+	+

+: 命令実行により変化します

-: 変化しません

N: A の下位のビット 3 の値になります。

Z: A の下位のビット 2 の値になります。

V: A の下位のビット 1 の値になります。

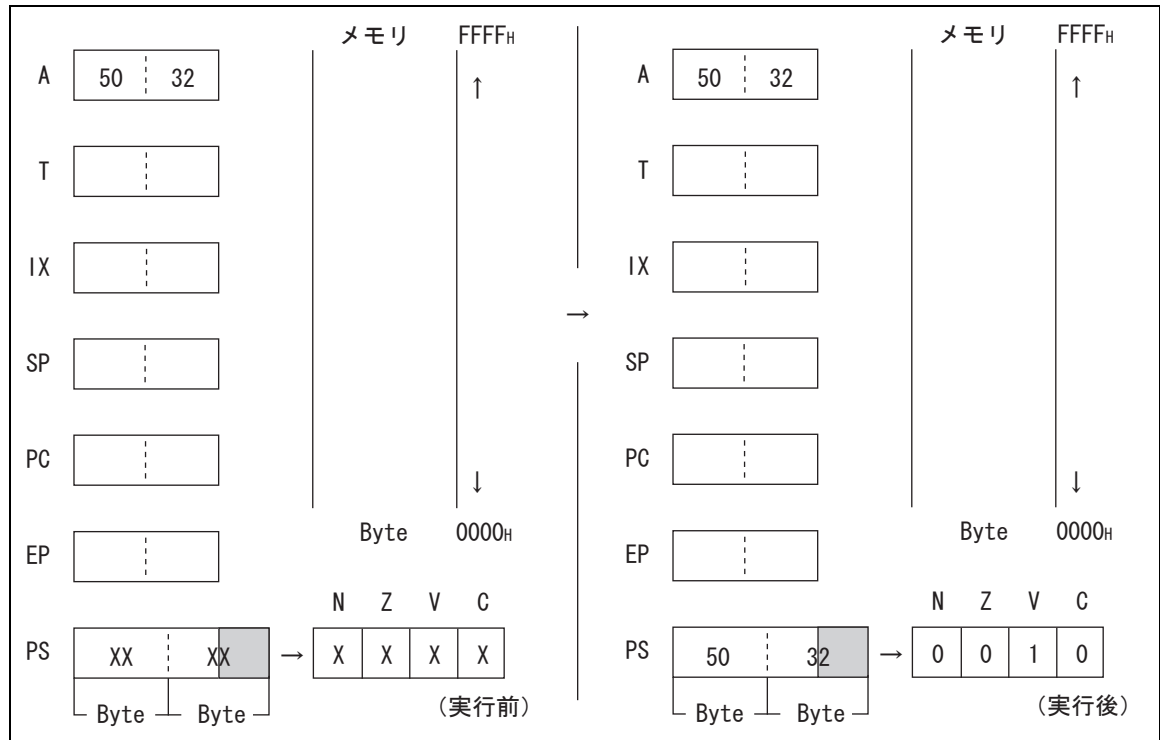
C: A の下位のビット 0 の値になります。

実行サイクル数: 1

バイト数: 1

オペコード: 71

実行例 : MOVW PS, A





## 6.58 MOVW (MOVE Immediate Word Data to Stack Pointer)

ワードの即値データを SP へ転送します。

### MOVW (MOVE Immediate Word Data to Stack Pointer)

オペレーション

(SP) ← d16 (ワード転送)

アセンブラ形式

MOVW SP, #d16

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

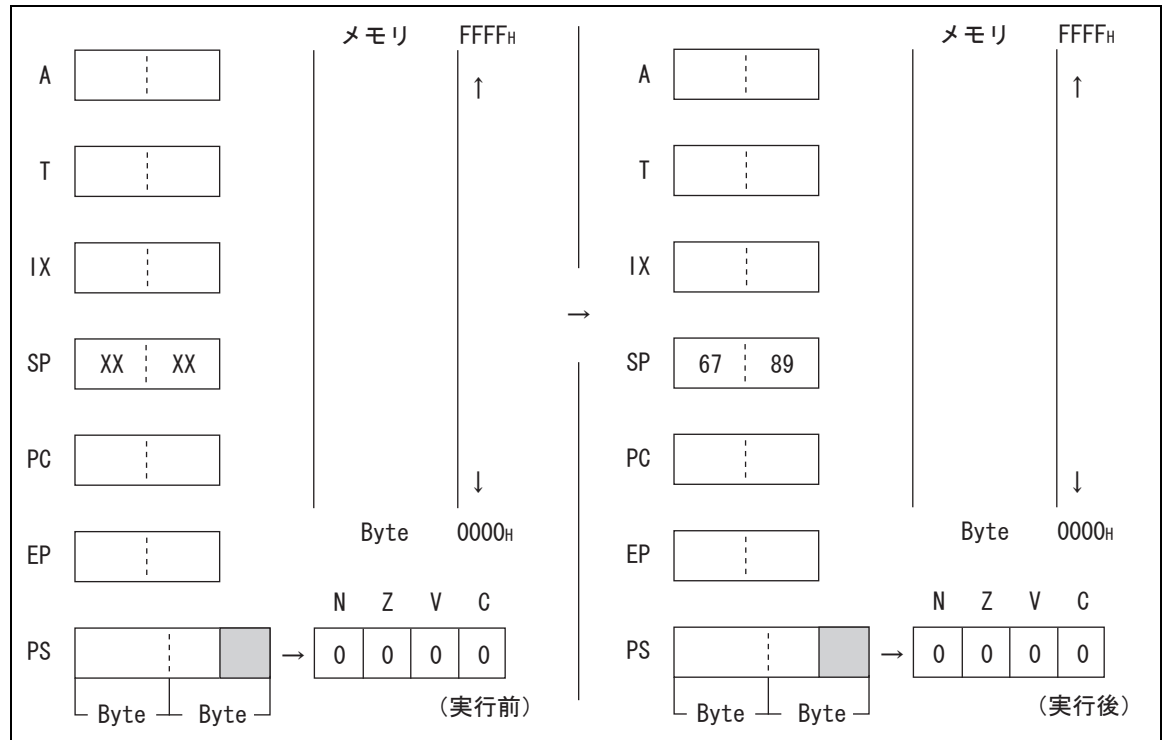
C: 変化しません

実行サイクル数: 3

バイト数: 3

オペコード: E5

実行例 : MOVW SP, #6789H



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

A のワードデータを SP へ転送します。

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

オペレーション

(SP) ← (A) (ワード転送)

アセンブラ形式

MOVW SP, A

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

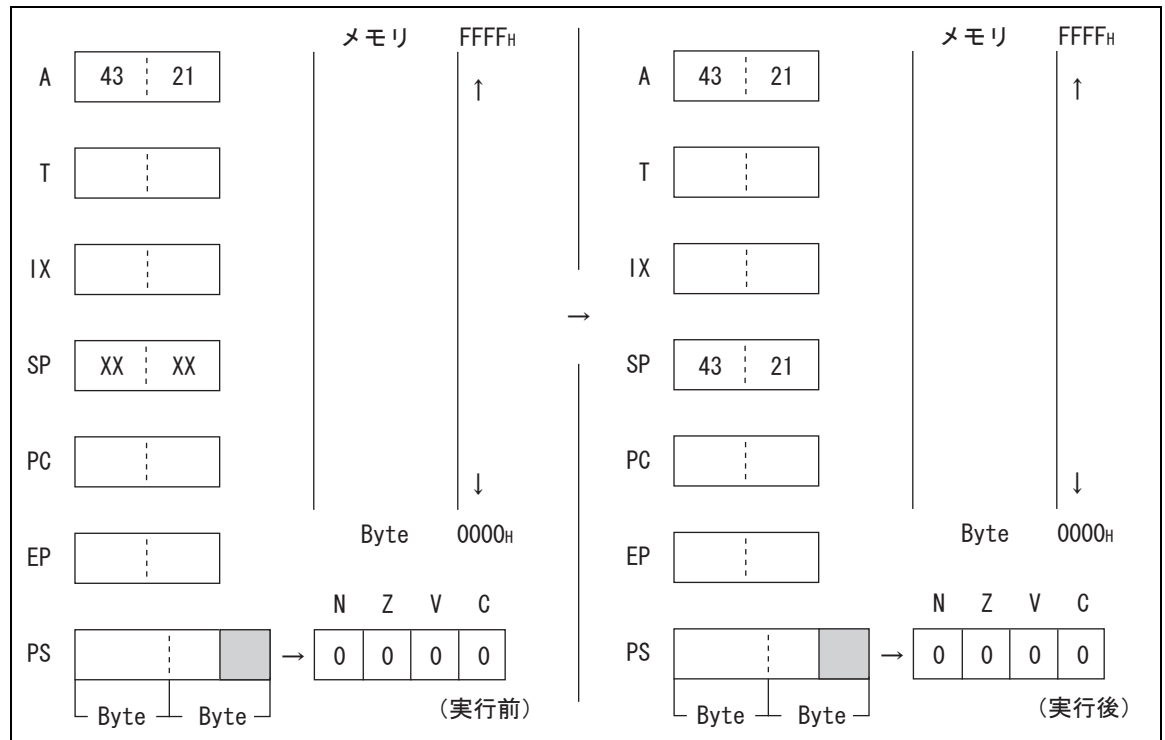
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: E1

実行例 : MOVW SP, A



## 6.60 MULU (MULTiPLY Unsigned)

AL と TL のバイトデータを符号なし 2 進数として乗算します。  
 結果を A のワードデータに戻します。

### MULU (MULTiPLY Unsigned)

オペレーション

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

アセンブラ形式

MULU A

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

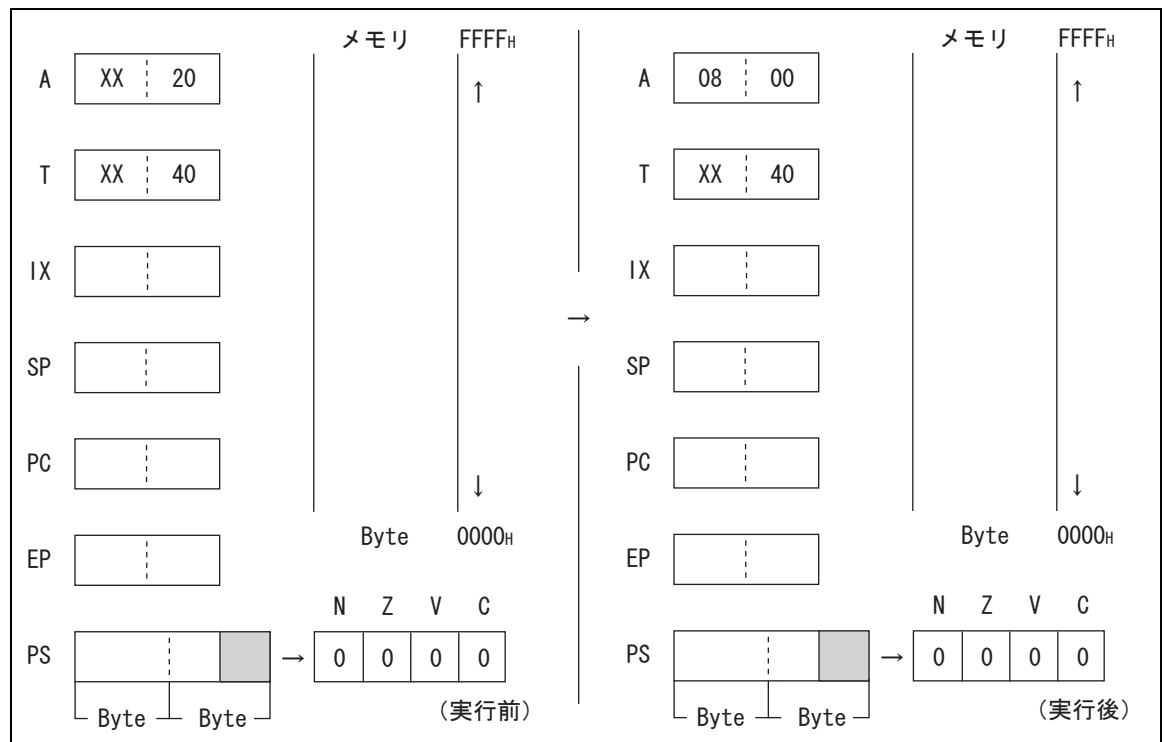
C: 変化しません

実行サイクル数: 8

バイト数: 1

オペコード: 01

実行例 : MULU A



## 6.61 NOP (NoOperation)

無操作を示します。

### NOP (NoOperation)

オペレーション

アセンブラ形式

NOP

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

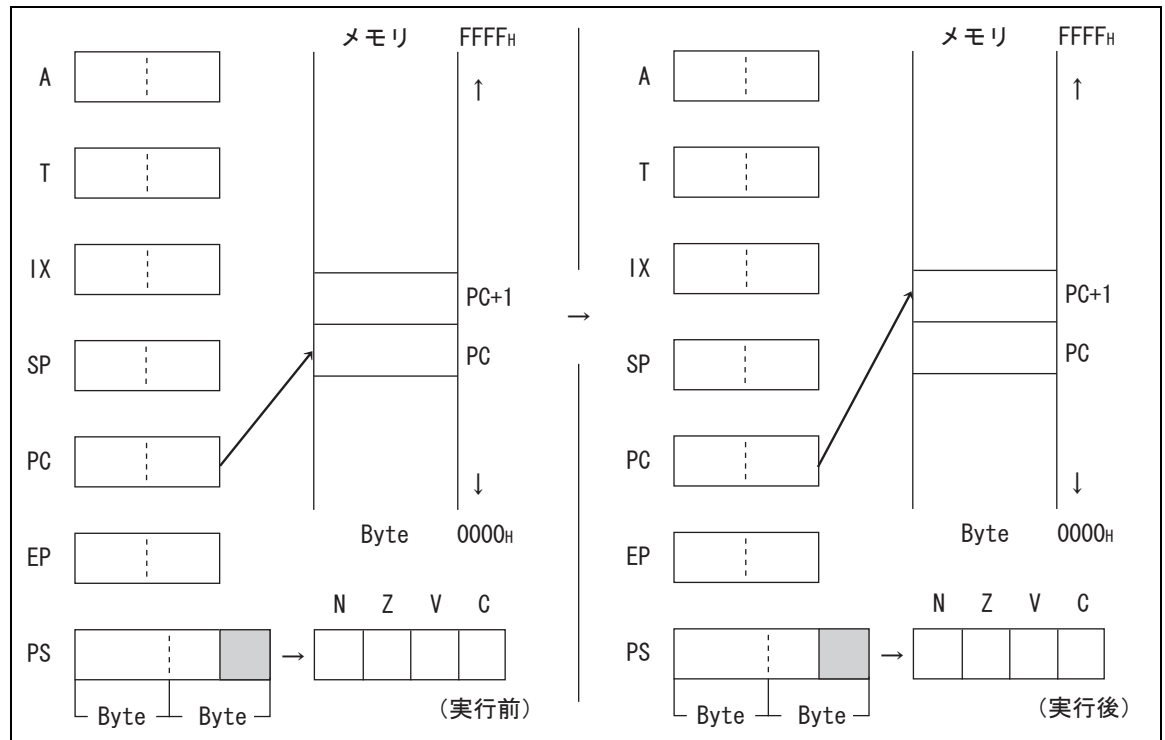
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: 00

実行例 : NOP





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

AL と TL とのバイトデータをビットごとに論理和をとり、結果を AL に戻します。  
 AH の内容は変化しません。

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

オペレーション

$(AL) \leftarrow (AL) \vee (TL)$  (バイト論理和)

アセンブラ形式

OR A

コンディションコード (CCR)

N	Z	V	C
+	+	R	-

+: 命令実行により変化します

-: 変化しません

R: 命令実行により 0 となります

N: 演算結果の AL が MSB=1 ならば 1 となり、それ以外は 0 となります

Z: 演算結果が 00<sub>H</sub> ならば 1 となり、それ以外は 0 となります

V: 常に 0 となります

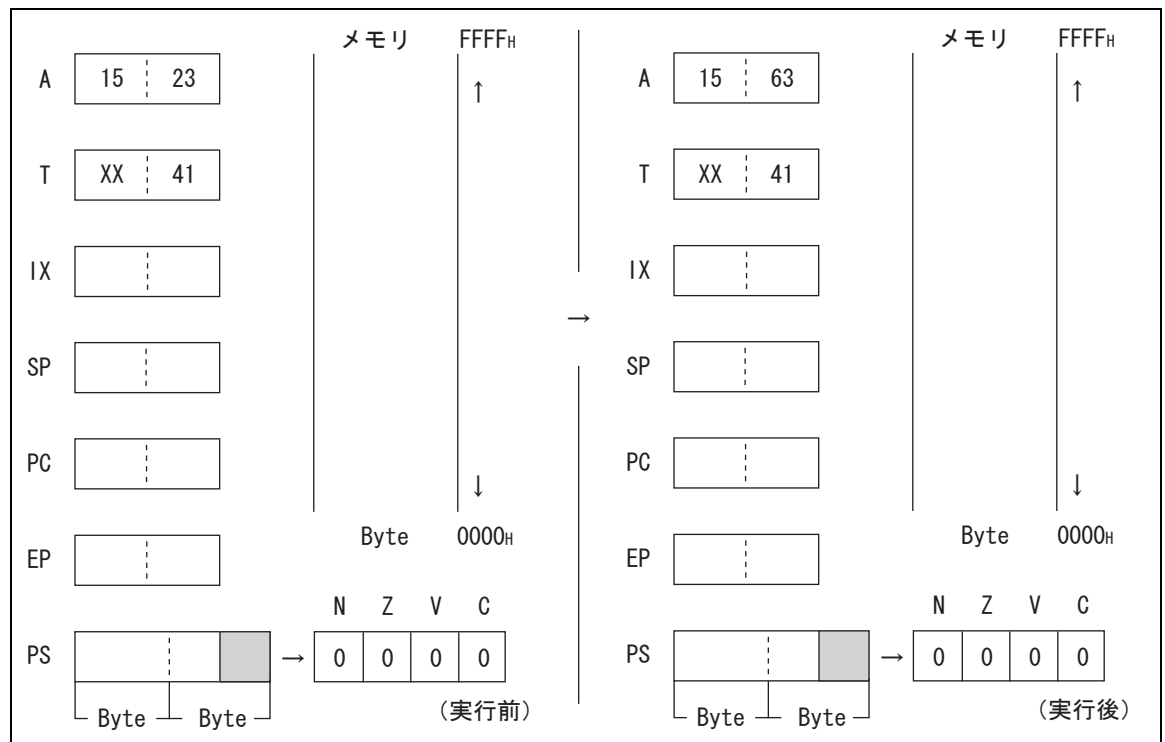
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: 72

実行例 : OR A



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

AL と EA メモリ ( 各種アドレスで表現されるメモリ ) とのビットごとに論理和をとり , 結果を AL に戻します。

AH の内容は変化しません。

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

オペレーション

$(AL) \leftarrow (AL) \vee (EA)$  ( バイト論理和 )

アセンブラ形式

OR A, EA

コンディションコード (CCR)

N	Z	V	C
+	+	R	-

+: 命令実行により変化します

-: 変化しません

R: 命令実行により 0 となります

N: 演算結果の AL が MSB=1 ならば 1 となり , それ以外は 0 となります

Z: 演算結果が 00<sub>H</sub> ならば 1 となり , それ以外は 0 となります

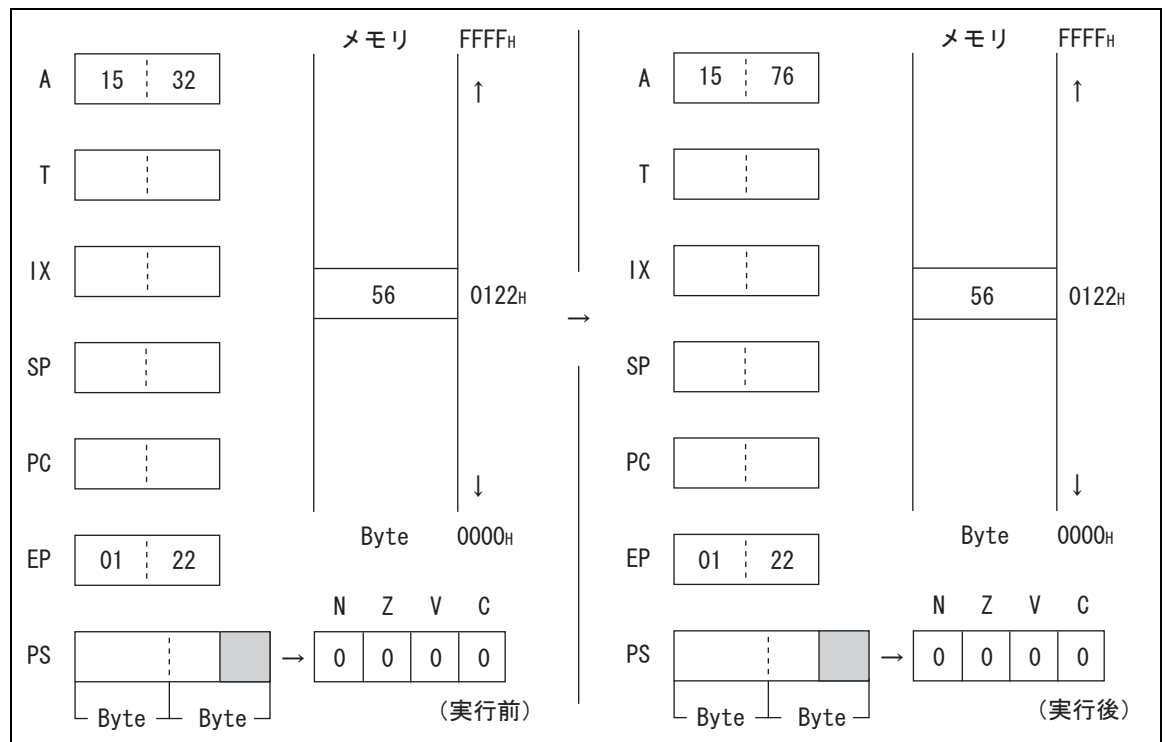
V: 常に 0 となります

C: 変化しません

表 6-15. 実行サイクル数 / バイト数 / オペコード

EA	#d8	dir	@IX+off	@EP	Ri
実行サイクル数	2	3	3	2	2
バイト数	2	2	2	1	1
オペコード	74	75	76	77	78 ~ 7F

実行例 : OR A, @EP



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

A と T とのワードデータをビットごとに論理和をとり、結果を A に戻します。

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

オペレーション

$(A) \leftarrow (A) \vee (T)$  (ワード論理和)

アセンブラ形式

ORW A

コンディションコード (CCR)

N	Z	V	C
+	+	R	-

+: 命令実行により変化します

-: 変化しません

R: 命令実行により 0 となります

N: 演算結果の A が MSB=1 ならば 1 となり、それ以外は 0 となります

Z: 演算結果が 0000<sub>H</sub> ならば 1 となり、それ以外は 0 となります

V: 常に 0 となります

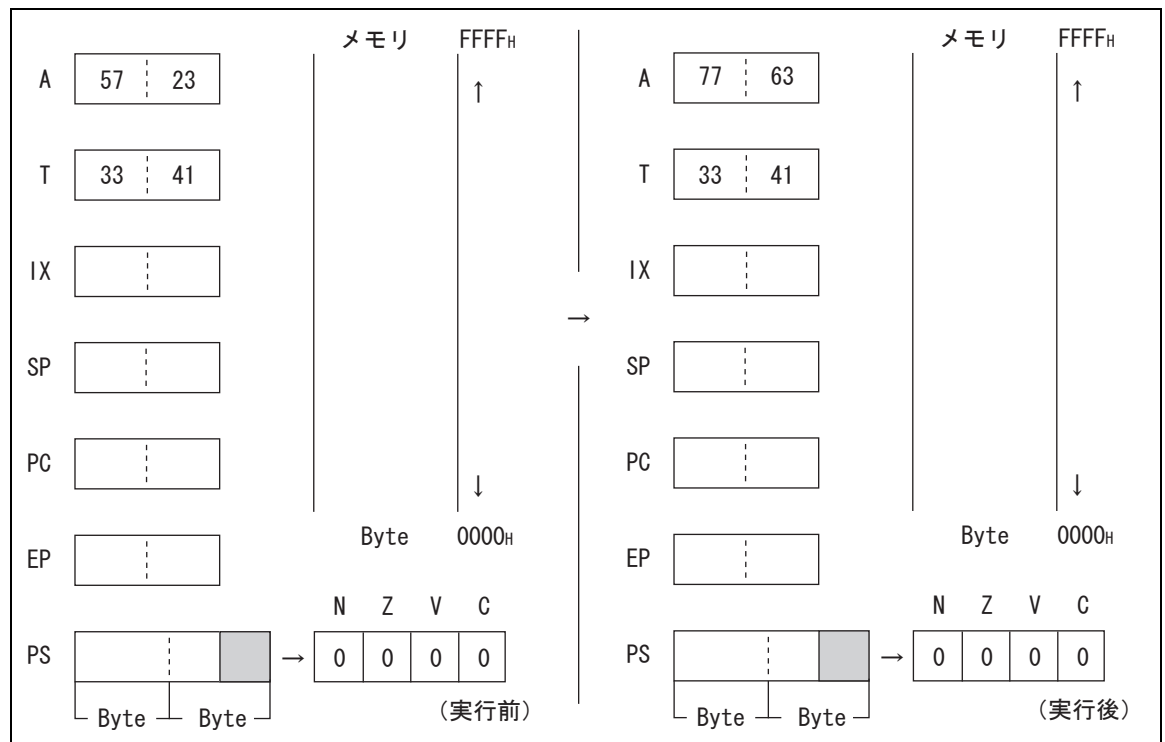
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: 73

実行例 : ORW A



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

SP の値から 2 ワード減算し、その後 SP で示されるメモリへ dr のワード値を転送します。

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

オペレーション

(SP) ← (SP) - 2 (ワード減算)

((SP)) ← (dr) (ワード転送)

アセンブラ形式

PUSHW dr

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

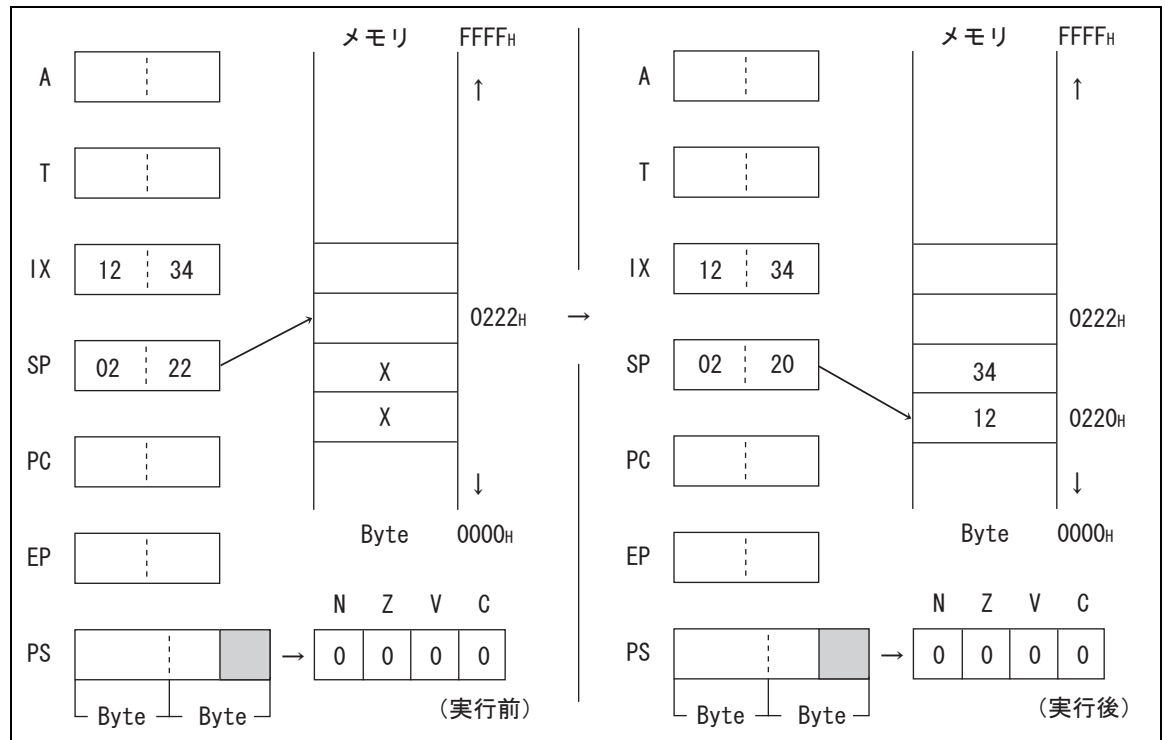
V: 変化しません

C: 変化しません

表 6-16. 実行サイクル数 / バイト数 / オペコード

DR	A	IX
実行サイクル数	4	4
バイト数	1	1
オペコード	40	41

実行例 : PUSHW IX





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

SP で示されるメモリからワード値を dr へ転送します。  
 その後 SP の値に 2 をワード加算します。

### POPW (POP Word Data of Inherent Register from Stack Memory)

オペレーション

$(dr) \leftarrow ((SP))$  (ワード転送)

$(SP) \leftarrow (SP) + 2$  (ワード加算)

アセンブラ形式

POPW dr

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

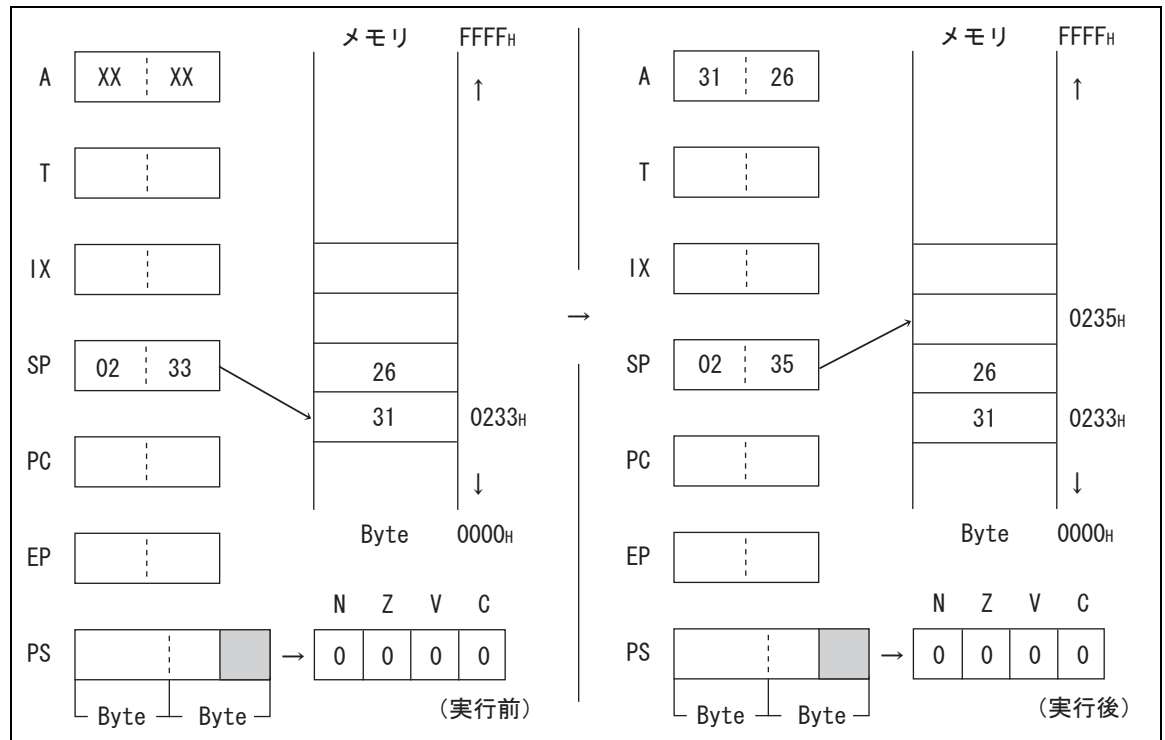
V: 変化しません

C: 変化しません

表 6-17. 実行サイクル数 / バイト数 / オペコード

DR	A	IX
実行サイクル数	3	3
バイト数	1	1
オペコード	50	51

実行例 : POPW A



## 6.67 RET (RETurn from subroutine)

スタックに退避してあった PC の内容を復帰します。  
 CALLV, CALL 命令と対で使うと, それぞれの命令の次の命令に戻ります。

### RET (RETurn from subroutine)

オペレーション

$(PC) \leftarrow ((SP))$  (ワード転送)

$(SP) \leftarrow (SP) + 2$  (ワード加算)

アセンブラ形式

RET

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

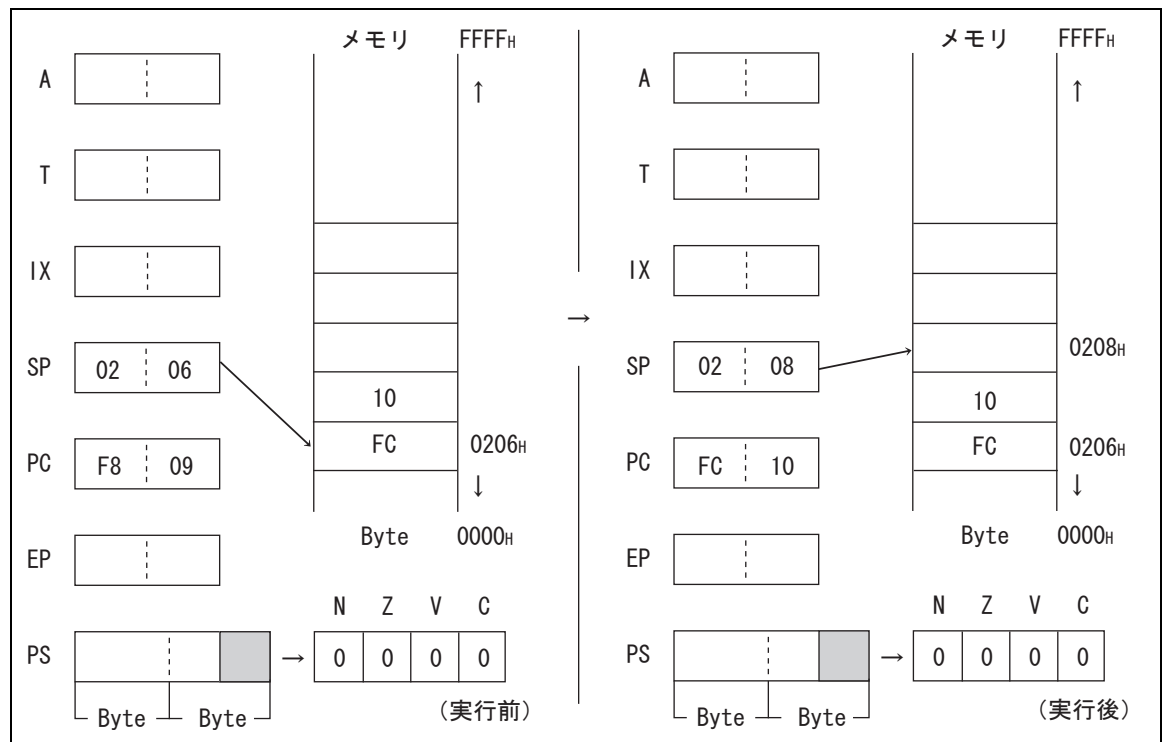
C: 変化しません

実行サイクル数: 6

バイト数: 1

オペコード: 20

実行例 : RET



## 6.68 RETI (RETurn from Interrupt)

スタックに退避してあった PS, PC の内容を復帰します。  
 インタラプト以前の状態に PS, PC を戻します。

### RETI (RETurn from Interrupt)

オペレーション

$(PS) \leftarrow ((SP)), (PC) \leftarrow ((SP + 2))$  (ワード転送)

$(SP) \leftarrow (SP) + 4$  (ワード加算)

アセンブラ形式

RETI

コンディションコード (CCR)

N	Z	V	C
+	+	+	+

+: 命令実行により変化します

-: 変化しません

N: 退避されていた N の値になります。

Z: 退避されていた Z の値になります。

V: 退避されていた V の値になります。

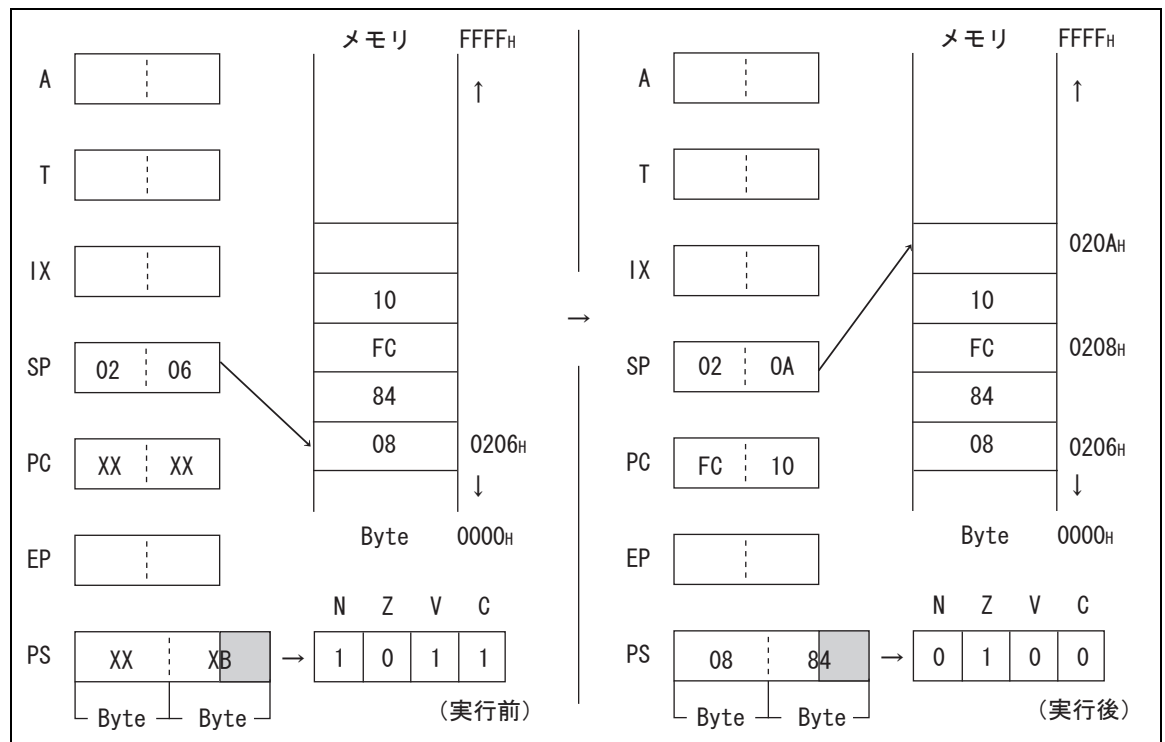
C: 退避されていた C の値になります。

実行サイクル数: 8

バイト数: 1

オペコード: 30

実行例 : RETI

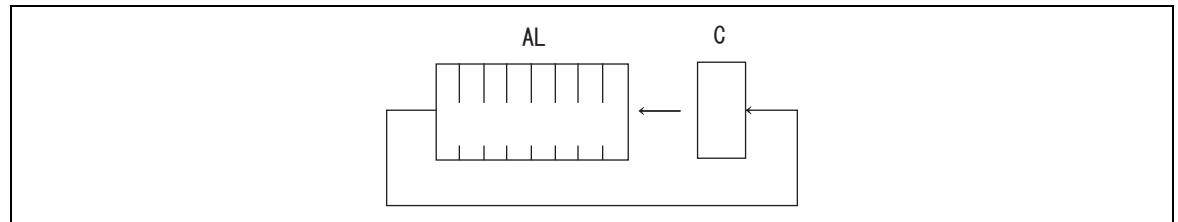


## 6.69 ROLC (Rotate Byte Data of Accumulator with Carry to Left)

キャリーを含めて AL のバイトデータを左へ 1 ビットシフトします。  
 AH の内容は変化しません。

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

オペレーション



アセンブラ形式

ROLC A

コンディションコード (CCR)

N	Z	V	C
+	+	-	+

+: 命令実行により変化します

-: 変化しません

N: シフトの結果, MSB=1 ならば 1 となり, それ以外は 0 となります

Z: シフト結果が 00<sub>H</sub> ならば 1 となり, それ以外は 0 となります

V: 変化しません

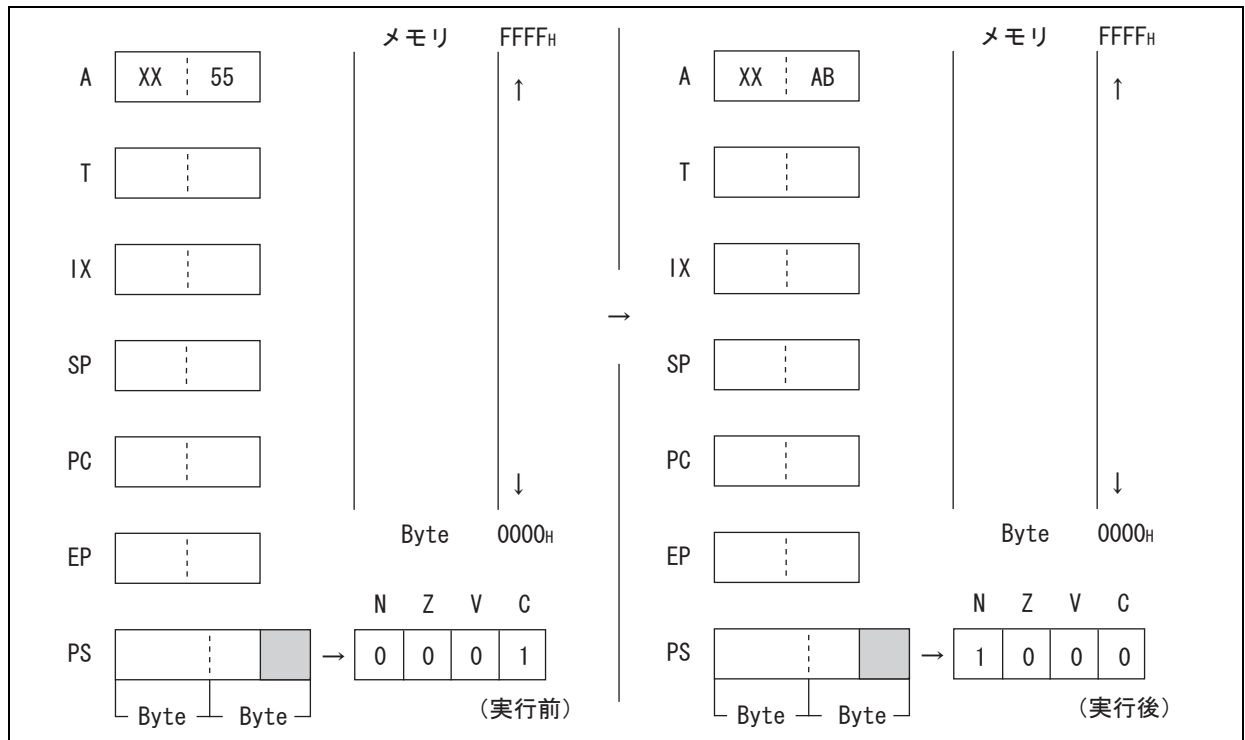
C: シフト前の A のビット 7 が入ります

実行サイクル数: 1

バイト数: 1

オペコード: 02

実行例 : ROLC A



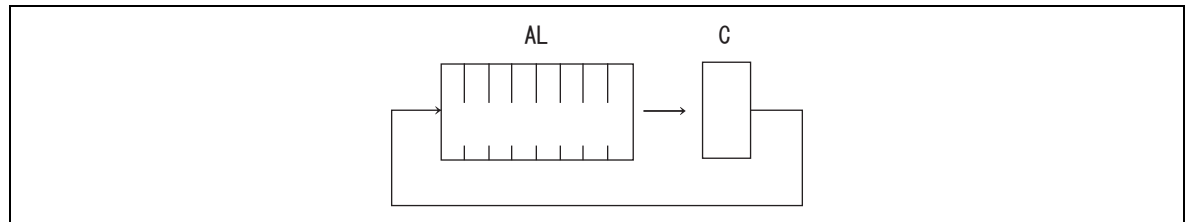


## 6.70 RORC (Rotate Byte Data of Accumulator with Carry to Right)

キャリーを含めて AL のバイトデータを右へ 1 ビットシフトします。  
 AH の内容は変化しません。

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

オペレーション



アセンブラ形式

RORC A

コンディションコード (CCR)

N	Z	V	C
+	+	-	+

+: 命令実行により変化します

-: 変化しません

N: シフトの結果, MSB=1 ならば 1 となり, それ以外は 0 となります

Z: シフト結果が 00<sub>H</sub> ならば 1 となり, それ以外は 0 となります

V: 変化しません

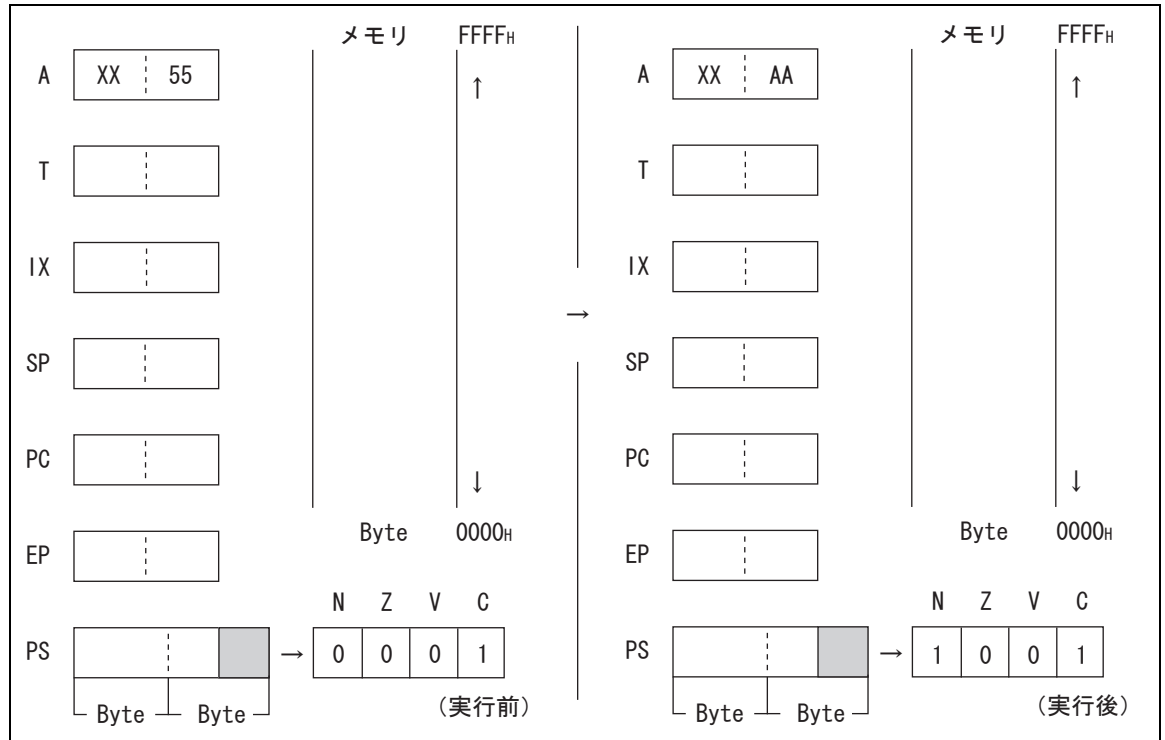
C: シフト前の A の LSB が入ります

実行サイクル数: 1

バイト数: 1

オペコード: 03

実行例 : RORC A



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

TL のバイトデータから AL のバイトデータを減算して、さらにキャリーを減算し、その結果を AL に戻します。

AH は変化しません。

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

オペレーション

$(AL) \leftarrow (TL) - (AL) - C$  (バイト減算, キャリー付き)

アセンブラ形式

SUBC A

コンディションコード (CCR)

N	Z	V	C
+	+	+	+

+: 命令実行により変化します

-: 変化しません

N: 演算結果の AL が MSB=1 ならば 1 となり、それ以外は 0 となります

Z: 演算結果が 00<sub>H</sub> ならば 1 となり、それ以外は 0 となります

V: 演算の結果オーバーフローが発生したときに 1 となり、それ以外は 0 となります

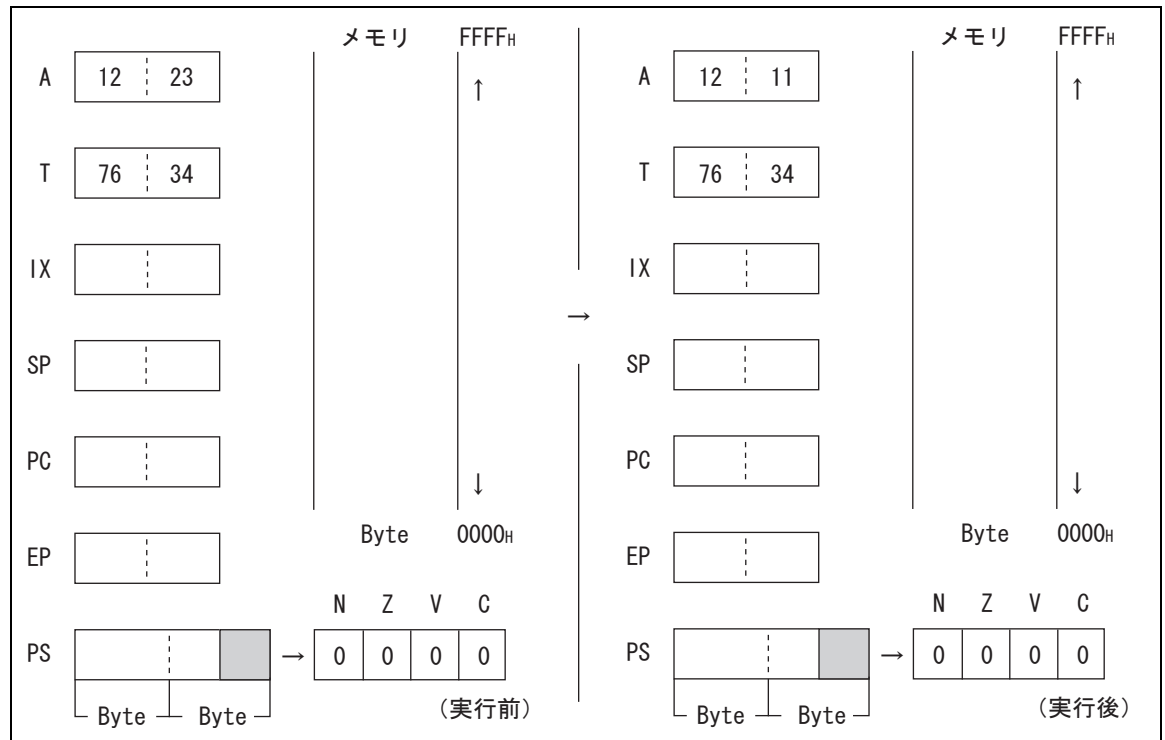
C: 演算の結果キャリーが発生したときに 1 となり、それ以外は 0 となります

実行サイクル数: 1

バイト数: 1

オペコード: 32

実行例 : SUBC A



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

AL のバイトデータから EA メモリ ( 各種アドレッシングで表現されるメモリ ) のバイトデータを減算して、さらにキャリーを減算し、その結果を AL に戻します。

AH は変化しません。

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

オペレーション

$(AL) \leftarrow (AL) - (EA) - C$  ( バイト減算 , キャリー付き )

アセンブラ形式

SUBC A, EA

コンディションコード (CCR)

N	Z	V	C
+	+	+	+

+: 命令実行により変化します

-: 変化しません

N: 演算結果の AL が MSB=1 ならば 1 となり、それ以外は 0 となります

Z: 演算結果が 00<sub>H</sub> ならば 1 となり、それ以外は 0 となります

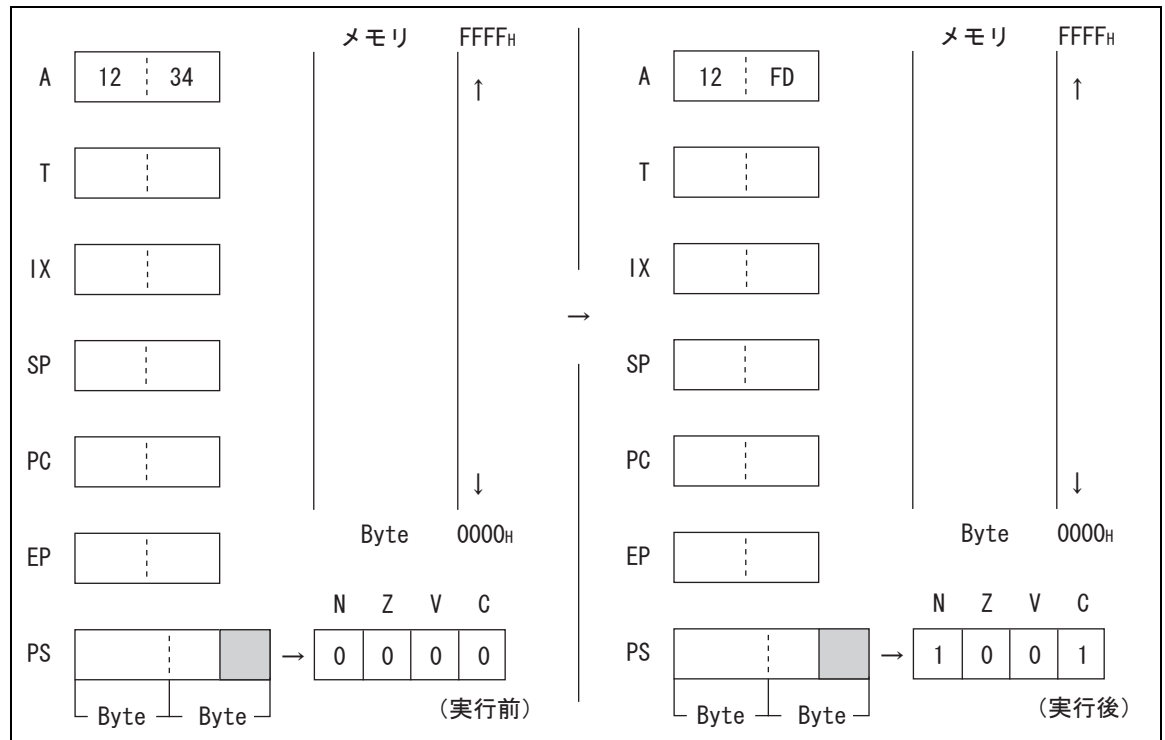
V: 演算の結果オーバーフローが発生したときに 1 となり、それ以外は 0 となります

C: 演算の結果キャリーが発生したときに 1 となり、それ以外は 0 となります

表 6-18. 実行サイクル数 / バイト数 / オペコード

EA	#d8	dir	@IX+off	@EP	Ri
実行サイクル数	2	3	3	2	2
バイト数	2	2	2	1	1
オペコード	34	35	36	37	38 ~ 3F

実行例 : SUBC A, #37H



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

T のワードデータから A のワードデータを減算して、さらにキャリーを減算し、その結果を A に戻します。

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

オペレーション

$(A) \leftarrow (T) - (A) - C$  (ワード減算, キャリー付き)

アセンブラ形式

SUBCW A

コンディショニングコード (CCR)

N	Z	V	C
+	+	+	+

+: 命令実行により変化します

-: 変化しません

N: 演算結果の A が MSB=1 ならば 1 となり、それ以外は 0 となります

Z: 演算結果が 0000<sub>H</sub> ならば 1 となり、それ以外は 0 となります

V: 演算の結果オーバーフローが発生したときに 1 となり、それ以外は 0 となります

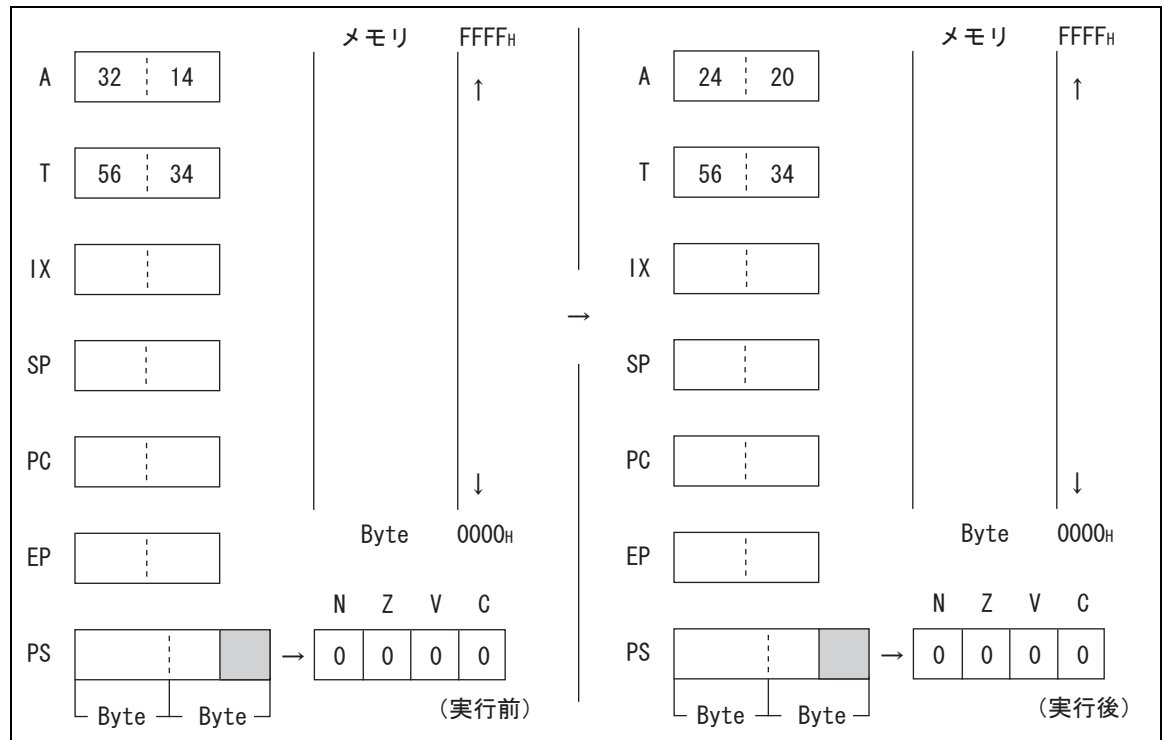
C: 演算の結果キャリーが発生したときに 1 となり、それ以外は 0 となります

実行サイクル数: 1

バイト数: 1

オペコード: 33

実行例 : SUBCW A





## 6.74 SETB (Set Direct Memory Bit)

ダイレクト領域の 1 ビット ( ニーモニックの下位 3 ビット (b) で示される ) の内容を 1 にします。

### SETB (Set Direct Memory Bit)

オペレーション

(dir:b) ← 1

アセンブラ形式

SETB dir:b

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

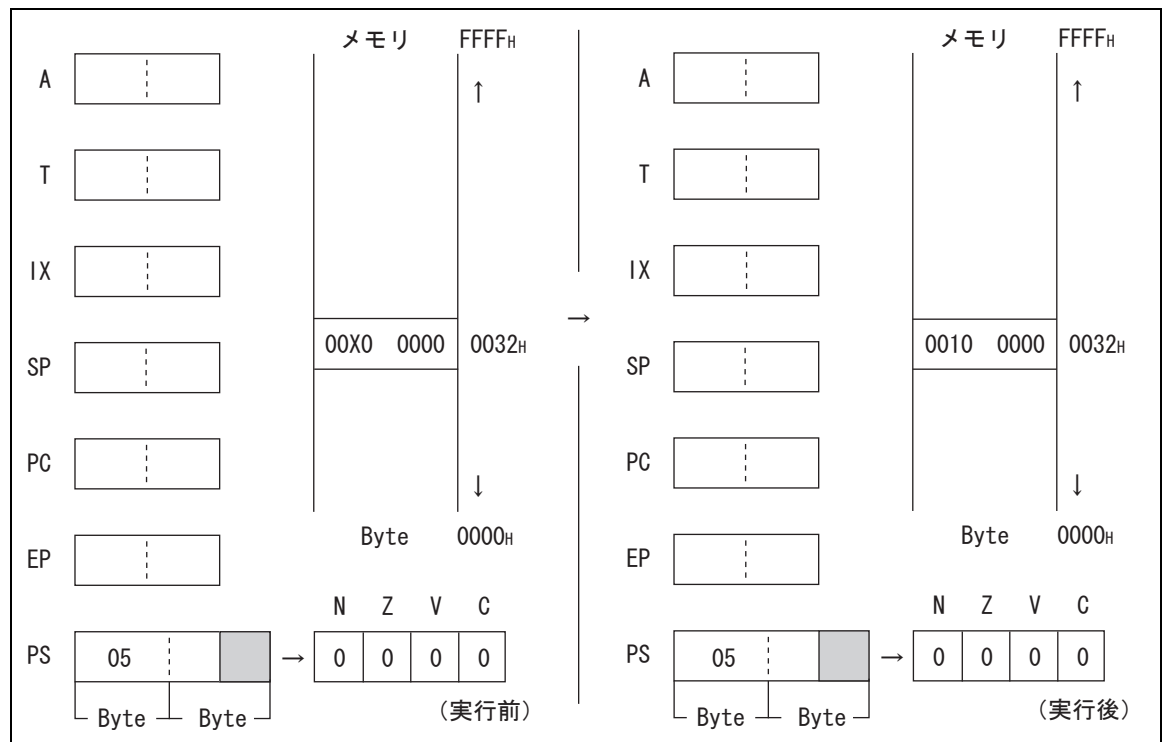
C: 変化しません

実行サイクル数: 4

バイト数: 2

オペコード: A8 ~ AF

実行例 : SETB 32H:5



## 6.75 SETC (SET Carry flag)

C フラグを 1 にします。

### SETC (SET Carry flag)

オペレーション

(C) ← 1

アセンブラ形式

SETC

コンディションコード (CCR)

N	Z	V	C
-	-	-	S

+: 命令実行により変化します

-: 変化しません

S: 命令実行により 1 になります

N: 変化しません

Z: 変化しません

V: 変化しません

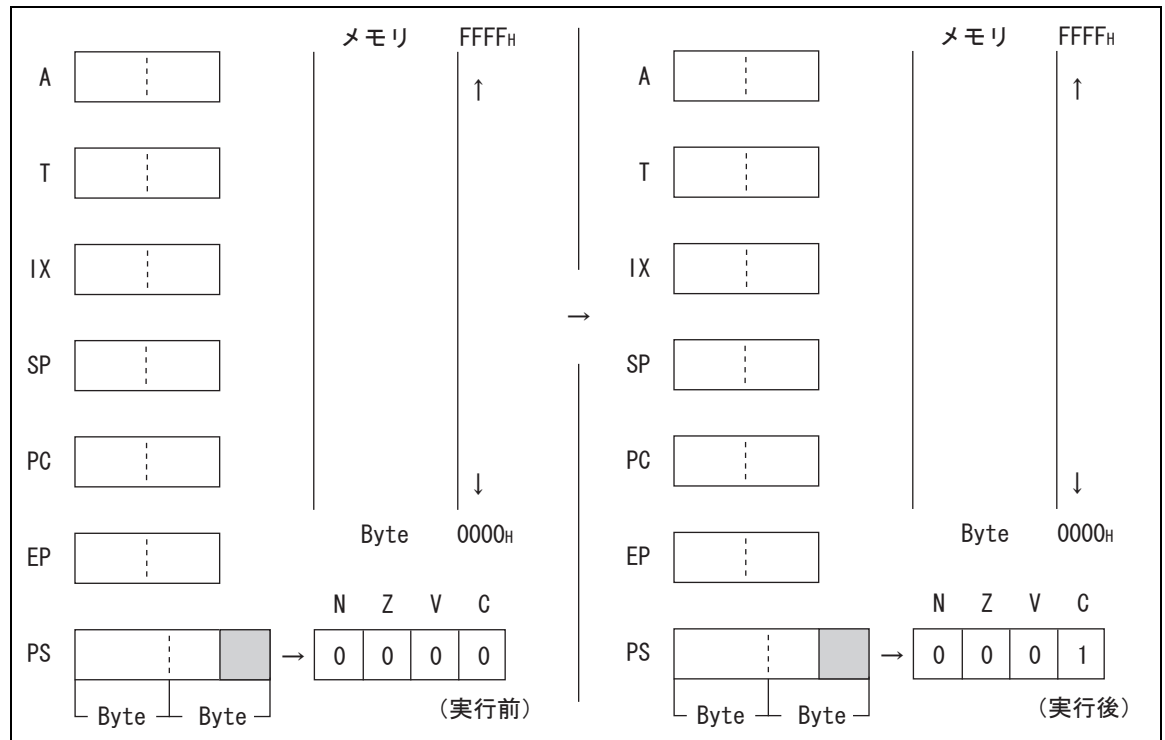
C: 1 になります

実行サイクル数: 1

バイト数: 1

オペコード: 91

実行例 : SETC



## 6.76 SETI (SET Interrupt flag)

I フラグを 1 にします ( 割込みを許可します )。

### SETI (SET Interrupt flag)

オペレーション

(I) ← 1

アセンブラ形式

SETI

コンディションコード (CCR)

I	N	Z	V	C
S	-	-	-	-

+: 命令実行により変化します

-: 変化しません

S: 命令実行により 1 になります

I: 1 になります

N: 変化しません

Z: 変化しません

V: 変化しません

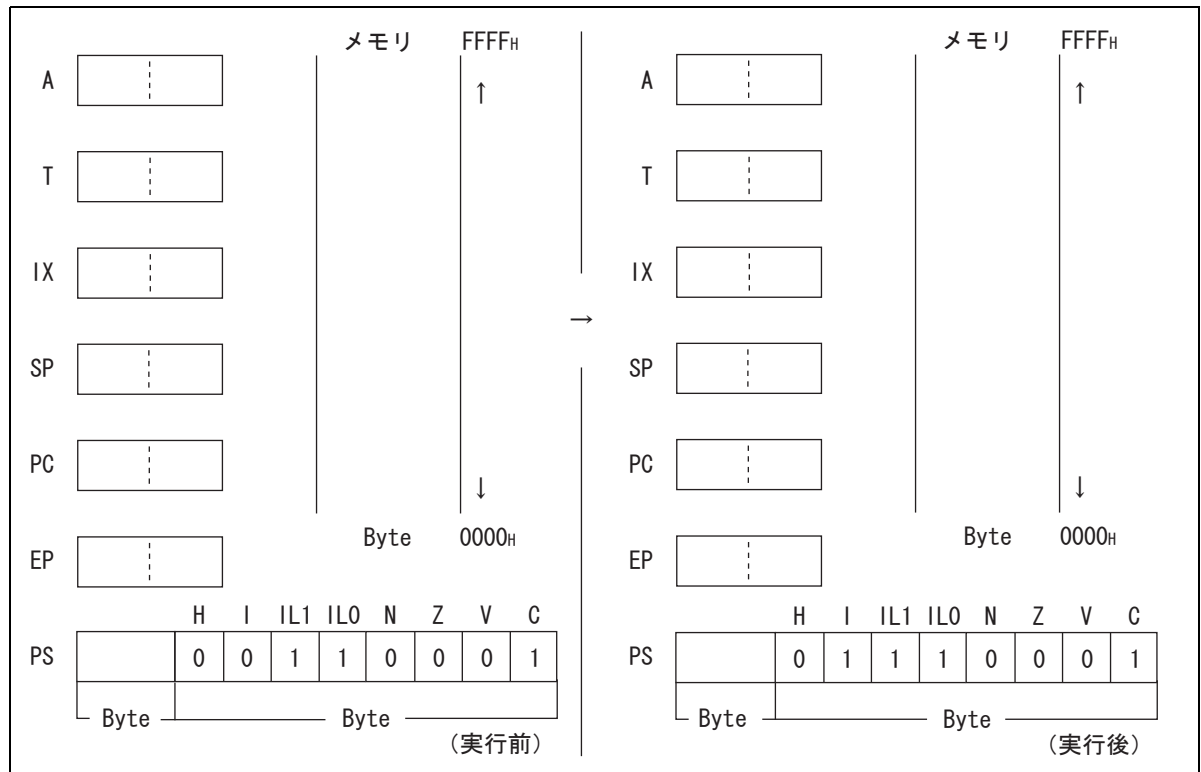
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: 90

実行例 : SETI



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

AH のバイトデータと AL のバイトデータを交換します。

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

オペレーション

(AH) ↔ (AL) ( バイトデータの交換 )

アセンブラ形式

SWAP

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

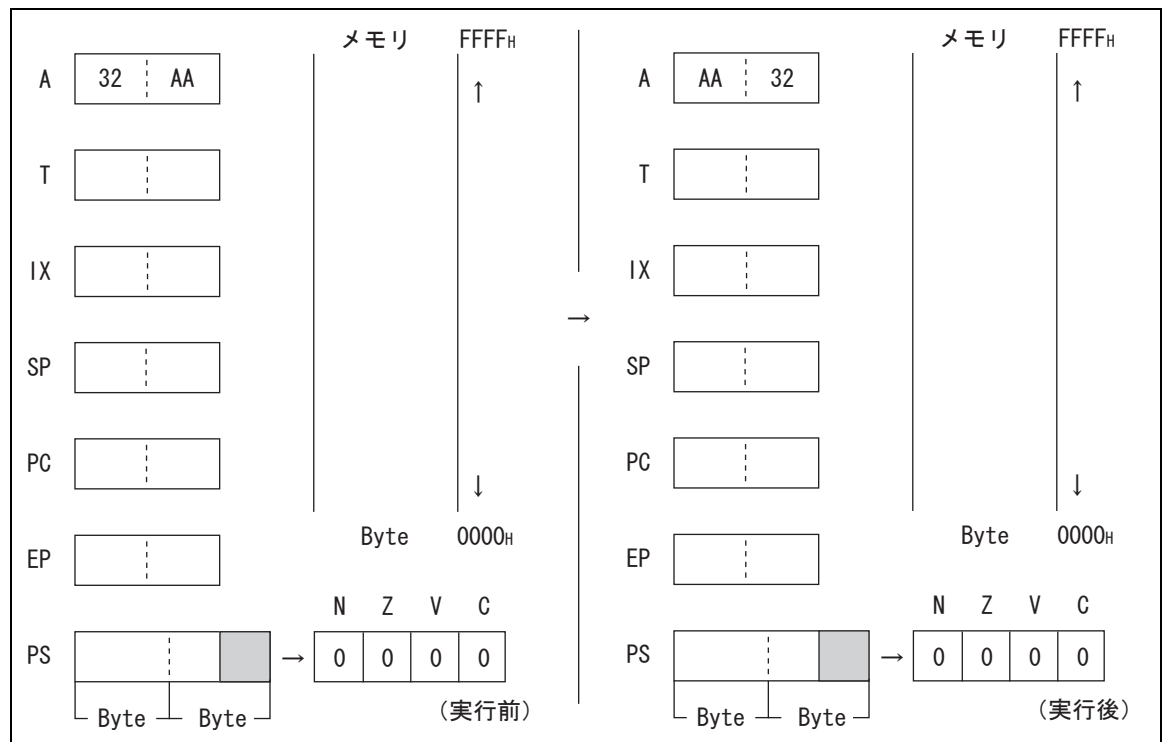
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: 10

実行例 : SWAP





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

AL のバイトデータと TL のバイトデータを交換します。

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

オペレーション

(AL) ↔ (TL) ( バイトデータの交換 )

アセンブラ形式

XCH A, T

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

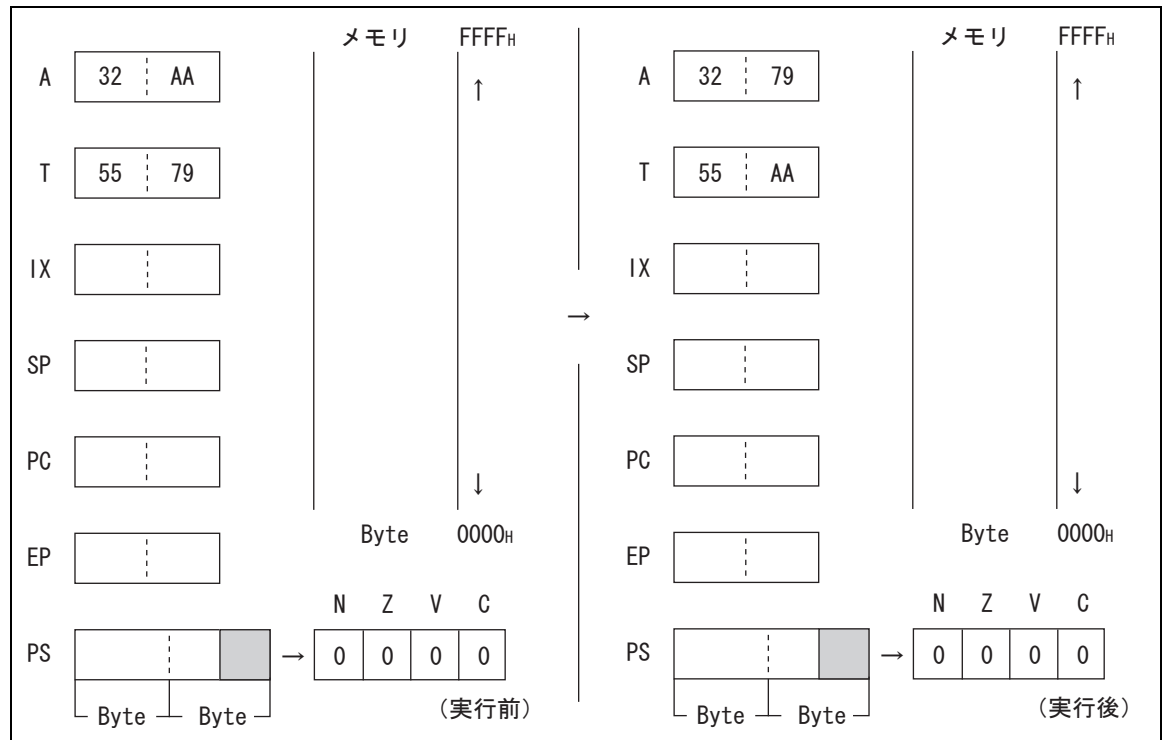
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: 42

実行例 : XCH A, T



## 6.79 XCHW (eXCHange Word Data Accumulator and Extrapointer)

A のワードデータと EP のワードデータを交換します。

### XCHW (eXCHange Word Data Accumulator and Extrapointer)

オペレーション

(A) ↔ (EP) (ワードデータの交換)

アセンブラ形式

XCHW A, EP

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

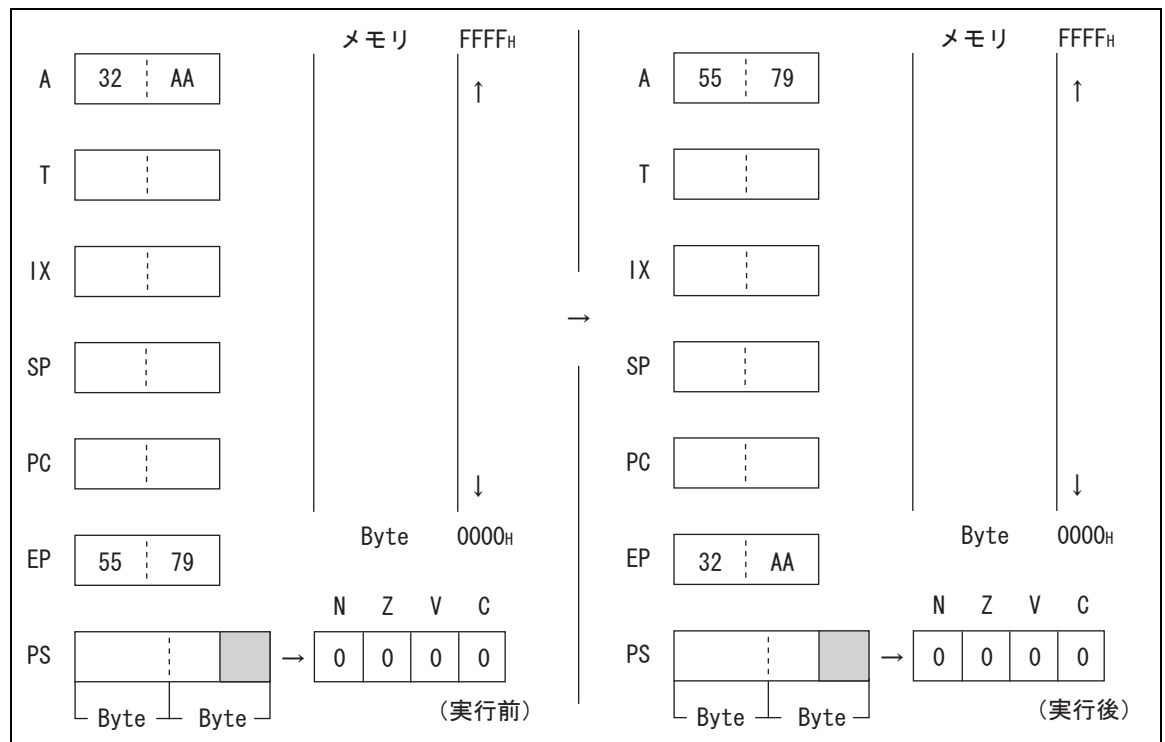
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: F7

実行例 : XCHW A, EP



## 6.80 XCHW (eXCHange Word Data Accumulator and Index Register)

A のワードデータと IX のワードデータを交換します。

### XCHW (eXCHange Word Data Accumulator and Index Register)

オペレーション

(A) ↔ (IX) (ワードデータの交換)

アセンブラ形式

XCHW A, IX

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

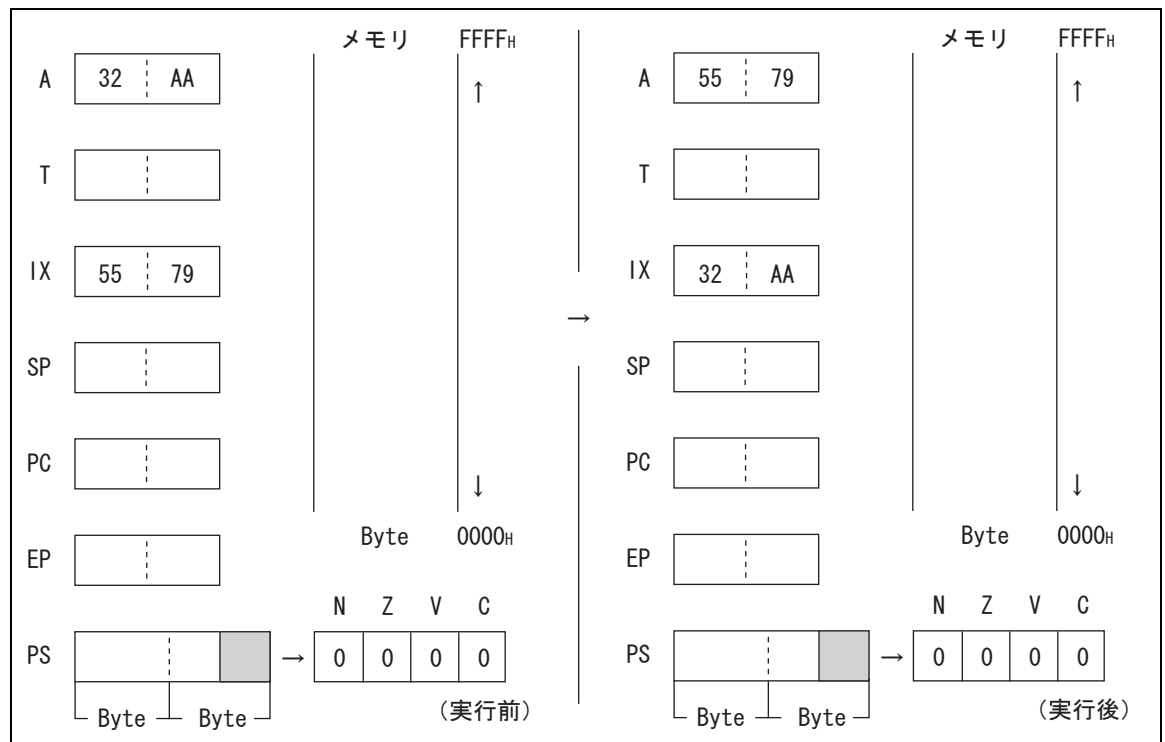
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: F6

実行例 : XCHW A, IX



## 6.81 XCHW (eXCHange Word Data Accumulator and Program Counter)

A のワードデータと PC のワードデータを交換します。

### XCHW (eXCHange Word Data Accumulator and Program Counter)

オペレーション

(PC) ← (A) (ワード転送)

(A) ← (PC) + 1 (ワード加算, ワード転送)

アセンブラ形式

XCHW A, PC

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

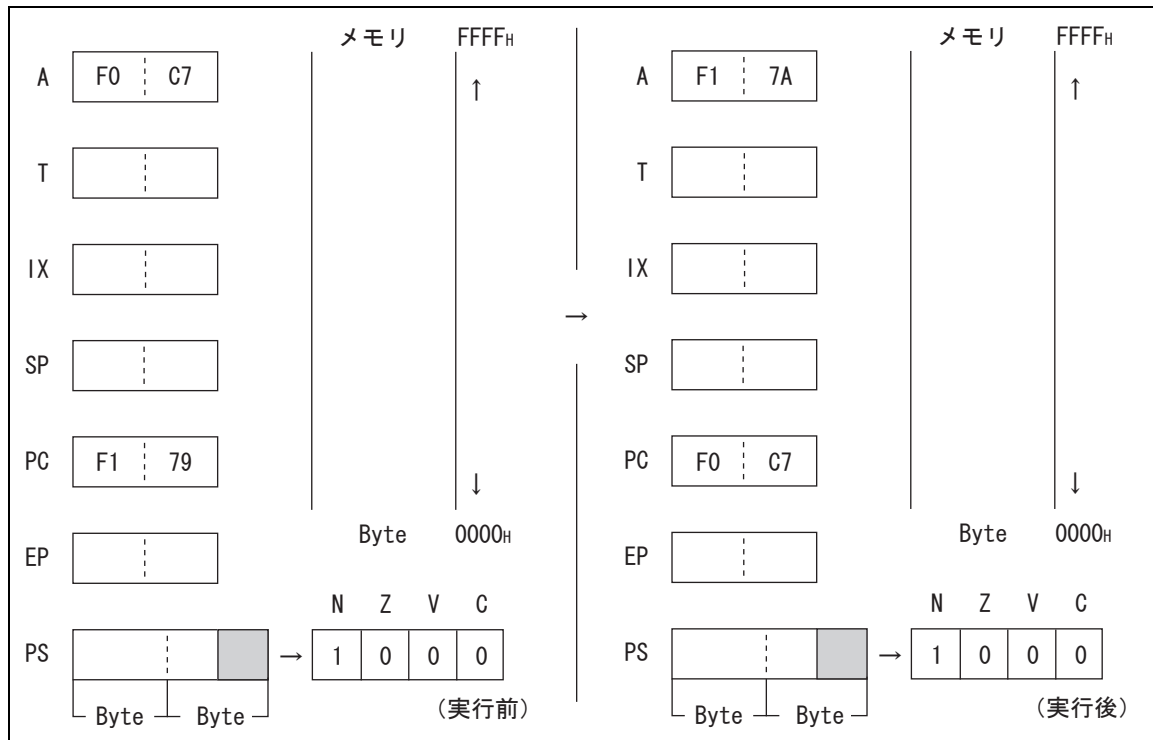
C: 変化しません

実行サイクル数: 3

バイト数: 1

オペコード: F4

実行例 : XCHW A, PC





## 6.82 XCHW (eXCHange Word Data Accumulator and Stack Pointer)

A のワードデータと SP のワードデータを交換します。

### XCHW (eXCHange Word Data Accumulator and Stack Pointer)

オペレーション

(A) ↔ (SP) (ワードデータの交換)

アセンブラ形式

XCHW A, SP

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

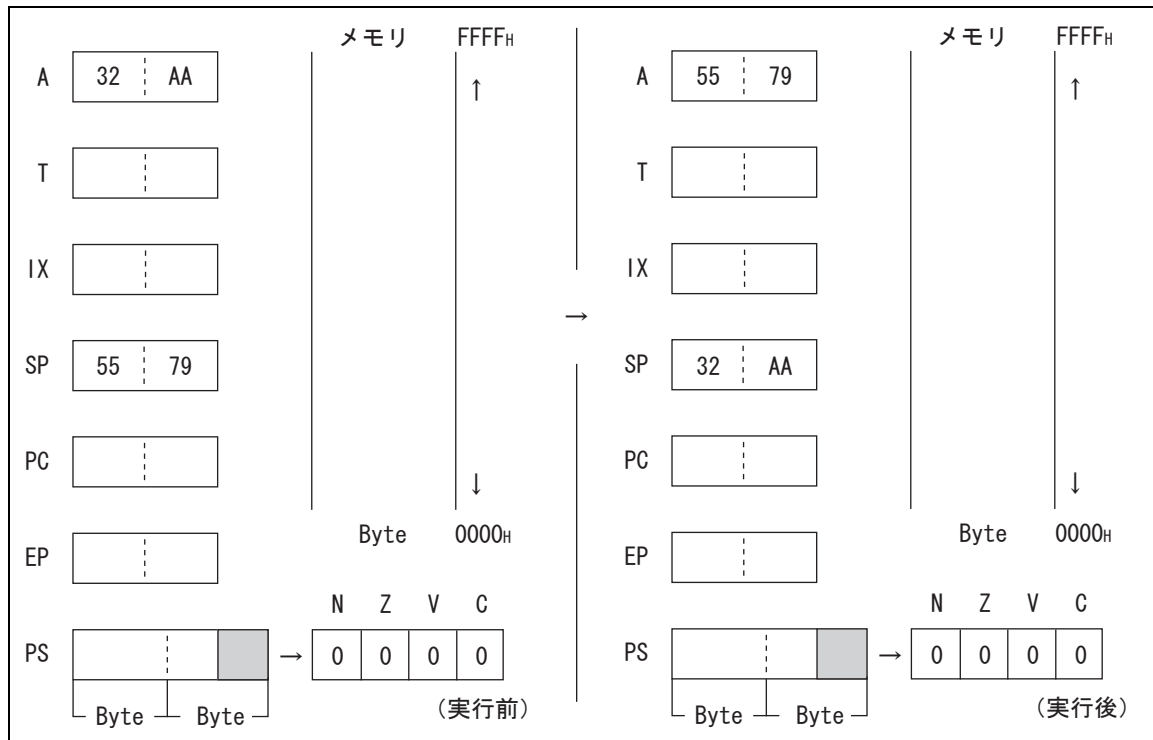
C: 変化しません

実行サイクル数: 2

バイト数: 1

オペコード: F5

実行例 : XCHW A, SP



## 6.83 XCHW (eXCHange Word Data Accumulator and Temporary Accumulator)

A のワードデータと T のワードデータを交換します。

### XCHW (eXCHange Word Data Accumulator and Temporary Accumulator)

オペレーション

(A) ↔ (T) (ワードデータの交換)

アセンブラ形式

XCHW A, T

コンディションコード (CCR)

N	Z	V	C
-	-	-	-

+: 命令実行により変化します

-: 変化しません

N: 変化しません

Z: 変化しません

V: 変化しません

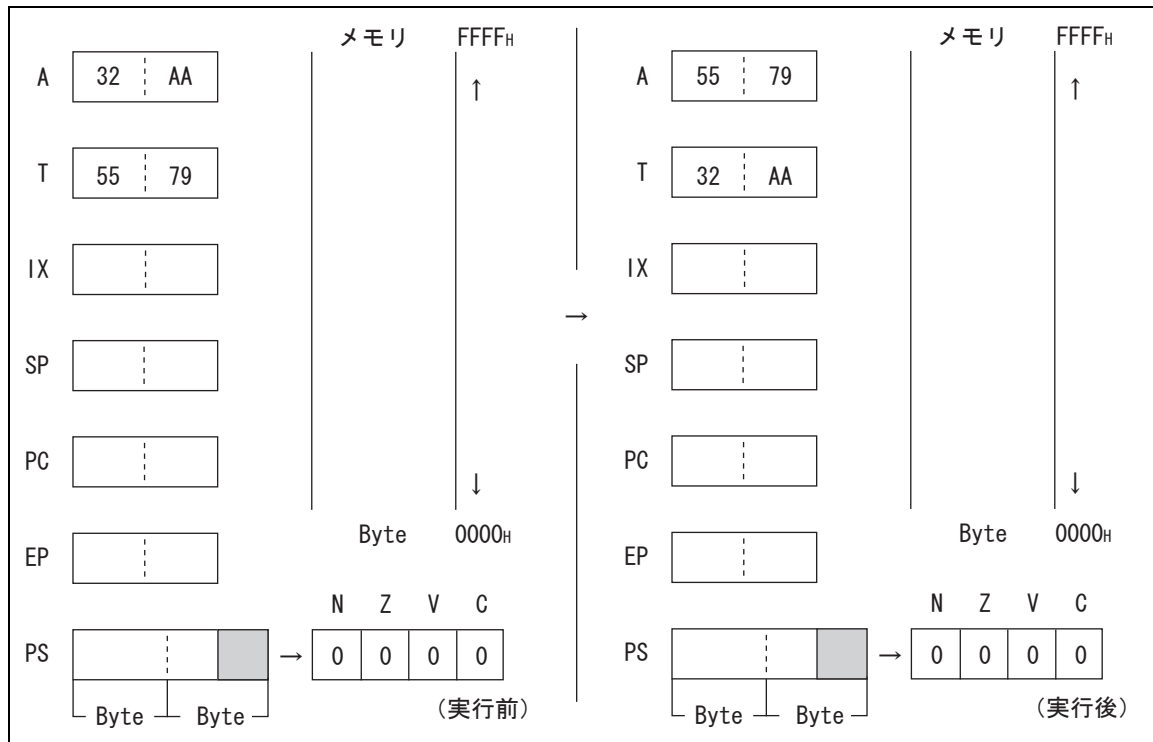
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: 43

実行例 : XCHW A, T



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

AL のバイトデータと TL とのバイトデータをビットごとに排他的論理和をとり、結果を AL に戻します。

AH の内容は変化しません。

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

オペレーション

(AL) ← (AL) ∨ (TL) (バイト排他的論理和)

アセンブラ形式

XOR A

コンディションコード (CCR)

N	Z	V	C
+	+	R	-

+: 命令実行により変化します

-: 変化しません

R: 命令実行により 0 となります

N: 演算結果の AL が MSB=1 ならば 1 となり、それ以外は 0 となります

Z: 演算結果が 00<sub>H</sub> ならば 1 となり、それ以外は 0 となります

V: 常に 0 となります

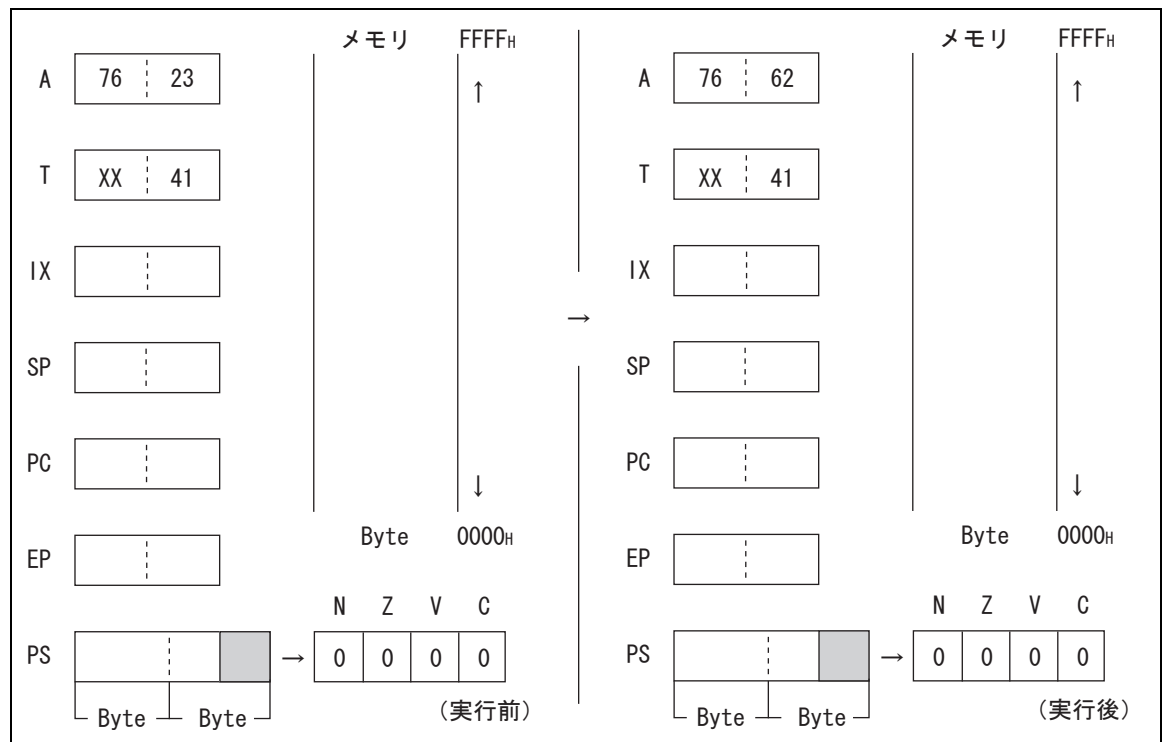
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: 52

実行例 : XOR A



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

AL のバイトデータと EA メモリ ( 各種アドレッシングで表現されるメモリ ) のバイトデータをビットごとに排他的論理和をとり、結果を AL に戻します。

AH の内容は変化しません。

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

オペレーション

(AL) ← (AL) ∨ (EA) ( バイト排他的論理和 )

アセンブラ形式

XOR A, EA

コンディションコード (CCR)

N	Z	V	C
+	+	R	-

+: 命令実行により変化します

-: 変化しません

R: 命令実行により 0 となります

N: 演算結果の AL が MSB=1 ならば 1 となり、それ以外は 0 となります

Z: 演算結果が 00<sub>H</sub> ならば 1 となり、それ以外は 0 となります

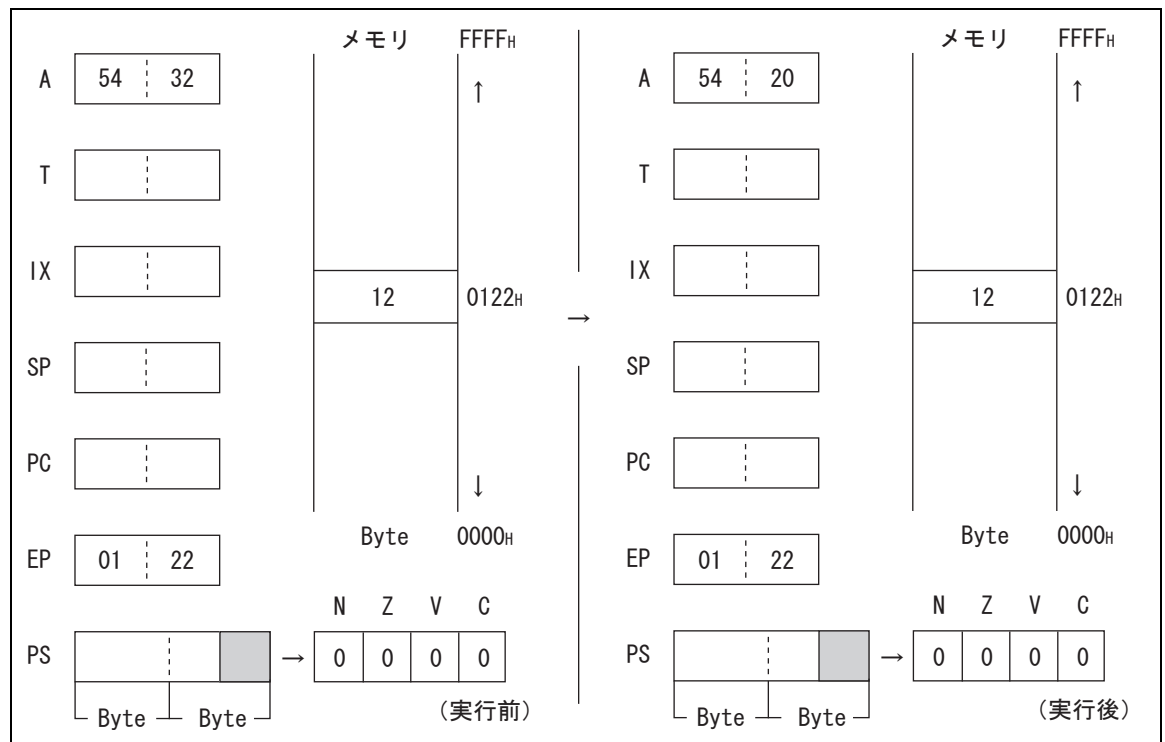
V: 常に 0 となります

C: 変化しません

表 6-19. 実行サイクル数 / バイト数 / オペコード

EA	#d8	dir	@IX+off	@EP	Ri
実行サイクル数	2	3	3	2	2
バイト数	2	2	2	1	1
オペコード	54	55	56	57	58 ~ 5F

実行例 : XOR A, @EP





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

A のワードデータと T のワードデータをビットごとに排他的論理和をとり、結果を A に戻します。

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

オペレーション

$(A) \leftarrow (A) \vee (T)$  (ワード排他的論理和)

アセンブラ形式

XORW A

コンディションコード (CCR)

N	Z	V	C
+	+	R	-

+: 命令実行により変化します

-: 変化しません

R: 命令実行により 0 となります

N: 演算結果の A が MSB=1 ならば 1 となり、それ以外は 0 となります

Z: 演算結果が 0000<sub>H</sub> ならば 1 となり、それ以外は 0 となります

V: 常に 0 となります

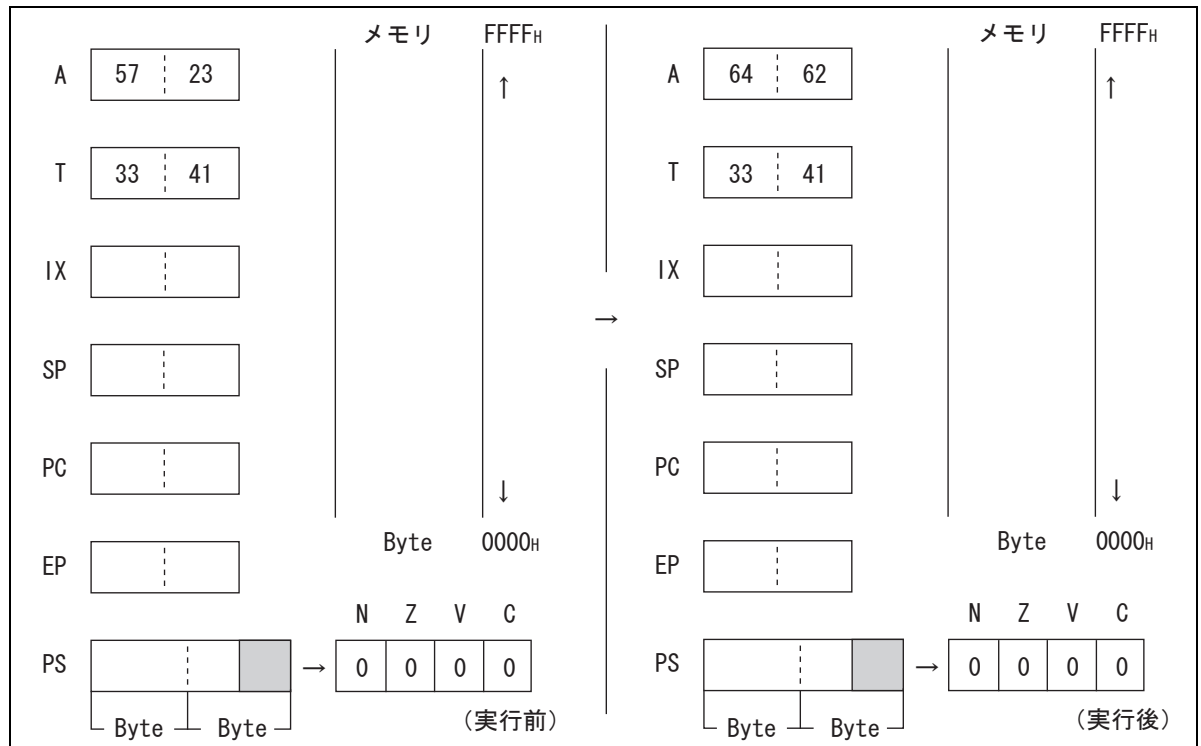
C: 変化しません

実行サイクル数: 1

バイト数: 1

オペコード: 53

実行例 : XORW A



# 主な変更内容



Spansion Publication Number: CM26-00301-2

ページ	変更内容
15	第3章 レジスタ ( "3.7 ダイレクトバンク " の追加 )
31	4.2 割込み許可 / 禁止 / 優先順位機構 ■ 割込み許可 / 禁止 / 優先順位機構 ● 周辺の割込み許可フラグによる要求許可チェック ( " 割込みの発生元での " → " 割込み発生元からの " )
35	4.4 多重割込み ■ 多重割込み ( " A/D " → " A/D コンバータ " )
37	4.5 リセット動作 ( " 実行します " → " リセット処理を実行します " )
178	6.65 PUSHW (PUSH Word Data of Inherent Register to Stack Memory) ( " PS で示されるメモリへ dr のワード値を転送します。その後 SP の値から 2 をワード減算します。 " ) → " SP の値から 2 ワード減算し、その後 SP で示されるメモリへ dr のワード値を転送します。 " )
178	6.65 PUSHW (PUSH Word Data of Inherent Register to Stack Memory) ( " ((SP)) ← (dr) (ワード転送) " → " (SP) ← (SP) - 2 (ワード減算) " ) ( " (SP) ← (SP) - 2 (ワード減算) " → " ((SP)) ← (dr) (ワード転送) " )
234	付録 A.2 動作一覧表 表 A.2-4 動作一覧表 ( その他 ) ( " (SP) ← (SP)-2, ((SP)) ← (A) " 追加 ) ( " (A) ← ((SP)), (SP) ← (SP)+2 " 追加 ) ( " (SP) ← (SP)-2, ((SP)) ← (IX) " 追加 ) ( " (IX) ← ((SP)), (SP) ← (SP)+2 " 追加 )

ページ	変更内容
234	付録 A.2 動作一覧表 表 A.2-4 動作一覧表 ( その他 ) ( " No operation " 追加 ) ( " (C) ← 0 " 追加 ) ( " (C) ← 1 " 追加 ) ( " (I) ← 0 " 追加 ) ( " (I) ← 1 " 追加 )

注意事項 : 以降の変更点に関しては、「改訂履歴」を参照してください。

# A. 命令一覧表



付録では、命令一覧表、バス動作一覧表および命令マップについて記載しています。

- A. 命令一覧表
- B. バス動作一覧表
- C. 命令マップ

## 付録 A 命令一覧表

ここでは、アセンブラで使用される命令の一覧を記載します。

A.1 F<sup>2</sup>MC-8FX CPU の命令概要

A.2 動作一覧表

A.3 フラグ変化表

## A.1 F<sup>2</sup>MC-8FX CPU の命令概要

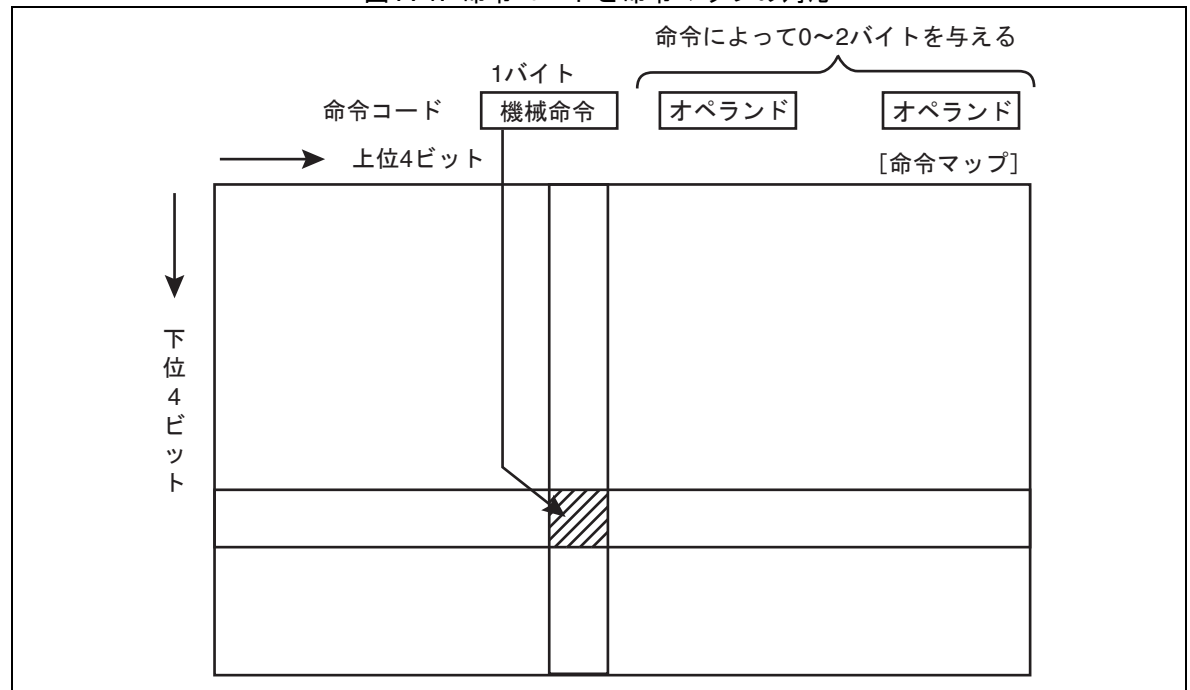
F<sup>2</sup>MC-8FX CPU の命令について説明します。

### F<sup>2</sup>MC-8FX CPU の命令概要

F<sup>2</sup>MC-8FX CPU には、140 種類の 1 バイト機械命令 (マップとしては 256 バイト) があり、命令とそれに続くオペランドによって命令コードを構成します。

図 A-1. に命令コードと命令マップの対応を示します。

図 A-1. 命令コードと命令マップの対応



F<sup>2</sup>MC-8FX CPU の命令の特徴として、以下のものが挙げられます。

命令は転送系、演算系、分岐系、その他の 4 つに分類されます。

アドレス指定は各種の方法があり、命令の選択とオペランド指定により 10 種類のアドレッシングを選択できます。

ビット操作命令を備えており、リードモディファイライト動作が可能です。

特殊な動作を指示する命令があります。

## 命令一覧表の記号

表 A-1. に命令コードの記述で使用している記号の説明を示します。

表 A-1. 命令一覧表の記号の説明 (Sheet 1 of 2 )

表記	意味
dir	ダイレクトアドレス (8 ビット長)
off	オフセット (8 ビット長)
ext	エクステンドアドレス (16 ビット長)
#vct	ベクタテーブル番号 (3 ビット長)
#d8	イミディエートデータ (8 ビット長)
#d16	イミディエートデータ (16 ビット長)
dir:b	ビットダイレクトアドレス (8 ビット長 : 3 ビット長)
rel	分岐相対アドレス (8 ビット長)
@	レジスタ間接 (例 : @A, @IX, @EP)
A	アキュムレータ (使用する命令によって 8 ビット長か 16 ビット長かが決まる)
AH	アキュムレータの上位 8 ビット (8 ビット長)
AL	アキュムレータの下位 8 ビット (8 ビット長)
T	テンポラリアキュムレータ (使用する命令によって 8 ビット長か 16 ビット長かが決まる)
TH	テンポラリアキュムレータの上位 8 ビット (8 ビット長)
TL	テンポラリアキュムレータの下位 8 ビット (8 ビット長)
IX	インデックスレジスタ (16 ビット長)
EP	エクストラポインタ (16 ビット長)
PC	プログラムカウンタ (16 ビット長)
SP	スタックポインタ (16 ビット長)
PS	プログラムステータス (16 ビット長)
dr	アキュムレータまたはインデックスレジスタのいずれか (16 ビット長)
CCR	コンディションコードレジスタ (8 ビット長)
RP	レジスタバンクポインタ (5 ビット長)
DP	ダイレクトバンクポインタ (3 ビット長)
Ri	汎用レジスタ (8 ビット長, i = 0 ~ 7)
×	× が即値データそのものであることを示す (使用する命令によって 8 ビット長か 16 ビット長かが決まる)



表 A-1. 命令一覧表の記号の説明 (Sheet 2 of 2)

表記	意味
(×)	× の中身がアクセスの対象であることを示す (使用する命令によって 8 ビット長か 16 ビット長かが決まる)
((×))	× の中身が示すアドレスがアクセスの対象であることを示す (使用する命令によって 8 ビット長か 16 ビット長かが決まる)

### 命令一覧表の項目

表 A-2. に命令一覧表の項目について説明を示します。

表 A-2. 命令一覧表の項目

項目	説明
MNEMONIC	命令のアセンブル記述を表します。
RD	内部バスのリード状態を表します。
WR	内部バスのライト状態を表します。
RMW	内部バスのリードモディファイライト信号を表します。
~	命令のサイクル数を表します。1 命令サイクルは 1 マシンサイクルです。 ( 注意事項 ) 命令サイクル数は、直前の命令によって 1 サイクル延期される場合があります。 また、IO 領域へのアクセスでは、命令のサイクル数が延長される場合があります。
#	命令のバイト数を表します。
動作	命令の動作を表します。
TL, TH, AH	TL, TH および AH の、各命令実行時の内容の変化を示します。欄内の記号は以下のものを、それぞれ表します。 <ul style="list-style-type: none"> <li>• -: 変化しないこと</li> <li>• dH: 動作に記述したデータの上位 8 ビット</li> <li>• AL, AH: 命令直前の AL または AH の内容になること</li> <li>• 00: 00 になること</li> </ul>
N, Z, V, C	各命令実行時に変化するフラグを表します。欄内の記号は以下のものを、それぞれ表します。 <ul style="list-style-type: none"> <li>• -: 変化しないこと</li> <li>• +: 変化すること</li> <li>• R: 0 になること</li> <li>• S: 1 になること</li> </ul>
OPCODE	命令のコードを表します。該当命令が複数コードを占める場合は、次のような記載規約に則っています。 48 ~ 4F: 48, 49, ..., 4F を示します。

## A.2 動作一覧表

表 A-3. に転送系の, 表 A-4. に演算系の, 表 A-5. に分岐点の, 表 A-6. にその他の動作一覧表を示します。

### 動作一覧表

表 A-3. 動作一覧表 ( 転送系 ) (Sheet 1 of 3 )

No	MNEMONIC	~	#	動作	TL	TH	AH	NZVC	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 ~ 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 ~ 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 ~ 8F
17	MOVW dir, A	4	2	(dir) ← (AH), (dir+1) ← (AL)	—	—	—	-----	D5
18	MOVW @IX+off, A	4	2	((IX)+off) ← (AH), ((IX)+off+1) ← (AL)	—	—	—	-----	D6
19	MOVW ext, A	5	3	(ext) ← (AH), (ext+1) ← (AL)	—	—	—	-----	D4
20	MOVW @EP, A	3	1	((EP)) ← (AH), ((EP)+1) ← (AL)	—	—	—	-----	D7

表 A-3. 動作一覧表 ( 転送系 ) (Sheet 2 of 3 )

No	MNEMONIC	~	#	動作	TL	TH	AH	NZVC	OP CODE
21	MOVW EP, A	1	1	(EP) ← (A)	—	—	—	----	E3
22	MOVW A, #d16	3	3	(A) ← d16	AL	AH	dH	++--	E4
23	MOVW A, dir	4	2	(AH) ← (dir), (AL) ← (dir+1)	AL	AH	dH	++--	C5
24	MOVW A, @IX+off	4	2	(AH) ← ((IX)+off), (AL) ← ((IX)+off+1)	AL	AH	dH	++--	C6
25	MOVW A, ext	5	3	(AH) ← (ext), (AL) ← (ext+1)	AL	AH	dH	++--	C4
26	MOVW A, @A	3	1	(AH) ← ((A)), (AL) ← ((A)+1)	AL	AH	dH	++--	93
27	MOVW A, @EP	3	1	(AH) ← ((EP)), (AL) ← ((EP)+1)	AL	AH	dH	++--	C7
28	MOVW A, EP	1	1	(A) ← (EP)	—	—	dH	----	F3
29	MOVW EP, #d16	3	3	(EP) ← d16	—	—	—	----	E7
30	MOVW IX, A	1	1	(IX) ← (A)	—	—	—	----	E2
31	MOVW A, IX	1	1	(A) ← (IX)	—	—	dH	----	F2
32	MOVW SP, A	1	1	(SP) ← (A)	—	—	—	----	E1
33	MOVW A, SP	1	1	(A) ← (SP)	—	—	dH	----	F1
34	MOV @A, T	2	1	((A)) ← (T)	—	—	—	----	82
35	MOVW @A, T	3	1	((A)) ← (TH), ((A)+1) ← (TL)	—	—	—	----	83
36	MOVW IX, #d16	3	3	(IX) ← d16	—	—	—	----	E6
37	MOVW A, PS	1	1	(A) ← (PS)	—	—	dH	----	70
38	MOVW PS, A	1	1	(PS) ← (A)	—	—	—	++++	71
39	MOVW SP, #d16	3	3	(SP) ← d16	—	—	—	----	E5
40	SWAP	1	1	(AH) ↔ (AL)	—	—	AL	----	10
41	SETB dir:b	4	2	(dir):b ← 1	—	—	—	----	A8 ~ AF
42	CLRB dir:b	4	2	(dir):b ← 0	—	—	—	----	A0 ~ A7
43	XCH A, T	1	1	(AL) ↔ (TL)	AL	—	—	----	42
44	XCHW A, T	1	1	(A) ↔ (T)	AL	AH	dH	----	43

表 A-3. 動作一覧表 ( 転送系 ) (Sheet 3 of 3 )

No	MNEMONIC	～	#	動作	TL	TH	AH	NZVC	OP CODE
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

( 注意事項 )

A へのバイト転送動作時には, T ← A は low byte のみです。

複数オペランド命令でのオペランドは, MNEMONIC で表示された順に格納されるものとします。

表 A-4. 動作一覧表 ( 演算系 ) (Sheet 1 of 3 )

No	MNEMONIC	～	#	動作	TL	TH	AH	NZVC	OP CODE
1	ADDC A, Ri	2	1	(A) ← (A)+(Ri)+C	—	—	—	++++	28 ~ 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 ~ 3F
9	SUBC A, #d8	2	2	(A) ← (A)-d8-C	—	—	—	++++	34
10	SUBC A, dir	3	2	(A) ← (A)-(dir)-C	—	—	—	++++	35
11	SUBC A, @IX+off	3	2	(A) ← (A)-((IX)+off)-C	—	—	—	++++	36
12	SUBC A, @EP	2	1	(A) ← (A)-((EP))-C	—	—	—	++++	37
13	SUBCW A	1	1	(A) ← (T)-(A)-C	—	—	dH	++++	33
14	SUBC A	1	1	(AL) ← (TL)-(AL)-C	—	—	—	++++	32
15	IINC Ri	3	1	(Ri) ← (Ri)+1	—	—	—	+++-	C8 ~ CF
16	INCW EP	1	1	(EP) ← (EP)+1	—	—	—	----	C3
17	INCW IX	1	1	(IX) ← (IX)+1	—	—	—	----	C2

表 A-4. 動作一覧表 (演算系) (Sheet 2 of 3)

No	MNEMONIC	~	#	動作	TL	TH	AH	NZVC	OP CODE
18	INCW A	1	1	$(A) \leftarrow (A)+1$	—	—	dH	++--	C0
19	DEC Ri	3	1	$(Ri) \leftarrow (Ri)-1$	—	—	—	+++—	D8 ~ 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	$(A) \leftarrow (T)/(A),$ 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
31	ROLC A	1	1		—	—	—	++-+	02
32	CMP A, #d8	2	2	$(A) - d8$	—	—	—	++++	14
33	CMP A, dir	3	2	$(A) - (dir)$	—	—	—	++++	15
34	CMP A, @EP	2	1	$(A) - ((EP))$	—	—	—	++++	17
35	CMP A, @IX+off	3	2	$(A) - ((IX)+off)$	—	—	—	++++	16
36	CMP A, Ri	2	1	$(A) - (Ri)$	—	—	—	++++	18 ~ 1F
37	DAA	1	1	decimal adjust for addition	—	—	—	++++	84
38	DAS	1	1	decimal adjust for subtraction	—	—	—	++++	94
39	XOR A	1	1	$(A) \leftarrow (AL) \vee (TL)$	—	—	—	++R—	52
40	XOR A, #d8	2	2	$(A) \leftarrow (AL) \vee d8$	—	—	—	++R—	54
41	XOR A, dir	3	2	$(A) \leftarrow (AL) \vee (dir)$	—	—	—	++R—	55

表 A-4. 動作一覧表 (演算系) (Sheet 3 of 3)

No	MNEMONIC	~	#	動作	TL	TH	AH	NZVC	OP CODE
42	XOR A, @EP	2	1	$(A) \leftarrow (AL) \vee ((EP))$	—	—	—	++ R —	57
43	XOR A, @IX+off	3	2	$(A) \leftarrow (AL) \vee ((IX)+off)$	—	—	—	++ R —	56
44	XOR A, Ri	2	1	$(A) \leftarrow (AL) \vee (Ri)$	—	—	—	++ R —	58 ~ 5F
45	AND A	1	1	$(A) \leftarrow (AL) \wedge (TL)$	—	—	—	++ R —	62
46	AND A, #d8	2	2	$(A) \leftarrow (AL) \wedge d8$	—	—	—	++ R —	64
47	AND A, dir	3	2	$(A) \leftarrow (AL) \wedge (dir)$	—	—	—	++ R —	65
48	AND A, @EP	2	1	$(A) \leftarrow (AL) \wedge ((EP))$	—	—	—	++ R —	67
49	AND A, @IX+off	3	2	$(A) \leftarrow (AL) \wedge ((IX)+off)$	—	—	—	++ R —	66
50	AND A, Ri	2	1	$(A) \leftarrow (AL) \wedge (Ri)$	—	—	—	++ R —	68 ~ 6F
51	OR A	1	1	$(A) \leftarrow (AL) \vee (TL)$	—	—	—	++ R —	72
52	OR A, #d8	2	2	$(A) \leftarrow (AL) \vee d8$	—	—	—	++ R —	74
53	OR A, dir	3	2	$(A) \leftarrow (AL) \vee (dir)$	—	—	—	++ R —	75
54	OR A, @EP	2	1	$(A) \leftarrow (AL) \vee ((EP))$	—	—	—	++ R —	77
55	OR A, @IX,off	3	2	$(A) \leftarrow (AL) \vee ((IX)+off)$	—	—	—	++ R —	76
56	OR A, Ri	2	1	$(A) \leftarrow (AL) \vee (Ri)$	—	—	—	++ R —	78 ~ 7F
57	CMP dir, #d8	4	3	$(dir) - d8$	—	—	—	++++	95
58	CMP @EP, #d8	3	2	$((EP)) - d8$	—	—	—	++++	97
59	CMP @IX+off, #d8	4	3	$((IX)+off) - d8$	—	—	—	++++	96
60	CMP Ri, #d8	3	2	$(Ri) - d8$	—	—	—	++++	98 ~ 9F
61	INCW SP	1	1	$(SP) \leftarrow (SP) + 1$	—	—	—	----	C1
62	DECW SP	1	1	$(SP) \leftarrow (SP) - 1$	—	—	—	----	D1

表 A-5. 動作一覧表 ( 分岐系 )

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

表 A-6. 動作一覧表 ( その他 )

No	MNEMONIC	~	#	動作	TL	TH	AH	NZVC	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 フラグ変化表

表 A-7. に転送系命令の, 表 A-8. に演算系命令の, 表 A-9. に分岐系命令の, 表 A-10. にその他の命令  
フラグ変化表を示します。

### フラグ変化表

表 A-7. フラグ変化表 ( 転送系命令 ) (Sheet 1 of 2 )

命令	フラグ変化
MOV dir, A MOV @IX+off, A MOV ext, A MOV @EP, A MOV Ri, A	N : 変化しません Z : 変化しません V : 変化しません C : 変化しません
MOV , #d8 MOV A, dir MOV A, @IX+off MOV A, ext MOV A, @A MOV A, @EP MOV A, Ri	N : 転送したデータが負なら 1, それ以外なら 0 Z : 転送したデータが "0" なら 1, それ以外なら 0 V : 変化しません C : 変化しません
MOV dir, #d8 MOV @IX+off, #d8 MOV @EP, #d8 MOV Ri, #d8	N : 変化しません Z : 変化しません V : 変化しません C : 変化しません
MOVW dir, A MOVW @IX+off, A MOVW ext, A MOVW @EP, A	N : 変化しません Z : 変化しません V : 変化しません C : 変化しません
MOVW A, #d16 MOVW A, dir MOVW A, @IX+off MOVW A, ext MOVW A, @A MOVW A, @EP	N : 転送したデータが負なら 1, それ以外なら 0 Z : 転送したデータが "0" なら 1, それ以外なら 0 V : 変化しません C : 変化しません

表 A-7. フラグ変化表 ( 転送系命令 ) (Sheet 2 of 2 )

命令	フラグ変化
MOVW A, EP MOVW EP, #d16 MOVW IX, A MOVW A, IX MOVW SP, A MOVW A, SP MOVW SP, #d16	N: 変化しません Z: 変化しません V: 変化しません C: 変化しません
MOV @A, T MOVW @A, T	N: 変化しません Z: 変化しません V: 変化しません C: 変化しません
MOVW IX, #d16 MOVW A, PS MOVW A, PC JMP @A	N: 変化しません Z: 変化しません V: 変化しません C: 変化しません
MOVW PS, A	N: A のビット 3 が "1" なら 1, "0" なら 0 Z: A のビット 2 が "1" なら 1, "0" なら 0 V: A のビット 1 が "1" なら 1, "0" なら 0 C: A のビット 0 が "1" なら 1, "0" なら 0
SETB dir:b CLRb dir:b	N: 変化しません Z: 変化しません V: 変化しません C: 変化しません
SWAP XCH A, T	N: 変化しません Z: 変化しません V: 変化しません C: 変化しません
XCHW A, T XCHW A, EP XCHW A, IX XCHW A, SP XCHW A, PC	N: 変化しません Z: 変化しません V: 変化しません C: 変化しません

表 A-8. フラグ変化表 ( 演算系命令 ) (Sheet 1 of 4 )

命令	フラグ変化
ADDC A, Ri ADDC A, #d8 ADDC A, dir ADDC A, @IX+off ADDC A, @EP	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : オーバフローが発生すれば 1, それ以外なら 0 C : キャリーが発生すれば 1, それ以外なら 0
ADDC A ADDCW A	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : オーバフローが発生すれば 1, それ以外なら 0 C : キャリーが発生すれば 1, それ以外なら 0
SUBC A, Ri SUBC A, #d8 SUBC A, dir SUBC A, @IX+off SUBC A, @EP	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : オーバフローが発生すれば 1, それ以外なら 0 C : ボローが発生すれば 1, それ以外なら 0
SUBC A SUBCW A	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : オーバフローが発生すれば 1, それ以外なら 0 C : ボローが発生すれば 1, それ以外なら 0
INC Ri	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : オーバフローが発生すれば 1, それ以外なら 0 C : 変化しません
INCW EP INCW IX INCW SP	N : 変化しません Z : 変化しません V : 変化しません C : 変化しません
INCW A	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : 変化しません C : 変化しません

表 A-8. フラグ変化表 ( 演算系命令 ) (Sheet 2 of 4 )

命令	フラグ変化
DEC Ri	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : オーバフローが発生すれば 1, それ以外なら 0 C : 変化しません
DECW EP DECW IX DECW SP	N : 変化しません Z : 変化しません V : 変化しません C : 変化しません
DECW A	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : 変化しません C : 変化しません
MULU A	N : 変化しません Z : 変化しません V : 変化しません C : 変化しません
DIVU A	N : 変化しません Z : 演算前の A が 0000 <sub>H</sub> なら 1, それ以外なら 0 V : 変化しません C : 変化しません
ANDW A	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : 常に 0 C : 変化しません
AND A, #d8 AND A, dir AND A, @EP AND A, @IX+off AND A, Ri	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : 常に 0 C : 変化しません

表 A-8. フラグ変化表 ( 演算系命令 ) (Sheet 3 of 4 )

命令	フラグ変化
ORW A	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : 常に 0 C : 変化しません
OR A, #d8 OR A, dir OR A, @EP OR A, @IX+off OR A, Ri	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : 常に 0 C : 変化しません
XORW A	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : 常に 0 C : 変化しません
XOR A, #d8 XOR A, dir XOR A, @EP XOR A, @IX+off XOR A, Ri	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : 常に 0 C : 変化しません
CMP A	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : オーバフローが発生すれば 1, それ以外なら 0 C : ボローが発生すれば 1, それ以外なら 0
CMPW A	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : オーバフローが発生すれば 1, それ以外なら 0 C : ボローが発生すれば 1, それ以外なら 0
CMP A, #d8 CMP A, dir CMP A, @EP CMP A, @IX+off CMP A, Ri	N : 演算結果が負なら 1, それ以外なら 0 Z : 演算結果が "0" なら 1, それ以外なら 0 V : オーバフローが発生すれば 1, それ以外なら 0 C : ボローが発生すれば 1, それ以外なら 0

表 A-8. フラグ変化表 ( 演算系命令 ) (Sheet 4 of 4 )

命令	フラグ変化
CMP dir, #d8	N : 演算結果が負なら 1, それ以外なら 0
CMP @EP, #d8	Z : 演算結果が "0" なら 1, それ以外なら 0
CMP @IX+off, #d8	V : オーバフローが発生すれば 1, それ以外なら 0
CMP Ri, #d8	C : ボローが発生すれば 1, それ以外なら 0
RORC A	N : 演算結果が負なら 1, それ以外なら 0
ROLC A	Z : 演算結果が "0" なら 1, それ以外なら 0 V : 変化しません C : 演算前の A のビット 0 (RORA のとき) またはビット 7 (ROLA のとき) が入ります
DAA	N : 演算結果が負なら 1, それ以外なら 0
DAS	Z : 演算結果が "0" なら 1, それ以外なら 0 V : オーバフローが発生すれば 1, それ以外なら 0 C : キャリー ( ボロー ) が発生すれば 1, それ以外なら 0

表 A-9. フラグ変化表 ( 分岐系命令 )

命令	フラグ変化
BZ rel/BEQ rel	N : 変化しません
BNZ rel/BNE rel	Z : 変化しません
BC rel/BLO rel	V : 変化しません
BNC rel/BHS rel	C : 変化しません
BN rel	
BP rel	
BLT rel	
BGE rel	
JMP addr16	N : 変化しません Z : 変化しません V : 変化しません C : 変化しません
BBC dir:b, rel	N : 変化しません
BBS dir:b, rel	Z : ビット b が "0" なら 1, "1" なら 0 V : 変化しません C : 変化しません
CALL addr16	N : 変化しません
CALLV #vct	Z : 変化しません
RET	V : 変化しません C : 変化しません
RETI	N : 退避されていた CCR の N の値が入ります Z : 退避されていた CCR の Z の値が入ります V : 退避されていた CCR の V の値が入ります C : 退避されていた CCR の C の値が入ります

表 A-10. フラグ変化表 ( その他の命令 )

命令	フラグ変化
PUSHW A PUSHW IX	N: 変化しません Z: 変化しません V: 変化しません C: 変化しません
POPW A POPW IX	N: 変化しません Z: 変化しません V: 変化しません C: 変化しません
NOP	N: 変化しません Z: 変化しません V: 変化しません C: 変化しません
CLRC	N: 変化しません Z: 変化しません V: 変化しません C: "0" になります
SETC	N: 変化しません Z: 変化しません V: 変化しません C: "1" になります
CLRI	N: 変化しません Z: 変化しません V: 変化しません C: 変化しません I: "0" になります
SETI	N: 変化しません Z: 変化しません V: 変化しません C: 変化しません I: "1" になります



## B. バス動作一覧表



表 B-1. にバス動作一覧表を示します。

### バス動作一覧表

表 B-1. バス動作一覧表 (Sheet 1 of 1 1 )

CODE	MNEMONIC	～	サイクル	アドレスバス	データバス	RD	WR	RMW
00	NOP	1	1	N +2	次の次の命令	1	0	0
80	CLRI							
90	SETI							
81	CLRC							
91	SETC							
10	SWAP	1	1	N +2	次の次の命令	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	次の次の命令	1	0	0
23	ADDCW A							
33	SUBCW A							
43	XCHW A, T							
53	XORW A							
63	ANDW A							
73	ORW A							

表 B-1. バス動作一覧表 (Sheet 2 of 1 1 )

CODE	MNEMONIC	～	サイクル	アドレスバス	データバス	RD	WR	RMW
04	MOV A, #d8	2	1	N +2	次の命令	1	0	0
14	CMP A, #d8		2	N +3	次の次の命令	1	0	0
24	ADDC A, #d8							
34	SUBC A, #d8							
54	XOR A, #d8							
64	AND A, #d8							
74	OR A, #d8							
05	MOV A, dir	3	1	N +2	次の命令	1	0	0
15	CMP A, dir		2	dir アドレス	データ	1	0	0
25	ADDC A, dir		3	N +3	次の次の命令	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	次の命令	1	0	0
			2	dir アドレス	データ	0	1	0
			3	N +3	次の次の命令	1	0	0
06	MOV A, @IX+off	3	1	N +2	次の命令	1	0	0
16	CMP A, @IX+off		2	N +3	次の次の命令	1	0	0
26	ADDC A, @IX+off		3	(IX)+off アドレス	データ	1	0	0
36	SUBC A, @IX+off							
56	XOR A, @IX+off							
66	AND A, @IX+off							
76	OR A, @IX+off							

表 B-1. バス動作一覧表 (Sheet 3 of 1 1 )

CODE	MNEMONIC	～	サイクル	アドレスバス	データバス	RD	WR	RMW
46	MOV @IX+off, A	3	1	N +2	次の命令	1	0	0
			2	N +3	次の次の命令	1	0	0
			3	(IX)+off アドレス	データ	0	1	0
07	MOV A, @EP	2	1	N +2	次の次の命令	1	0	0
17	CMP A, @EP		2	(EP) アドレス	データ	1	0	0
27	ADDC A, @EP							
37	SUBC A, @EP							
57	XOR A, @EP							
67	AND A, @EP							
77	OR A, @EP							
47	MOV @EP, A	2	1	N +2	次の次の命令	1	0	0
			2	(EP) アドレス	データ	0	1	0
08 - 0F	MOV A, Ri	2	1	N +2	次の次の命令	1	0	0
18 - 1F	CMP A, Ri		2	Rn アドレス	データ	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	次の次の命令	1	0	0
			2	Rn アドレス	データ	0	1	0
C0	INCW A	1	1	N +2	次の次の命令	1	0	0
D0	DECW A							
C1	INCW SP							
D1	DECW SP							
C2	INCW IX							
D2	DECW IX							
C3	INCW EP							
D3	DECW EP							

表 B-1. バス動作一覧表 (Sheet 4 of 1 1 )

CODE	MNEMONIC	～	サイクル	アドレスバス	データバス	RD	WR	RMW
F0	MOVW A, PC	2	1	N +2	次の次の命令	1	0	0
			2	—	—	0	0	0
E1	MOVW SP, A	1	1	N +2	次の次の命令	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	N +2 のデータ	1	0	0
			2	分岐アドレス	次の命令	1	0	0
			3	分岐アドレス +1	次の次の命令	1	0	0
F5	XCHW A, SP	1	1	N +2	次の次の命令	1	0	0
F6	XCHW A, IX							
F7	XCHW A, EP							
F4	XCHW A, PC	3	1	N +2	N +2 のデータ	1	0	0
			2	分岐アドレス	次の命令	1	0	0
			3	分岐アドレス +1	次の次の命令	1	0	0
A0 - A7	CLRB dir:n	4	1	N +2	次の命令	1	0	1
A8 - AF	SETB dir:n		2	dir アドレス	データ	1	0	1
			3	dir アドレス	データ	0	1	0
			4	N +3	次の次の命令	1	0	0

表 B-1. バス動作一覧表 (Sheet 5 of 1 1 )

CODE	MNEMONIC	～	サイクル	アドレスバス	データバス	RD	WR	RMW
B0 - B7	BBC dir:n, rel	分岐時						
B8 - BF	BBS dir:n, rel	5	1	N +2	rel	1	0	0
			2	dir アドレス	データ	1	0	0
			3	N +3	N+3 のデータ	1	0	0
			4	分岐アドレス	次の命令	1	0	0
			5	分岐アドレス +1	次の次の命令	1	0	0
		非分岐時						
		5	1	N +2	rel	1	0	0
			2	dir アドレス	データ	1	0	0
			3	N +3	次の命令	1	0	0
			4	—	—	0	0	0
			5	N +4	次の次の命令	1	0	0
60	MOV A, ext	4	1	N +2	ext (L byte)	1	0	0
			2	N +3	次の命令	1	0	0
			3	ext アドレス	データ	1	0	0
			4	N +4	次の次の命令	1	0	0
61	MOV ext, A	4	1	N +2	ext (L byte)	1	0	0
			2	N +3	次の命令	1	0	0
			3	ext アドレス	データ	0	1	0
			4	N +4	次の次の命令	1	0	0
C4	MOVW A, ext	5	1	N +2	ext (L byte)	1	0	0
			2	N +3	次の命令	1	0	0
			3	ext アドレス	データ (H byte)	1	0	0
			4	ext+1 アドレス	データ (L byte)	1	0	0
			5	N +4	次の次の命令	1	0	0
D4	MOVW ext, A	5	1	N +2	ext (L byte)	1	0	0
			2	N +3	次の命令	1	0	0
			3	ext アドレス	データ (H byte)	0	1	0
			4	ext+1 アドレス	データ (L byte)	0	1	0
			5	N +4	次の次の命令	1	0	0

表 B-1. バス動作一覧表 (Sheet 6 of 1 1 )

CODE	MNEMONIC	～	サイクル	アドレスバス	データバス	RD	WR	RMW
C5	MOVW A, dir	4	1	N +2	次の命令	1	0	0
			2	dir アドレス	データ (H byte)	1	0	0
			3	dir+1 アドレス	データ (L byte)	1	0	0
			4	N +3	次の次の命令	1	0	0
D5	MOVW dir, A	4	1	N +2	次の命令	1	0	0
			2	dir アドレス	データ (H byte)	0	1	0
			3	dir+1 アドレス	データ (L byte)	0	1	0
			4	N +3	次の次の命令	1	0	0
C6	MOVW A, @IX+off	4	1	N +2	次の命令	1	0	0
			2	N +3	次の次の命令	1	0	0
			3	(IX)+off アドレス	データ (H byte)	1	0	0
			4	(IX)+off+ 1 アドレス	データ (L byte)	1	0	0
D6	MOVW @IX+off, A	4	1	N +2	次の命令	1	0	0
			2	N +3	次の次の命令	1	0	0
			3	(IX)+off アドレス	データ (H byte)	0	1	0
			4	(IX)+off+1 アドレス	データ (L byte)	0	1	0
C7	MOVW A, @EP	3	1	N +2	次の次の命令	1	0	0
			2	(EP) アドレス	データ (H byte)	1	0	0
			3	(EP)+1 アドレス	データ (L byte)	1	0	0
D7	MOVW @EP, A	3	1	N +2	次の次の命令	1	0	0
			2	(EP) アドレス	データ (H byte)	0	1	0
			3	(EP)+1 アドレス	データ (L byte)	0	1	0
85	MOV dir, #d8	4	1	N +2	# d8	1	0	0
			2	dir アドレス	データ	0	1	0
			3	N +3	次の命令	1	0	0
			4	N +4	次の次の命令	1	0	0

表 B-1. バス動作一覧表 (Sheet 7 of 1 1 )

CODE	MNEMONIC	～	サイクル	アドレスバス	データバス	RD	WR	RMW
95	CMP dir, #d8	4	1	N +2	# d8	1	0	0
			2	dir アドレス	データ	1	0	0
			3	N +3	次の命令	1	0	0
86	MOV @IX+off, #d8	4	1	N +2	#d8	1	0	0
			2	N +3	次の命令	1	0	0
			3	(IX)+off アドレス	データ	0	1	0
			4	N +4	次の次の命令	1	0	0
96	CMP @IX+off, #d8	4	1	N +2	#d8	1	0	0
			2	N +3	次の命令	1	0	0
			3	(IX)+off アドレス	データ	1	0	0
			4	N +4	次の次の命令	1	0	0
87	MOV @EP, #d8	3	1	N +2	次の命令	1	0	0
			2	(EP) アドレス	データ	0	1	0
			3	N +3	次の次の命令	1	0	0
97	CMP @EP, #d8	3	1	N +2	次の命令	1	0	0
			2	(EP) アドレス	データ	1	0	0
			3	N +3	次の次の命令	1	0	0
88 - 8F	MOV Ri, #d8	3	1	N +2	次の命令	1	0	0
			2	Rn アドレス	データ	0	1	0
			3	N +3	次の次の命令	1	0	0
98 - 9F	CMP Ri, #d8	3	1	N +2	次の命令	1	0	0
			2	Rn アドレス	データ	1	0	0
			3	N +3	次の次の命令	1	0	0
82	MOV @A, T	2	1	N +2	次の次の命令	1	0	0
			2	(A) アドレス	データ	0	1	0
92	MOV A, @A	2	1	N +2	次の次の命令	1	0	0
			2	(A) アドレス	データ	1	0	0

表 B-1. バス動作一覧表 (Sheet 8 of 1 1 )

CODE	MNEMONIC	～	サイクル	アドレスバス	データバス	RD	WR	RMW
83	MOVW @A, T	3	1	N +2	次の次の命令	1	0	0
			2	(A) アドレス	データ (H byte)	0	1	0
			3	(A) +1 アドレス	データ (L byte)	0	1	0
93	MOVW A, @A	3	1	N +2	次の次の命令	1	0	0
			2	(A) アドレス	データ (H byte)	1	0	0
			3	(A) +1 アドレス	データ (L byte)	1	0	0
E4	MOVW A, #d16	3	1	N +2	データ (L byte)	1	0	0
E5	MOVW SP, #d16		2	N +3	次の命令	1	0	0
E6	MOVW IX, #d16		3	N +4	次の次の命令	1	0	0
E7	MOVW EP, #d16							
84	DAA	1	1	N +2	次の次の命令	1	0	0
94	DAS							
02	ROL C A							
03	ROR C A							
70	MOVW A, PS							
71	MOVW PS, A							
C8 - CF	INC Ri	3	1	N +2	次の次の命令	1	0	1
D8 - DF	DEC Ri		2	Rn アドレス	データ	1	0	1
			3	Rn アドレス	データ	0	1	0
E8 - EF	CALLV #n	7	1	N +2	N +2 のデータ	1	0	0
			2	ベクタアドレス	ベクタ (H)	1	0	0
			3	ベクタアドレス +1	ベクタ (L)	1	0	0
			4	SP -1	復帰アドレス (L)	0	1	0
			5	SP -2	復帰アドレス (H)	0	1	0
			6	分岐先アドレス	次の命令	1	0	0
			7	分岐先アドレス +1	次の次の命令	1	0	0



表 B-1. バス動作一覧表 (Sheet 9 of 1 1 )

CODE	MNEMONIC	～	サイクル	アドレスバス	データバス	RD	WR	RMW
F8	BNC rel	分岐時						
F9	BC rel	4	1	N +2	N +2 のデータ	1	0	0
FA	BP rel		2	N +3	N +3 のデータ	1	0	0
FB	BN rel		3	分岐先アドレス	次の命令	1	0	0
FC	BNZ rel		4	分岐先アドレス +1	次の次の命令	1	0	0
FD	BZ rel	非分岐時						
FE	BGE rel	2	1	N +2	次の命令	1	0	0
FF	BLT rel		2	N +3	次の次の命令	1	0	0
40	PUSHW A	4	1	N +2	次の次の命令	1	0	0
41	PUSHW IX		2	—	—	0	0	0
			3	SP -1	退避データ (L)	0	1	0
			4	SP -2	退避データ (H)	0	1	0
50	POPW A	3	1	N +2	次の次の命令	1	0	0
51	POPW IX		2	SP	復帰データ (H)	1	0	0
			3	SP +1	復帰データ (L)	1	0	0
20	RET	6	1	N +2	N +2 のデータ	1	0	0
			2	SP	復帰アドレス (H)	1	0	0
			3	SP +1	復帰アドレス (L)	1	0	0
			4	—	—	0	0	0
			5	復帰アドレス	次の命令	1	0	0
			6	復帰アドレス +1	次の次の命令	1	0	0
30	RETI	8	1	N +2	N +2 のデータ	1	0	0
			2	SP	PSH (RP, DP)	1	0	0
			3	SP +1	PSL (CCR)	1	0	0
			4	SP +2	復帰アドレス (H)	1	0	0
			5	SP +3	復帰アドレス (L)	1	0	0
			6	—	—	0	0	0
			7	復帰アドレス	次の命令	1	0	0
			8	復帰アドレス +1	次の次の命令	1	0	0

表 B-1. バス動作一覧表 (Sheet 1 0 of 1 1 )

CODE	MNEMONIC	～	サイクル	アドレスバス	データバス	RD	WR	RMW
31	CALL ext	6	1	N +2	分岐先アドレス (L)	1	0	0
			2	—	—	0	0	0
			3	SP -1	復帰アドレス (L)	0	1	0
			4	SP -2	復帰アドレス (H)	0	1	0
			5	分岐先アドレス	次の命令	1	0	0
			6	分岐先アドレス +1	次の次の命令	1	0	0
21	JMP ext	4	1	N +2	分岐先アドレス (L)	1	0	0
			2	—	—	0	0	0
			3	分岐先アドレス	次の命令	1	0	0
			4	分岐先アドレス +1	次の次の命令	1	0	0
01	MULU A	8	1	N +2	次の次の命令	1	0	0
			2	—	—	0	0	0
			～					
			8	—	—	0	0	0
11	DIVU A	17	1	N +2	次の次の命令	1	0	0
			2	—	—	0	0	0
			～					
			17	—	—	0	0	0
—	RESET	7	1	—	—	0	0	0
			2	0FFFD <sub>H</sub>	モードデータ	1	0	0
			3	0FFFE <sub>H</sub>	リセットベクタ (H)	1	0	0
			4	0FFF <sub>H</sub>	リセットベクタ (L)	1	0	0
			5	—	—	0	0	0
			6	スタートアドレス	次の命令	1	0	0
			7	スタートアドレス +1	次の次の命令	1	0	0

表 B-1. バス動作一覧表 (Sheet 1 1 of 1 1 )

CODE	MNEMONIC	～	サイクル	アドレスバス	データバス	RD	WR	RMW
—	INTERRUPT	9	1	N +2	N +2 のデータ	1	0	0
			2	ベクタアドレス	ベクタ (H)	1	0	0
			3	ベクタアドレス +1	ベクタ (L)	1	0	0
			4	SP -1	復帰アドレス (L)	0	1	0
			5	SP -2	復帰アドレス (H)	0	1	0
			6	SP -3	PSL (CCR)	0	1	0
			7	SP -4	PSH (RP, DP)	0	1	0
			8	分岐先アドレス	次の命令	1	0	0
			9	分岐先アドレス +1	次の次の命令	1	0	0

-: 無効なバスサイクル

N: 実行中の命令が格納されているアドレス

(注意事項)

命令のサイクルは、直前の命令によって 1 サイクル延長される場合があります。また、IO 領域へのアクセスでは、命令のサイクル数が延長される場合があります。

## C. 命令マップ



表 C-1. に命令マップを示します。

# 命令マップ

表 C-1. 命令マップ

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

# 改訂履歴



## Document Revision History

文書名 : F <sup>2</sup> MC-8FX Programming Specifications				
文書番号 : 002-07003				
版	ECN 番号	発行日	変更者	変更内容
**	-	02/06/2008	GERR	サイプレスとして Spansion 取扱説明書 CM26-00301-2 をドキュメントコード 002-07003 に登録しました。 本版の内容およびフォーマットに変更はありません。
*A	5693237	04/12/2017	GERR	これは英語版の 002-07004 Rev. *A を翻訳した日本語版です。
*B	6533971	04/24/2019	HTER	タイトルとカテゴリを変更 表 A-4 を更新 (No.42, 43, 60)