お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

RENESAS

ユーザーズ・マニュアル

μPD170××シリーズ

４ビット・シングルチップ・マイクロコントローラ

共通事項

資料番号　U13262JJ2V0UM00（第２版）
（旧資料番号　IEU-786）
発行年月　March 1998　N　CP(K)

© NEC Corporation 1991

〔メ　モ〕

目次要約

第１章　概　　説　…　19

第２章　プログラム・メモリ（ROM）　…　21

第３章　プログラム・カウンタ（PC）　…　31

第４章　アドレス・スタック（ASK）　…　35

第５章　データ・メモリ（RAM）　…　49

第６章　システム・レジスタ（SYSREG）　…　53

第７章　ジェネラル・レジスタ（GR）　…　89

第８章　演算論理ユニット（ALU）　…　101

第９章　レジスタ・ファイル（RF）　…　131

第10章　データ・バッファ（DBF）　…　139

第11章　汎用ポート　…　151

第12章　割り込み機能　…　155

第13章　スタンバイ機能　…　179

第14章　リセット機能　…　191

第15章　命令セット　…　213

付録Ａ　開発ツール　…　291

付録Ｂ　マスクROMの発注方法　…　293

付録Ｃ　命令索引　…　295

付録Ｄ　改版履歴　…　297

CMOSデバイスの一般的注意事項

①静電気対策（MOS全般）

注意　MOSデバイス取り扱いの際は静電気防止を心がけてください。

MOSデバイスは強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には，NECが出荷梱包に使用している導電性のトレーやマガジン・ケース，または導電性の緩衝材，金属ケースなどを利用し，組み立て工程にはアースを施してください。プラスチック板上に放置したり，端子を触ったりしないでください。

また，MOSデバイスを実装したボードについても同様の扱いをしてください。

②未使用入力の処理（CMOS特有）

注意　CMOSデバイスの入力レベルは固定してください。

バイポーラやNMOSのデバイスと異なり，CMOSデバイスの入力に何も接続しない状態で動作させると，ノイズなどに起因する中間レベル入力が生じ，内部で貫通電流が流れて誤動作を引き起こす恐れがあります。プルアップかプルダウンによって入力レベルを固定してください。また，未使用端子が出力となる可能性（タイミングは規定しません）を考慮すると，個別に抵抗を介してV_{DD}またはGNDに接続することが有効です。

資料中に「未使用端子の処理」について記載のある製品については，その内容を守ってください。

③初期化以前の状態（MOS全般）

注意　電源投入時，MOSデバイスの初期状態は不定です。

分子レベルのイオン注入量等で特性が決定するため，初期状態は製造工程の管理外です。電源投入時の端子の出力状態や入出力設定，レジスタ内容などは保証しておりません。ただし，リセット動作やモード設定で定義している項目については，これらの動作ののちに保証の対象となります。

リセット機能を持つデバイスの電源投入後は，まずリセット動作を実行してください。

***SIMPLEHOST*, *emIC-17K*は，日本電気株式会社の登録商標です。**

PC/ATは，米国IBM社の商標です。

Windowsは，米国Microsoft Corporationの米国およびその他の国における登録商標または商標です。

注意： μPD170××シリーズはI^2Cバス・インタフェース回路を内蔵しています。
I^2Cバス・インタフェースを使用される場合には，カスタム・コードをご発注いただく時に，事前にその旨ご申告下さい。申告に基づき，以下の特典が受けられます。
日本電気株式会社のI^2Cバス対応部品をご購入いただくことにより，これらの部品をI^2Cシステムに使用する実施権がフィリップス社I^2C特許に基づき許諾されることになります。ただし，これらのI^2Cシステムはフィリップス社によって設定されたI^2C標準規格に合致しているものとします。

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

本製品が外国為替および外国貿易管理法の規定による戦略物資等（または役務）に該当するか否かは，ユーザ（仕様を決定した者）が判定してください。

○**本資料の内容は，後日変更する場合があります。**
○文書による当社の承諾なしに本資料の転載複製を禁じます。
○本資料に記載された製品の使用もしくは本資料に記載の情報の使用に際して，当社は当社もしくは第三者の知的所有権その他の権利に対する保証または実施権の許諾を行うものではありません。上記使用に起因する第三者所有の権利にかかわる問題が発生した場合，当社はその責を負うものではありませんのでご了承ください。
○当社は品質，信頼性の向上に努めていますが，半導体製品はある確率で故障が発生します。当社半導体製品の故障により結果として，人身事故，火災事故，社会的な損害等を生じさせない冗長設計，延焼対策設計，誤動作防止設計等安全設計に十分ご注意願います。
○当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定して頂く「特定水準」に分類しております。また，各品質水準は以下に示す用途に製品が使われることを意図しておりますので，当社製品の品質水準をご確認の上ご使用願います。
標準水準：コンピュータ，OA機器，通信機器，計測機器，AV機器，家電，工作機械，パーソナル機器，産業用ロボット
特別水準：輸送機器（自動車，列車，船舶等），交通用信号機器，防災／防犯装置，各種安全装置，生命維持を直接の目的としない医療機器
特定水準：航空機器，航空宇宙機器，海底中継機器，原子力制御システム，生命維持のための医療機器，生命維持のための装置またはシステム等
当社製品のデータ・シート／データ・ブック等の資料で，特に品質水準の表示がない場合は標準水準製品であることを表します。当社製品を上記の「標準水準」の用途以外でご使用をお考えのお客様は，必ず事前に当社販売窓口までご相談頂きますようお願い致します。
○この製品は耐放射線設計をしておりません。

M7 94.11

本版で改訂された主な箇所

箇　所	内　容
全　般	アセンブラを変更（AS17K→RA17K）
全　般	インサーキット・エミュレータ　IE-17K-ETを追加
p.8	**はじめに**の**関連資料**の表を変更
旧版p.2, 3	**第１章　概　　説**から**μPD170××の製品展開**の図と**機能一覧表**を削除
p.79	**６.７.３　ジェネラル・レジスタ・ポインタ使用時の注意**を追加
p.102	**図８－１　ALUブロックの構成**にプログラム・ステータス・ワード（PSWORD）とステータス・フリップフロップの関係の図を追加
p.104	**表８－１　ALU処理命令一覧**に命令の動作と説明を追加
p.106	**８.２.３　ステータス・フリップフロップの機能**に次の説明を追加 **（１）Zフラグ** **（２）CYフラグ** **（３）CMPフラグ** **（４）BCDフラグ**
p.108	**表８－２　２進４ビット演算結果とBCD演算結果**を変更
p.110	**表８－３　算術演算の種類**を追加
p.121	**表８－４　論理演算**を追加
p.124	**表８－６　ビット判断命令**を追加
p.126	**表８－７　比較判断命令**を追加
p.138	**９.４.２　レジスタ・ファイルのシンボル定義と予約語**に**例３**を追加
p.160	**12.2.6　割り込みイネーブル・フリップフロップ（INTE）**に説明を追加
p.183	次の項に備考を追加 **13.4.3　キー入力によるホールト解除** **13.4.4　タイマ・キャリー（ベーシック・タイマ0キャリー）によるホールト解除**
p.183, 184	**13.4.5　割り込みによるホールト解除**に備考と注意を追加
旧版 p.190-192	**旧版12.6　ホールト状態とクロック・ストップ状態時の消費電流**を削除
旧版 p.219-222	**旧版第14章　ワン・タイムPROM品**を削除
p.198	**14.4　パワーオン・リセット**に説明を追加
p.283	**15.5.9（３）SYSCAL entry**を追加
p.291, 292	**Ａ.１　ハードウエア**と**Ａ.２　ソフトウエア**を変更
p.295	**Ｃ.１　命令索引（機能別）**を追加
p.297	**付録Ｄ　改版履歴**を追加

本文欄外の★印は，本版で改訂された主な箇所を示しています。

巻末にアンケート・コーナを設けております。このドキュメントに対するご意見をお気軽にお寄せください。

はじめに

ご利用対象者　このマニュアルは，μPD170××シリーズの機能を理解し，それを用いたアプリケーション・システムを設計するユーザを対象としています。

目　的　このマニュアルはμPD170××シリーズの共通部分を抜き出したもので，プログラムを作る際の参考資料としていただくことを目的としています。

読み方　このマニュアルの読者には，電気，論理回路，マイクロコンピュータの一般知識を必要とします。
レジスタの数やメモリの容量などは品種によって異なるため，このマニュアルでは，最大数や許容範囲などで記述してあります。品種ごとの数値は各品種のデータ・シートによって確認してください。
周辺ハードウエアについても，このマニュアルには記載していないので，各品種のデータ・シートを読んでください。

●一通りμPD170××シリーズの機能を理解しようとするとき
→目次に従って読んでください。

●ニモニックが分かっているときの命令機能を調べるとき
→**付録C　命令索引**をご利用ください。

●ニモニックは知らないが大体の機能が分かっている命令を調べたいとき
→**15.3　命令セット一覧**でその命令のニモニックを調べ，**15.5　命令の個別説明**でその機能を調べてください。

●μPD170××シリーズの電気的特性を知りたいとき
→別冊のデータ・シートを参照してください。

凡　例

データ表記の重み	：左が上位桁，右が下位桁
アクティブ・ロウの表記	：$\overline{\times\times\times}$（端子，信号名称に上線）
メモリ・マップのアドレス	：上部－下位，下部－上位
注	：本文中につけた注の説明
注　意	：気をつけて読んでいただきたい内容
備　考	：本文の補足説明
数の表記	：２進数…××××または××××B 10進数…××××または××××D 16進数…××××H

★ **関連資料**　次の資料とあわせてご利用ください。
また各デバイスのデータ・シート，およびSEボードとデバイス・ファイルのユーザーズ・マニュアルもご用意しています。

資料名		資料番号	
		和文	英文
17Kシリーズ／DTS標準品　セレクション・ガイド		U10317J	U10317E
RA17K　ユーザーズ・マニュアル		U10305J	U10305E
IE-17K/IE17K-ET CLICE/CLICE-ET ユーザーズ・マニュアル		U10063J	U10063E
SIMPLEHOST® *emlc-17K*®/RA17K対応版 ユーザーズ・マニュアル	入門編	U10445J	U10445E
	レファレンス編	U10496J	U10496E
17Kシリーズ　プロジェクト・マネージャ　ユーザーズ・マニュアル	レファレンス編	U12810J	EEU-1527
MAKE/CNV17K　ユーザーズ・マニュアル		U10596J	U10596E
emlC-17K　ユーザーズ・マニュアル	レファレンス編	U12829J	U12829E
LK17K　ユーザーズ・マニュアル		U12518J	U12518E
DOC17K　ユーザーズ・マニュアル		EEU-5006	EEU-1536

目　　次

第１章　概　　説 … 19

1.1　μPD170××シリーズの内部構成 … 20

第２章　プログラム・メモリ（ROM） … 21

2.1　プログラム・メモリの構成 … 21
2.2　プログラム・メモリの機能 … 22
2.3　プログラムの流れ … 22
2.4　プログラムの分岐 … 23
2.4.1　直接分岐 … 23
2.4.2　間接分岐 … 23
2.4.3　ディバグ時の注意 … 23
2.5　サブルーチン … 25
2.5.1　直接サブルーチン・コール … 25
2.5.2　間接サブルーチン・コール … 25
2.6　システム・コール … 28
2.7　テーブル参照 … 30
2.8　分岐命令とサブルーチン・コール命令の使用時の注意 … 30

第３章　プログラム・カウンタ（PC） … 31

3.1　プログラム・カウンタの構成 … 31
3.2　プログラム・カウンタの機能 … 31
3.2.1　分岐命令（BR）実行時 … 31
3.2.2　サブルーチン・コール命令（CALL）およびサブルーチン・リターン命令（RET, RETSK）実行時 … 32
3.2.3　テーブル参照命令（MOVT）実行時 … 32
3.2.4　割り込み受け付け時および割り込みリターン命令（RETI）実行時 … 32
3.2.5　スキップ命令実行時 … 32
3.2.6　リセット時 … 33
3.2.7　システム・コール時 … 33
3.3　セグメント・レジスタ（SGR） … 34
3.4　プログラム・カウンタ使用時の注意 … 34

第４章　アドレス・スタック（ASK） … 35

4.1　アドレス・スタックの構成 … 35
4.2　アドレス・スタックの機能 … 37
4.3　スタック・ポインタ（SP） … 37
4.3.1　スタック・ポインタの構成 … 37
4.3.2　スタック・ポインタの動作 … 38
4.4　アドレス・スタック・レジスタ … 40
4.5　サブルーチン命令，テーブル参照命令，および割り込み実行時のアドレス・スタック動作 … 41
4.5.1　サブルーチン・コール命令（CALL）とリターン命令（RET, RETSK） … 41
4.5.2　テーブル参照命令（MOVT DBF, @AR） … 43

4.5.3　システム・コール命令（SYSCAL）とリターン命令（RET, RETSK）　…　44
4.5.4　割り込み受け付け時とリターン命令（RETI）　…　45
4.6　アドレス・スタックのネスティング・レベルとPUSH命令およびPOP命令　…　47

第5章　データ・メモリ（RAM）　…　49

5.1　データ・メモリの構成　…　49
5.2　データ・メモリのアドレス指定時の注意　…　51

第6章　システム・レジスタ（SYSREG）　…　53

6.1　システム・レジスタの構成　…　53
6.2　システム・レジスタの機能　…　55
6.2.1　各レジスタの機能　…　55
6.2.2　システム・レジスタ操作命令　…　55
6.3　アドレス・レジスタ（AR）　…　56
6.3.1　アドレス・レジスタの構成　…　56
6.3.2　アドレス・レジスタの機能　…　56
6.3.3　テーブル参照命令（MOVT DBF, @AR）　…　56
6.3.4　スタック操作命令（PUSH AR, POP AR）　…　57
6.3.5　間接分岐命令（BR @AR）　…　57
6.3.6　間接サブルーチン・コール命令（CALL @AR）　…　57
6.3.7　アドレス・レジスタとデータ・バッファ　…　59
6.4　ウインドウ・レジスタ（WR）　…　60
6.4.1　ウインドウ・レジスタの構成　…　60
6.4.2　ウインドウ・レジスタの機能　…　60
6.4.3　PEEK WR, rf命令　…　60
6.4.4　POKE rf, WR命令　…　60
6.5　バンク・レジスタ（BANK）　…　62
6.5.1　バンク・レジスタの構成　…　62
6.5.2　バンク・レジスタの機能　…　62
6.6　インデクス・レジスタ（IX）とデータ・メモリ・ロウ・アドレス・ポインタ（MP：メモリ・ポインタ）　…　65
6.6.1　インデクス・レジスタとデータ・メモリ・ロウ・アドレス・ポインタの構成　…　65
6.6.2　インデクス・レジスタとデータ・メモリ・ロウ・アドレス・ポインタの機能　…　66
6.6.3　MPE = 0，IXE = 0のとき（データ・メモリ修飾なし）　…　68
6.6.4　MPE = 1，IXE = 0のとき（ななめ間接転送）　…　70
6.6.5　MPE = 0，IXE = 1のとき（データ・メモリ・アドレス・インデクス修飾）　…　72
6.6.6　MPE = 1，IXE = 1のとき　…　77
6.7　ジェネラル・レジスタ・ポインタ（RP）　…　79
6.7.1　ジェネラル・レジスタ・ポインタの構成　…　79
6.7.2　ジェネラル・レジスタ・ポインタの機能　…　79
6.7.3　ジェネラル・レジスタ・ポインタ使用時の注意　…　79
6.8　プログラム・ステータス・ワード（PSWORD）　…　81
6.8.1　プログラム・ステータス・ワードの構成　…　81
6.8.2　プログラム・ステータス・ワードの機能　…　82
6.8.3　インデクス・イネーブル・フラグ（IXE）　…　83
6.8.4　ゼロ・フラグ（Z）とコンペア・フラグ（CMP）　…　83
6.8.5　キャリー・フラグ（CY）　…　84
6.8.6　バイナリ・コーデッド・デシマル・フラグ（BCD）　…　84
6.8.7　算術演算時の注意　…　84

6.9　システム・レジスタ使用時の注意　…　85
6.9.1　システム・レジスタの予約語　…　85
6.9.2　“０”に固定されているシステム・レジスタの取り扱い　…　87

第７章　ジェネラル・レジスタ（GR）　…　89

7.1　ジェネラル・レジスタの構成　…　89
7.2　ジェネラル・レジスタの機能　…　91
7.3　ジェネラル・レジスタ使用時の注意　…　91
7.3.1　ジェネラル・レジスタのアドレス指定　…　91
7.3.2　ジェネラル・レジスタのロウ・アドレスについて　…　91
7.3.3　イミーディエト・データとの演算　…　93
7.4　各命令におけるジェネラル・レジスタとデータ・メモリのアドレス指定と動作　…　94

第８章　演算論理ユニット（ALU）　…　101

8.1　ALUブロックの構成　…　101
8.2　ALUブロックの機能　…　101
8.2.1　ALUの機能　…　101
8.2.2　一時記憶レジスタAおよびBの機能　…　106
8.2.3　ステータス・フリップフロップの機能　…　106
8.2.4　２進４ビット演算　…　107
8.2.5　BCD演算　…　107
8.2.6　ALUブロック処理手順　…　109
8.3　算術演算（２進４ビット加減算およびBCD加減算）　…　110
8.3.1　CMPフラグ＝０，BCDフラグ＝０のときの加減算　…　111
8.3.2　CMPフラグ＝１，BCDフラグ＝０のときの加減算　…　113
8.3.3　CMPフラグ＝０，BCDフラグ＝１のときの加減算　…　115
8.3.4　CMPフラグ＝１，BCDフラグ＝１のときの加減算　…　119
8.3.5　算術演算使用時の注意　…　119
8.4　論理演算　…　120
8.4.1　論理和（Logical OR）　…　121
8.4.2　論理積（Logical AND）　…　122
8.4.3　排他的論理和（Logical Exclusive OR）　…　123
8.5　ビット判断　…　124
8.5.1　Trueビット（１）判断　…　125
8.5.2　Falseビット（０）判断　…　125
8.6　比較判断　…　126
8.6.1　“等しい”の判断　…　127
8.6.2　“等しくない”の判断　…　127
8.6.3　“以上”の判断　…　128
8.6.4　“未満”の判断　…　128
8.7　回転処理　…　129
8.7.1　右回転処理　…　129
8.7.2　左回転処理　…　130

第９章　レジスタ・ファイル（RF）　…　131

9.1　レジスタ・ファイルの構成　…　131
9.2　レジスタ・ファイルの機能　…　133
9.2.1　レジスタ・ファイルの機能　…　133
9.2.2　レジスタ・ファイルの操作命令　…　133

９.３　コントロール・レジスタ　…　135
９.３.１　コントロール・レジスタの構成　…　135
９.３.２　コントロール・レジスタの各周辺ハードウエア制御機能　…　135
９.４　レジスタ・ファイル使用時の注意　…　136
９.４.１　コントロール・レジスタ（読み出し専用および未使用レジスタ）操作時の注意　…　136
９.４.２　レジスタ・ファイルのシンボル定義と予約語　…　137
９.４.３　アセンブラ（RA17K）組み込みマクロ命令使用時の注意　…　138

第10章　データ・バッファ（DBF）　…　139

10.１　データ・バッファの構成　…　139
10.２　データ・バッファの機能　…　141
10.３　データ・バッファ使用時の注意　…　142
10.３.１　書き込み専用，読み出し専用のレジスタおよび未使用アドレスの操作時について　…　142
10.３.２　周辺レジスタのアドレス指定について　…　142
10.４　データ・バッファとテーブル参照　…　143
10.４.１　テーブル参照動作　…　143
10.４.２　テーブル参照プログラム例　…　144
10.５　データ・バッファと周辺ハードウエア　…　148
10.５.１　周辺ハードウエアの制御方法　…　148
10.５.２　周辺レジスタとのデータ転送時のデータ長の取り扱い　…　149

第11章　汎用ポート　…　151

11.１　汎用ポートの構成　…　151
11.２　汎用ポートの機能　…　153
11.２.１　汎用ポート・データ・レジスタ（ポート・レジスタ）　…　153

第12章　割り込み機能　…　155

12.１　割り込みブロックの構成　…　155
12.２　割り込みの機能　…　157
12.２.１　周辺ハードウエア　…　157
12.２.２　割り込み要求処理ブロック　…　157
12.２.３　インタラプト・リクエスト・フラグ（IRQ×××）の構成と機能　…　158
12.２.４　インタラプト・パーミッション・フラグ（IP×××）の構成と機能　…　159
12.２.５　スタック・ポインタ，アドレス・スタック・レジスタとプログラム・カウンタ　…　160
12.２.６　割り込みイネーブル・フリップフロップ（INTE）　…　160
12.２.７　ベクタ・アドレス・ジェネレータ（VAG）　…　160
12.２.８　割り込みスタック　…　161
12.３　割り込み受け付け動作　…　163
12.３.１　割り込み受け付け動作と優先順位　…　163
12.３.２　割り込み受け付け時のタイミング・チャート　…　165
12.４　割り込み受け付け後の動作　…　169
12.５　割り込み処理ルーチン　…　170
12.５.１　退避処理　…　171
12.５.２　復帰処理　…　171
12.５.３　割り込み処理ルーチンでの注意　…　173

12.6　多重割り込み　…　174
12.6.1　割り込み要因の優先順位　…　174
12.6.2　割り込み用スタックによる割り込みレベルの制限　…　175

第13章　スタンバイ機能　…　179

13.1　スタンバイ・ブロックの構成　…　179
13.2　スタンバイ機能　…　180
13.3　CE端子によるデバイス動作モード　…　180
13.3.1　内蔵する周辺ハードウエアの動作制御　…　180
13.3.2　クロック・ストップ命令の有効，無効の制御　…　180
13.3.3　デバイスのリセット　…　180
13.3.4　CE端子への信号入力　…　181
13.4　ホールト機能　…　181
13.4.1　ホールト状態　…　181
13.4.2　ホールト解除条件　…　182
13.4.3　キー入力によるホールト解除　…　183
13.4.4　タイマ・キャリー（ベーシック・タイマ0キャリー）によるホールト解除　…　183
13.4.5　割り込みによるホールト解除　…　183
13.4.6　複数の解除条件が同時に設定された場合　…　185
13.5　クロック・ストップ機能　…　187
13.5.1　クロック・ストップ状態　…　187
13.5.2　クロック・ストップ状態の解除　…　187
13.5.3　CE端子がハイ・レベルの場合にクロック・ストップ命令によって起こる不具合と対策　…　189

第14章　リセット機能　…　191

14.1　リセット・ブロックの構成　…　191
14.2　リセットの機能　…　192
14.3　CEリセット　…　193
14.3.1　クロック・ストップ（STOPs命令）を使用しないときのCEリセット　…　193
14.3.2　クロック・ストップ（STOPs命令）を使用したときのCEリセット　…　194
14.3.3　CEリセット時の注意　…　195
14.4　パワーオン・リセット　…　198
14.4.1　通常動作時のパワーオン・リセット　…　199
14.4.2　クロック・ストップ状態中のパワーオン・リセット　…　199
14.4.3　電源電圧V_{DD}が0 Vから立ち上がったときのパワーオン・リセット　…　199
14.5　CEリセットとパワーオン・リセットの関係　…　201
14.5.1　V_{DD}端子とCE端子が同時に立ち上がったとき　…　201
14.5.2　パワーオン・リセットの強制ホールト中にCE端子が立ち上がったとき　…　201
14.5.3　パワーオン・リセット後にCE端子が立ち上がったとき　…　201
14.5.4　電源電圧V_{DD}立ち上げ時の注意　…　203
14.6　停電検出　…　205
14.6.1　停電検出回路　…　205
14.6.2　TMCYフラグによる停電検出時の注意　…　208
14.6.3　RAMジャッジ方法による停電検出　…　210
14.6.4　RAMジャッジによる停電検出時の注意　…　212

第15章　命令セット　…　213

15.1　命令セット概要　…　213

15.2　凡　　例　…　214
15.3　命令セット一覧　…　215
15.4　アセンブラ（RA17K）組み込みマクロ命令　…　217
15.5　命令の個別説明　…　218
15.5.1　加算命令　…　218
15.5.2　減算命令　…　231
15.5.3　論理演算命令　…　240
15.5.4　判断命令　…　245
15.5.5　比較命令　…　247
15.5.6　回転命令　…　250
15.5.7　転送命令　…　252
15.5.8　分岐命令　…　273
15.5.9　サブルーチン命令　…　278
15.5.10　割り込み命令　…　287
15.5.11　その他の命令　…　289

付録A　開発ツール　…　291

A.1　ハードウエア　…　291
A.2　ソフトウエア　…　292

付録B　マスクROMの発注方法　…　293

付録C　命令索引　…　295

C.1　命令索引（機能別）　…　295
C.2　命令索引（アルファベット順）　…　296

付録D　改版履歴　…　297

図の目次（1/3）

図番号	タイトル，ページ
2－1	プログラム・メモリ（ROM）の構成例 … 21
2－2	分岐命令の動作とマシン・コード例 … 24
2－3	サブルーチン・コール命令の動作例 … 26
2－4	サブルーチン・コール命令の使用例 … 27
2－5	システム・コール命令の使用例 … 28
3－1	プログラム・カウンタの構成 … 31
3－2	各命令におけるプログラム・カウンタの設定 … 33
4－1	スタックの構成 … 36
4－2	スタック・ポインタの構成 … 38
4－3	サブルーチン・コール時のスタック動作例 … 42
4－4	テーブル参照時のスタック動作例 … 43
4－5	割り込み時のスタック動作例 … 46
4－6	PUSH, POP命令によるスタック動作例 … 48
5－1	データ・メモリの構成 … 50
6－1	システム・レジスタのデータ・メモリ上の配置 … 53
6－2	システム・レジスタの構成 … 54
6－3	アドレス・レジスタの構成 … 56
6－4	アドレス・レジスタとデータ・バッファとの間のデータ転送 … 59
6－5	ウインドウ・レジスタの構成 … 60
6－6	PEEK, POKE命令の動作 … 61
6－7	バンク・レジスタの構成 … 62
6－8	データ・メモリのバンク指定 … 63
6－9	インデクス・レジスタとデータ・メモリ・ロウ・アドレス・ポインタの構成 … 65
6－10	MPE＝0，IXE＝0のときの動作例 … 69
6－11	MPE＝1，IXE＝0のときの動作例 … 71
6－12	MPE＝0，IXE＝1のときの動作例 … 73
6－13	MPE＝0，IXE＝1のときのジェネラル・レジスタ間接転送動作例 … 75
6－14	MPE＝0，IXE＝1のときの動作例（配列の処理） … 76
6－15	MPE＝1，IXE＝1のときのジェネラル・レジスタ間接転送動作例 … 78
6－16	ジェネラル・レジスタ・ポインタの構成 … 79
6－17	ジェネラル・レジスタの構成 … 80
6－18	プログラム・ステータス・ワードの構成 … 81
6－19	プログラム・ステータス・ワードの機能概要 … 82
7－1	ジェネラル・レジスタの構成 … 90
7－2	ジェネラル・レジスタのロウ・アドレス指定例 … 92

図の目次（2/3）

図番号	タイトル，ページ
7－3	ジェネラル・レジスタとデータ・メモリのアドレス指定例 … 94
7－4	データ・メモリとジェネラル・レジスタの演算例1 … 95
7－5	データ・メモリとジェネラル・レジスタの演算例2 … 96
7－6	ジェネラル・レジスタへの転送例 … 97
7－7	ジェネラル・レジスタ間接転送例 … 98
7－8	ジェネラル・レジスタのロウ・アドレス変更例 … 100
8－1	ALUブロックの構成 … 102
9－1	レジスタ・ファイルとデータ・メモリの関係 … 132
9－2	レジスタ・ファイルの構成 … 132
9－3	PEEK, POKE命令によるレジスタ・ファイルのアクセス例 … 134
10－1	データ・バッファの配置 … 139
10－2	データ・バッファの構成 … 140
10－3	データ・バッファと周辺ハードウエアおよびテーブル参照との関係（例） … 141
10－4	テーブル参照動作例 … 143
10－5	データ・バッファと周辺ハードウエアとの間のデータ転送例1 … 149
10－6	データ・バッファと周辺ハードウエアとの間のデータ転送例2 … 150
11－1	汎用ポートのブロック図 … 152
11－2	ポート・レジスタと各端子の関係 … 153
12－1	割り込みブロックの構成例 … 156
12－2	インタラプト・リクエスト・フラグの構成例 … 158
12－3	インタラプト・パーミッション・フラグの構成例 … 159
12－4	割り込みスタックの構成例 … 161
12－5	割り込みスタックの動作例（最大スタック・レベル = 2の場合） … 162
12－6	割り込み受け付け動作のフロー・チャート … 164
12－7	割り込み受け付けタイミング・チャート … 166
12－8	割り込み処理ルーチン内の退避と復帰 … 172
12－9	“ PEEK ”および“ POKE ”命令を用いたときのシステム・レジスタやコントロール・レジスタの退避動作 … 173
12－10	多重割り込み例 … 174
12－11	多重割り込み時の割り込みスタック … 176
12－12	最大スタック・レベル以上の多重割り込み使用例（割り込みスタックが2レベルのとき） … 177
12－13	17Kシリーズのエミュレータ（IE-17K, IE-17K-ET）使用時の最大スタック・レベル以上の割り込みスタック動作 … 178

図の目次（3/3）

図番号	タイトル，ページ
13－1	スタンバイ・ブロックの構成例 … 179
13－2	ホールト解除条件 … 182
13－3	CEリセットによるクロック・ストップの解除 … 188
13－4	パワーオン・リセット時のクロック・ストップ解除 … 188
13－5	CE端子入力によって起こるクロック・ストップ命令の誤動作と対策 … 190
14－1	リセット・ブロックの構成例 … 191
14－2	クロック・ストップを使用しないときのCEリセット動作 … 193
14－3	クロック・ストップを使用するときのCEリセット動作 … 194
14－4	パワーオン・リセットの動作 … 198
14－5	パワーオン・リセットと電源電圧V_{DD} … 200
14－6	パワーオン・リセットとCEリセットの関係 … 202
14－7	V_{DD}立ち上げ時の注意 … 203
14－8	クロック・ストップ状態からの復帰 … 204
14－9	停電検出フロー・チャート … 205
14－10	TMCYフラグの状態遷移 … 206
14－11	TMCYフラグの動作 … 207
14－12	V_{DD}とデータ・メモリの破壊 … 212

表の目次

表番号	タイトル，ページ
４－１	スタック・ポインタ（SP）の動作 … 39
４－２	サブルーチン・コール命令とリターン命令の動作 … 41
４－３	テーブル参照命令の動作 … 43
４－４	システム・コール時の動作 … 44
４－５	割り込み受け付け時とリターン命令の動作 … 45
４－６	PUSH命令およびPOP命令の動作 … 47
６－１	インデクス・レジスタとデータ・メモリ・ロウ・アドレス・ポインタによるデータ・メモリ・アドレスの修飾 … 67
６－２	コンペア・フラグ（CMP）の状態とゼロ・フラグ（Z）のセット，リセット条件 … 83
７－１	ジェネラル・レジスタとデータ・メモリ間の操作命令 … 94
８－１	ALU処理命令一覧 … 104
８－２	２進４ビット演算結果とBCD演算結果 … 108
８－３	算術演算の種類 … 110
８－４	論理演算 … 121
８－５	論理演算の真理値表 … 121
８－６	ビット判断命令 … 124
８－７	比較判断命令 … 126
14－１	内部リセット信号の各リセットの関係 … 192
14－２	停電検出回路とRAMジャッジによる停電検出の比較 … 210

第1章　概　　説

μPD170××はNECのAV，家電用の４ビット・コントローラ「17Kシリーズ」の中で，特にTV，AM/FMラジオ用に，選局機能を持ったことを特徴とするライン・アップです。

それぞれ選局に必要なPLL回路や，ボルテージ・シンセサイザの回路を持ち，すべての「17Kシリーズ」に共通なCPU（中央演算ユニット）も，これらの選局回路との接続や選局回路に関するレジスタの割り付けがμPD170××独特のものになっています。

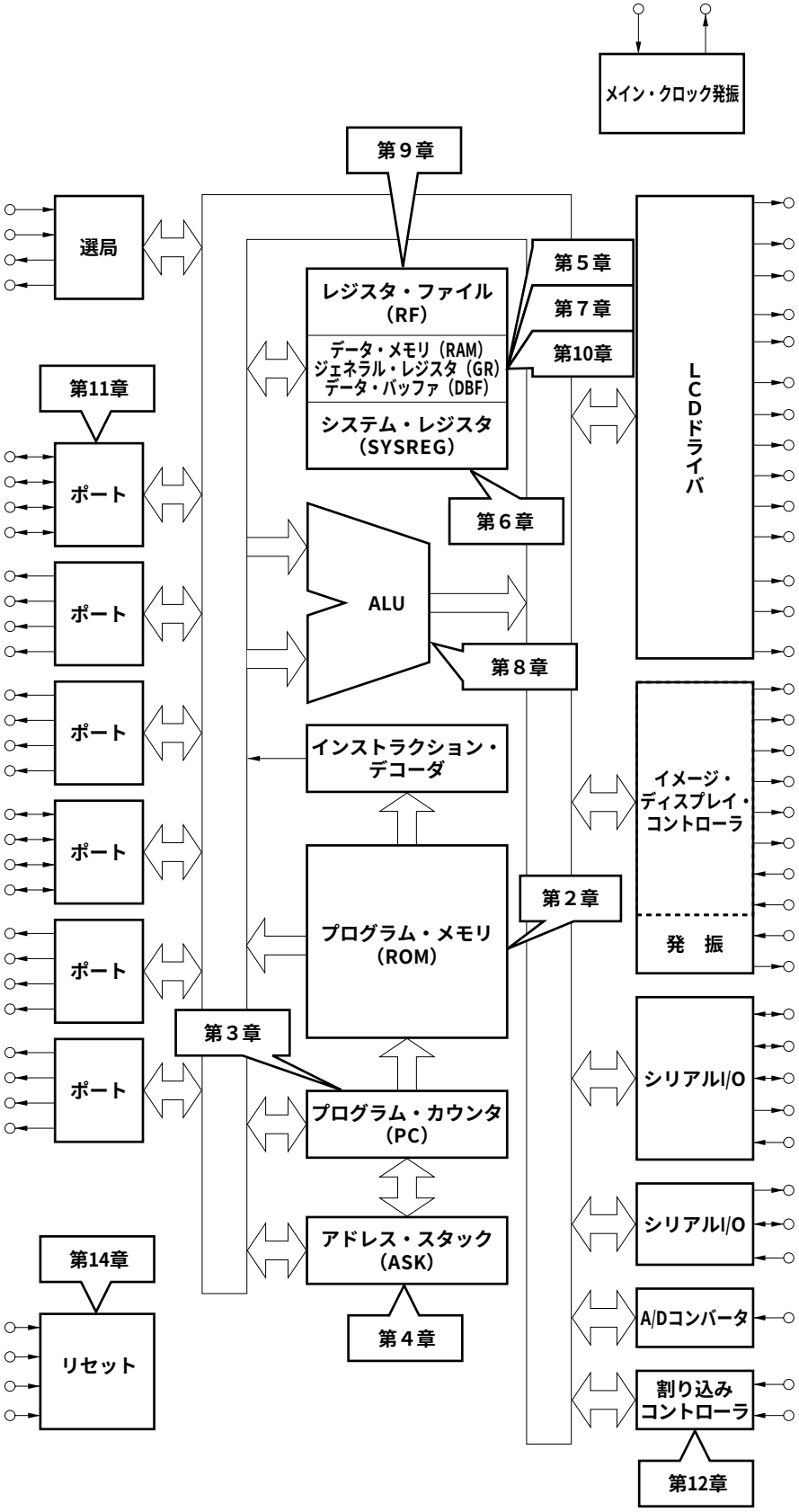
選局回路は，品種個々で異なります。このマニュアルは，μPD170××の共通事項に関するものですので，選局回路については個別データ・シートを読んでください。

そのほか，品種によってプリスケーラや，LCD（液晶表示板）駆動回路，タイマ，オン・スクリーン表示機能などを持ちますが，これらについても個別データ・シートを読んでください。

μPD17P0××は，それぞれμPD170××のワン・タイムPROM版です。システム開発時の評価や，小規模生産にご利用ください。

1.1　μPD170××シリーズの内部構成

次の図は，μPD170××シリーズの内部構成を類型化したもので，実在する品種のものではありません。



第２章　プログラム・メモリ（ROM）

プログラム・メモリは，CPU（中央演算ユニット）が実行する“プログラム”や，あらかじめ決まっている定数データを書き込んでおくところです。

μPD170××はマスクROM（リード・オンリ・メモリ）を，μPD17P0××はEPROM（電気的書き込み可能ROM）をそれぞれプログラム・メモリとして持っています。

２.１　プログラム・メモリの構成

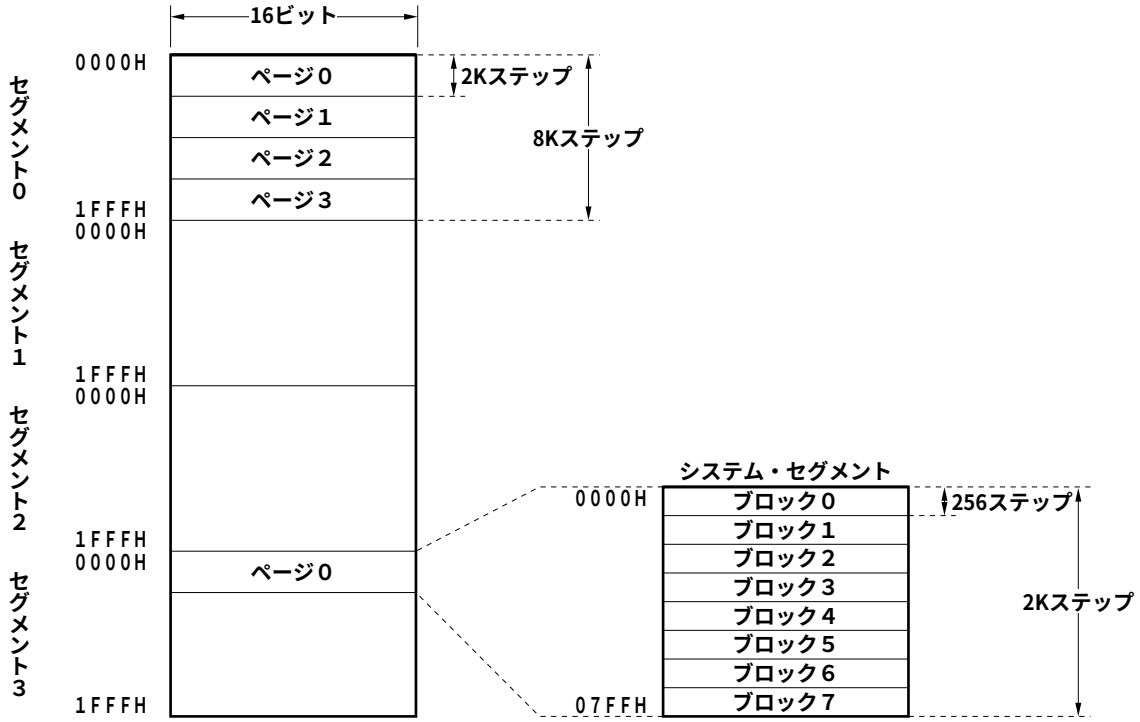
図２－１のように，プログラム・メモリ（ROM）は16ビットを１ステップとして，“番地（アドレス）”が付けられています。

そして，2Kステップごとに“ページ”と呼ぶ単位に分けられています。

さらに，４ページごとに“セグメント”に分けられています。セグメントあたり0000H-1FFFH番地まであります。ただし，セグメントによっては8Kステップよりも少ないものもあります。

セグメントの中のあるものは“システム・セグメント”と呼ばれ，256ステップごとに“ブロック”に分けられています。

図２－１　プログラム・メモリ（ROM）の構成例



２.２　プログラム・メモリの機能

プログラム・メモリには，大別して次の３つの機能があります。

（１）プログラムを格納しておく
（２）定数データを格納しておく
（３）周辺機能のためのデータを格納しておく

プログラムとは，CPU（Central Processing Unit：実際にマイクロコントローラを制御する）を動作させる“命令”の集まりであり，プログラムに書かれた“命令”に従い順次CPUが処理を実行していきます。すなわち，CPUはプログラム・メモリに格納されているプログラムから順次“命令”を読み出していき，各“命令”に従って処理を実行します。

“命令”はすべて16ビット長の“１語命令”であるため，プログラム・メモリの１つの番地に１つの命令を格納することができます。

定数データとは，あらかじめ決まっているデータです。定数データは，専用命令であるMOVT命令を使用することによりデータ・メモリ（RAM）上のデータ・バッファ（DBF）にプログラム・メモリの内容を読み込むことができます。このようにプログラム・メモリ上の定数データを読み込むことを“テーブル参照”と呼びます。

プログラム・メモリは読み出し専用のメモリであるため“命令”により書き換えることはできません。したがってプログラム・メモリには通常ROM（Read Only Memory）を使います。

２.３　プログラムの流れ

プログラムの流れは，プログラム・メモリの番地を指定するプログラム・カウンタ（PC）によって制御されます。

プログラム・メモリに格納されているプログラムは，通常0000H番地から１番地ごとに実行されます。しかし，条件によって異なるプログラムを実行させるような場合はプログラムの流れを分岐する必要があります。このような場合は，分岐命令（BR）を使用してプログラム・カウンタの内容を変更します。

また，同一のプログラムを何度も実行させたい場合は，そのたびに同一プログラムを用いているとプログラム・メモリの効率が低下します。このような場合はそのプログラムを汎用化してつくっておき，専用命令であるCALL命令によりこのプログラムを呼び出すことにより何度も同一のプログラムを実行させることができます。このプログラムのことを“サブルーチン”と呼びます。サブルーチンに対して通常実行しているプログラムを“メイン・ルーチン”と呼びます。

また，プログラムの流れとはまったく関係なく，ある現象が起きたときに実行させたいプログラムがあるときは，割り込み機能を使用します。割り込み機能を使うと，指定の現象によって割り込み要求が発生したとき，現在のプログラムの流れとは関係なくあらかじめ決められた番地（ベクタ・アドレスと呼ぶ）へ分岐させることができます。

２.４　プログラムの分岐

プログラムの分岐は分岐命令（BR）で行います。

分岐命令（BR）には，命令のオペランドで指定したプログラム・メモリのアドレス（addr）に直接分岐する直接分岐命令（BR addr）と，後述するアドレス・レジスタ（AR）の内容で指定したプログラム・メモリのアドレスへ分岐する間接分岐命令（BR @AR）の２種類があります。

第３章　プログラム・カウンタ（PC）も参照してください。

２.４.１　直接分岐

直接分岐命令では命令のオペレーション・コードの下位２ビットと命令のオペランド11ビットの合計13ビットで分岐先のプログラム・メモリ・アドレスを指定します。したがって直接分岐命令における分岐先アドレスの範囲は同一のセグメント内全体です（最大0000H-1FFFH番地）。直接分岐命令では，セグメント間を越える分岐はできません。

２.４.２　間接分岐

間接分岐命令は，図２－２の（２）のようにアドレス・レジスタにプログラムで分岐先のアドレスを設定しますから，アドレス・レジスタのビット数によって分岐先の指定範囲が異なります。

なお，間接分岐命令はセグメント間を越える分岐もできます。

６.３　アドレス・レジスタ（AR）を参照してください。

２.４.３　ディバグ時の注意

図２－２の（１）のように，直接分岐命令は，分岐先のページによってオペレーション・コードが異なります。

たとえば，ページ０への直接分岐命令のオペレーション・コードは“0CH”でありページ１内へのオペレーション・コードは“0DH”，ページ２内へのオペレーション・コードは“0EH，ページ３内へのオペレーション・コードは“0FH”です。

これは，直接分岐命令のオペランド“addr”が11ビットしかないのでオペレーション・コードの下位２ビットを分岐先のアドレスとして使用しているためです。

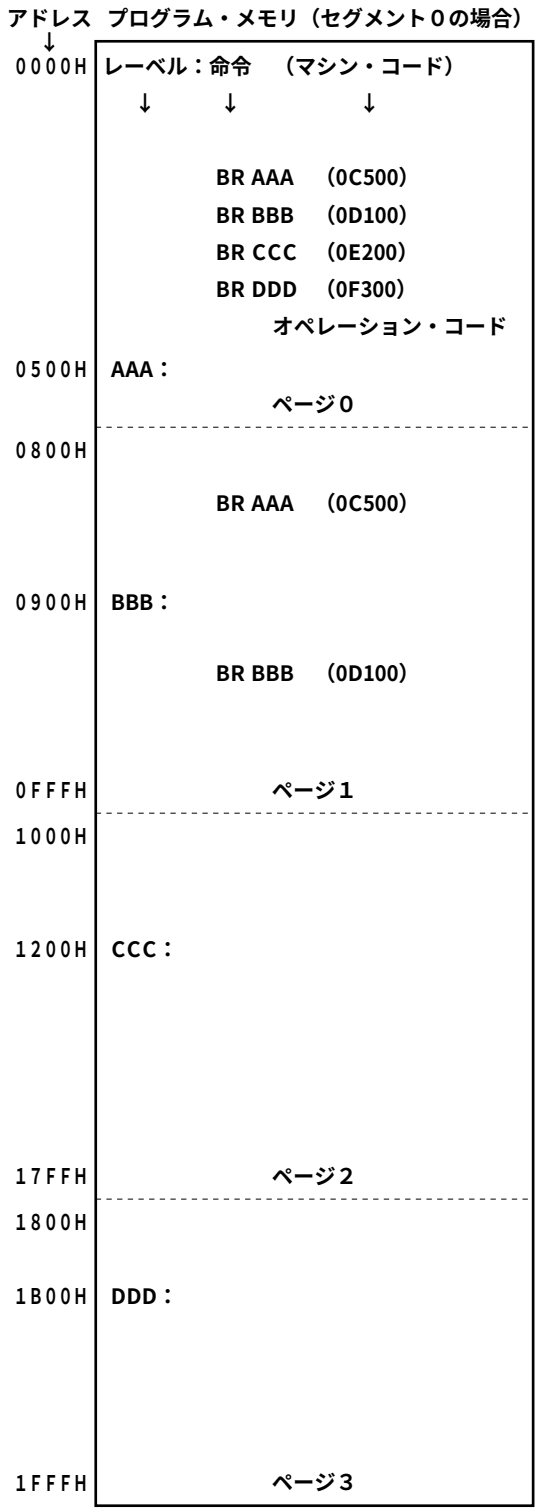
このオペレーション・コードは，アセンブルを17Kシリーズのアセンブラ（RA17K）ですれば，アセンブラがレーベルで指定されたジャンプ先を参照して自動的に変換します。

ディバグ時にパッチ修正を行う場合には，プログラマ自身が分岐するページを判断し，オペレーション・コードを変換しなければなりません。

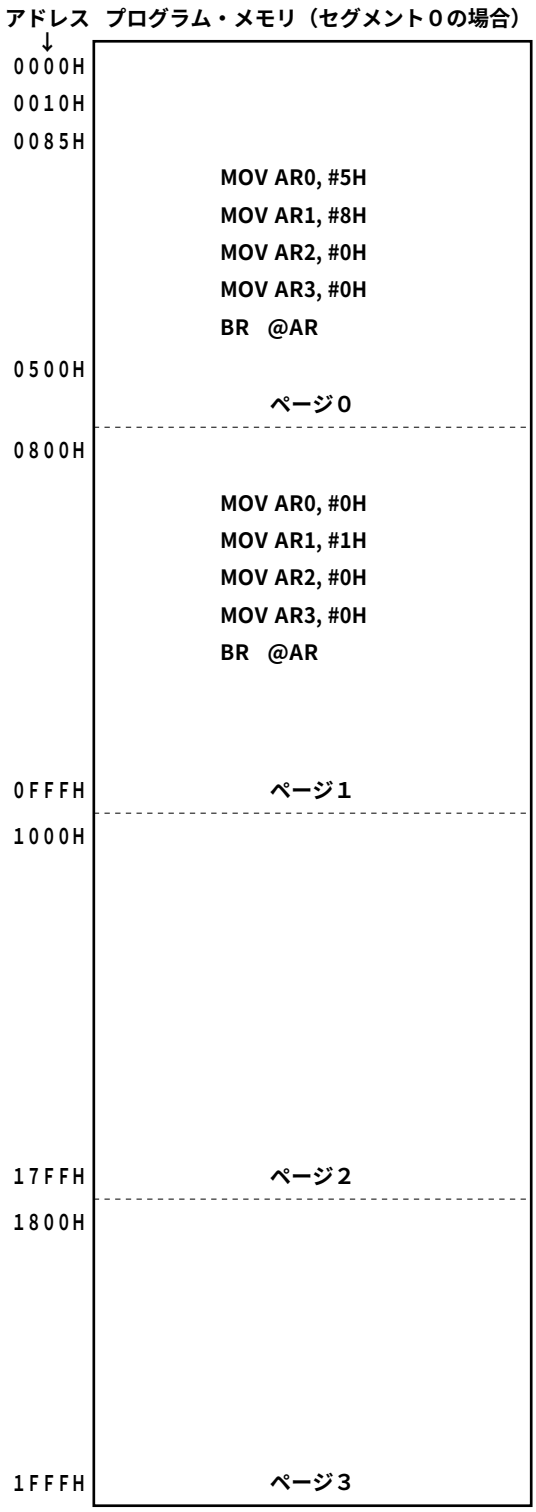
たとえば，図２－２の（１）でBBBのアドレス0900H番地をアドレス0910Hにパッチ修正する場合は，“BR BBB”命令のマシン・コードを“0D110”と入力します。

図２－２　分岐命令の動作とマシン・コード例

（１）直接分岐（BR addr）



（２）間接分岐（BR @AR）



２.５　サブルーチン

サブルーチンへの分岐には専用命令であるサブルーチン・コール命令（CALL）とサブルーチン・リターン命令（RET, RETSK）を使います。

サブルーチン・コール命令には，命令のオペランド（addr）で指定するアドレスに分岐する直接サブルーチン・コール命令（CALL addr）とアドレス・レジスタに設定した値で指定するアドレスに分岐する間接サブルーチン・コール命令（CALL @AR）とがあります。

また，システム・セグメントに分岐するシステム・コール命令（SYSCAL entry）があります。

サブルーチンからの復帰命令にはRET命令とRETSK命令を使用します。RETおよびRETSK命令を実行することによりサブルーチン・コール命令（CALL）を実行した次のプログラム・メモリ・アドレスへ復帰します。このときRETSK命令は復帰した最初の命令をノー・オペレーション命令（NOP）として実行します。

第３章　プログラム・カウンタ（PC）も参照してください。

２.５.１　直接サブルーチン・コール

直接サブルーチン・コール命令は，命令のオペランド11ビットで分岐先のプログラム・メモリ・アドレスを指定します。したがって，直接サブルーチン・コール命令を使用する場合は，図２－３の（１）のようにその呼び出し番地つまりサブルーチンの先頭番地はページ０内（0000H-07FFH番地）に置く必要があります。ページ０以外（0800H-1FFFH番地）に先頭番地のあるサブルーチンは呼び出すことができません。

しかし，直接サブルーチン・コール命令やサブルーチン・リターン命令（RET, RETSK）はページ０以外にあってもかまいません。

直接サブルーチン・コール命令では，セグメント間を越えるサブルーチン・コールはできません。

例１．サブルーチンからのリターン命令がページ１にある場合

図２－４に示すように，サブルーチンの先頭番地がページ０内にあれば，戻り番地，リターン命令ともにどのページがあってもかまいません。

サブルーチンの先頭番地がページ０にある限り，CALL命令はページの概念なく使用することができます。しかしプログラムの作成上，サブルーチンの先頭番地をページ０内に置くことができない場合は，例２のようにしてください。

２．サブルーチンの先頭番地がページ１にある場合

図２－４に示すように，ページ０内に分岐命令（BR）を設け，このBR命令を介して，実際のサブルーチン（SUB1）を呼び出します。

２.５.２　間接サブルーチン・コール

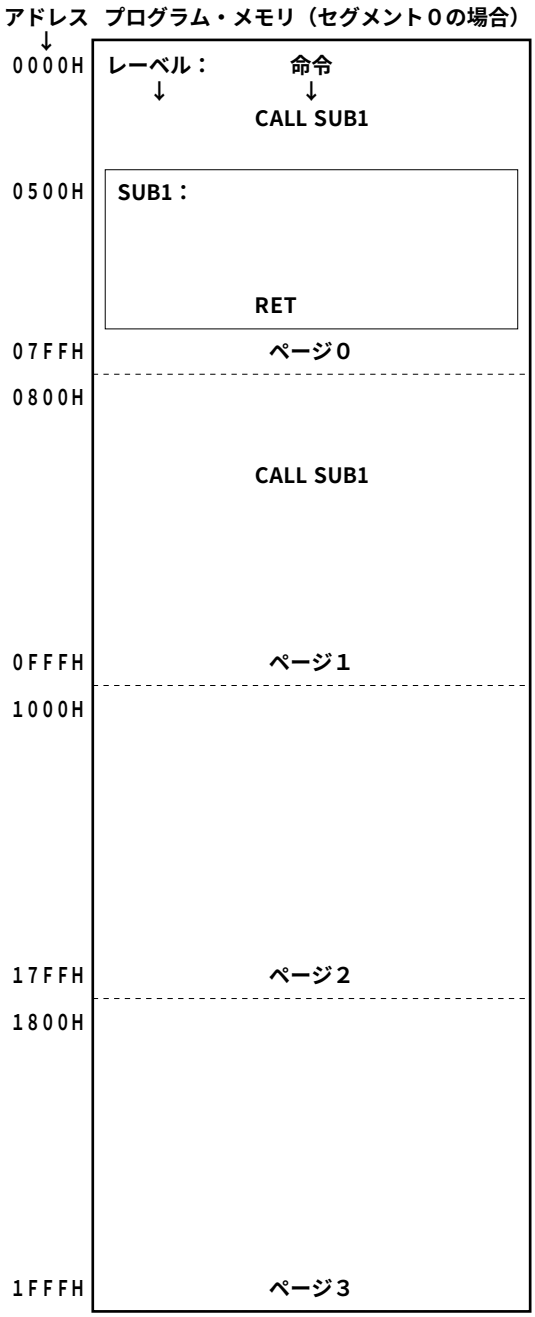
間接サブルーチン・コール命令（CALL @AR）は，図２－３の（２）のようにアドレス・レジスタ（AR）に分岐先のアドレスを指定します。したがって，この命令で分岐できるプログラム・メモリ・アドレスの範囲はアドレス・レジスタのビット数で品種ごとに決まります。

６.３　アドレス・レジスタ（AR）を参照してください。

なお，間接サブルーチン・コール命令は，セグメント間を越えるサブルーチン・コールが可能です。

図２－３　サブルーチン・コール命令の動作例

（1）直接サブルーチン・コール（CALL addr）



（2）間接サブルーチン・コール（CALL @AR）

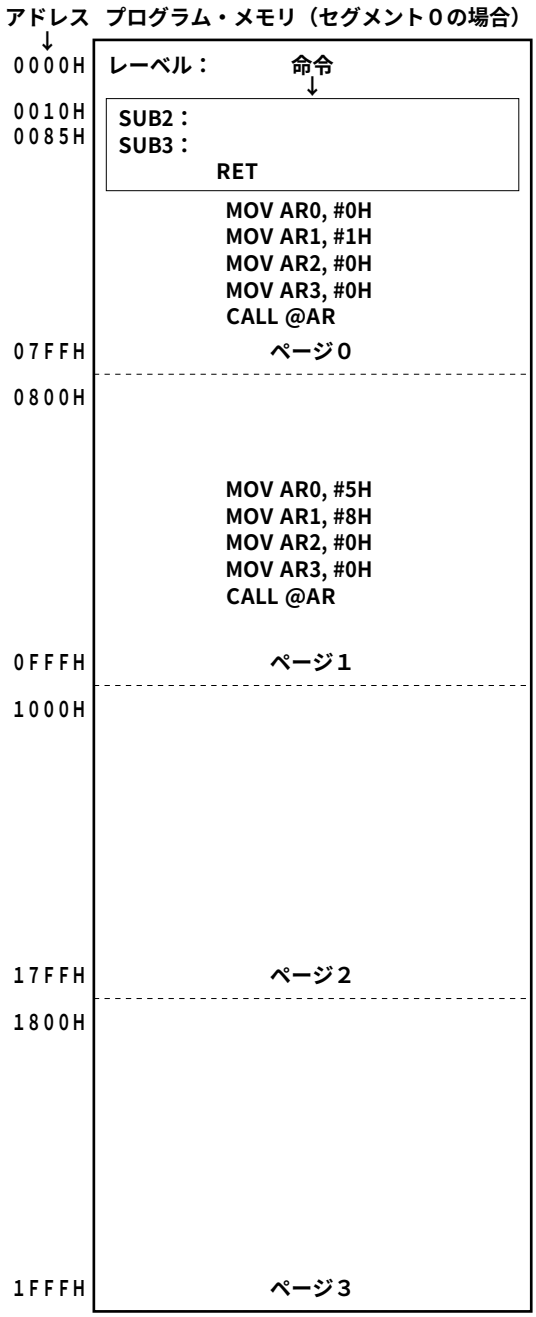
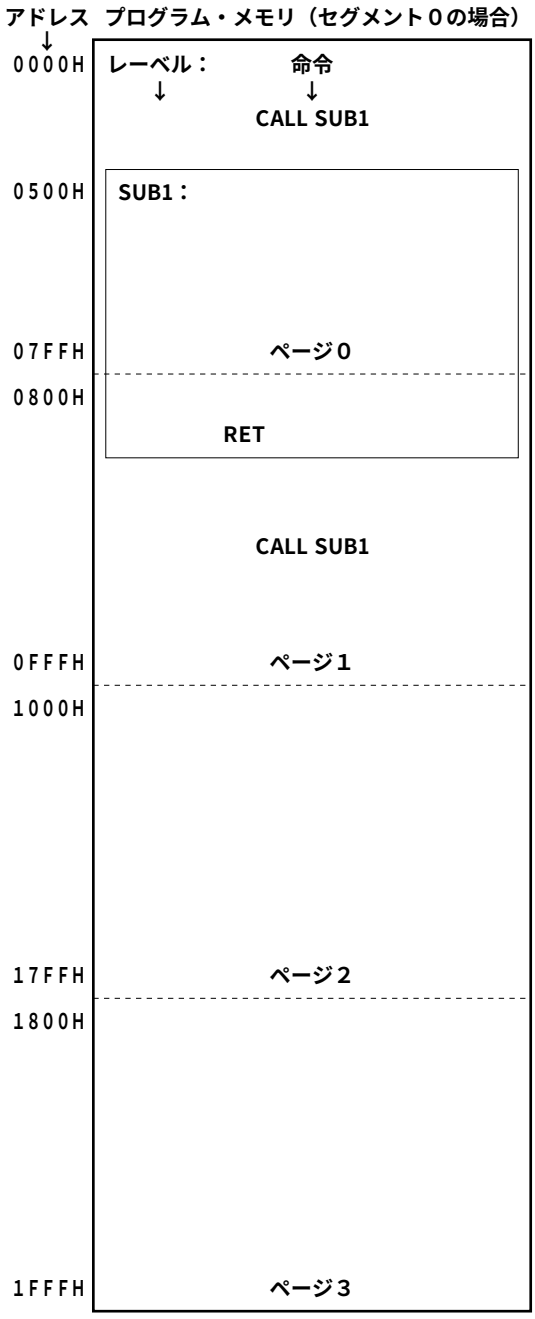
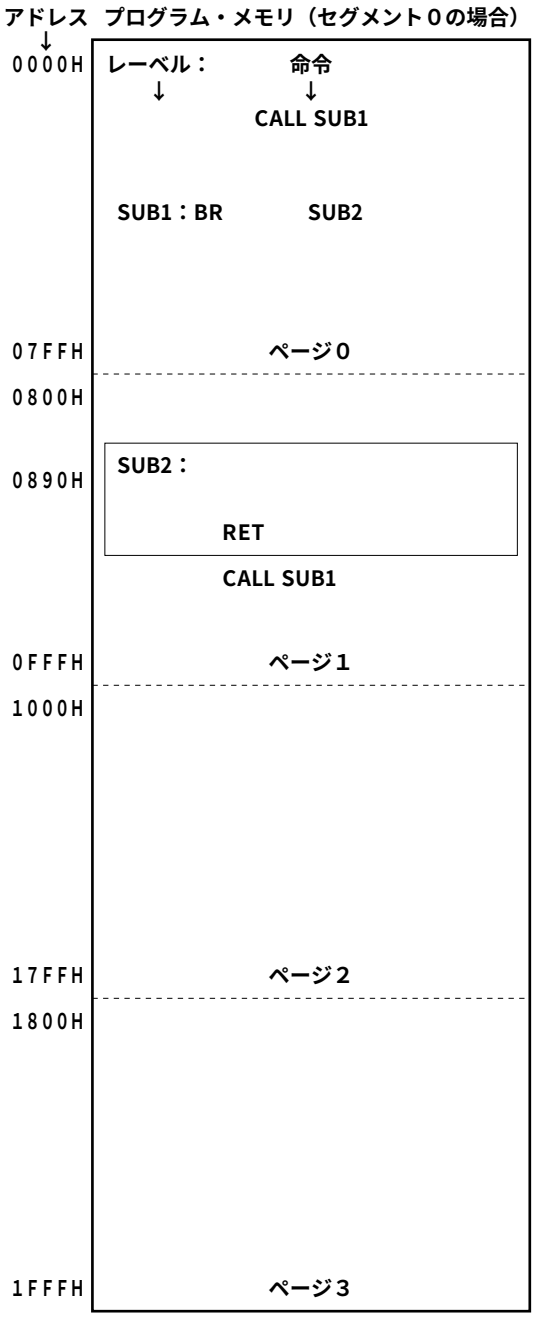


図２－４　サブルーチン・コール命令の使用例

（1）サブルーチン･リターン命令がページ１にあるとき



（2）サブルーチンの先頭番地がページ１にあるとき



2.6　システム・コール

システム・コール命令（SYSCAL entry）は，各セグメントからシステム・セグメント内のサブルーチンに１命令で分岐できます。

システム・コール命令は，命令のオペランド“ entry ”により，分岐先のプログラム・メモリ・アドレスを指定できます。命令のオペランド“ entry ”の７ビットのうち，上位３ビットがブロックを指定し，下位４ビットがアドレスを指定します。したがって，指定できる範囲はシステム・セグメントのページ０にある各ブロック（ブロック０-ブロック７）の先頭にある16ステップだけです。詳細は，**３.２.７　システム・コール時**を参照してください。

図２－５の　部分はシステム・コールが可能なプログラム・メモリ範囲です。SYSCAL命令やサブルーチン・リターン命令（RET, RETSK）はどこにあってもかまいません。

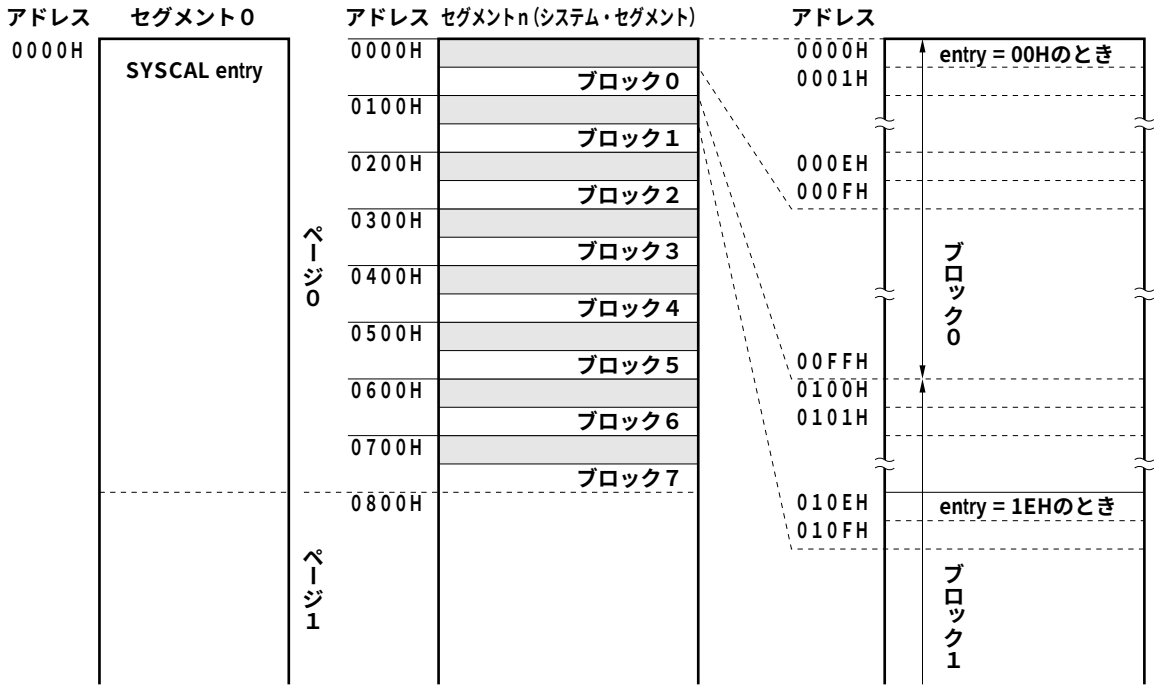
例１． SYSCAL命令がセグメント０内にある場合

図２－５に示すように，セグメント０内のSYSCAL命令を実行すると，オペランド“ entry ”で指定するシステム・セグメントに分岐します。

このとき，オペランド“ entry ”が00H（entry = 00H）であれば，ブロック０のアドレス00H（0000H番地）に分岐します。

また，オペランド“ entry ”が1EH（entry = 1EH）であれば，ブロック１のアドレス0EH（010EH番地）に分岐します。

図２－５　システム・コール命令の使用例



例２．システム・コール命令を使ったプログラム例

```
　；
　；モジュール１（セグメント０）
　：

EXTERN LAB ENTRY0
EXTERN LAB ENTRY1
EXTERN LAB ENTRY2
EXTERN LAB ENTRY3

SYSCAL .DL.（ENTRY0 SHR 4 AND 0070H）OR（ENTRY0 AND 000FH）
SYSCAL .DL.（ENTRY1 SHR 4 AND 0070H）OR（ENTRY1 AND 000FH）
SYSCAL .DL.（ENTRY2 SHR 4 AND 0070H）OR（ENTRY2 AND 000FH）
SYSCAL .DL.（ENTRY3 SHR 4 AND 0070H）OR（ENTRY3 AND 000FH）

；
；モジュール２（セグメント１）
：
CSEG 1
PUBLIC ENTRY0, ENTRY1, ENTRY2, ENTRY3

ORG 2000H

ENTRY0：
  BR SUB0
ENTRY1：
  BR SUB1

ORG 2100H

ENTRY2：
  BR SUB2
ENTRY3：
  BE SUB3
```

２.７　テーブル参照

テーブル参照は，プログラム・メモリ内の定数データを参照するときに使用します。MOVT DBF, @AR命令を実行すると，アドレス・レジスタで指定するプログラム・メモリ・アドレスの内容がデータ・バッファ（DBF）に格納されます。

プログラム・メモリの内容は16ビットで構成されているので，MOVT命令でデータ・バッファに格納される定数データは16ビット（４ワード）になります。また，MOVT命令により参照できるプログラム・メモリ・アドレスは各品種のアドレス・レジスタが指定できる範囲です。

テーブル参照を行うと，スタックが１レベル使用されます。

６.３　アドレス・レジスタ（AR）および**10.４　データ・バッファとテーブル参照**を参照してください。

２.８　分岐命令とサブルーチン・コール命令の使用時の注意

17Kシリーズのアセンブラ（RA17K）を使うとき，例１のように分岐命令（BR）とサブルーチン・コール命令（CALL）のオペランドに直接プログラム・メモリ・アドレス（数値によるアドレス）を指定すると，エラーが発生します。

これは，プログラム修正時などにおけるバグ要因を低減する目的でアセンブラに組み込まれています。

分岐命令（BR）とサブルーチン・コール命令（CALL）のオペランドにはレーベルを使用してください。

例１．エラーとなる場合

```
;①
        BR        005H      ;アセンブラでエラーになる。
;②
        CALL      00F0H     ;
```

２．エラーとならない場合

```
;③
    LOOP1：                 ;プログラムの中にレーベルを用い，そのレーベルに対してBR
        BR        LOOP1     ;命令やCALL命令を行う。
;④
    SUB1：                  ;
        CALL      SUB1      ;
;⑤
    LOOP2 LAB     0005H     ;0005Hをレーベル型としてLOOP2に割り当てる。
        BR        LOOP2     ;
;⑥
        BR. LD.   0005H     ;オペランドの数値をレーベル型に変換する。
                            ;ただし，この方法はバグ要因を低減するためにはあまり使用するべき
                            ;ではない。
```

詳しくはRA17Kアセンブラ　ユーザーズ・マニュアル（U10305J）を参照してください。

第３章　プログラム・カウンタ（PC）

プログラム・カウンタは，プログラム・メモリの番地を指定します。

３.１　プログラム・カウンタの構成

プログラム・カウンタは，図３－１のように13ビットのバイナリ・カウンタと最大３ビットのセグメント・レジスタ（SGR）で構成されています。

図３－１　プログラム・カウンタの構成

MSB　　LSB

SGR2	SGR1	SGR0	PC_{12}	PC_{11}	PC_{10}	PC_9	PC_8	PC_7	PC_6	PC_5	PC_4	PC_3	PC_2	PC_1	PC_0

３.２　プログラム・カウンタの機能

プログラム・カウンタはプログラム・メモリに書き込まれている複数の命令または定数データから，実際に実行する命令または使用する定数データの番地を指定します。

通常は命令をフェッチするごとに１ずつインクリメントされます。また，分岐命令（BR），サブルーチン・コール命令（CALL），リターン命令（RET, RETSK, RETI），テーブル参照命令（MOVT）を実行したとき，および割り込みを受け付けたときにはそれぞれ指定の値を格納し，その番地の命令を実行します。

３.２.１-３.２.７に各命令実行時のプログラム・カウンタの動作を示します。

３.２.１　分岐命令（BR）実行時

分岐命令には，分岐先を命令のオペランドで直接指定する直接分岐命令（BR addr）と，アドレス・レジスタARの内容で指定する間接分岐命令（BR @AR）とがあります。

直接分岐命令（BR addr）は，図３－２のように命令のオペランド（下位11ビット）で指定する値をそのままプログラム・カウンタに書き込みます。さらに命令のオペレーション・コード（上位５ビット）の中の下位２ビットがプログラム・カウンタのビット12（b_{12}）およびビット11（b_{11}）として付け加わるので，分岐できる範囲は計13ビットで指定できる同じセグメント内（0000H-1FFFH番地）のプログラム・メモリです。

命令のオペレーション・コードはアセンブラ（RA17K）が自動的に分岐先の所在箇所を探して決定します。

間接分岐命令は（BR @AR）は，図３－２のようにシステム・レジスタ内のアドレス・レジスタARの内容をプログラム・カウンタに書き込んで分岐先を指定します。アドレス・レジスタの下位13ビットはプログラム・カウンタに書き込まれ，それより上位のビット（最大３ビット）はセグメント・レジスタに書き込まれます。したがって，分岐できる範囲はアドレス・レジスタのビット数によって品種ごとに決まります。

３.２.２　サブルーチン・コール命令（CALL）およびサブルーチン・リターン命令（RET, RETSK）実行時

サブルーチン・コール命令には，分岐先を命令のオペランドで直接指定する直接サブルーチン・コール命令（CALL addr）と，アドレス・レジスタ（AR）の内容で指定する間接サブルーチン命令（CALL @AR）とがあります。

直接サブルーチン・コール命令（CALL addr）は，図３－２のように命令のオペランドの値をそのままプログラム・カウンタに書き込みます。オペランドは11ビットなので，プログラム・カウンタの上位２ビット（b_{11}, b_{12}）は０に固定されて，直接サブルーチン・コール命令で分岐できる範囲はプログラム・メモリ上でページ０内の0000H-07FFH番地です。

間接サブルーチン・コール命令（CALL @AR）は，図３－２のように，システム・レジスタ内のアドレス・レジスタの内容をプログラム・カウンタに書き込んで分岐先を指定します。したがって，間接サブルーチン・コールで分岐できる範囲はアドレス・レジスタのビット数によって品種ごとに決まります。

サブルーチン・リターン命令（RET, RETSK）は，スタック・ポインタSPが指定するアドレス・スタック・レジスタASRの内容，すなわち戻り番地をプログラム・カウンタに書き込みます。

アドレス・スタックについては，**第４章　アドレス・スタック（ASK）**を参照してください。

３.２.３　テーブル参照命令（MOVT）実行時

テーブル参照命令（MOVT DBF, @AR）は，図３－２のようにプログラム・カウンタにアドレス・レジスタの内容を書き込み，それで指定するプログラム・メモリの内容をデータ・バッファに読み出して参照します。したがって，テーブル参照ができる範囲はアドレス・レジスタのビット数によって品種ごとに決まります。

プログラム・メモリの内容をデータ・バッファに読み出したあと，テーブル参照命令の次の命令の番地をプログラム・カウンタに書き込み，続きのプログラムを実行します。テーブル参照命令は，この戻り番地の保持のためにスタックを１レベル使いますので，サブルーチンや，割り込み中に使うときは許容スタック・レベルを越えないように注意してください。

テーブル参照命令は，１命令の実行に２インストラクション・サイクルかかります。

３.２.４　割り込み受け付け時および割り込みリターン命令（RETI）実行時

割り込みを受け付けると，図３－２のように各割り込みごとに固有のベクタ・アドレス（分岐先番地）をプログラム・カウンタに書き込みます。

割り込み処理後，割り込みリターン命令（RETI）を実行すると，スタック・ポインタが指定するアドレス・スタックに書き込んである戻り番地をプログラム・カウンタに復帰します。

第12章　割り込み機能も参照してください。

３.２.５　スキップ命令実行時

スキップ命令（SKT, SKF, SKEなど）実行時は，スキップ条件の内容にかかわらずスキップ命令の次の番地がプログラム・カウンタに格納されます。サブルーチン・リターン・スキップ命令（RETSK）実行時もスタック・ポインタで指定するアドレス・スタック・レジスタ（ASR）の内容をプログラム・カウンタに格納します。

このとき，実行した命令がスキップすべき条件であれば（RETSKは常時）スキップ命令の次の命令はノー・オペレーション命令（NOP）として実行します。したがって，スキップ命令を実行したとき次の命令をスキップしなくても実行する命令数は同じになります。

３.２.６　リセット時

パワーオン・リセット（V_{DD} = ロウ・レベル→ハイ・レベル）時およびCEリセット（CE = ロウ・レベル→ハイ・レベル）時は，プログラム・カウンタの内容は0000Hにリセットされ，プログラム・カウンタ内のセグメント・レジスタ（SGR）の内容は０にリセットされます。したがって，このときプログラムはセグメント０の０番地から実行します。

また，クロック・ストップ命令（STOP s命令）実行時は，この命令の番地でプログラムがストップしますが，クロック・ストップ命令の解除（CE = ロウ・レベル→ハイ・レベルなど）後はセグメント０の０番地からプログラムを実行します。

第14章　リセット機能も参照してください。

３.２.７　システム・コール時

システム・コール（SYSCAL entry）実行時は，命令のオペランド“ entry ”をプログラム・カウンタに格納します。オペランドは７ビットなので，プログラム・カウンタ内のセグメント・レジスタ（SGR）にはシステム・セグメントのセグメント値，残りの６ビットには“ ０ ”を書き込んで，システム・セグメントのアドレスを指定します。

図３－２　各命令におけるプログラム・カウンタの設定

命令＼プログラム・カウンタ		プログラム・カウンタ（PC）の内容															
		SGR2	SGR1	SGR0	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
BR addr	ページ０	変化しない			0	0	命令のオペランド（addr）										
	ページ１				0	1											
	ページ２				1	0											
	ページ３				1	1											
CALL addr					0	0	命令のオペランド（addr）										
BR　@AR CALL @AR MOVT DBF, @AR		アドレス・レジスタの内容（ARのb_{13}-b_0）															
RET RETSK RETI		スタック・ポインタ（SP）で指定される アドレス・スタック・レジスタ（ASR）の内容 （戻り番地）															
割り込み受け付け時		0	0	0	各種割り込みのベクタ・アドレス												
パワーオン・リセット，CEリセット		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SYSCAL entry		システム・セグメントの値			0	0	命令のオペランド			0	0	0	0	命令のオペランド			

３.３　セグメント・レジスタ（SGR）

セグメント・レジスタは，プログラム・メモリのセグメントを指定するレジスタです。

また，図３－２のSGR0-SGR2は，各命令実行時のセグメント・レジスタの動作です。

セグメント・レジスタがセットされるのは“ SYSCAL ”命令を実行したとき，および間接分岐，間接サブルーチン・コールをしたときです。

パワーオン・リセット時およびCEリセット時にはセグメント・レジスタは０にリセットされます。

３.４　プログラム・カウンタ使用時の注意

プログラム・カウンタは，命令を１つフェッチするごとに＋１されます。

そのため，プログラム・メモリの最終番地（1FFFH番地）の命令が分岐命令（BR）や復帰命令（RET, RETSK, RETI）以外の命令であると，プログラム・カウンタが次に指定する番地は0000Hになって動作不良を起こす場合があります。

そのほかに，プログラム・メモリが少なく，たとえば最終番地が0FEFH番地でしかもその内容が分岐命令（BR）や復帰命令（RET, RETSK, RETI）以外の命令であるとプログラム・カウンタが次に指定する番地は0FF0Hとなって動作不良を起こします。

したがって，各セグメントの最終番地には，分岐命令（BR）を書き込んでください。ただし，復帰命令である場合は，そのままでかまいません。

なお，プログラム・カウンタの値が許容範囲外になるような命令を実行すると，アセンブラ（RA17K）はエラーを発生し，警告します。

第４章　アドレス・スタック（ASK）

アドレス・スタックは，サブルーチン・コール時や割り込み受け付け時およびテーブル参照命令実行時にプログラムの戻り番地を待避するためのレジスタです。

スタックには，このほかに割り込み受け付け時にシステム・レジスタの内容を待避する割り込みスタックがあります。割り込みスタックについては**12.2.8　割り込みスタック**を参照してください。

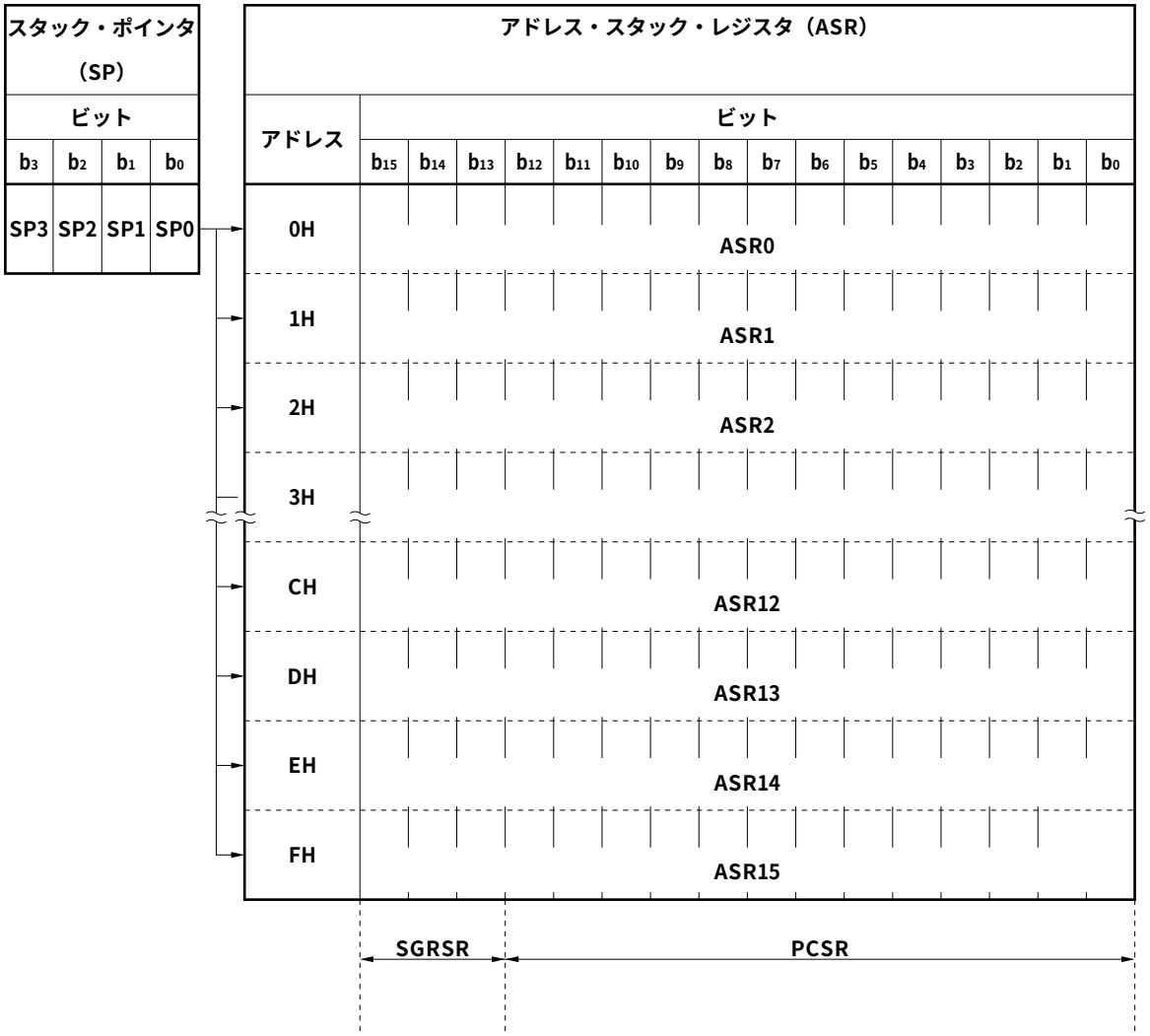
４.１　アドレス・スタックの構成

図４－１のようにアドレス・スタックは，スタック・ポインタとアドレス・スタック・レジスタとで構成されています。スタック・ポインタは４ビットのバイナリ・カウンタで構成されています。

アドレス・スタック・レジスタ（ASR）は，最大16ビット×最大16個のレジスタASR0-ASRn（n≦15）で構成されています。

図４－１のように各アドレス・スタック・レジスタの下位13ビットはプログラム・カウンタ・スタック（PCSR）で，それより上位の最大３ビットはセグメント・レジスタ・スタック（SGRSR）です。

図４－１　スタックの構成


スタック・ポインタ（SP）
ビット
b3 b2 b1 b0
SP3 SP2 SP1 SP0
アドレス・スタック・レジスタ（ASR）
アドレス
ビット
b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0
0H
ASR0
1H
ASR1
2H
ASR2
3H
CH
ASR12
DH
ASR13
EH
ASR14
FH
ASR15
SGRSR
PCSR

４.２　アドレス・スタックの機能

アドレス・スタックは，サブルーチン・コール時や割り込み受け付け時およびテーブル参照命令実行時にサブルーチンおよび割り込みルーチンからの戻り番地を退避するレジスタです。

サブルーチン・コール命令（CALL addr, CALL @AR）を実行すると，アドレス・スタック・レジスタASR0-ASRn（n≦15）にサブルーチン・コール命令の次のプログラム・メモリ番地，すなわち戻り番地を退避します。サブルーチン実行後，サブルーチン・リターン命令（RET, RETSK）を実行すると，戻り番地をアドレス・スタック・レジスタからプログラム・カウンタに復帰します。

そのほか，テーブル参照命令（MOVT DEF, @AR）実行時も，アドレス・スタックを使います。

アドレス・スタックは，スタック操作命令（POP AR, PUSH AR）でも操作できます。

４.３-**４.４**にスタック・ポインタ，アドレス・スタック・レジスタの機能を示します。

４.３　スタック・ポインタ（SP）

スタック・ポインタは，最大16個のアドレス・スタック・レジスタASR0-ASRn（n≦15）のうち，どのアドレス・スタック・レジスタを使うのかを指定するためのレジスタです。

４.３.１　スタック・ポインタの構成

図４－２のように，スタック・ポインタは最大４ビットのフラグで構成されるレジスタです。

スタック・ポインタはレジスタ・ファイル内のコントロール・レジスタにあります。コントロール・レジスタについては**第９章　レジスタ・ファイル（RF）**を参照してください。

図４－２　スタック・ポインタの構成

名　　称	フラグ記号				アドレス	Read/Write
	b_3	b_2	b_1	b_0		
スタック・ポインタ SP	SP3	SP2	SP1	SP0	品種により異なる	R/W

				アドレス・スタック・レジスタ（ASR）のアドレスを指定
0	0	0	0	アドレス０（ASR0）
0	0	0	1	アドレス１（ASR1）
0	0	1	0	アドレス２（ASR2）
0	0	1	1	アドレス３（ASR3）
0	1	0	0	アドレス４（ASR4）
0	1	0	1	アドレス５（ASR5）
0	1	1	0	アドレス６（ASR6）
0	1	1	1	アドレス７（ASR7）
1	0	0	0	アドレス８（ASR8）
1	0	0	1	アドレス９（ASR9）
1	0	1	0	アドレス10（ASR10）
1	0	1	1	アドレス11（ASR11）
1	1	0	0	アドレス12（ASR12）
1	1	0	1	アドレス13（ASR13）
1	1	1	0	アドレス14（ASR14）
1	1	1	1	アドレス15（ASR15）

4.3.2　スタック・ポインタの動作

スタック・ポインタは表４－１に示すようにサブルーチン・コール命令（CALL addr, CALL@AR），システム・コール命令（SYSCAL），テーブル参照命令（MOVT DBF, @AR）の第１命令サイクル，スタック操作命令（PUSH AR）実行時および割り込み受け付け時に－１されます。また，サブルーチン・リターン命令（RET, RETSK），テーブル参照命令（MOVT DBF, @AR）の第２命令サイクル，スタック操作命令（POP AR）および割り込みリターン命令（RETI）実行時に＋１されます。

表４－１　スタック・ポインタ（SP）の動作

命　　令	スタック・ポインタの値
CALL addr CALL @AR SYSCAL entry MOVT DBF, @AR PUSH AR 割り込み受け付け時	SP－1
RET RETSK MOVT DBF, @AR POP AR RETI	SP＋1

各命令実行時の具体的動作については**４.５　サブルーチン命令，テーブル参照命令，および割り込み実行時のアドレス・スタック動作**を参照してください。

スタック・ポインタは，最大４ビットのバイナリ・カウンタですから，とりうる値は0H-nH（n≦F）の最大16とおりになります。しかし，品種によってはスタック・レジスタの数がスタック・ポインタの指定範囲よりも少ないものがあり，その場合スタック・ポインタがスタック・レジスタ外を指定すると動作不良を起こします。**４.６　アドレス・スタックのネスティング・レベルとPUSH命令およびPOP命令**を参照してください。

また，スタック・ポインタはレジスタ・ファイル上にあるので，PEEK命令およびPOKE命令により直接値を書き込んだり読み出したりできます。このときスタック・ポインタの値は変わりますが，アドレス・スタック・レジスタの値は変化しません。

４.４　アドレス・スタック・レジスタ

アドレス・スタック・レジスタは，サブルーチン・コール命令実行時やテーブル参照命令実行時に戻り番地を退避するために使用します。また，割り込み受け付け時には，プログラムの戻り番地およびプログラム・ステータス・ワード（PSWORD）が自動的に退避されます。

アドレス・スタック・レジスタは，サブルーチン・コール命令（CALL addr, CALL @AR），システム・コール命令（SYSCAL entry），テーブル参照命令（MOVT DBF, @AR）の第１命令サイクル実行時および割り込み受け付け時に，プログラム・カウンタ（PC）の値に＋１した値，つまり戻り番地を格納します。またスタック操作命令（PUSH AR）実行時にはアドレス・レジスタ（AR）の内容が格納されます。データが格納されるアドレス・スタック・レジスタは，命令実行時のスタック・ポインタ（SP）の値に－１した値で指定されます。

サブルーチン・リターン命令（RET, RETSK），割り込みリターン命令（RETI）およびテーブル参照命令（MOVT DBF, @AR）の第２命令サイクル実行時は，スタック・ポインタで指定されるアドレス・スタック・レジスタの内容をプログラム・カウンタに復帰して，スタック・ポインタの値を＋１します。またスタック操作命令（POP AR）実行時は，スタック・ポインタで指定されたアドレス・スタック・レジスタの値をアドレス・レジスタに転送して，スタック・ポインタの値を＋１します。

各命令実行時の具体的動作については**４.５　サブルーチン命令，テーブル参照命令，および割り込み実行時のアドレス・スタック動作**を参照してください。

アドレス・スタック・レジスタはASR0-ASRn（n≦15）の最大16個です。各品種の最大スタック・レベルを越えるサブルーチン・コールや割り込みをすると動作不良を起こします。**４.６　アドレス・スタックのネスティング・レベルとPUSH命令およびPOP命令**および**12.６　多重割り込み**を参照してください。

４.５　サブルーチン命令，テーブル参照命令，および割り込み実行時のアドレス・スタック動作

４.５.１-４.５.４にそれぞれの場合のアドレス・スタック動作を示します。

４.５.１　サブルーチン・コール命令（CALL）とリターン命令（RET, RETSK）

表４－２にサブルーチン・コール命令とリターン命令実行時のスタック・ポインタ，アドレス・スタック・レジスタおよびプログラム・カウンタの動作を示します。

表４－２　サブルーチン・コール命令とリターン命令の動作

命　令	動　　作
CALL addr	①プログラム・カウンタ（PC）の値を＋１ ②スタック・ポインタ（SP）の値を－１ ③スタック・ポインタ（SP）で指定されるアドレス・スタック・レジスタ（ASR）にプログラム・カウンタ（PC）の値を退避 ④命令のオペランド（addr）で指定される値をプログラム・カウンタに転送
RET RETSK	①スタック・ポインタ（SP）で指定されるアドレス・スタック・レジスタ（ASR）の値をプログラム・カウンタ（PC）に復帰 ②スタック・ポインタ（SP）の値を＋１

RETSK命令実行時は復帰後の最初の命令はノー・オペレーション命令（NOP）になります。

図４－３の動作例ではメイン・ルーチンの100H番地に30H番地のサブルーチンを呼び出すCALL命令があり，35H番地に50H番地のサブルーチンを呼び出すCALL命令があることを示しています。

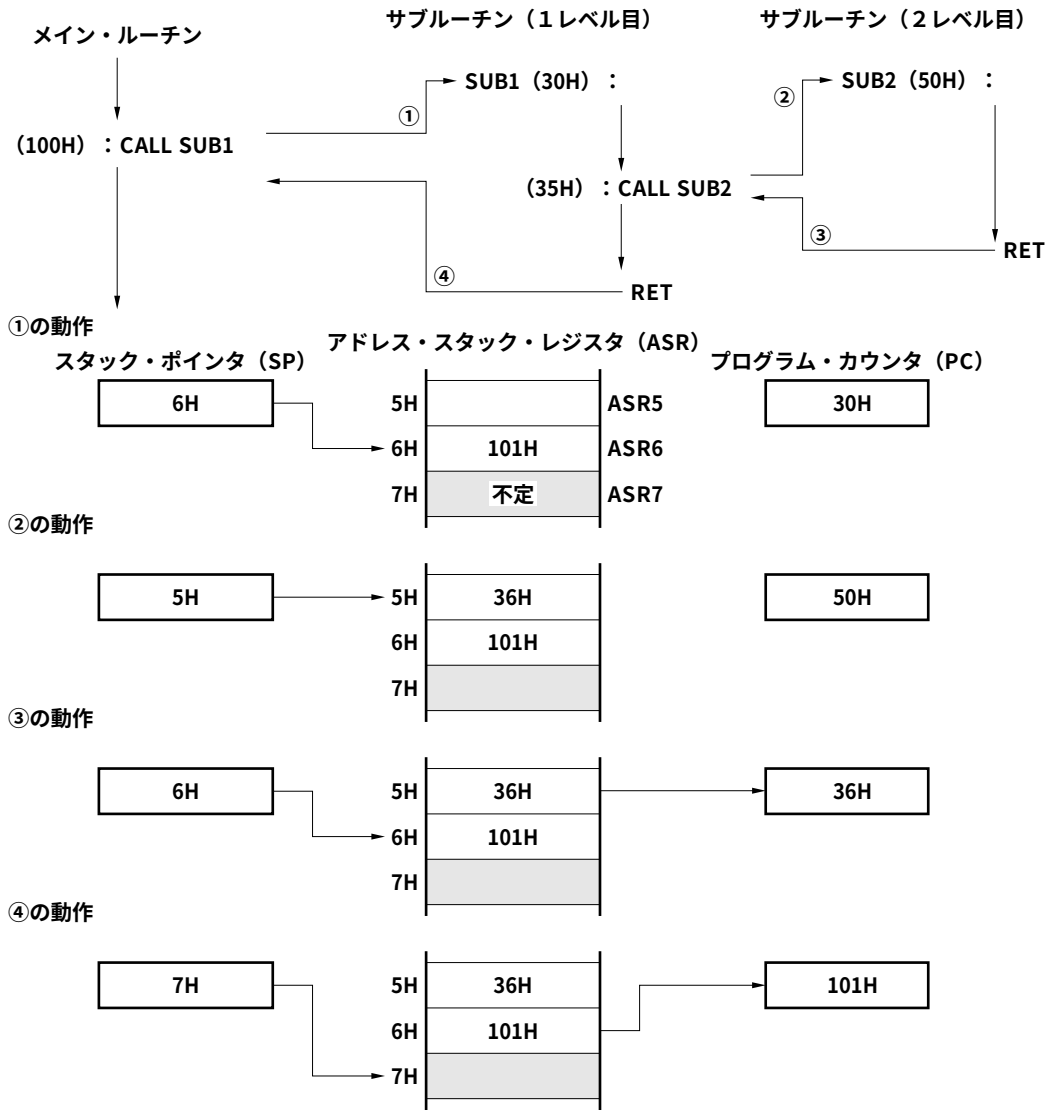
この30H番地から始まるサブルーチンを“１レベル目”，50H番地から始まるサブルーチンを“２レベル目”のサブルーチンと呼びます。また図中の矢印はプログラムの流れを示しています。

この例において100H番地の命令を実行する直前のスタック・ポインタの値が7Hであったと仮定すると，100H番地のCALL命令を実行することにより，まずプログラム・カウンタが101Hになり同時にスタック・ポインタの値が－１されて6Hになります。

次に，アドレス・スタック・レジスタの6H番地（ASR6）に１レベル目のサブルーチンからの戻り番地である101Hが退避され，プログラム・カウンタにはCALL命令のオペランドである30Hが転送されます。

次に35H番地のCALL命令が実行されると，スタック・ポインタの値は－１されて5Hになり，アドレス・スタック・レジスタの5H番地（ASR5）には２レベル目のサブルーチンからの戻り番地である36Hが退避され，プログラム・カウンタにはCALL命令のオペランドである50Hが転送されます。次に２レベル目のサブルーチン内でRET命令が実行されるとスタック・ポインタで指定されているアドレス・スタック・レジスタの5H番地（ASR5）の内容36Hがプログラム・カウンタに復帰され，スタック・ポインタの値は＋１されて6Hになります。続いて１レベル目のサブルーチンのRET命令が実行されるとプログラム・カウンタには，メインルーチンの戻り番地である101Hが復帰され，スタック・ポインタの値は＋１されて7Hになります。

図４－３　サブルーチン・コール時のスタック動作例



４.５.２　テーブル参照命令（MOVT DBF, @AR）

表４－３はテーブル参照命令実行時の動作を示します。

表４－３　テーブル参照命令の動作

命　令	命令サイクル	動　　作
MOVT DBF, @AR	第１	①プログラム・カウンタ（PC）の値を＋１ ②スタック・ポインタ（SP）の値を－１ ③スタック・ポインタ（SP）で指定されるアドレス・スタック・レジスタ（ASR）にプログラム・カウンタ（PC）の値を退避 ④アドレス・レジスタ（AR）の値をプログラム・カウンタ（PC）へ転送
	第２	⑤プログラム・カウンタ（PC）で指定されるプログラム・メモリ（ROM）の内容をデータ・バッファ（DBF）へ転送 ⑥スタック・ポインタ（SP）で指定されるアドレス・スタック・レジスタ（ASR）の値をプログラム・カウンタ（PC）へ復帰 ⑦スタック・ポインタ（SP）の値を＋１

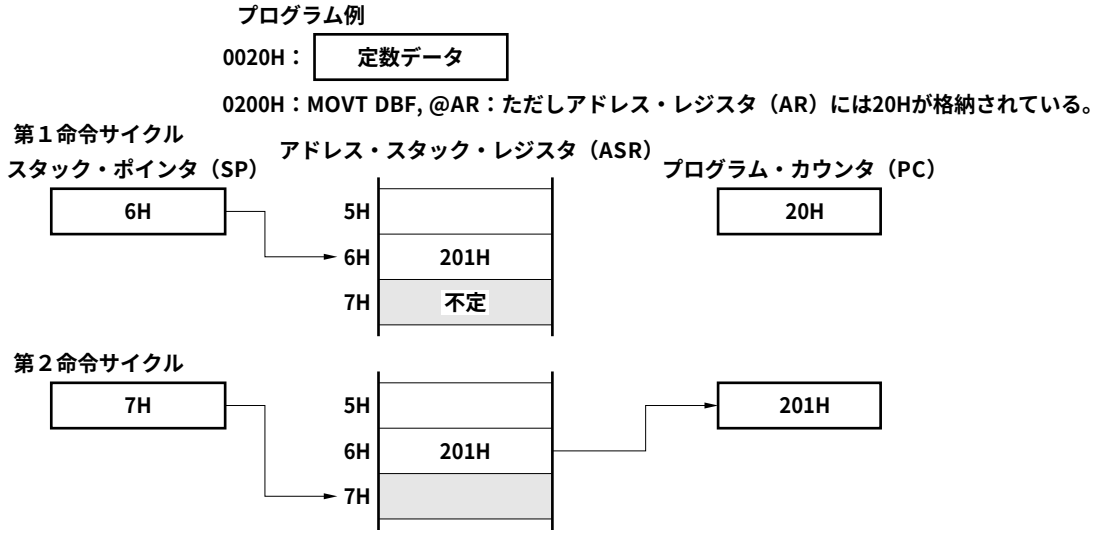
図４－４の動作例ではテーブル参照命令は200H番地にあり，参照する定数データが格納してあるプログラム・メモリ・アドレスが20Hであり，200H番地の“ MOVT DBF, @AR ”命令を実行する直前のスタック・ポインタの値が7Hであると仮定します。

200H番地の“ MOVT DBF, @AR ”命令を実行すると，第１命令サイクルでスタック・ポインタの値は－１されて6Hになり，アドレス・スタック・レジスタの6H番地には“ MOVT DBF, @AR ”命令の次のアドレスである201Hを退避し，プログラム・カウンタには定数データが格納してあるプログラム・アドレス20Hを転送します。

このアドレス20Hはアドレス・レジスタで指定します。

次に第２命令サイクルでプログラム・カウンタの内容である20H番地の定数データをデータ・バッファに転送し，プログラム・カウンタにはアドレス・スタック・レジスタの内容である201Hを復帰します。また，スタック・ポインタの値は＋１されて7Hになります。

図４－４　テーブル参照時のスタック動作例



４.５.３　システム・コール命令（SYSCAL）とリターン命令（RET, RETSK）

表４－４は，システム・コール命令とリターン命令実行時のスタック・ポインタ（SP），アドレス・スタック・レジスタ（ASR）およびプログラム・カウンタ（PC）の動作を示します。

表４－４　システム・コール時の動作

命　　令	動　　作
SYSCAL entry	①プログラム・カウンタ（PC）の値を＋１ ②スタック・ポインタ（SP）の値を－１ ③スタック・ポインタ（SP）で指定されるアドレス・スタック・レジスタ（ASR）に，プログラム・カウンタ（PC）およびセグメント・レジスタ（SGR）の値を退避。 ④セグメント・レジスタ（SGR）を１にセット ⑤命令のオペランド（entry）で指定される値を，プログラム・カウンタ（PC）のビットb_{10}-b_8およびb_3-b_0へ転送
RET RETSK	①スタック・ポインタ（SP）で設定されるアドレス・スタック・レジスタ（ASR）の値をプログラム・カウンタ（PC）およびセグメント・レジスタ（SGR）に復帰 ②スタック・ポインタ（SP）の値を＋１ ③RETSKのみ：復帰後の最初の命令は，ノー・オペレーション（NOP）命令として処理し，次の命令にすすみます（スキップ動作）。

システム・コール命令実行時のスタック動作は，セグメント・レジスタを１にセットすることを除けば通常のサブルーチン・コール動作と同等です。

ただし，サブルーチン・コール命令とは，コールされるプログラム・メモリ・アドレスの指定が異なるため注意してください。

４.５.４　割り込み受け付け時とリターン命令（RETI）

表４－５は割り込み受け付け時とリターン命令実行時のスタック動作を示します。

表４－５　割り込み受け付け時とリターン命令の動作

命　令	動　　作
割り込み受け付け時	①プログラム・カウンタ（PC）の値を＋１ ただし，割り込み受け付け時の命令が分岐命令（BR命令）およびサブルーチン・コール命令（CALL命令）であるときは，分岐またはコールするプログラム・メモリ（ROM）のアドレスになります。 ②スタック・ポインタ（SP）の値を－１ ③スタック・ポインタ（SP）で指定されるアドレス・スタック・レジスタにプログラム・カウンタ（PC）およびセグメント・レジスタ（SGR）の値を退避 ④割り込みスタック・レジスタへPSWORD（BCD, CMP, CY, Z, IXE）の各フラグとBANKの値を退避 ⑤ベクタ・アドレスをプログラム・カウンタ（PC）へ転送，セグメント・レジスタ（SGR）をリセット
RETI	①PSWORD（BCD, CMP, CY, Z, IXE）とBANK へ割り込みスタック・レジスタの値を復帰 ②スタック・ポインタ（SP）で指定されるアドレス・スタック・レジスタの値をプログラム・カウンタ（PC）およびセグメント・レジスタ（SGR）へ復帰 ③スタック・ポインタ（SP）の値を＋１

図４－５の動作例では，スタック・ポインタの値が7Hで，300H番地の命令を実行中に割り込みを受け付けたと仮定します。

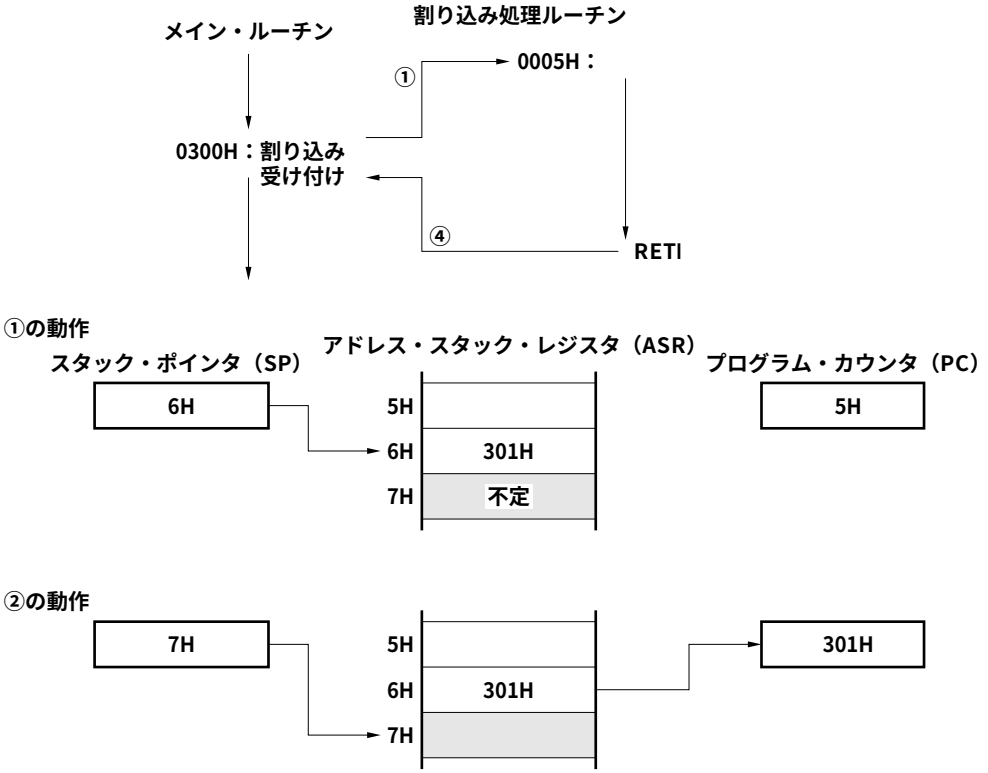
このとき，300H番地の命令実行後，スタック・ポインタの値は－１されて6Hになり，アドレス・スタック・レジスタの6H番地（ASR6）には次に実行するはずであった301Hを退避し，割り込みスタック・レジスタにはバンク・レジスタの下位２ビットとインデクス・イネーブル・フラグ１ビットを退避します。プログラム・カウンタにはINT_0端子の割り込みベクタ・アドレスである0005Hを転送し，0005H番地の命令を実行します。

次に割り込み処理ルーチン内でリターン命令（RETI）を実行すると，割り込みスタック・レジスタの内容をバンク・レジスタとインデクス・イネーブル・フラグに復帰します。また，プログラム・カウンタにはアドレス・スタック・レジスタの内容の301Hを復帰し，スタック・ポインタは＋１されて7Hになります。

割り込み動作については，**第12章　割り込み機能**の項を参照してください。

★　また割り込み要因とそれに対するベクタ・アドレスは各品種により異なります。各品種のデータ・シートを参照してください。

図４－５　割り込み時のスタック動作例



４.６　アドレス・スタックのネスティング・レベルとPUSH命令およびPOP命令

スタック・ポインタはサブルーチン・コール命令やリターン命令などで単純に＋１，－１される３ビットのバイナリ・カウンタとして動作しています。

したがってスタック・ポインタの値が0Hのときに CALL命令，SYSCAL命令，MOVT命令の実行および割り込み受け付けを行うとスタック・ポインタの値は－１されて7Hになり，アドレス・スタック・レジスタの7H番地であるASR7にサブルーチンあるいは割り込み処理ルーチンからの戻り番地やアドレス・レジスタの値を書き込む動作を行います。

しかしアドレス・スタック・レジスタの7H番地であるASR7は実際には存在しないため，戻り番地やアドレス・レジスタの値を書き込むことができません。

したがって，スタック・ポインタの値が7Hのときにリターン命令を実行するとアドレス・スタック・レジスタの7H番地であるASR7の内容をプログラム・カウンタおよびセグメント・レジスタに転送します。

このとき，アドレス・スタック・レジスタの7H番地であるASR7から読み出される内容は“不定”となるため，プログラムの流れが正常に復帰しなくなります。

このような事態を避けるため，PUSH命令やPOP命令を用いてアドレス・スタック・レジスタの値を退避することにより対処します。

表４－６はPUSH命令およびPOP命令の動作を示します。

表４－６　PUSH命令およびPOP命令の動作

命　令	動　　作
POP	①スタック・ポインタ（SP）で指定されるアドレス・スタック・レジスタの値をアドレス・レジスタ（AR）に転送 ②スタック・ポインタ（SP）の値を＋１
PUSH	①スタック・ポインタ（SP）の値を－１ ②スタック・ポインタ（SP）で指定されるアドレス・スタック・レジスタにアドレス・レジスタ（AR）の値を転送

図４－６の動作例では６レベル目のサブルーチンの10H番地に30H番地から始まる７レベル目のサブルーチンを呼び出すCALL命令があり，35H番地に50H番地から始まる８レベル目のサブルーチンを呼び出すCALL命令があることを表しています。

図中の矢印はプログラムの流れを表しています。

この例において10H番地を実行する直前のスタック・ポインタの値は1Hとなっています。10H番地のCALL命令を実行することによりスタック・ポインタの値は－１されて0Hになり，アドレス・スタック・レジスタの0Hには７レベル目のサブルーチンからの戻り番地である11Hを退避します。またプログラム・カウンタにはCALL命令のオペランドである30Hを転送します。

次に７レベル目のサブルーチン内のPOP命令を実行すると，スタック・ポインタの値は＋１されて1Hになり，アドレス・レジスタにアドレス・スタック・レジスタの0H番地の内容の11Hを転送します。

次に35H番地のCALL命令を実行するとスタック・ポインタの値は－１されて0Hになり，アドレス・スタック・レジスタの0H番地には８レベル目のサブルーチンからの戻り番地である41Hが退避されます。またプログラム・カウンタにはCALL命令のオペランドである50Hを転送します。

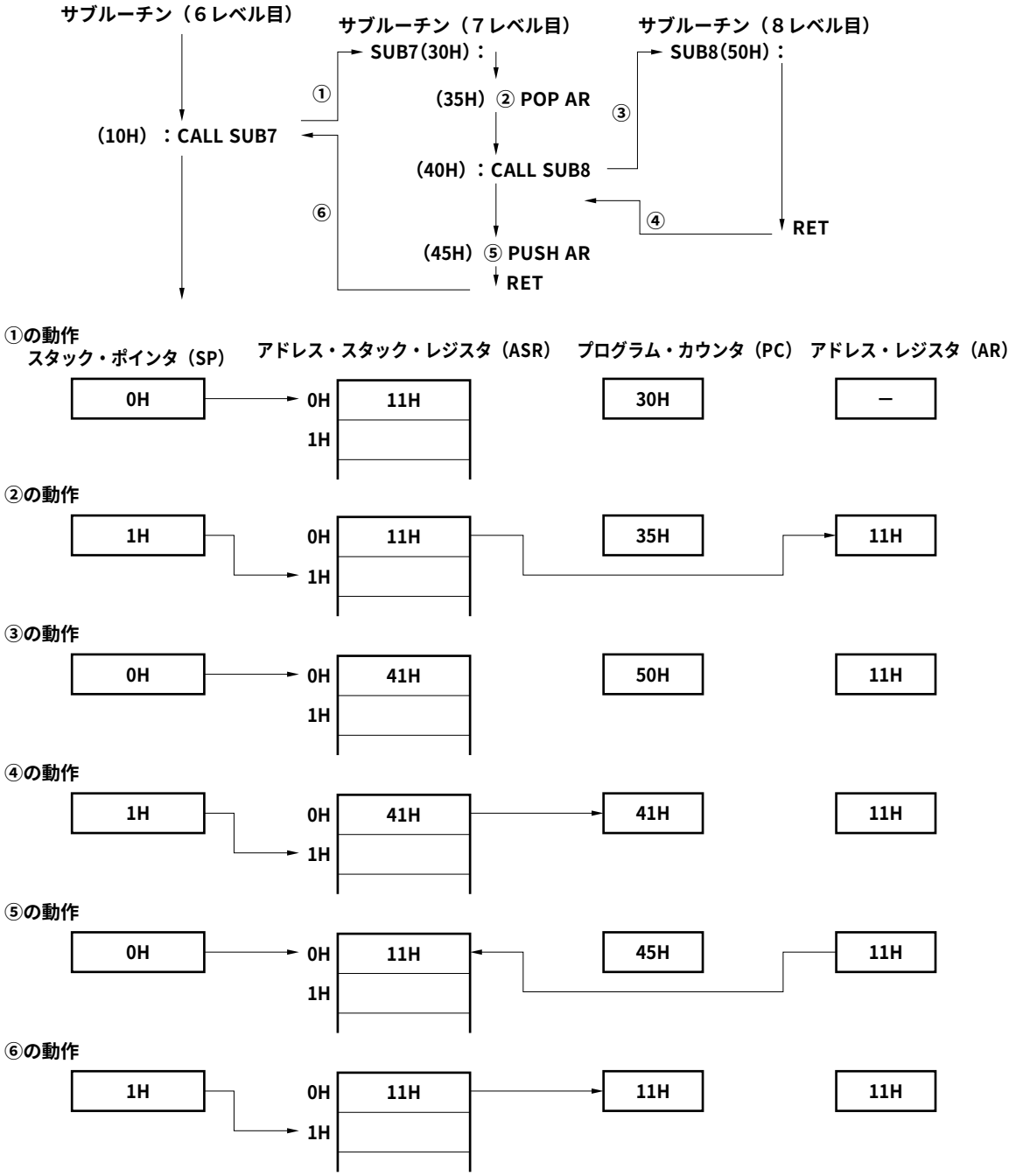
次に８レベル目のサブルーチン内でRET命令を実行するとアドレス・スタック・レジスタの0H番地の内容41Hがプログラム・カウンタに復帰し，スタック・ポインタの値は＋１されて1Hになります。

次に７レベル目のサブルーチンのPUSH命令を実行するとスタック・ポインタは－１されて0Hになり，アドレス・スタック・レジスタの0H番地に，アドレス・レジスタの内容の11番地，つまり６レベル目のサブルーチンへの戻り番地を転送します。

次に７レベル目のサブルーチン内のRET命令を実行するとアドレス・スタック・レジスタの0H番地の内容の11Hをプログラム・カウンタに復帰し，スタック・ポインタの値は＋１されて1Hになります。

以上のような方法でアドレス・スタックのネスティング・レベルを８レベルとすることができます。

図４－６　PUSH, POP命令によるスタック動作例



第５章　データ・メモリ（RAM）

データ・メモリは，演算・制御等のデータを記憶するメモリです。命令により常時データの書き込みや読み出しができます。

５.１　データ・メモリの構成

図５－１のように，データ・メモリは“ バンク ”と呼ぶ単位で最大16に分割されています。各バンクはそれぞれBANK0, BANK1, …, BANKn（n≦15）と呼びます。

各バンクは４ビット単位のデータごとに番地（アドレス）が割り付けられており，上位３ビットを“ ロウ・アドレス ”，下位４ビットを“ カラム・アドレス ”と呼びます。たとえば，ロウ・アドレスが1Hでカラム・アドレスが0AHのデータ・メモリはアドレス1AHのデータ・メモリと呼びます。また１つのアドレスは４ビットのメモリで構成されており，これを“ １ニブル ”と呼びます。

また，データ・メモリは，機能別に次の（１）-（５）に示すブロックに分けられます。

（１）システム・レジスタ（SYSREG）の構成

★　データ・メモリのアドレス74H-7FHに割り当てられた12ニブルで構成されています。システム・レジスタはバンクに無関係に割り当てられています。すなわち，どのバンクであってもアドレス74H-7FHには同一のシステム・レジスタが存在します。

（２）データ・バッファ（DBF）の構成

データ・メモリのBANK0のアドレス0CH-0FHに割り当てられた４ニブルで構成されています。

（３）ポート・データ・レジスタ（ポート・レジスタ）の構成

データ・メモリの各バンクの一部を使って構成されます。

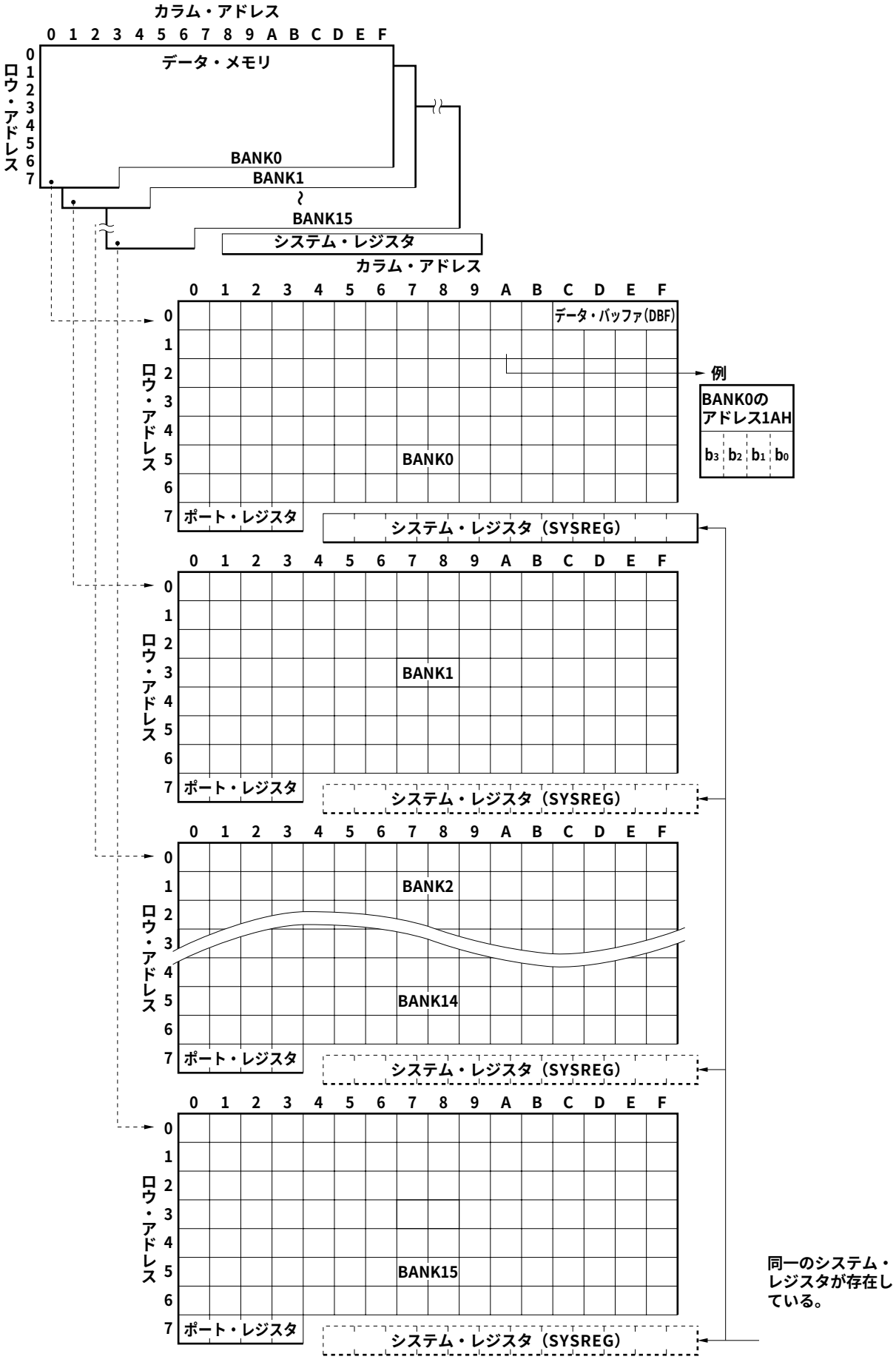
（４）汎用データ・メモリの構成

データ・メモリ内のシステム・レジスタとポート・レジスタ以外の部分です。112ニブルで構成されます。

（５）実装されていないデータ・メモリ

ポート・レジスタなどで，実際にポートが割り付けられていないデータ・メモリは“ ０ ”に固定されています。

図５－１　データ・メモリの構成



５.２　データ・メモリのアドレス指定時の注意

17Kシリーズのアセンブラ（RA17K）を使う場合は，例１のようにデータ・メモリ操作命令のオペランドに直接データ・メモリ・アドレスを数値で記述するとエラーが発生します。

これは，プログラム修正時などにバグ要因を低減する目的であらかじめアセンブラに組み込まれています。

例１．エラーになる場合

```
;①
    MOV     2FH, #0001B  ;アドレス2FHを直接指定する。
;②
    MOV   0.2FH, #0001B  ;BANK0のアドレス2FHを直接指定する。
```

エラーにならない場合

```
;③
    M02F MEM 0.2FH       ;BANK0のアドレス2FHをメモリ型としてM02Fにシンボル定義する。
    MOV M02F, #0001B     ;
;④
    MOV.MD.2FH, #0001B ;アドレス2FHを.MD.によりメモリ型に変換する。
                       ;ただし，バグ要因を低減するためにはこのような方法は避けるべきであ
                       ;る。
```

したがってアセンブラ疑似命令であるMEM命令（シンボル定義疑似命令）を用いてあらかじめ，データ・メモリ・アドレスをシンボル定義しておく必要があります。

データ・メモリのシンボル定義には，例２のようにデータ・メモリのバンクも定義する必要があります。

これは，アセンブラ上で自動的にデータ・メモリ・マップを作成するときに使用されます。

ただし，このときたとえば例２のようにBANK2にシンボル定義されているデータ・メモリを，プログラム上BANK1の範囲で使うと，BANK1のデータ・メモリが操作されます。

２．

```
M1    MEM 0.15H     ;
M2    MEM 1.15H     ;  シンボル定義疑似命令
M3    MEM 2.15H     ;
```

バンク　ロウ・アドレス　カラム・アドレス

```
BANK1               ;  アセンブラ組み込みマクロ命令　BANK←１
MOV   M1, #0000B    ;  M1, M2, M3は①で別のバンクにシンボル定義されているが，プログラム
MOV   M2, #0000B    ;  上はBANK1となっているため，これらの３つの命令はすべてBANK1のア
MOV   M3, #0000B    ;  ドレス15Hのデータ・メモリに０を書き込む。
```

〔メ　モ〕

第６章　システム・レジスタ（SYSREG）

システム・レジスタは，直接CPUの制御をするためのレジスタでデータ・メモリ上にあります。

６.１　システム・レジスタの構成

図６－１のように，システム・レジスタは，データ・メモリ・アドレスの74H-7FH番地に，バンクとは無関係にあります。つまり，どのバンクでも74H-7FH番地は同一のシステム・レジスタです。

システム・レジスタはデータ・メモリ上にあるので，すべてのデータ・メモリ操作命令が使えます。

また，システム・レジスタをジェネラル・レジスタに指定することもできます。

図６－２のようにシステム・レジスタは次の７種類のレジスタで構成されています。

アドレス・レジスタ（AR）
ウインドウ・レジスタ(WR)
バンク・レジスタ（BANK）
インデクス・レジスタ（IX）
データ・メモリ・ロウ・アドレス・ポインタ（MP）
ジェネラル・レジスタ・ポインタ（RP）
プログラム・ステータス・ワード（PSWORD）

図６－１　システム・レジスタのデータ・メモリ上の配置

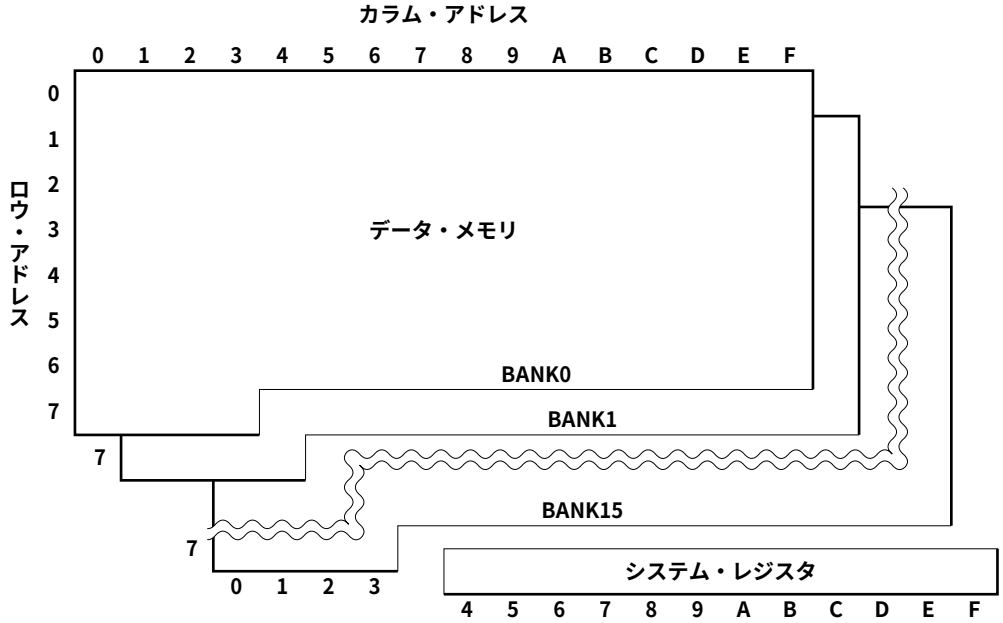
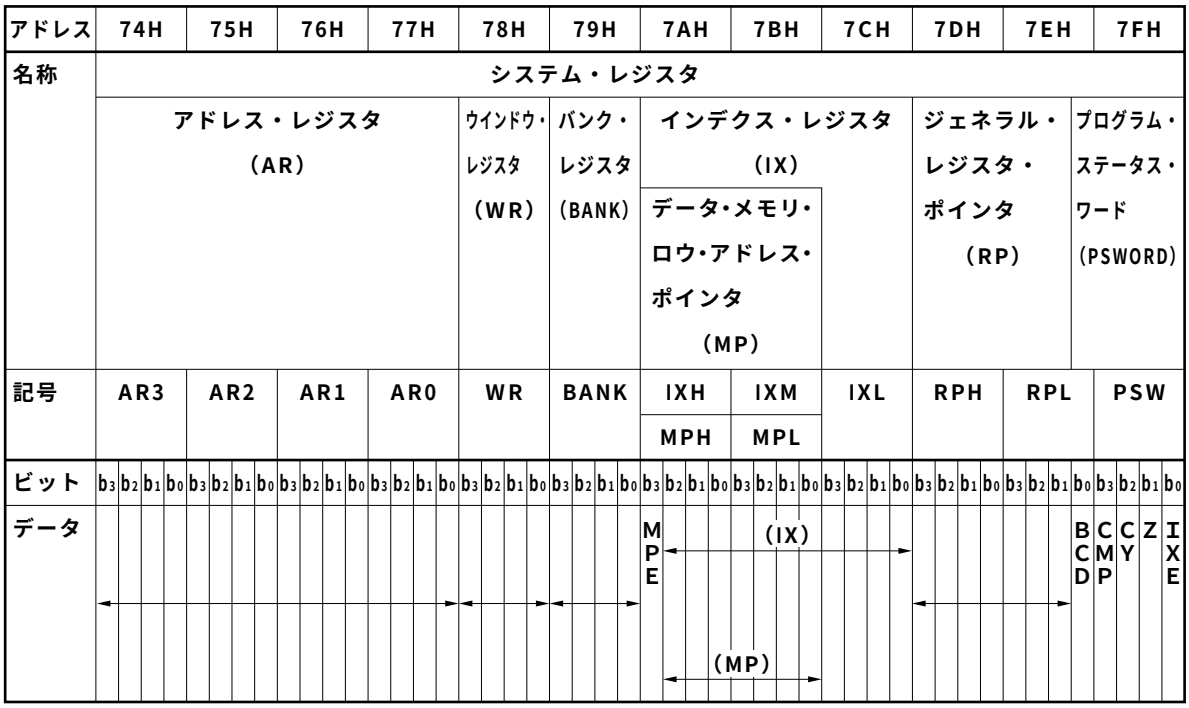


図６－２　システム・レジスタの構成



6.2　システム・レジスタの機能

6.2.1　各レジスタの機能

システム・レジスタの各レジスタの機能の概略は次の（1）-（7）のとおりです。
各レジスタの機能の詳細は**6.4-6.9**です。

（1）アドレス・レジスタ（AR）

プログラム・メモリのアドレスを間接指定します。

（2）ウインドウ・レジスタ（WR）

レジスタ・ファイルとのデータ転送を行います。

（3）バンク・レジスタ（BANK）

データ・メモリのバンクを指定します。

（4）インデクス・レジスタ（IX）

データ・メモリのアドレスの修飾を行います。

（5）データ・メモリ・ロウ・アドレス・ポインタ（MP）

ジェネラル・レジスタ間接転送時のロウ・アドレスを指定します。

（6）ジェネラル・レジスタ・ポインタ（RP）

ジェネラル・レジスタのバンクとロウ・アドレスを指定します。

（7）プログラム・ステータス・ワード（PSWORD）

演算，転送命令の条件を設定します。

6.2.2　システム・レジスタ操作命令

システム・レジスタはデータ・メモリ上にあるので，すべてのデータ・メモリ操作命令が使えます。またシステム・レジスタのうち，アドレス・レジスタおよびインデクス・レジスタについては次の専用命令があります。

INC AR；アドレス・レジスタの内容を“１”ずつインクリメントします。
アドレス・レジスタは有効ビット数が14ビットなので，命令を実行するたびに“１”ずつインクリメントしていき，1FFFHになると次に0000Hになります。
したがって，8 Kステップを越えるインクリメントはできません。

INC IX ；インデクス・レジスタの内容を“１”ずつインクリメントします。
インデクス・レジスタは有効ビット数が11ビットなので，命令を実行するたびに“１”ずつインクリメントしていき，7FFHになると次に000Hになります。

6.3　アドレス・レジスタ（AR）

6.3.1　アドレス・レジスタの構成

図６－３のようにアドレス・レジスタはシステム・レジスタの74H-77H（AR3-AR0）の16ビットで構成されています。

★ **図６－３　アドレス・レジスタの構成**

アドレス	74H				75H				76H				77H			
名称	アドレス・レジスタ（AR）															
記号	AR3				AR2				AR1				AR0			
ビット	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
データ	〈MSB〉															〈LSB〉

6.3.2　アドレス・レジスタの機能

アドレス・レジスタは間接分岐命令（BR @AR），間接サブルーチン・コール命令（CALL @AR），テーブル参照命令（MOVT DBF, @AR）およびスタック操作命令（PUSH AR, POP AR）実行時にプログラム・メモリ・アドレスを指定します。

6.3.3-**6.3.4**に各命令時の動作を説明します。

アドレス・レジスタには１ずつインクリメントできる専用命令（INC AR）があります。この命令を使うと，アドレス・レジスタは13ビット単位でデータをインクリメントできます。アドレス・レジスタの内容が1FFFHのときに“INC AR”命令を実行すると，アドレス・レジスタの内容は0000Hになります。

また，プログラム・メモリの範囲を越えるアドレスを設定しないように注意してください。

6.3.3　テーブル参照命令（MOVT DBF, @AR）

“MOVT DBF, @AR”命令を実行することにより，アドレス・レジスタの内容で指定するプログラム・メモリ・アドレスの定数データ（16ビット）をデータ・メモリ上のデータ・バッファ（DBF；BANK0の0CH-0FH番地）に読み込みます。

データ・バッファに読み込むことができるプログラム・メモリの定数データ格納番地は各品種のアドレス・レジスタが指定できる範囲です。

10.4　データ・バッファとテーブル参照も参照してください。

例

```
アドレス　レーベル
0000H     DATA1  ：16ビットの定数データ
                  MOV   AR0, ＃0FH ；AR0に0FHを書き込む。
                  MOV   AR1, ＃0H  ；AR1に0Hを書き込む。
                  MOV   AR2, ＃0H  ；AR2に0Hを書き込む。
                  MOV   AR3, ＃0H  ；AR3に0Hを書き込む。
                  MOVT  DBF, @AR   ；データ・バッファにプログラム・メモリ
                                   ；の000FH番地の定数データを読み込む。
```

6.3.4　スタック操作命令（PUSH AR, POP AR）

“ PUSH AR ”命令を実行することにより，スタック・ポインタの値を－１とし，その値で指定するアドレス・スタック・レジスタにアドレス・レジスタ（AR）の内容を格納します。

“ POP AR ”命令を実行することにより，スタック・ポインタで指定するアドレス・スタック・レジスタの内容をアドレス・レジスタに転送して，スタック・ポインタを＋１します。

詳しくは**第４章　アドレス・スタック（ASK）**を参照してください。

6.3.5　間接分岐命令（BR @AR）

“ BR @AR ”命令により，アドレス・レジスタで指定するプログラム・メモリ番地へ分岐することができます。

例

```
MOV  AR0, ＃0FH     ；AR0に0FHを書き込む。
MOV  AR1, ＃0H      ；AR1に0Hを書き込む。
MOV  AR2, ＃0H      ；AR2に0Hを書き込む。
MOV  AR3, ＃0H      ；AR3に0Hを書き込む。
BR   @AR            ；プログラムは000FHに分岐する。
```

6.3.6　間接サブルーチン・コール命令（CALL @AR）

“ CALL @AR ”命令により，アドレス・レジスタが指定するプログラム・メモリ番地のサブルーチンを呼び出すことができます。

例１．

```
アドレス　レーベル
000FH     SUB：
                    サブルーチン処理


                    RET
                    MOV  AR0, #0FH      ；AR0に0FHを書き込む。
                    MOV  AR1, #0H       ；AR1に0Hを書き込む。
                    MOV  AR2, #0H       ；AR2に0Hを書き込む。
                    MOV  AR3, #0H       ；AR3に0Hを書き込む。
                    CALL @AR            ；000FH番地のサブルーチンをコール
```

上記の例１では，間接サブルーチン・コールするアドレスを“MOV”命令で指定しています。
このような方法では，たとえば間接サブルーチン・コールする回数が多いとき，プログラム・メモリの効率が悪くなります。
したがって，次の例２のように“POP”，“PUSH”命令およびテーブル参照を用いると便利です。

２．

```
SUBENTRY：
        DI
        POP   AR
        MOVT DBF, @AR
        INC   AR
        PUSH AR
        EI
        PUT   AR, DBF
        BR    @AR
SUB1    ：
SUB2    ：
MAIN    ：
        CALL  SUBENTRY
        DW    .DL.SUB1
        CALL  SUBENTRY
        DW    .DL.SUB2
```

６.３.７　アドレス・レジスタとデータ・バッファ

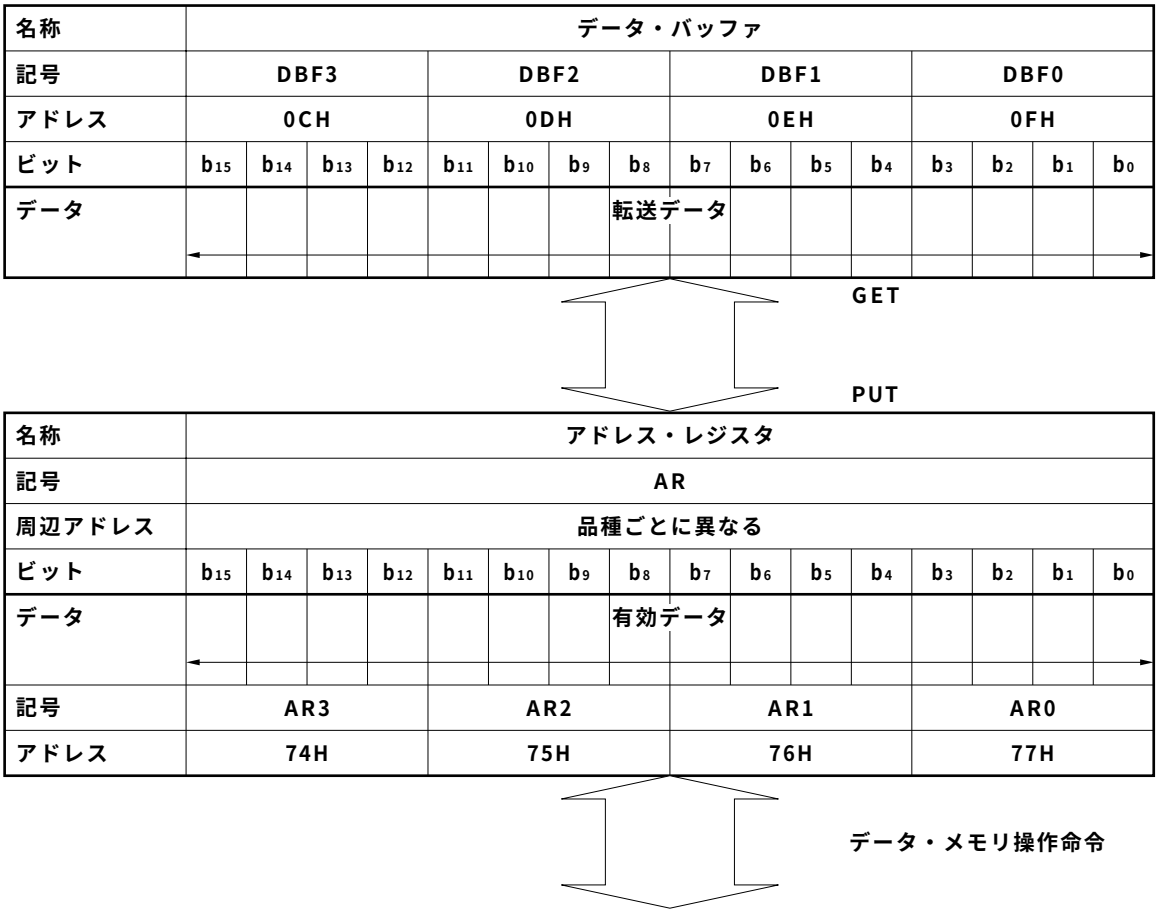
アドレス・レジスタは，データ・メモリ操作命令により直接データを操作できますが，周辺ハードウエアの一部として，データ・バッファを介してのデータ転送も可能です。

すなわち，アドレス・レジスタは，データ・メモリ操作命令の他に，図６－４のように“ PUT ”および“ GET ”命令を用いることにより，データ・バッファを介してデータの読み込みおよび書き込みができます。

次に，アドレス・レジスタとデータ・バッファとの関係を示します。

データ・バッファについては，**第10章　データ・バッファ（DBF）**の項を参照してください。

図６－４　アドレス・レジスタとデータ・バッファとの間のデータ転送



アドレス・レジスタへ直接データを読み書きするとき

６.４　ウインドウ・レジスタ（WR）

６.４.１　ウインドウ・レジスタの構成

図６－５のように，ウインドウ・レジスタはシステム・レジスタの78H番地の４ビットで構成されています。

図６－５　ウインドウ・レジスタの構成

アドレス	78H			
名称	ウインドウ・レジスタ（WR）			
記号	WR			
ビット	b_3	b_2	b_1	b_0
データ	〈MSB〉			〈LSB〉

６.４.２　ウインドウ・レジスタの機能

ウインドウ・レジスタはレジスタ・ファイル（RF）とのデータ転送に使用します。

レジスタ・ファイルとのデータ転送は，専用命令である“ PEEK WR, rf ”および“ POKE rf, WR ”でします。

６.４.３，６.４.４に各命令の動作を説明します。

第９章　レジスタ・ファイル（RF）も参照してください。

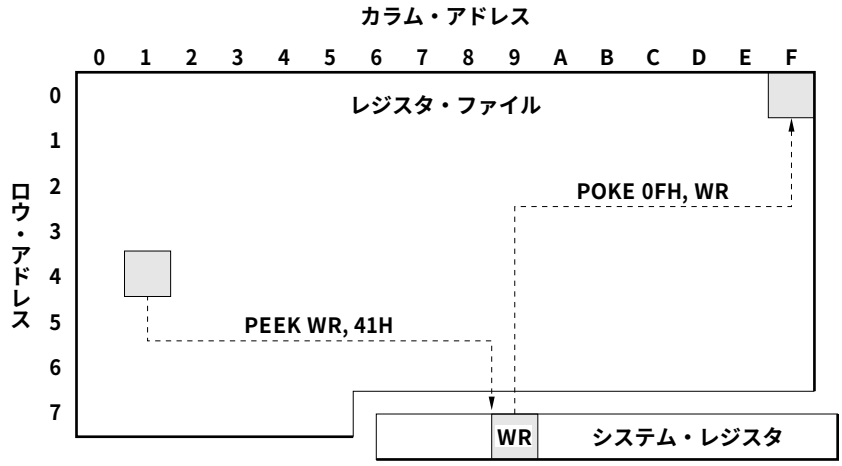
６.４.３　PEEK WR, rf命令

図６－６のように，PEEK WR, rf命令（rf：レジスタ・ファイルのアドレス）により，rfでアドレス指定するレジスタ・ファイルの内容を，ウインドウ・レジスタに転送します。

６.４.４　POKE rf, WR命令

図６－６のように，POKE rf, WR命令（rf：レジスタ・ファイルのアドレス）により，ウインドウ・レジスタの内容を，rfでアドレス指定するレジスタ・ファイルに転送します。

図６－６　PEEK, POKE命令の動作



6.5　バンク・レジスタ（BANK）

6.5.1　バンク・レジスタの構成

図６－７のように，バンク・レジスタは，システム・レジスタの中の79H番地（BANK）の４ビットです。

図６－７　バンク・レジスタの構成

アドレス	79H			
名称	バンク・レジスタ（BANK）			
記号	BANK			
ビット	b_3	b_2	b_1	b_0
データ	〈MSB〉			〈LSB〉

6.5.2　バンク・レジスタの機能

バンク・レジスタはデータ・メモリのバンク切り替えをします。

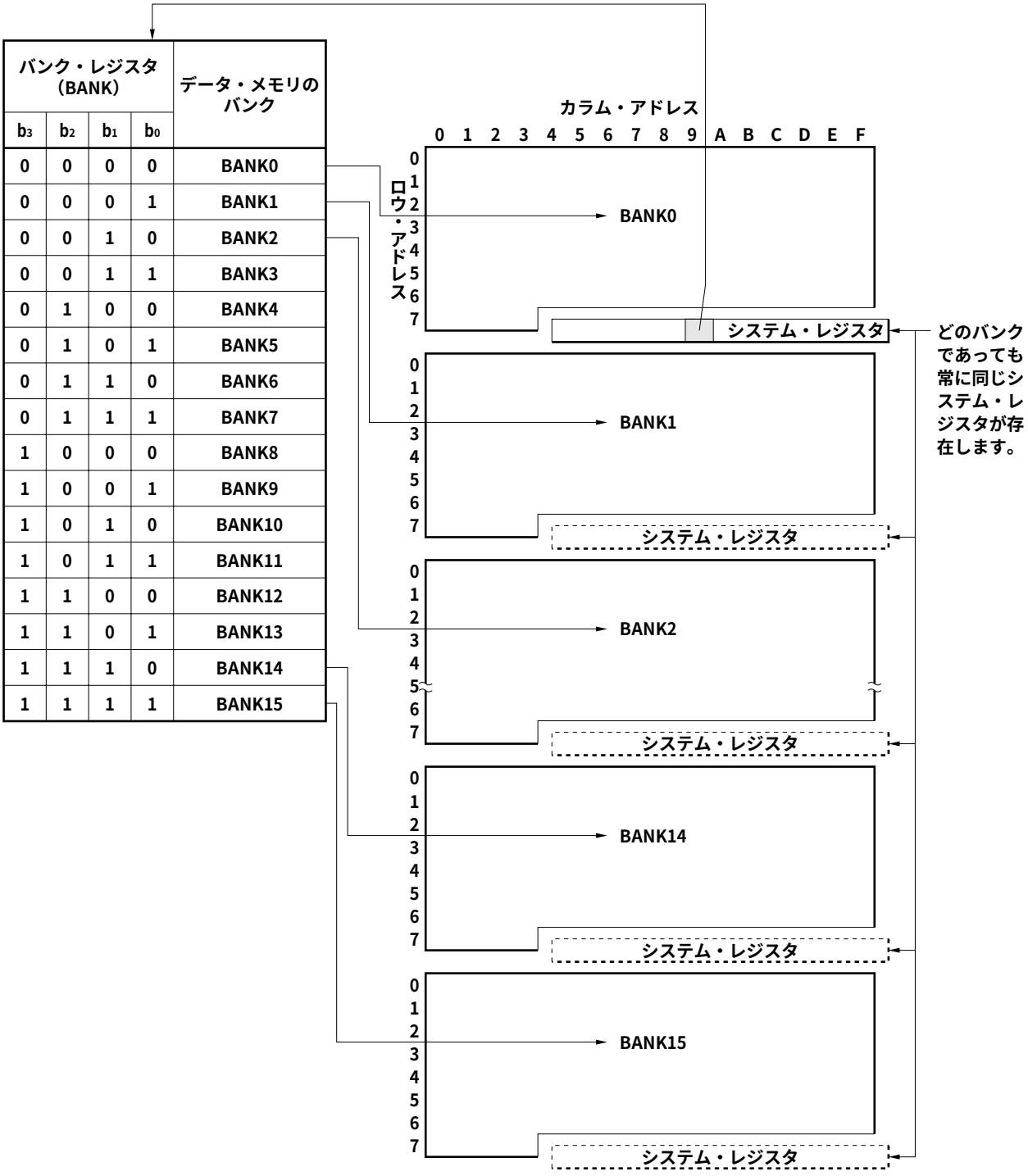
図６－８のように，データ・メモリは，バンク・レジスタにより最大16のバンクに分けられており，データ・メモリ操作命令は，バンク・レジスタが指定するバンク内のデータ・メモリを操作します。

したがって，たとえば現在BANK0のときにBANK1のデータ・メモリを操作するためには，バンク・レジスタに0001Hを書き込み，バンクをBANK1に切り替えておく必要があります。

このときデータ・メモリの74H-7FH番地に配置されているシステム・レジスタにはバンクの概念がなく，すべてのバンクにおいて同一のシステム・レジスタです。これはBANK1で“MOV BANK, #0”命令を行っても，BANK2で“MOV BANK, #０”命令を行っても，結局バンク・レジスタ（BANK：78H番地）に０が書き込まれます。すなわち，バンク・レジスタを操作する場合は，そのときのバンク指定に関係ありません。

図6－8　データ・メモリのバンク指定

バンク・レジスタ(BANK)				データ・メモリのバンク
b_3	b_2	b_1	b_0	
0	0	0	0	BANK0
0	0	0	1	BANK1
0	0	1	0	BANK2
0	0	1	1	BANK3
0	1	0	0	BANK4
0	1	0	1	BANK5
0	1	1	0	BANK6
0	1	1	1	BANK7
1	0	0	0	BANK8
1	0	0	1	BANK9
1	0	1	0	BANK10
1	0	1	1	BANK11
1	1	0	0	BANK12
1	1	0	1	BANK13
1	1	1	0	BANK14
1	1	1	1	BANK15



★　　バンクを操作する命令として，17Kシリーズのアセンブラ（RA17K）には組み込みマクロ命令“ BANKn ”（0≦n≦4）があります。

次にバンクとデータ・メモリの操作例を示します。

例

```
M000    MEM 0.00H       ; シンボル定義
M100    MEM 1.00H       ;
BANK0                   ; MOV BANK, #0000Bと同等
MOV     M000, #0101B    ; BANK0の00H番地に0101Bを書き込む。
BANK1                   ; MOV BANK, #0001Bと同等
MOV     M100, #0101B    ; BANK1の00H番地に0101Bを書き込む。
MOV     M000, #0101B    ; BANK1の00H番地に0101Bを書き込む。
                        ; これは，データ・メモリM000はシンボル定義でBANK0に定義されて
                        ; いるが，プログラム実行時にはそのとき指定されているバンクになる
                        ; ということです。
```

6.6　インデクス・レジスタ（IX）とデータ・メモリ・ロウ・アドレス・ポインタ（MP：メモリ・ポインタ）

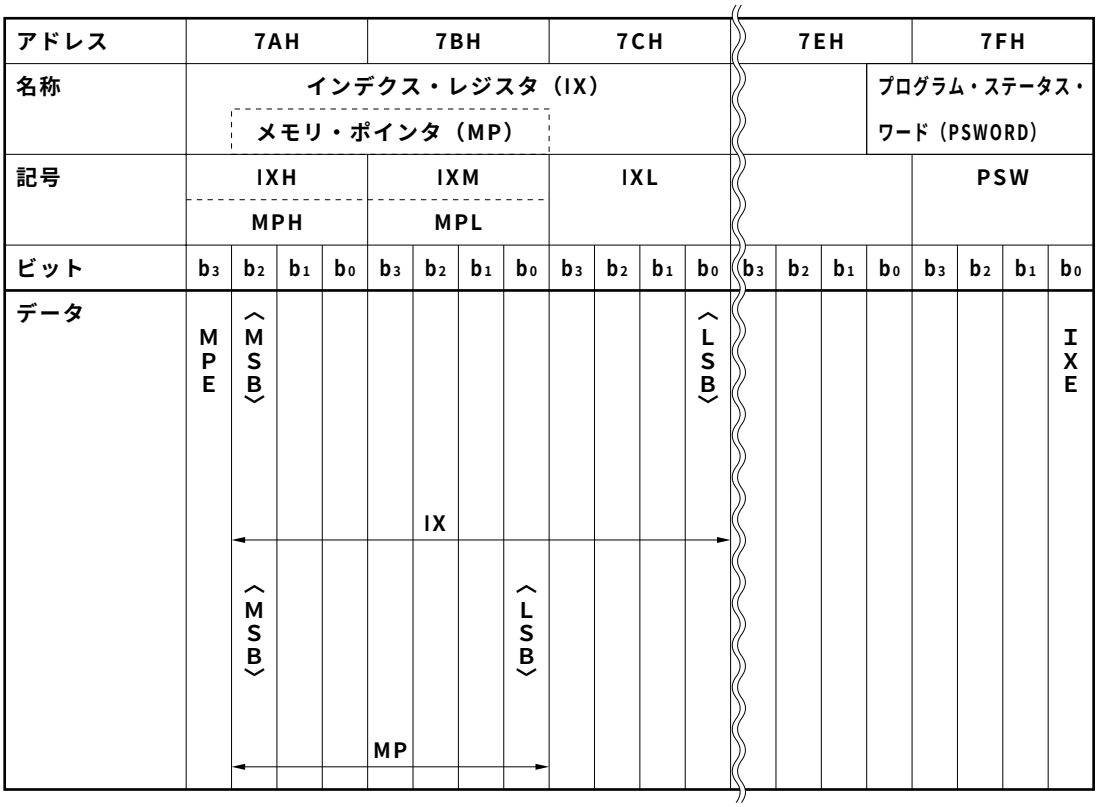
6.6.1　インデクス・レジスタとデータ・メモリ・ロウ・アドレス・ポインタの構成

図6－9のように，インデクス・レジスタは，システム・レジスタの7AHの下位3ビット(IXH）と7BH，7CH（IXM, IXL）の計11ビットと，7FH（PSW）の最下位ビットにあるインデクス・イネーブル・フラグ（IXE）から構成されています。

データ・メモリ・ロウ・アドレス・ポインタ（メモリ・ポインタ）は，7AH（MPH）の下位3ビットと7BH（MPL）の計7ビットと，7AH（MPH）の最上位ビットにあるデータ・メモリ・ロウ・アドレス・ポインタ・イネーブル・フラグ（メモリ・ポインタ・イネーブル・フラグ：MPE）とから構成されています。

すなわち，インデクス・レジスタの上位7ビットとデータ・メモリ・ロウ・アドレス・ポインタとは共用です。

図6－9　インデクス・レジスタとデータ・メモリ・ロウ・アドレス・ポインタの構成



６.６.２　インデクス・レジスタとデータ・メモリ・ロウ・アドレス・ポインタの機能

（１），（２）にインデクス・レジスタとデータ・メモリ・ロウ・アドレス・ポインタの機能を示します。

（１）インデクス・レジスタ

インデクス・レジスタは，データ・メモリ操作命令を実行すると命令で指定されたデータ・メモリのバンクとアドレスをインデクス・レジスタの内容によりアドレス修飾します。

ただし，インデクス・レジスタによるアドレス修飾はインデクス・イネーブル・フラグ（IXE）がセットされているときのみ有効になります。

アドレス修飾の方法はデータ・メモリのバンクとアドレスをインデクス・レジスタの内容とOR演算し，その演算結果で指定されるアドレス（実アドレスと呼ぶ）のデータ・メモリに対して命令を実行します。

インデクス・レジスタによるアドレス修飾は，すべてのデータ・メモリ操作命令を対象にできます。

また，次の命令はインデクス・レジスタの修飾対象となりません。

MOVT	DBF, @AR	BR	addr	INC	AR	EI
PEEK	WR, rf	BR	@AR	INC	IX	DI
POKE	rf, WR	CALL	addr	RORC	r	
GET	DBF, p	CALL	@AR	STOP	s	
PUT	p, DBF	RET		HALT	h	
PUSH	AR	RETSK		NOP		
POP	AR	RETI				

（２）データ・メモリ・ロウ・アドレス・ポインタ

データ・メモリ・ロウ・アドレス・ポインタはジェネラル・レジスタ間接転送命令（MOV @r, mおよびMOV m, @r）を実行すると，間接転送先のアドレスをデータ・メモリ・ロウ・アドレス・ポインタの内容でアドレス修飾します。

ただし，データ・メモリ・ロウ・アドレス・ポインタによるアドレス修飾はデータ・メモリ・ロウ・アドレス・ポインタ・イネーブル・フラグ（メモリ・ポインタ・イネーブル・フラグ：MPE）がセット（１）されているときのみ有効になります。

また，ジェネラル・レジスタ間接転送命令以外ではアドレス修飾の対象になりません。

アドレス修飾の方法は間接転送先のバンクとロウ・アドレスをデータ・メモリ・ロウ・アドレス・ポインタの内容で置き換えます。

表６－１にインデクス・レジスタとデータ・メモリ・ロウ・アドレス・ポインタによるデータ・メモリ・アドレスの修飾および間接転送アドレスの修飾を示します。

また，６.６.３-６.６.６にインデクス・レジスタとデータ・メモリ・ロウ・アドレス・ポインタによるデータ・メモリのアドレス修飾の各動作について説明します。

表6－1　インデクス・レジスタとデータ・メモリ・ロウ・アドレス・ポインタによるデータ・メモリ・アドレスの修飾

IXE	MPE		rで指定されるジェネラル・レジスタ・アドレス			mで指定されるデータ・メモリ・アドレス			@rで指定される間接転送アドレス		
			バンク	ロウ・アドレス	カラム・アドレス	バンク	ロウ・アドレス	カラム・アドレス	バンク	ロウ・アドレス	カラム・アドレス
			b_3 b_2 b_1 b_0	b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_3 b_2 b_1 b_0	b_2 b_1 b_0	b_3 b_2 b_1 b_0
0	0		RP		r	BANK		m	BANK	m_R	(r)
0	1			同上			同上		MP		(r)
1	0			同上		BANK		m	BANK	m_R	
							Logical OR		Logical OR		(r)
							IX		IXH	IXM	
1	1			同上			同上		MP		(r)
アドレス修飾される命令群											
加減算	ADD ADDC SUB SUBC		r			m					
						m, ＃n4					
論理	AND OR XOR		r			m					
						m, ＃n4					
比較	SKE SKGE SKLT SKNE					m, ＃n4					
判断	SKT SKF					m, ＃n					
転送	LD ST		r			m					
	MOV					m, ＃n4					
			@r			m			間接転送アスドレス		

BANK：バンク・レジスタ
IX　：インデクス・レジスタ
　IXE：インデクス・イネーブル・フラグ
　IXH：インデクス・レジスタのビット10-ビット８
　IXM：インデクス・レジスタのビット７-ビット４
　IXL：インデクス・レジスタのビット３-ビット０
m　：m_R,m_Cで示されるデータ・メモリ・アドレス
　m_R：データ・メモリ・ロウ・アドレス（上位）
MP　：データ・メモリ・ロウ・アドレス・ポインタ
　MPE：メモリ・ポインタ・イネーブル・フラグ
r　：ジェネラル・レジスタ・カラム・アドレス
　RP：ジェネラル・レジスタ・ポインタ
（×）：×でアドレスされる内容
　　×：m, rなどのダイレクト・アドレス
　　：BANKなどのレジスタ

★ **備考**　IXEとMPEの設定は各品種によって異なります。各品種のデータ・シートを参照してください。

６.６.３　MPE＝０，IXE＝０のとき（データ・メモリ修飾なし）

表６－１に示したようにデータ・メモリ・アドレスはインデクス・レジスタと，データ・メモリ・ロウ・アドレス・ポインタの影響を受けません。

（１）データ・メモリ操作命令

例１．ジェネラル・レジスタがロウ・アドレス０にあるとき

```
R003     MEM 0.03H
M061     MEM 0.61H
ADD      R003, M061
```

上の命令を実行すると，図６－10に示すようにジェネラル・レジスタR003とデータ・メモリM061の内容を加算し，結果をジェネラル・レジスタR003に格納します。

（２）ジェネラル・レジスタ間接転送

例２．ジェネラル・レジスタがロウ・アドレス０にあるとき

```
R005     MEM 0.05H
M034     MEM 0.34H
MOV      R005, ＃8          ；R005←８
MOV   @R005, M034           ；レジスタ間接転送
```

上の命令を実行すると図６－10に示すようにデータ・メモリM034の内容がデータ・メモリの38H番地へ転送されます。

すなわち“MOV @r, m”命令は，mで指定されるデータ・メモリの内容をmと同じロウ・アドレスの@rで指定される間接アドレスのデータ・メモリへ転送します。

間接転送アドレスは，ロウ・アドレスがmと同一（上記ではロウ・アドレス３）で，カラム・アドレスがrで指定されるジェネラル・レジスタの内容（上記では８）になります。すなわち上記では38Hとなります。

例３．ジェネラル・レジスタがロウ・アドレス０にあるとき

```
R00B    MEM 0.0BH
M034    MEM 0.34H
MOV     R00B, ＃0EH       ；R00B←0EH
MOV     M034, @R00B       ；レジスタ間接転送
```

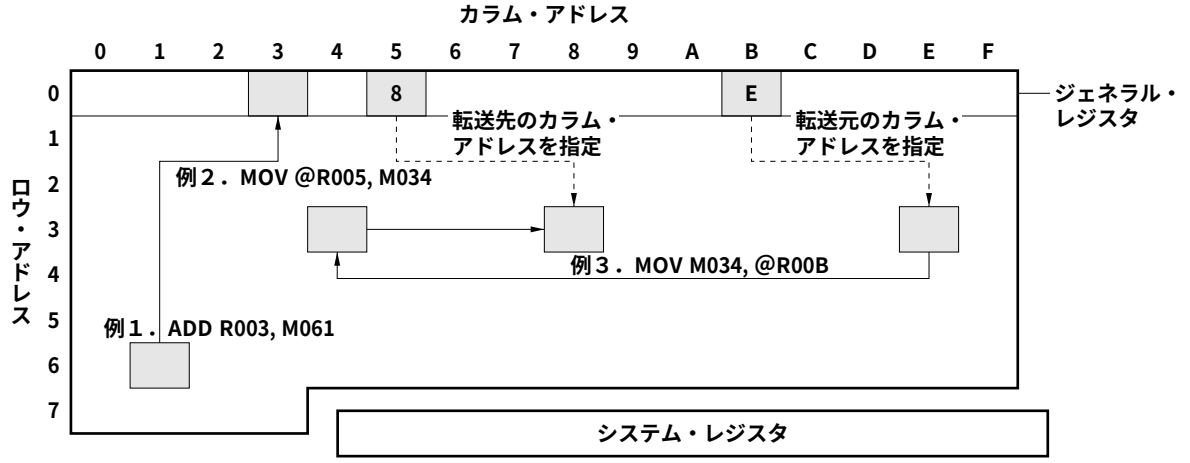
上の命令を実行すると，図６－10に示すようにデータ・メモリM034へ，アドレス3EH番地のデータ・メモリの内容を転送します。

すなわち“MOV m, @r”命令はmで指定されるデータ・メモリへmと同じロウ・アドレスの@rで指定される間接アドレスのデータ・メモリの内容を転送します。

間接転送アドレスは，ロウ・アドレスがmと同一（上記ではロウ・アドレス３）で，カラム・アドレスがrで指定されるジェネラル・レジスタの内容（上記では0EH）になります。すなわち上記ではアドレス3EHとなります。

これは例２と比べると転送するデータ・メモリ・アドレスのソース（転送元）とディスティネーション（転送先）が，入れ替わったことになります。

図６－10　MPE＝０，IXE＝０のときの動作例



例１のアドレス生成

ADD R003, M061

	バンク	ロウ・アドレス	カラム・アドレス
データ・メモリ・アドレス　M	0000	110	0001
ジェネラル・レジスタ・アドレス　R	0000	000	0011

例２のアドレス生成

MOV @R005, M034

	バンク	ロウ・アドレス	カラム・アドレス
データ・メモリ・アドレス　M	0000	011	0100
ジェネラル・レジスタ・アドレス　R	0000	000	0101
間接転送アドレス　@R	0000	011	1000
		Mと同一	Rの内容

６.６.４　MPE＝１，IXE＝０のとき（ななめ間接転送）

表６－１に示したようにジェネラル・レジスタ間接転送命令（MOV @r, mおよびMOV m, @r）を行ったときのみ，@rで指定される間接転送アドレスのバンクとロウ・アドレスがデータ・メモリ・ロウ・アドレス・ポインタの値になります。

例１．ジェネラル・レジスタがロウ・アドレス０のとき

```
R005    MEM 0.05H
M034    MEM 0.34H
MOV     MPL, ＃0110B      ; MP←６
MOV     MPH, ＃1000B      ; MPE←１
MOV     R005, ＃８        ; R005←８
MOV     @R005, M034       ; レジスタ間接転送
```

上の命令を実行すると図６－11に示すように，データ・メモリM034の内容が，データ・メモリの68H番地に転送されます。

すなわちMPE＝１のときに“MOV @r, m”命令を実行すると，mで指定されるデータ・メモリの内容をメモリ・ポインタで指定したロウ・アドレスの@rで指定されるカラム・アドレスへ転送します。

このとき@rで指定される間接アドレスは，バンクとロウ・アドレスがデータ・メモリ・ロウ・アドレス・ポインタの値（上記例ではロウ・アドレス６）で，カラム・アドレスがrで指定されるジェネラル・レジスタの内容（上記例では８）になります。

すなわち上記では68Hとなります。

これは**６.６.３**例２のMPE＝０のときと比べると，@rで指定される間接アドレスのバンクとロウ・アドレスをデータ・メモリ・ロウ・アドレス・ポインタで指定する点が異なります（**６.６.３**例２では間接アドレスのバンクとロウ・アドレスはmと同じになる）。

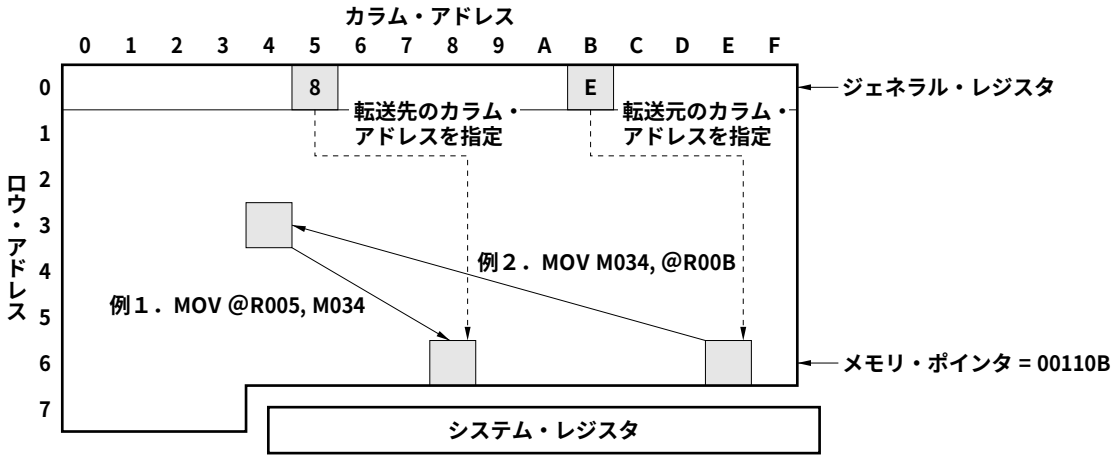
したがって，MPE＝１とすることによりジェネラル・レジスタ間接転送をななめに行うことが可能になります。

例２．ジェネラル・レジスタがロウ・アドレス０のとき

```
R00B    MEM 0.0BH
M034    MEM 0.34H
MOV     MPL, #0110B      ; MP←6
MOV     MPH, #1000B      ; MPE←1
MOV     R00B, #0EH       ; R00B←0EH
MOV     M034, @R00B      ; レジスタ間接転送
```

上の命令を実行すると，図６－11に示すようにデータ・メモリM034へ，アドレス6EH番地のデータ・メモリの内容を転送します。

図６－11　MPE＝１，IXE＝０のときの動作例



例１のアドレス生成

MOV @R005, M034

	バンク	ロウ・アドレス	カラム・アドレス
データ・メモリ・アドレス　M	0000	011	0100
ジェネラル・レジスタ・アドレス　R	0000	000	0101
間接転送アドレス　@R	0000	110	1000
	MPの内容		Rの内容

例２のアドレス生成

MOV M034, @R00B

	バンク	ロウ・アドレス	カラム・アドレス
データ・メモリ・アドレス　M	0000	011	0100
ジェネラル・レジスタ・アドレス　R	0000	000	1011
間接転送アドレス　@R	0000	110	1110
	MPの内容		Rの内容

６.６.５　MPE＝０，IXE＝１のとき（データ・メモリ・アドレス・インデクス修飾）

表６－１に示したようにデータ・メモリ操作命令を行うと，命令のオペランド“ｍ”で直接指定されたすべてのデータ・メモリのバンクとアドレスが，インデクス・レジスタによりアドレス修飾されます。

また，ジェネラル・レジスタ間接転送命令（MOV @r, mおよびMOV m, @r）を行ったときは，@rで指定される間接転送アドレスのバンクとロウ・アドレスもインデクス・レジスタによりアドレス修飾されます。

アドレス修飾の方法は，データ・メモリ・アドレスとインデクス・レジスタの内容がOR演算され，その演算結果で指定されるデータ・メモリ・アドレス（実アドレスと呼ぶ）に対して命令が実行されます。

例１．ジェネラル・レジスタがロウ・アドレス０のとき

```
R003    MEM 0.03H
M061    MEM 0.61H
MOV     IXL, #0010B             ; IX←00000010010B
MOV     IXM, #0001B             ;
MOV     IXH, #0000B             ; MPE←0
OR      PSW, #. DF.IXE AND 0FH  ; IXE←1
ADD     R003, M061
```

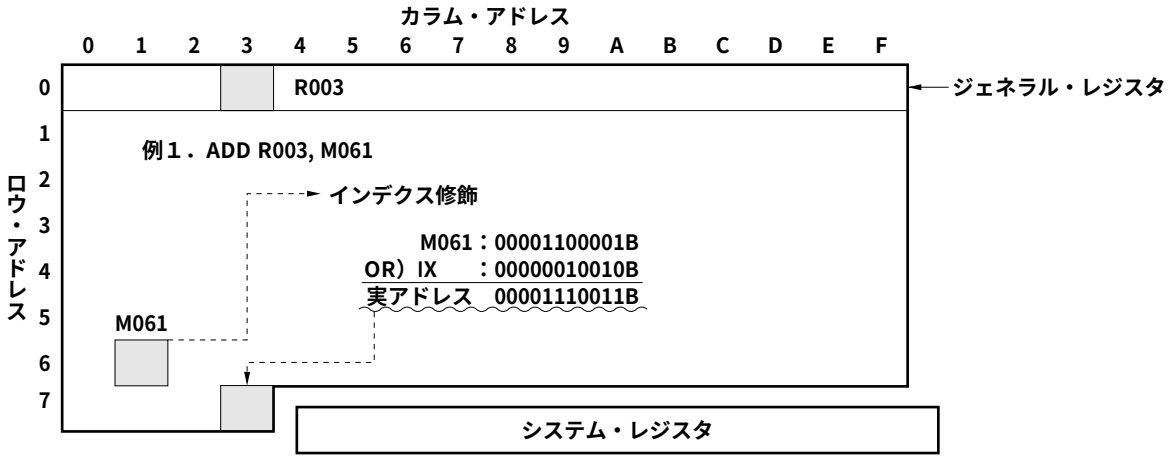
例１の命令を実行すると，図６－12に示すようにアドレスが73H番地のデータ・メモリ（実アドレス）の内容とジェネラル・レジスタR003（アドレスの03H番地）の内容を加算し，結果をジェネラル・レジスタR003へ格納します。

すなわち，“ADD r, m”命令を実行すると“ｍ”で指定されたデータ・メモリ・アドレス（上記では61H番地）がインデクス修飾されます。

修飾の方法はデータ・メモリM061のアドレスである61H番地（２進で00001100001B）と，インデクス・レジスタの値（上記例では00000010010B）をOR演算し，その結果の00001110011Bを実アドレス（73H番地）として，実アドレスに対して命令を実行します。

これはIXE＝０のとき（６.６.３の例）と比べると命令のオペランド“ｍ”で直接指定されるデータ・メモリのアドレスがインデクス・レジスタにより修飾（OR演算）されたことになります。

図６－12　MPE＝０，IXE＝１のときの動作例



例１のアドレス生成

ADD R003, M061

		バンク	ロウ・アドレス	カラム・アドレス
データ・メモリ・アドレス　M		0000	110	0001
ジェネラル・レジスタ・アドレス　R		0000	000	0011
インデクス修飾	M061	0000	110	0001
		BANK	m	
	IX	0000	001	0010
		IXH	IXM	IXL
	実アドレス（OR演算）	0000	111	0011

このアドレスに対して命令が実行される

例２．ジェネラル・レジスタ間接転送

BANK0でジェネラル・レジスタがロウ・アドレス０であるとする。

```
R005    MEM 0.05H
M034    MEM 0.34H
MOV     IXL, #0001B             ; IX←00000000001B
MOV     IXM, #0000B             ;
MOV     IXH, #0000B             ; MPE←０
OR      PSW, #.DF.IXE AND 0FH   ; IXE←１
MOV     R005, # ８              ; R005←８
MOV     @R005, M034             ; レジスタ間接転送
```

上の命令を実行すると図６－13に示すように，データ・メモリのアドレス35H番地の内容が，データ・メモリの38H番地に転送されます。

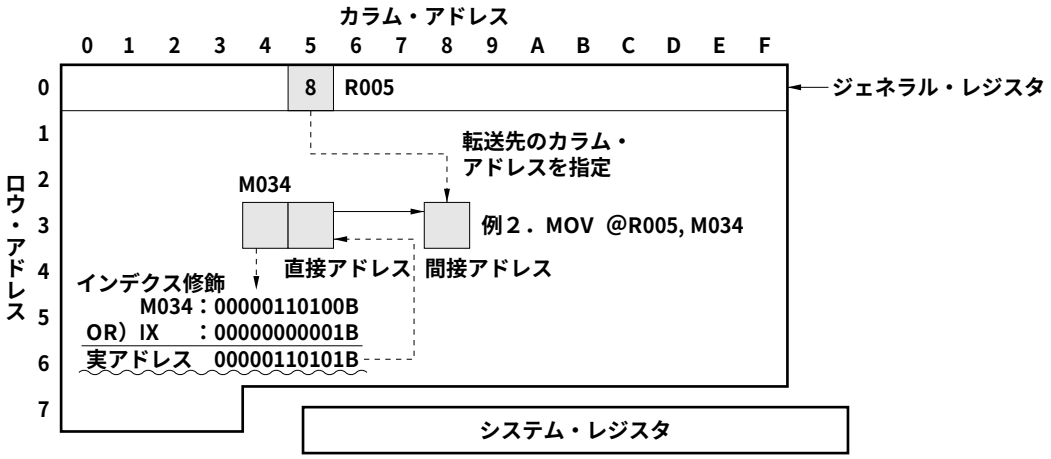
すなわち，IXE＝１のときに“MOV @r, m”命令を実行すると，“m”で指定されるデータ・メモリ・アドレス（直接アドレス）をインデクス・レジスタの内容でアドレス修飾し，“@r”で指定される間接アドレスのバンクとロウ・アドレスもインデクス・レジスタで修飾されます。

“m”で指定される直接アドレスはバンク，ロウ・アドレスおよびカラム・アドレスのすべてが修飾され，“@r”で指定される間接アドレスは，バンクとロウ・アドレスが修飾されます。

すなわち上記では直接アドレスが35Hとなり，間接アドレスが38H番地になります。

これは**6.6.3**例３のIXE＝０のときと比べると，“m”で指定される直接アドレスのバンク，ロウ・アドレスおよびカラム・アドレスがインデクス・レジスタでアドレス修飾され，このアドレス修飾されたデータ・メモリ・アドレスと同一ロウ・アドレスにジェネラル・レジスタ間接転送することになります（**6.6.3**例３では直接アドレスは修飾されない）。

図６－13　MPE＝０，IXE＝１のときのジェネラル・レジスタ間接転送動作例



例３．すべてのデータ・メモリの内容を０にクリアする

```
M000        MEM 0.00H
        MOV  IXL, ＃０              ; IX←０
        MOV  IXM, ＃０              ;
        MOV  IXH, ＃０              ; MPE←０
LOOP：
        OR   PSW, ＃.DF.IXE AND 0FH ; IXE←１
        MOV  M000, ＃０             ; IXで指定されたデータ・メモリを０にする。
        INC  IX                     ; IX←IX＋１
        AND  PSW, ＃1110B           ; IXE←０；IXEは7FH番地のためIXでアドレス修飾されな
                                    ; い。
        SKE  IXM, ＃0111B           ; ロウ・アドレス７になったか。
        BR   LOOP                   ; ７でなければLOOP（ロウ・アドレスはクリアしない）
```

例４．配列の処理

図６－14に示すように，Aという８ビットのデータを１次元で定義したとする。

このとき

A（N）＝A（N）＋４　（０≦N≦15）

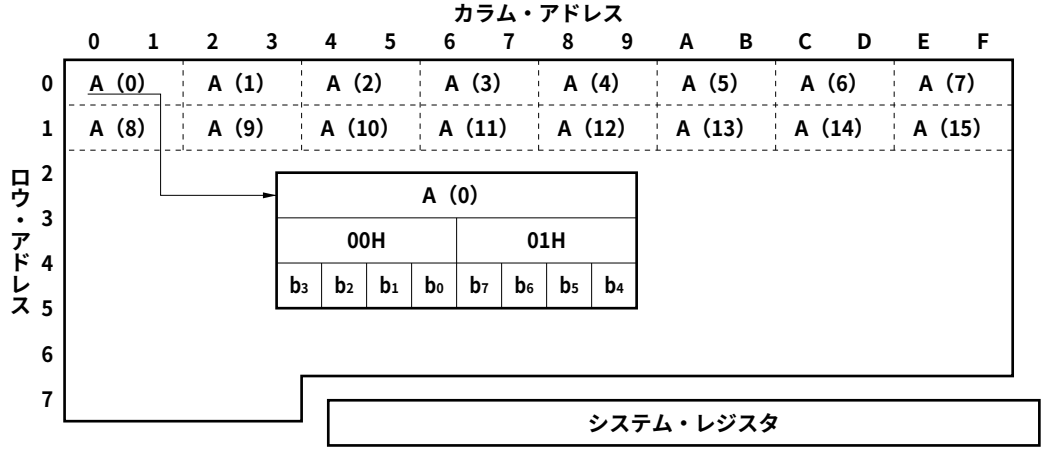
という演算を行うためには次の命令を実行すればよい。

ジェネラル・レジスタはロウ・アドレス７にあるとする。

```
M000    MEM 0.00H
M001    MEM 0.01H
MOV     IXH, #0                 ; IX←2N
MOV     IXM, #N SHR 3           ; なぜなら配列要素は８ビットであるためインデクス修飾するデ
MOV     IXL, #N SHL 1 AND 0FH   ; ータ・メモリ・アドレスをシフトする。
OR      PSW, #.DF.IXE AND 0FH   ; IXE←１
ADD     M000, #4                ; IXでインデクス修飾されるデータ・メモリM000およびM001に
ADDC    M001, #0                ; ４を加える。すなわちA（N）で指定される８ビットの配列に４
                                ; を加える。
```

上記の例のように配列A（N）のNの指定をするには，インデクス・レジスタにNの２倍の値を指定すればよいことになります。

図６－14　MPE＝０，IXE＝１のときの動作例（配列の処理）



６.６.６　MPE = １，IXE = １のとき

表６－１で示したようにデータ・メモリ操作命令を行うと，命令のオペランド“ m ”で直接指定されたデータ・メモリのアドレスが，すべてインデクス・レジスタによりアドレス修飾されます。

★　ジェネラル・レジスタ間接転送命令（MOV @r, mおよびMOV m, @r）を行ったときは，“ m ”で指定する直接アドレスがインデクス・レジスタで修飾され，“ @r ”で指定する間接アドレスをデータ・メモリ・ロウ・アドレス・ポインタで指定します。

例　BANK0でジェネラル・レジスタのロウ・アドレス０のとき

```
R005    MEM 0.05H
R034    MEM 0.34H
MOV     IXL, #0001B             ; (IX) ←00010000001B
MOV     IXM, #1000B             ; (MP) ←0001000B
MOV     IXH, #0000B             ;
MOV     R005, #8                ; R005← 8
OR      IXH, #1000B             ; MPE← 1
★ OR    PSW, #.DF.IXE AND 0FH   ; IXE← 1
MOV     @R005, M034             ; レジスタ間接転送
```

例の命令を実行すると，図６－15のようにデータ・メモリのBANK1の35H番地の内容を，データ・メモリのBANK1の08H番地に転送します。

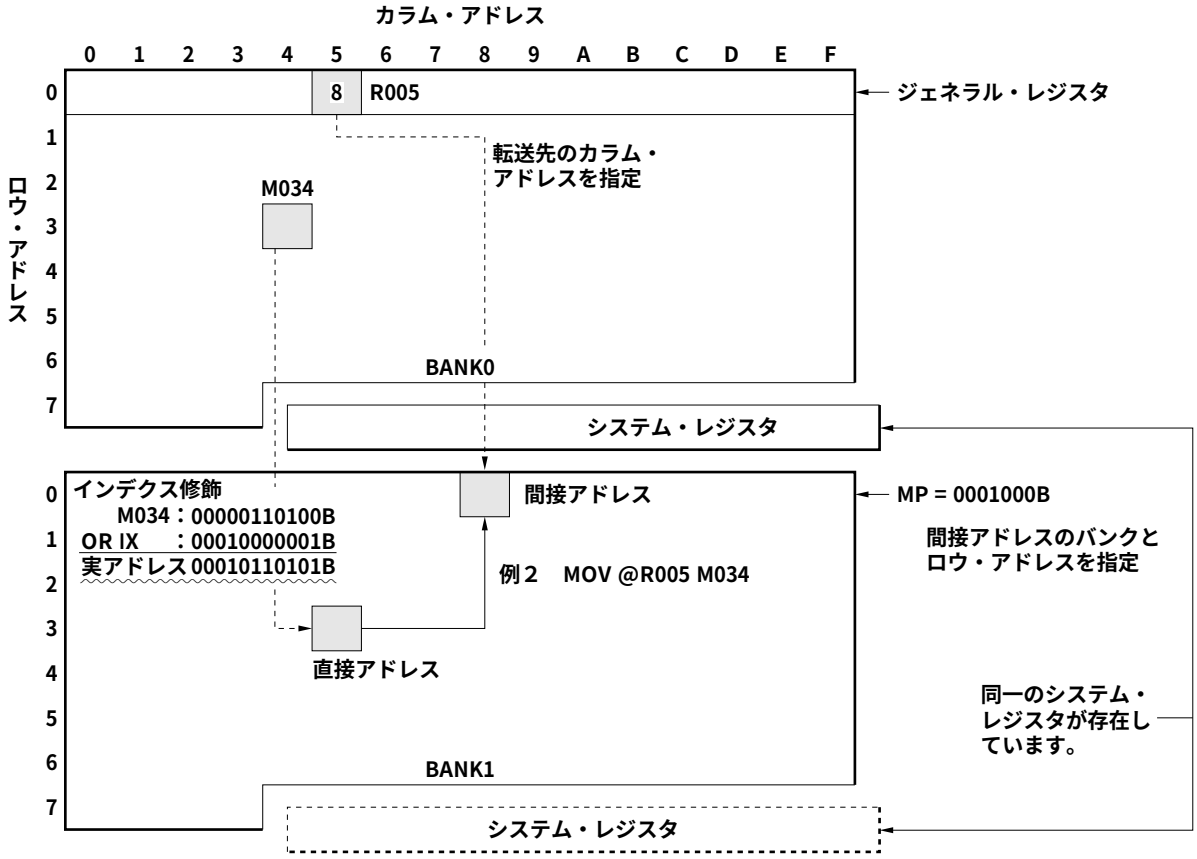
すなわち，MPE = 1，IXE = 1のときに“ MOV @r, m ”命令を実行すると，“ m ”で指定されるデータ・メモリ・アドレス（直接アドレス）をインデクス・レジスタの内容でアドレスを修飾し，“ @r ”で指定される間接アドレスはデータ・メモリ・ロウ・アドレス・ポインタの内容で指定します。

直接アドレスの修飾の方法は，“ m ”で指定されるデータ・メモリ・アドレスのバンク，ロウ・アドレスおよびカラム・アドレスのすべてをインデクス・レジスタの内容とOR演算し，@rで指定される間接アドレスは，データ・メモリ・ロウ・アドレス・ポインタの内容がバンクとロウ・アドレスになります。

すなわち，上記では直接アドレスがBANK1の35Hとなり，間接アドレスがBANK1の08H番地になります。

６.６.５のMPE = ０，IXE = １のときの例２との違いは，“ @r ”で指定される間接アドレスのバンクとロウ・アドレスをデータ・メモリ・ロウ・アドレス・ポインタの内容で指定しています（６.６.５の例２では間接アドレスはインデクス・レジスタで修飾されます）。

図６－15　MPE＝1，IXE＝1のときのジェネラル・レジスタ間接転送動作例



6.7　ジェネラル・レジスタ・ポインタ（RP）

6.7.1　ジェネラル・レジスタ・ポインタの構成

図６－16のように，ジェネラル・レジスタ・ポインタはシステム・レジスタの7DH番地（RPH）の４ビットと7EH番地（RPL）の上位３ビットの計７ビットで構成されています。

図６－16　ジェネラル・レジスタ・ポインタの構成

アドレス	7DH				7EH			
名称	ジェネラル・レジスタ・ポインタ（RP）							
記号	RPH				RPL			
ビット	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
データ	MSB						LSB	BCD

6.7.2　ジェネラル・レジスタ・ポインタの機能

ジェネラル・レジスタ・ポインタはデータ・メモリ上でジェネラル・レジスタを指定します。

ジェネラル・レジスタには，データ・メモリ上の同一ロウ・アドレスである16ニブルを指定できます。したがって，どのロウ・アドレスをジェネラル・レジスタにするのかを，図６－17のようにジェネラル・レジスタ・ポインタによって指定します。

ジェネラル・レジスタとして指定できるデータ・メモリのロウ・アドレスは，各品種のジェネラル・レジスタ・ポインタ（RP）によって決まります。

データ・メモリをジェネラル・レジスタに指定すると，ジェネラル・レジスタとデータ・メモリとの演算および転送ができます。

たとえば，次の命令を実行すると，命令のオペランド“ r ”でアドレスを指定するジェネラル・レジスタと，“ m ”でアドレスを指定するデータ・メモリとの間で加算や転送ができます。

```
ADD r, m
LD  r, m
```

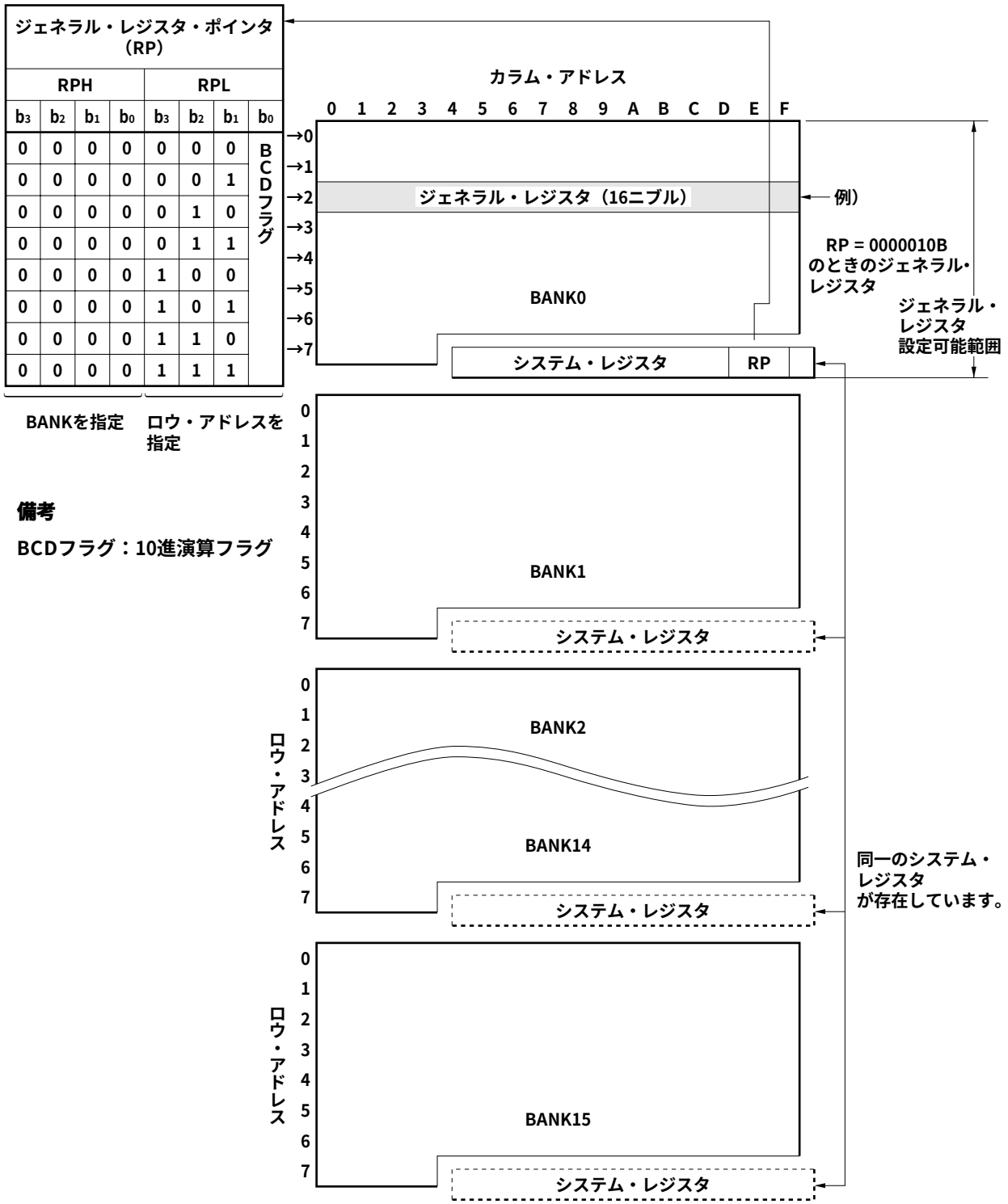
詳細は**第７章　ジェネラル・レジスタ（GR）**を参照してください。

★　6.7.3　ジェネラル・レジスタ・ポインタ使用時の注意

ジェネラル・レジスタ・ポインタであるアドレス7EH番地（RPL）の最下位ビットはプログラム・ステータス・ワードのBCDフラグに割り当てられています。

したがって，RPLを書き換える場合は，BCDフラグの値に注意してください。

図６－17　ジェネラル・レジスタの構成



備考

BCDフラグ：10進演算フラグ

６.８　プログラム・ステータス・ワード（PSWORD）

６.８.１　プログラム・ステータス・ワードの構成

図６－18のように，プログラム・ステータス・ワードはシステム・レジスタの7EH番地（RPL）の最下位ビットと7FH番地（PSW）の４ビットの計５ビットで構成されています。

プログラム・ステータス・ワードはさらに１ビットずつ機能が分かれており，それぞれバイナリ・デシマル・フラグ（BCD），コンペア・フラグ（CMP），キャリー・フラグ（CY），ゼロ・フラグ（Z）およびインデクス・イネーブル・フラグ（IXE）から構成されています。

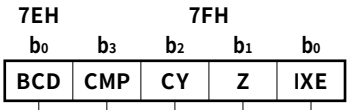
図６－18　プログラム・ステータス・ワードの構成

アドレス	7EH				7FH			
名称	（RP）			プログラム・ステータス・ワード（PSWORD）				
記号	RPL				PSW			
ビット	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
データ				BCD	CMP	CY	Z	IXE

６.８.２　プログラム・ステータス・ワードの機能

プログラム・ステータス・ワードの各フラグは，ALU（Arithmetic Logic Unit）での演算および転送命令の条件を設定したり，演算結果の状態を示すために使用します。図６－19にプログラム・ステータス・ワードの機能概要を示します。

図６－19　プログラム・ステータス・ワードの機能概要



インデクス・イネーブル・フラグ	このフラグがセットされるとインデクス修飾が可能です。
ゼロ・フラグ	算術演算の結果が“０”以外のときにリセットされる。セットの条件はCMPフラグの内容により異なる。 （１）CMP＝０のとき 算術演算の結果が“０”のときにセットされる。 （２）CMP＝１のとき Z＝１でかつ実行した算術演算の結果が“０”のときのみセットされる。
キャリー・フラグ	加算命令実行時のキャリー発生，または減算命令実行時のボロー発生でセット（１）され，それらが発生しないときにリセット（０）される。 またRORC命令が実行されるジェネラル・レジスタの最下位ビットが“１”のときセット（１）され，“０”のときにはリセット（０）される。
コンペア・フラグ	このフラグがセット（１）されているとき，算術演算の結果は，データ・メモリには格納されない。CMPフラグは，SKT, SKF命令を実行すると自動的にリセット（０）される。
BCDフラグ	このフラグがセット（１）されるとすべての算術演算は10進（BCD）で行われ，リセット（０）されると２進で行われる。

６.８.３　インデクス・イネーブル・フラグ（IXE）

IXEフラグは，データ・メモリ操作命令実行時にデータ・メモリのアドレスをインデクス修飾するために使用します。

IXEフラグをセット（１）すると，命令で指定されたデータ・メモリ・アドレスとインデクス・レジスタ（IX）の内容をOR演算し，そのOR演算結果を実アドレスとするデータ・メモリに対して命令を実行します。

詳しくは**６.６　インデクス・レジスタ（IX）とデータ・メモリ・ロウ・アドレス・ポインタ（MP：メモリ・ポインタ）**を参照してください。

６.８.４　ゼロ・フラグ（Z）とコンペア・フラグ（CMP）

Zフラグは算術演算の結果が０であることを示すフラグであり，CMPフラグは算術演算の結果をデータ・メモリやジェネラル・レジスタへ格納しないように設定するためのフラグです。

Zフラグのセットおよびリセット条件はCMPフラグの状態により表６－２のようになります。

表６－２　コンペア・フラグ（CMP）の状態とゼロ・フラグ（Z）のセット，リセット条件

条　　件	Zフラグの状態	
	CMPフラグが０のとき	CMPフラグが１のとき
リセット時	リセット	CMPフラグとともにリセット
データ・メモリ操作命令でZフラグに直接“０”が書き込まれたとき	リセット	リセット
データ・メモリ操作命令でZフラグに直接“１”が書き込まれたとき	セット	セット
算術演算の結果が“０”以外になったとき	リセット	リセット
算術演算の結果が“０”になったとき	セット	以前のZフラグの状態保持

ZフラグおよびCMPフラグはジェネラル・レジスタの内容とデータ・メモリの内容の比較を行うときなどに使用します。Zフラグは算術演算以外，CMPフラグはビット判断以外では変化しません。

６.８.５　キャリー・フラグ（CY）

CYフラグは加算命令および減算実行後のキャリーまたはボローの発生を示すフラグです。

CYフラグは，算術演算の結果，キャリーまたはボローが発生するとセット（１）され，発生しなければリセット（０）されます。

また，“ RORC r ”命令（rでアドレス指定されるジェネラル・レジスタの内容を右に１ビット分シフトする）を実行したとき，命令を実行する直前のCYフラグの値をジェネラル・レジスタの最上位ビットにシフトし，最下位ビットをCYフラグにシフトします。

CYフラグはキャリーやボローが発生した場合に次の命令をスキップしたいときなどに使用すると便利です。

★　算術演算および回転処理以外では変化しません。

６.８.６　バイナリ・コーデッド・デシマル・フラグ（BCD）

BCDフラグはBCD演算を行うためのフラグです。

BCDフラグをセット（１）することにより，すべての算術演算がBCD演算で行われます。リセット（０）すると，２進４ビットで行われます。

★　論理演算，ビット判断，比較判断，回転処理には影響を与えません。

６.８.７　算術演算時の注意

プログラム・ステータス・ワード（PSWORD）に対して算術演算（加算および減算）を行うと，算術演算の“ 結果 ”が格納されるので注意してください。

例
```
MOV   PSW,   #0001B
ADD   PSW,   #1111B
```

上記の命令を実行するとキャリーが発生するためPSWのビット２であるCYフラグがセット（１）されるように見えますが，算術演算の結果は0000Bであるため PSWには0000Bが格納されます。

６．９　システム・レジスタ使用時の注意

６．９．１　システム・レジスタの予約語

システム・レジスタはデータ・メモリ上に配置されているため，すべてのデータ・メモリ操作命令を使用できます。このとき，17Kシリーズのアセンブラ（RA17K）を使用するうえでは例１に示すように，命令のオペランドには直接データ・メモリ・アドレスを記述できないため，あらかじめデータ・メモリ・アドレスをシンボル定義しておく必要があります。

ここで，システム・レジスタはデータ・メモリでもありますが，通常の汎用データ・メモリと異なり専用の機能を持っているため，アセンブラ（RA17K）上であらかじめ“予約語”としてシンボル定義されています。

システム・レジスタの予約語はアドレス74H-7FH番地に割り当てられており，**図６－２　システム・レジスタの構成**に示した記号（AR3, AR2……PSW）で定義されています。

この予約語を使用すれば，例２に示すようにシンボル定義する必要はありません。

予約語については各品種のデータ・シートを参照してください。

```
例１．MOV     34H, ＃0101B   ；オペランドにデータ・メモリ・アドレスを34Hや76Hと記述すると，アセ
      MOV     76H, ＃1010B   ；ンブラでエラーとなる。
      M037    MEM 0.37H      ；汎用データ・メモリは，MEM疑似命令でデータ・メモリ・アドレスをシ
      MOV     M037, ＃0101B  ；ンボル定義する必要がある。

    ２．MOV     AR1, ＃1010B   ；予約語であるAR1（アドレス6H）を使用すればシンボル定義の必要はな
                               ；い。
                               ；予約語AR1は“AR1 MEM 0.76H”としてデバイス・ファイルにて定義さ
                               ；れている。
```

また，アセンブラ（RA17K）を使用する場合はアセンブラ内部にフラグ型シンボル操作命令として次のマクロ命令が組み込まれています。

```
SETn    ：フラグに“１”をセット
CLRn    ：フラグを“０”にリセット
SKTn    ：フラグがすべて“１”であればスキップ
SKFn    ：フラグがすべて“０”であればスキップ
NOTn    ：フラグを反転
INITFLG ：フラグをイニシャライズ
```

したがって，これらの組み込みマクロ命令を使用することにより，次の例３に示すようにデータ・メモリをフラグとして操作することができます。

★　プログラム・ステータス・ワードおよびメモリ・ポインタ・イネーブル・フラグはビット単位（フラグ単位）で機能が分けられているため，それぞれのビットに予約語が定義されています。この予約語はMPE, BCD, CMP, CY, Z, IXEとなります。

したがって，このフラグ型予約語を使用すれば例４に示すようにそのまま組み込みマクロ命令を使用できます。

例３．
```
F0003   FLG 0.00.3      ；フラグ型シンボル定義
        SET1 F0003      ；組み込みマクロ
```

マクロ展開
```
OR      .MF.F0003 SHR 4, #.DF.F0003 AND 0FH
                ；BANK0のアドレス00Hのビット３をセットする。
```

４．
```
        SET1 BCD        ；組み込みマクロ
```

マクロ展開
```
OR      .MF.BCD SHR 4, #.DF.BCD AND 0FH
                ；BCD フラグをセット
                ；BCDは“ BCD FLG 0.7EH.0 ”で定義されている。
```

```
        CLR2 Z, CY      ；同一アドレスのフラグ
```

マクロ展開
```
AND     .MF.Z SHR 4, #.DF.（NOT（Z OR CY）AND 0FH）
```

```
        CLR2 Z, BCD     ；異なるアドレスのフラグ
```

マクロ展開
```
AND     .MF.Z SHR 4, #.DF.（NOT Z AND 0FH）
AND     .MF.BCD SHR 4, #.DF.（NOT BCD AND 0FH）
```

６.９.２　“０”に固定されているシステム・レジスタの取り扱い

システム・レジスタの中であらかじめ“０”に固定されているデータ（**図６－２　システム・レジスタの構成**参照）については，デバイス動作，エミュレータ動作およびアセンブラ動作に注意が必要です。

これを，（１），（２）および（３）に示します。

（１）デバイス動作上

“０”に固定されているデータに書き込み命令を行っても，何ら影響を受けません。また読み出し命令を行ったときは常に“０”が読み出されます。

★ **（２）17Kシリーズのインサーキット・エミュレータ（IE-17K, IE-17K-ET）を使用するとき**

“０”に固定されているデータに“１”を書き込む命令が実行されると，エラーが発生します。

すなわち次に示すような命令が実行されると，インサーキット・エミュレータはエラーを発生します。

```
例１．MOV    BANK, ＃0100B  ；０に固定されているビット３に１を書き込む。

２．MOV    IXL, ＃1111B    ;
    MOV    IXM, ＃1111B    ;
    MOV    IXH, ＃0001B    ;
    ADD    IXL, ＃１       ;
    ADDC   IXM, ＃０       ;
    ADDC   IXH, ＃０       ;
```

ただし，“INC AR”および“INC IX”命令については，例２のように有効ビットがすべて“１”のときに実行されても，インサーキット・エミュレータはエラーを発生しません。これは，アドレス・レジスタとインデクス・レジスタの有効ビットがすべて“１”のときに“INC”命令が実行されるとこの命令により有効ビットがすべて“０”になるためです。

また，アドレス・レジスタに限り，上記の例１，例２のように“０”で固定されているデータに“１”を書き込んでも，インサーキット・エミュレータはエラーを発生しません。

★　（３）17Kシリーズのアセンブラ（RA17K）を使用するとき

“０”に固定されているデータに“１”を書き込む命令があってもエラーは出力されません。すなわち，例１で示した，

```
MOV     BANK, ＃0100B
```

命令を用いると，アセンブラ・エラーは発生ぜず，インサーキット・エミュレータ上で命令が実行されたときに初めてエミュレータ・エラーを発生します。

★　アセンブラ（RA17K）でエラーが発生しない理由としては，レジスタ間接転送等が行われるときは，データ・メモリ操作命令の対象となるデータ・メモリ・アドレスを判断できないなどの理由によります。

したがって，アセンブラ・エラーを発生するのは次の場合のみです。

“BANKn”組み込みマクロ命令で“n”に１以上の値を用いたとき

これは，BANKn命令を使用するときは，明らかにシステム・レジスタのバンク・レジスタを操作したいという意志のもとで使用されていると判断するためです。

第７章　ジェネラル・レジスタ（GR）

ジェネラル・レジスタは，データ・メモリ上に配置されるレジスタで，データ・メモリと直接，演算や転送を行います。

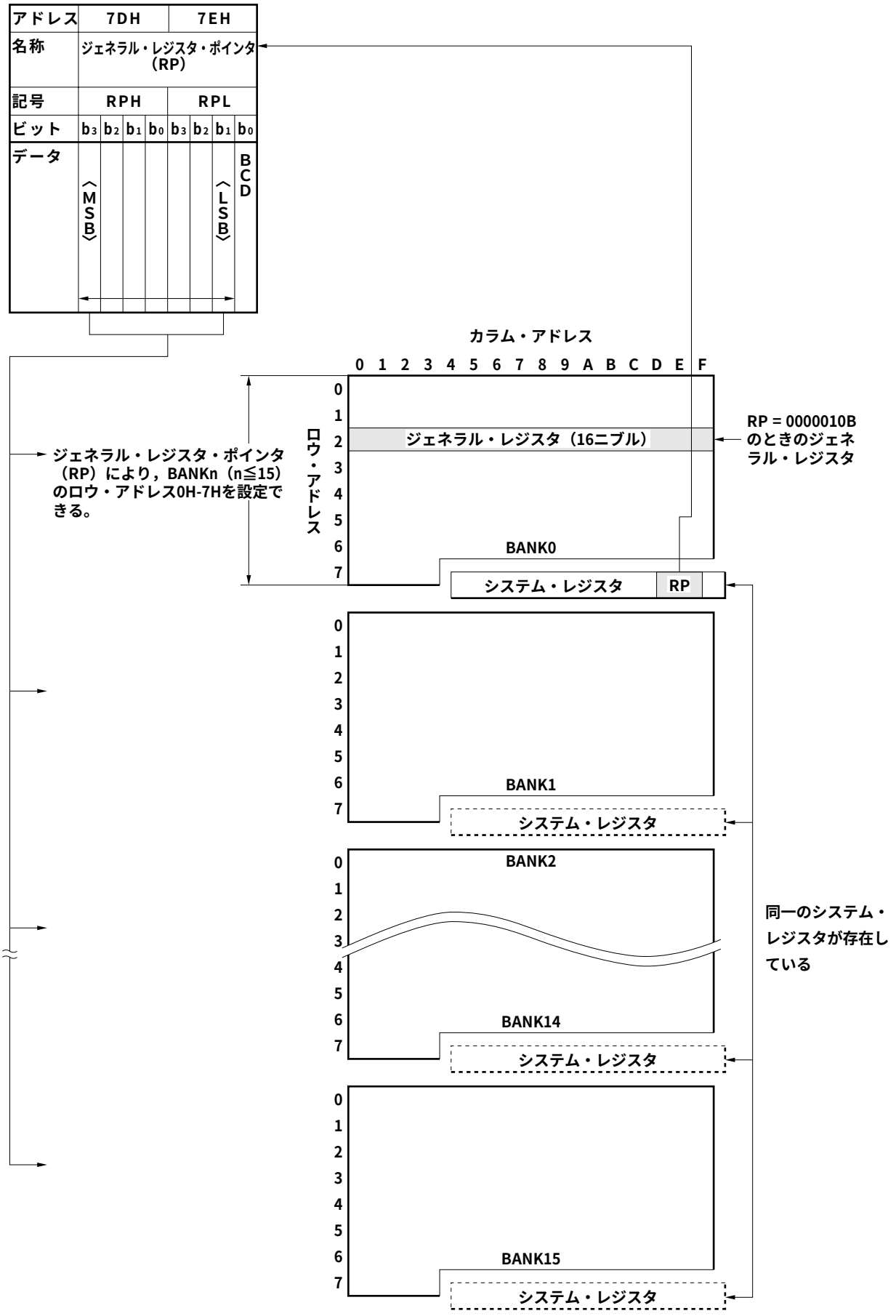
７.１　ジェネラル・レジスタの構成

ジェネラル・レジスタは，図７－１のようにデータ・メモリ上の16ニブル（16ワード×４ビット）で構成されます。

プログラムで任意のロウ・アドレスの１列をジェネラル・レジスタとして指定できます。指定のロウ・アドレスはシステム・レジスタのジェネラル・レジスタ・ポインタに設定します。

ジェネラル・レジスタ・ポインタについては，**６.７　ジェネラル・レジスタ・ポインタ（RP）**を参照してください。

図７－１　ジェネラル・レジスタの構成



７.２　ジェネラル・レジスタの機能

ジェネラル・レジスタを使うことにより，データ・メモリとジェネラル・レジスタとの間で演算や転送を１命令で行うことができます。

ジェネラル・レジスタはすなわちデータ・メモリであるため，言い換えれば１命令でデータ・メモリ同士の演算や転送が可能になります。

また，ジェネラル・レジスタは，データ・メモリ上にあるので他のデータ・メモリと同様にデータ・メモリ操作命令で制御できます。

データ・メモリ操作命令については**７.４　各命令におけるジェネラル・レジスタとデータ・メモリのアドレス指定と動作**を参照してください。

７.３　ジェネラル・レジスタ使用時の注意

ジェネラル・レジスタ使用時の注意を**７.３.１**-**７.３.３**に示します。

７.３.１　ジェネラル・レジスタのアドレス指定

★　17Kシリーズのアセンブラ（RA17K）を使用するとき，下記のように命令のオペランドにジェネラル・レジスタ・アドレスを直接指定するとエラーが発生します。

これは，データ・メモリと同様，プログラム修正時などのバグ要因を低減する目的で組み込まれています。

★　したがって，ジェネラル・レジスタのアドレスをあらかじめシンボルとして定義する必要があります。

例　エラーになる場合

```
LD      04H, 32H      ；ジェネラル・レジスタ・アドレスやデータ・メモリ・アドレスを直接数値で記
                      ；述する。
```

エラーにならない場合

```
R004   MEM   0.04H；アドレス04Hをメモリ型としてR004にシンボル定義する。
M032   MEM   0.32H；
LD     R004,  M032 ；
```

７.３.２　ジェネラル・レジスタのロウ・アドレスについて

ジェネラル・レジスタのロウ・アドレスは，ジェネラル・レジスタ・ポインタにより決まるため，命令のオペランド“ｒ”で指定するアドレスのバンクとロウ・アドレスは無視されます。

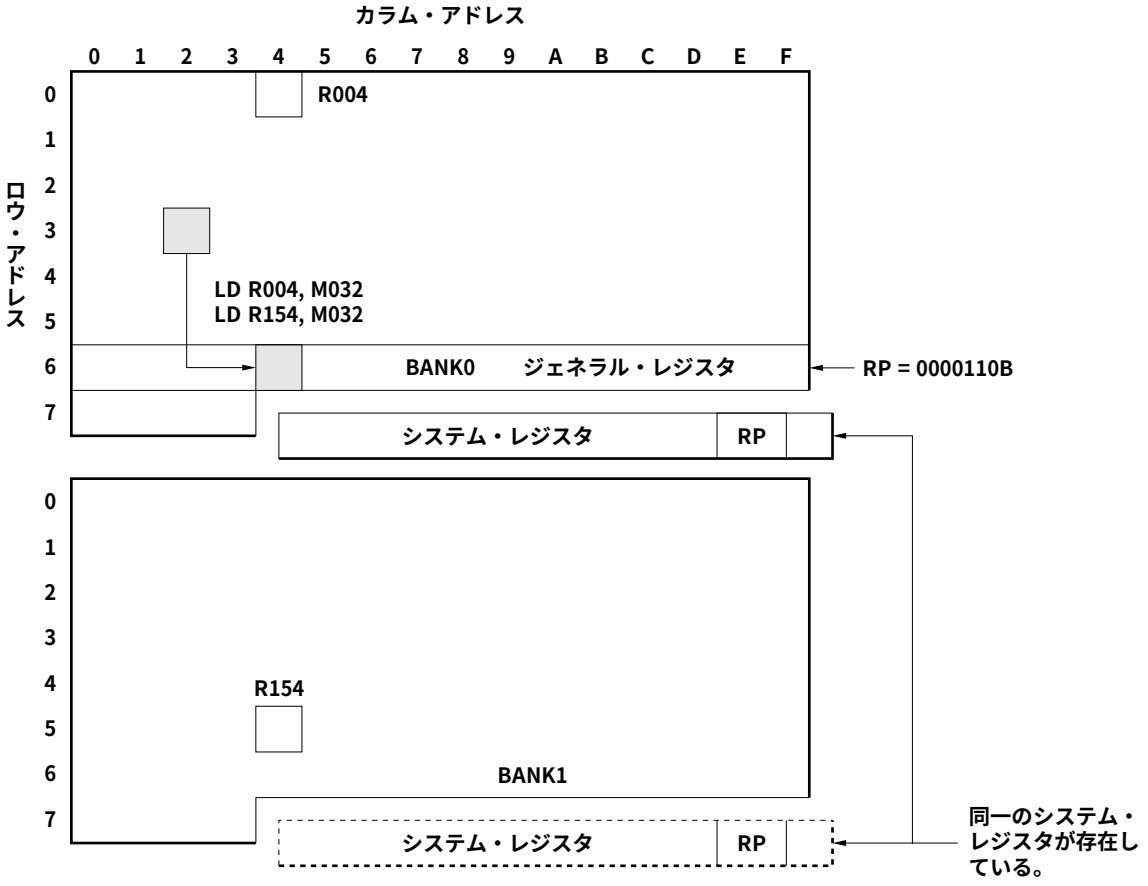
次のプログラム例を実行すると，図７－２のように①，②ともにジェネラル・レジスタであるBANK0の64H番地にデータ・メモリM032（BANK0の32H番地）の内容を転送します。

すなわち，①および②の命令において，ジェネラル・レジスタ・アドレスを指定するR004およびR154のバンクとロウ・アドレスは無視され，カラム・アドレスの4H番地のみ有効になります。

例　ジェネラル・レジスタのロウ・アドレス指定例

```
BANK0であるとする

R004   MEM 0.04H    ;
R154   MEM 1.54H    ;
M032   MEM 0.32H    ;
MOV    RPH, #0000B ;
MOV    RPL, #0110B ; RP←0000110B
;①
   LD   R004, M032
;②
   LD   R154, M032
```

図７－２　ジェネラル・レジスタのロウ・アドレス指定例



７.３.３　イミーディエト・データとの演算

ジェネラル・レジスタの内容とイミーディエト・データとの演算命令はありません。したがって，ジェネラル・レジスタに指定しているデータ・メモリの内容とイミーディエト・データとの演算命令を行うときは，ジェネラル・レジスタではなくデータ・メモリとして演算をするプログラムを組んでください。

例　ジェネラル・レジスタとイミーディエト・データの演算例

```
  BANK0であるとする

  R065     MEM  0.65H   ;
  M105     MEM  1.05H   ;
;①
    MOV   RPH, #0001B ;ジェネラル・レジスタをBANK0のロウ・アドレス6H番地に設定
    MOV   RPL, #0100B ;
★  BANK1               ;アセンブラ（RA17K）組み込みマクロ命令
;②
    ADD   R065, #3     ;
;③
    ADD   M105, #3     ;
```

このプログラムは，②ではBANK1の65H番地のデータ・メモリにイミーディエト・データの３を加算し，③ではBANK1の05H番地のデータ・メモリに３を加算します。
すなわち，①でジェネラル・レジスタをBANK0のロウ・アドレス6H番地に設定しているが，②の命令のオペランドであるR065はジェネラル・レジスタではなくデータ・メモリとして扱われます。
したがって②の命令でジェネラル・レジスタであるBANK0のアドレス6Hに加算を行いたい場合は，次のようなプログラムにする必要があります。

```
★  BANK0               ;アセンブラ（RA17K）組み込みマクロ命令
   ADD   R065, #3
```

７.４　各命令におけるジェネラル・レジスタとデータ・メモリのアドレス指定と動作

ジェネラル・レジスタとデータ・メモリとの演算および転送命令を表７－１に示します。

これらの命令によるアドレス指定は，“ADD r, m”命令を例にとると，図７－３のようにレジスタ・ポインタの内容と命令のオペランドrの値とでジェネラル・レジスタRを指定します。

また，この命令でデータ・メモリのアドレス“M”はバンク・レジスタの内容と命令のオペランドmとで指定されます。

したがって，この命令はこれらのジェネラル・レジスタの内容“（R）”とデータ・メモリの内容“（M）”とを加算し，結果をジェネラル・レジスタRに格納します。

上述したジェネラル・レジスタのアドレスの指定は，表７－１の他の命令でも同様です。例１，２，３はその動作例です。

表７－１　ジェネラル・レジスタとデータ・メモリ間の操作命令

命令群	命　　令	動　　作
加算	ADD r, m	(R)←(R)＋(M)
	ADDC r, m	(R)←(R)＋(M)＋(CY)
減算	SUB r, m	(R)←(R)－(M)
	SUBC r, m	(R)←(R)－(M)－(CY)
論理演算	AND r, m	(R)←(R)AND(M)
	OR r, m	(R)←(R)OR(M)
	XOR r, m	(R)←(R)XOR(M)
転送	LD r, m	(R)←(M)
	ST m, r	(M)←(R)
	MOV @r, m	[MP, (R)]←(M) or [m, (R)]←(M)
	MOV m, @r	M←[MP, (R)] or M←[H, (R)]
シフト	RORC r	(CY)を含めた右シフト

図７－３　ジェネラル・レジスタとデータ・メモリのアドレス指定例

命　　令	アドレス内容	生成アドレス										
		記号	バンク				ロウ・アドレス			カラム・アドレス			
			b_3	b_2	b_1	b_0	b_2	b_1	b_0	b_3	b_2	b_1	b_0
ADD r, m	rで指定するジェネラル・レジスタのアドレス	R	（RP）							r			
	mで指定するデータ・メモリのアドレス	M	（BANK）				m						

例１．データ・メモリとジェネラル・レジスタとの演算

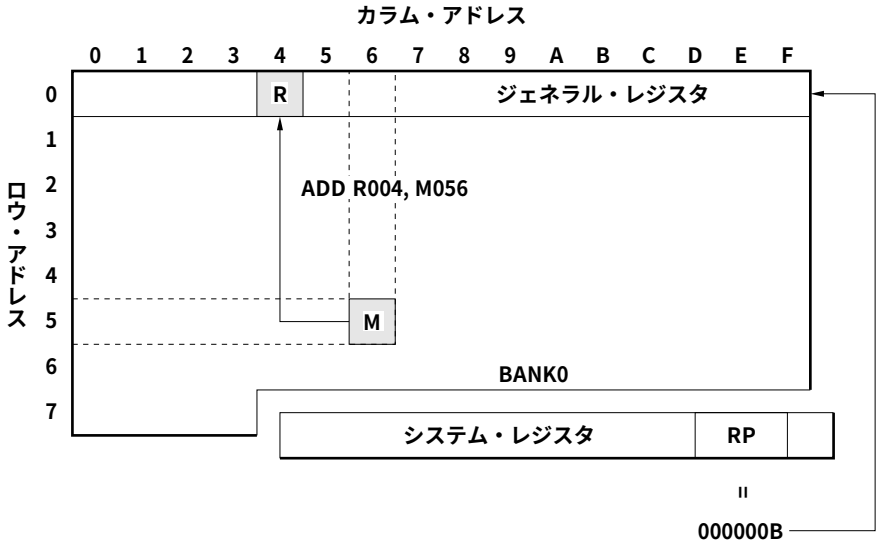
（1）データ・メモリとジェネラル・レジスタのバンクが等しいとき

BANK0でジェネラル・レジスタがBANK0のロウ・アドレス0Hであるとします。

```
R004   MEM  0.04H ；シンボル定義
M056   MEM  0.56H ；
ADD    R004, M056 ；データ・メモリとジェネラル・レジスタの加算
```

このプログラムを実行すると，図７－４のようにジェネラル・レジスタであるR004（BANK0の04H番地）の内容とデータ・メモリM056（56H番地）の内容を加算して結果をジェネラル・レジスタR004（04H）に格納します。

図７－４　データ・メモリとジェネラル・レジスタの演算例１



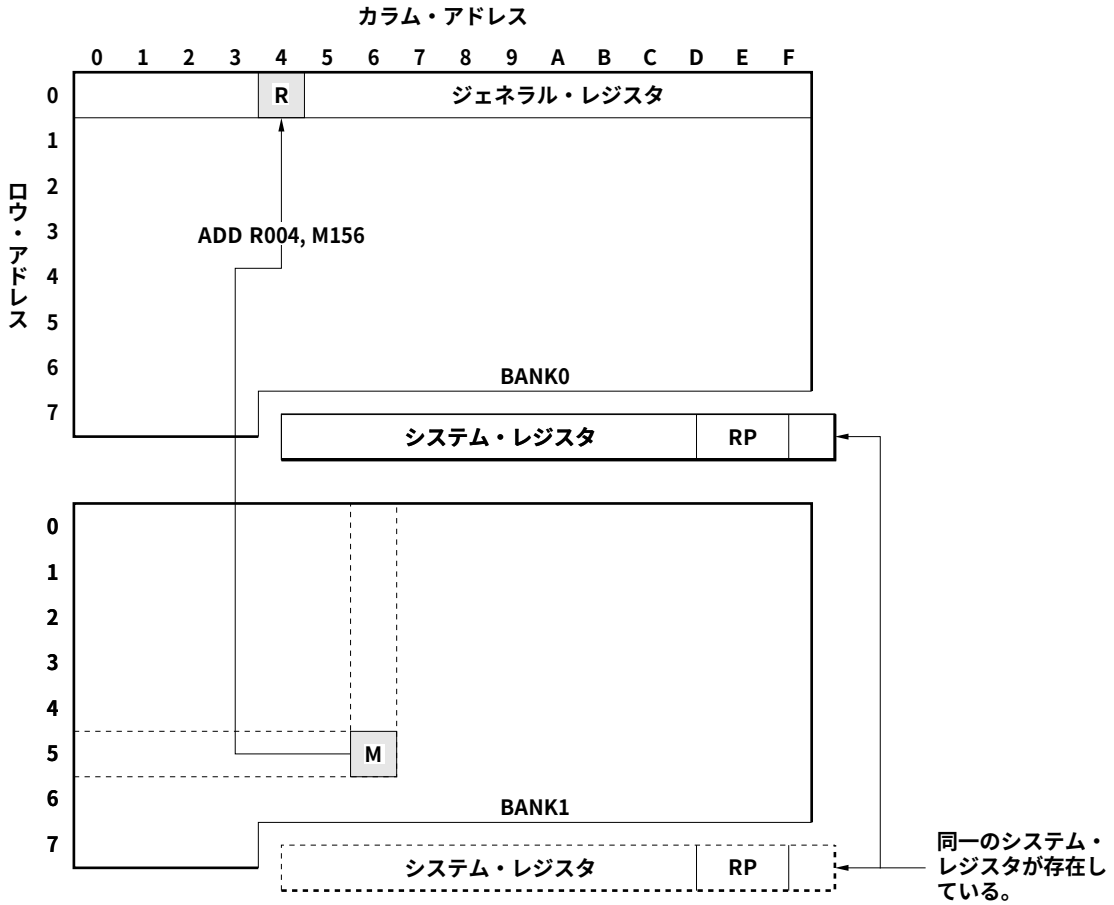
（2）データ・メモリとジェネラル・レジスタのバンクが異なるとき

BANK1でジェネラル・レジスタがBANK0のロウ・アドレス0Hであるとします。

```
R004   MEM  0.04H ；シンボル定義
M156   MEM  1.56H ；
BANK1             ；アセンブラ（RA17K）組み込みマクロ命令
ADD    R004, M156 ；データ・メモリとジェネラル・レジスタの加算
```

上の命令を実行すると，図７－５のようにジェネラル・レジスタであるR004（BANK0の04H番地）の内容とデータ・メモリM156（BANK1の56H番地）の内容を加算して結果をジェネラル・レジスタR004（04H）に格納します。
すなわち，選択されているバンクはBANK1ですがジェネラル・レジスタはBANK0のため結果としてBANK1とBANK0のデータ・メモリを１命令で加算したことになります。

図７－５　データ・メモリとジェネラル・レジスタの演算例２



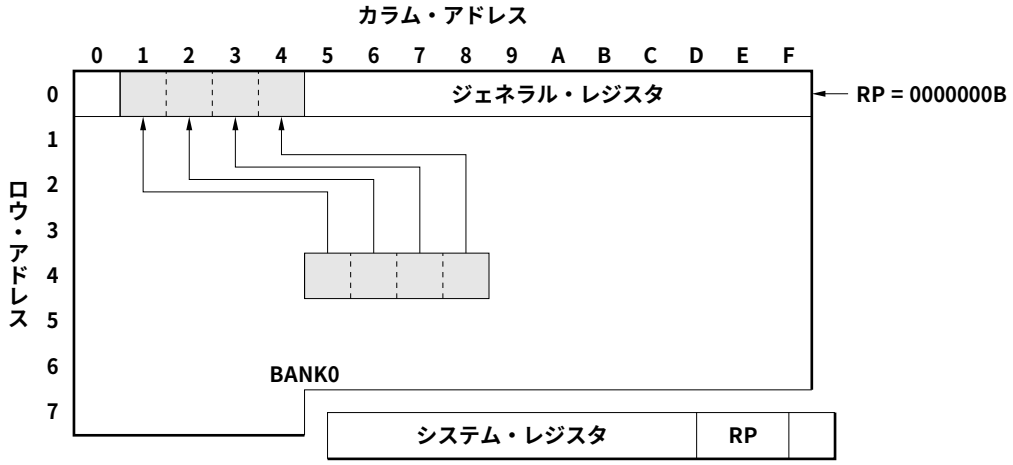
例２．ジェネラル・レジスタへの転送

BANK0で，ジェネラル・レジスタがロウ・アドレス0Hにあるとします。

```
R001   MEM  0.01H   ; シンボル定義
R002   MEM  0.02H   ;
R003   MEM  0.03H   ;
R004   MEM  0.04H   ;
M045   MEM  0.45H   ;
M046   MEM  0.46H   ;
M047   MEM  0.47H   ;
M048   MEM  0.48H   ;
LD     R001, M045
LD     R002, M046
LD     R003, M047
LD     R004, M048
```

上のプログラムは図７－６のようにデータ・メモリM045, M046, M047 M048（アドレス45H, 46H, 47H, 48H）の内容を，それぞれジェネラル・レジスタR001, R002, R003, R004（アドレス01H, 02H, 03H, 04H）に転送します。

図７－６　ジェネラル・レジスタへの転送例



例３．ジェネラル・レジスタ間接転送

BANK0でジェネラル・レジスタがBANK0のロウ・アドレス0Hであるとします。

```
R004   MEM  0.04H   ;
M052   MEM  0.52H   ;
MOV    R004, #8     ; (R004) ←8
MOV    @R004, M052 ; ジェネラル・レジスタ間接転送
```

上の命令を実行すると，図７－７のようにデータ・メモリM052（アドレス52H番地）内容を間接転送先のデータ・メモリ（この例ではアドレス58H番地）に転送します。

“ MOV @r, m ”命令はジェネラル・レジスタ間接転送と呼ばれ，mでアドレス指定されるデータ・メモリの内容を，@rでアドレス指定される（間接アドレスと呼ぶ）データ・メモリに転送します。

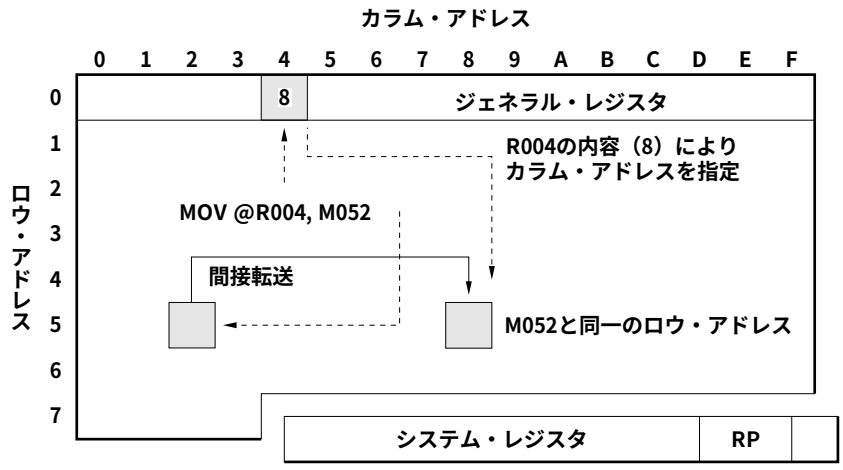
★ このとき，@rで指定される間接転送先のデータ・メモリ・アドレス（間接アドレス）は，次のようになります。

- ロウ・アドレス　…mで指定するデータ・メモリと同一のロウ・アドレス
- カラム・アドレス…rで指定するジェネラル・レジスタの内容

例３では，ロウ・アドレスが5H（アドレス52H番地のロウ・アドレス），カラム・アドレスが8H（アドレス08番地の内容が８）になり，データ・メモリ・アドレスが58H番地になります。

ジェネラル・レジスタ間接転送については**６.６　インデクス・レジスタ（IX）とデータ・メモリ・ロウ・アドレス・ポインタ（MP：メモリ・ポインタ）**も参照してください。

図７－７　ジェネラル・レジスタ間接転送例



例４．ジェネラル・レジスタのロウ・アドレスを変更

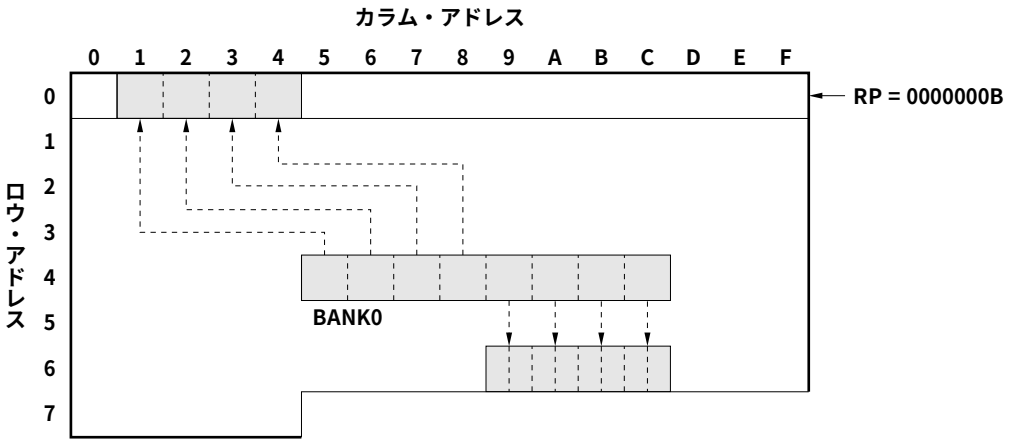
BANK0でジェネラル・レジスタがBANK0のロウ・アドレス0Hであるとします。

```
R001    MEM  0.01H     ;シンボル定義
R002    MEM  0.02H     ;
R003    MEM  0.03H     ;
R004    MEM  0.04H     ;
R005    MEM  0.05H     ;
R006    MEM  0.06H     ;
R007    MEM  0.07H     ;
R008    MEM  0.08H     ;
M045    MEM  0.45H     ;
M046    MEM  0.46H     ;
M047    MEM  0.47H     ;
M048    MEM  0.48H     ;
M049    MEM  0.49H     ;
M04A    MEM  0.4AH     ;
M04B    MEM  0.4BH     ;
M04C    MEM  0.4CH     ;
LD      R001, M045
LD      R002, M046
LD      R003, M047
LD      R004, M048
;①
    MOV RPH,  #0000B ;ジェネラル・レジスタ・ポインタに0000110Bを転送。すなわち，
    MOV RPL,  #0110B ;BANK0のロウ・アドレス6Hに設定。
    LD    R005, M049
    LD    R006, M04A
    LD    R007, M04B
    LD    R008, M04C
```

上のプログラムは図７－８のようにBANK0にある８ニブルのデータ・メモリM045-M04Cの内容を，BANK0の異なるロウ・アドレスに４ニブルずつ転送するプログラムです。このとき，ジェネラル・レジスタが固定で，たとえばBANK0のロウ・アドレス０にしか存在しないような場合，上記①のプログラムを次のプログラムのように一度８ニブルすべてをジェネラル・レジスタに転送したあとに再度データ・メモリに格納する命令が必要となります。
しかし，上のプログラムのようにジェネラル・レジスタ・ポインタでジェネラル・レジスタのロウ・アドレスを替えてやればLD命令だけで動作が終了できます。

```
M065   MEM  0.65H    ;シンボル定義
M066   MEM  0.66H    ;
M067   MEM  0.67H    ;
M068   MEM  0.68H    ;
LD     R005, M049
LD     R006, M04A
LD     R007, M04B
LD     R008, M04C
BANK1                ;アセンブラ（RA17K）組み込みマクロ命令
                     ;BANK←１
ST     M065, R005
ST     M066, R006
ST     M067, R007
ST     M068, R008
```

図７－８　ジェネラル・レジスタのロウ・アドレス変更例



第８章　演算論理ユニット（ALU）

ALUは４ビット・データの算術演算，論理演算，ビット判断，比較判断および回転処理を行います。

８.１　ALUブロックの構成

図８－１にALUブロックの構成を示します。

図８－１に示すようにALUブロックは４ビットのデータ処理を行うALU本体と，ALUの周辺回路である一時記憶用レジスタA, Bと，ALUの状態を制御するステータス・フリップフロップと，BCD演算使用時の10進補正回路から構成されています。

★　ステータス・フリップフロップは図８－１に示すように，ゼロ・フラグ用FF，キャリー・フラグ用FF，コンペア・フラグ用FF，およびBCDフラグ用FFから構成されています。

ステータス・フリップフロップはシステム・レジスタの中のプログラム・ステータス・ワード（PSWORD：アドレス7EH, 7FH）の各フラグであるゼロ・フラグ（Z），キャリー・フラグ（CY），コンペア・フラグ（CMP）およびBCDフラグ（BCD）と１対１に対応しています。

８.２　ALUブロックの機能

ALUはプログラムに書かれた命令により，算術演算，論理演算，ビット判断，比較判断および回転処理を行います。表８－１に各演算，判断，および回転処理命令の一覧を示します。

表８－１に示した各命令を実行することにより４ビット単位の演算，判断および回転処理または１桁の10進算術演算が１命令で実行できます。

８.２.１　ALUの機能

算術演算には加算と減算があります。算術演算はジェネラル・レジスタの内容とデータ・メモリの内容との演算，またはデータ・メモリの内容とイミーディエト・データとの演算が行えます。また，算術演算は２進数による４ビットの演算と10進数による１桁の演算（BCD演算）が可能です。

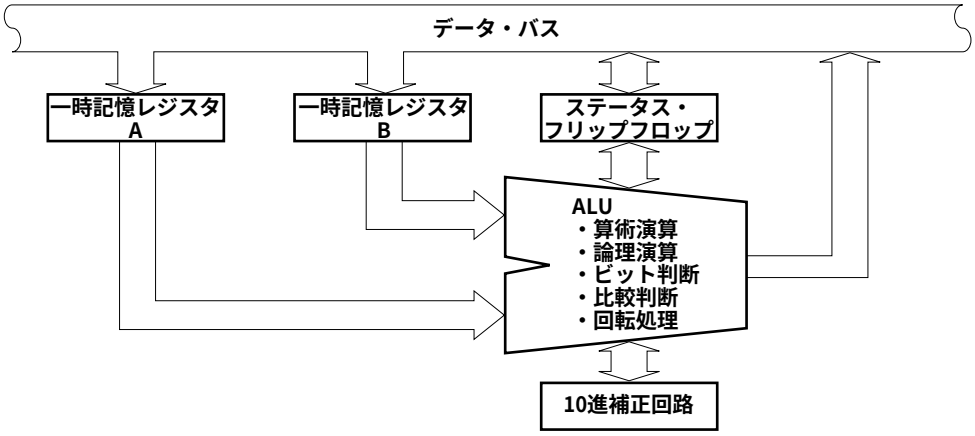
論理演算には論理積（AND），論理和（OR）および排他的論理和（XOR）があります。論理演算は，ジェネラル・レジスタの内容とデータ・メモリの内容との演算，またはデータ・メモリの内容とイミーディエト・データとの演算が行えます。

ビット判断は，データ・メモリの４ビット・データのうち“０”であるビットまたは“１”であるビットの判断を行います。

比較判断はデータ・メモリの内容とイミーディエト・データとの比較を行い，“等しい”，“等しくない”，“以上”および“未満”の判断を行います。

回転処理はジェネラル・レジスタの４ビット・データを下位ビットの方向へ１ビット，シフトします（右へ回転する）。

図８－１　ALUブロックの構成



★

アドレス	7EH				7FH
名　称	プログラム・ステータス・ワード（PSWORD）				
ビット	b_0	b_3	b_2	b_1	b_0
フラグ	BCD	CMP	CY	Z	IXE

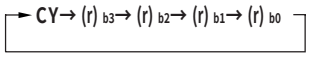
ステータス・フリップフロップ			
BCD フラグ用 FF	CMP フラグ用 FF	CY フラグ用 FF	Z フラグ用 FF

機能の概要
算術演算結果が０であることを示す
算術演算時のキャリーまたはボローを格納
算術演算結果を格納するかを指定
算術演算時に10進補正を行うかを指定

〔メ　モ〕

★

表８－１　ALU処理命令一覧（1/2）

ALU機能		命　令	動　　作	説　　明
算術演算	加算	ADD r, m	(r) ← (r) ＋ (m)	ジェネラル・レジスタとデータ・メモリの内容を加算。結果をジェネラル・レジスタへ格納。
		ADD m, #n4	(m) ← (m) ＋n4	データ・メモリとイミーディエト・データの内容を加算。結果をデータ・メモリへ格納。
		ADDC r, m	(r) ← (r) ＋ (m) ＋CY	ジェネラル・レジスタとデータ・メモリの内容をCYフラグとともに加算。結果をジェネラル・レジスタへ格納。
		ADDC m, #n4	(m) ← (m) ＋n4＋CY	データ・メモリとイミーディエト・データの内容をCYフラグとともに加算。結果をデータ・メモリへ格納。
	減算	SUB r, m	(r) ← (r) － (m)	ジェネラル・レジスタの内容からデータ・メモリの内容を減算。結果をジェネラル・レジスタへ格納。
		SUB m, #n4	(m) ← (m) －n4	データ・メモリの内容からイミーディエト・データを減算。結果をデータ・メモリへ格納。
		SUBC r, m	(r) ← (r) － (m) －CY	ジェネラル・レジスタの内容からデータ・メモリの内容とCYフラグを減算。結果をジェネラル・レジスタへ格納。
		SUBC m, #n4	(m) ← (m) －n4－CY	データ・メモリの内容からイミーディエト・データとCYフラグを減算。結果をデータ・メモリへ格納。
論理演算	論理和	OR r, m	(r) ← (r) ∨ (m)	ジェネラル・レジスタとデータ・メモリの内容をOR。結果をジェネラル・レジスタへ格納。
		OR m, #n4	(m) ← (m) ∨n4	データ・メモリとイミーディエト・データの内容をOR。結果をデータ・メモリへ格納。
	論理積	AND r, m	(r) ← (r) ∧ (m)	ジェネラル・レジスタとデータ・メモリの内容をAND。結果をジェネラル・レジスタへ格納。
		AND m, #n4	(m) ← (m) ∧n4	データ・メモリとイミーディエト・データの内容をAND。結果をデータ・メモリへ格納。
	排他的論理和	XOR r, m	(r) ← (r) ⊻ (m)	ジェネラル・レジスタとデータ・メモリの内容をXOR。結果をジェネラル・レジスタへ格納。
		XOR m, #n4	(m) ← (m) ⊻ n4	データ・メモリとイミーディエト・データの内容をXOR。結果をデータ・メモリへ格納。
ビット判断	True	SKT m, #n	CMP←0, if (m) ∧n = n, then skip	データ・メモリの内容のうち，nで指定されたビットがすべてTrue（１）ならスキップ。結果は格納されない。
	False	SKF m, #n	CMP←0, if (m) ∧n = 0, then skip	データ・メモリの内容のうち，nで指定されたビットがすべてFalse（０）ならスキップ。結果は格納されない。
比較判断	等しい	SKE m, #n4	(m) －n4, skip if zero	データ・メモリの内容がイミーディエト・データと等しいときスキップ。結果は格納されない。
	等しくない	SKNE m, #n4	(m) －n4, skip if not zero	データ・メモリの内容がイミーディエト・データと等しくないときスキップ。結果は格納されない。
	以上	SKGE m, #n4	(m) －n4, skip if not borrow	データ・メモリの内容がイミーディエト・データより以上のときスキップ。結果は格納されない。
	未満	SKLT m, #n4	(m) －n4, skip if borrow	データ・メモリの内容がイミーディエト・データより未満のときスキップ。結果は格納されない。
回転	右回転	RORC r		ジェネラル・レジスタの内容をCYフラグとともに右へ回転。結果をジェネラル・レジスタへ格納。

★

表８－１　ALU処理命令一覧（2/2）

<table>
<tr><th>ALU機能</th><th>プログラム・ステータス・ワード（PSWORD）による動作のちがい</th></tr>
<tr><td>算術演算</td><td>
<table>
<tr><th>BCDフラグの値</th><th>CMPフラグの値</th><th>演算動作</th><th>CYフラグ</th><th>Zフラグ</th><th>IXE = 1による修飾</th></tr>
<tr><td>0</td><td>0</td><td>２進演算
結果を格納する</td><td rowspan="4">キャリーボロー発生でセット，発生しなければリセット</td><td>演算結果0000Bでセット
0000B以外はリセット</td><td rowspan="4">あ　り</td></tr>
<tr><td>0</td><td>1</td><td>２進演算
結果を格納しない</td><td>演算結果0000Bで状態保持
0000B以外はリセット</td></tr>
<tr><td>1</td><td>0</td><td>BCD演算
結果を格納する</td><td>演算結果0000Bでセット
0000B以外はリセット</td></tr>
<tr><td>1</td><td>1</td><td>BCD演算
結果を格納しない</td><td>演算結果0000Bで状態保持
0000B以外はリセット</td></tr>
</table>
</td></tr>
<tr><td>論理演算</td><td>
<table>
<tr><td>Don' t care
（保持）</td><td>Don' t care
（保持）</td><td>変わらない</td><td>Don' t care
（保持）</td><td>Don' t care
（保持）</td><td>あ　り</td></tr>
</table>
</td></tr>
<tr><td>ビット判断</td><td>
<table>
<tr><td>Don' t care
（保持）</td><td>リセット
される</td><td>変わらない</td><td>Don' t care
（保持）</td><td>Don' t care
（保持）</td><td>あ　り</td></tr>
</table>
</td></tr>
<tr><td>比較判断</td><td>
<table>
<tr><td>Don' t care
（保持）</td><td>Don' t care
（保持）</td><td>変わらない</td><td>Don' t care
（保持）</td><td>Don' t care
（保持）</td><td>あ　り</td></tr>
</table>
</td></tr>
<tr><td>回転</td><td>
<table>
<tr><td>Don' t care
（保持）</td><td>Don' t care
（保持）</td><td>変わらない</td><td>ジェネラル・レジスタの b_0 の値</td><td>Don' t care
（保持）</td><td>あ　り</td></tr>
</table>
</td></tr>
</table>

８.２.２　一時記憶レジスタAおよびBの機能

一時記憶レジスタAおよびBは４ビット・データを一度に処理するために必要なレジスタであり，処理されるデータと，処理するデータを一時的に蓄えておくレジスタです。

８.２.３　ステータス・フリップフロップの機能

ステータス・フリップフロップはALUの動作制御および，処理されたデータの状態を格納するフリップフロップです。ステータス・フリップフロップはシステム・レジスタのプログラム・ステータス・ワード（PSWORD）の各フラグと１対１に対応しているため，システム・レジスタを操作すれば，ステータス・フリップフロップも同時に操作されます。次にプログラム・ステータス・ワードの各フラグについて説明します。

★　**（１）Zフラグ**

算術演算の結果が0000Bになるとセット（１）され，0000B以外になるとリセット（０）されます。
ただし，CMPフラグの状態により次のようにセット（１）される条件が異なります。

（ⅰ）CMPフラグ＝０のとき

演算結果が0000Bであればセット（１）され0000B以外であればリセット（０）されます。

（ⅱ）CMPフラグ＝１のとき

演算結果が0000Bであれば以前の状態を保持し，0000B以外であればリセット（０）されます。
算術演算以外では変化しません。

★　**（２）CYフラグ**

算術演算の結果，キャリーまたはボローが発生するとセット（１）され，発生しなければリセット（０）されます。
算術演算がキャリーまたはボローとともに演算を行う場合はCYフラグの内容を最下位ビットに演算します。
回転処理（RORC命令）を行うときは，そのときのCYフラグの内容をジェネラル・レジスタの最上位ビット（b_3）とし，ジェネラル・レジスタの最下位ビットの内容がCYフラグの内容になります。
算術演算および回転処理以外では変化しません。

★　**（３）CMPフラグ**

CMPフラグがセット（１）されているときに実行された算術演算は，結果がジェネラル・レジスタおよびデータ・メモリに格納されません。
ビット判断命令を実行するとCMPフラグはリセット（０）されます。
比較判断，論理演算，回転処理には影響を与えません。

★　**（４）BCDフラグ**

BCDフラグがセット（１）されているときは，すべての算術演算結果が10進補正されます。
リセット（０）されているときは２進４ビットで演算されます。
論理演算，ビット判断，比較判断，回転処理には影響を与えません。

これらのフラグは，プログラム・ステータス・ワード（PSWORD）を直接操作することにより値を変化させることも可能です。このとき，変化したフラグに対応するステータス・フリップフロップも同様に変化します。

８.２.４　２進４ビット演算

BCDフラグが０のとき，算術演算は，２進数による４ビット単位の演算を行います。

８.２.５　BCD演算

BCDフラグが１のとき算術演算は，２進４ビットで行った演算に対して10進補正を行います。２進４ビット演算結果とBCD演算結果の違いを表８－２に示します。10進補正を行った場合に加算結果が20以上になったとき，あるいは10進補正を行った場合に減算結果が－10～＋９以外になったときにはデータ・メモリに1010B（0AH）以上のデータが格納されます（表８－２の網掛け部分）。

★

表８－２　２進４ビット演算結果とBCD演算結果

演算結果	2進4ビット加算		BCD加算	
	CY	演算結果	CY	演算結果
0	0	0000	0	0000
1	0	0001	0	0001
2	0	0010	0	0010
3	0	0011	0	0011
4	0	0100	0	0100
5	0	0101	0	0101
6	0	0110	0	0110
7	0	0111	0	0111
8	0	1000	0	1000
9	0	1001	0	1001
10	0	1010	1	0000
11	0	1011	1	0001
12	0	1100	1	0010
13	0	1101	1	0011
14	0	1110	1	0100
15	0	1111	1	0101
16	1	0000	1	0110
17	1	0001	1	0111
18	1	0010	1	1000
19	1	0011	1	1001
20	1	0100	1	1110
21	1	0101	1	1111
22	1	0110	1	1100
23	1	0111	1	1101
24	1	1000	1	1110
25	1	1001	1	1111
26	1	1010	1	1100
27	1	1011	1	1101
28	1	1100	1	1010
29	1	1101	1	1011
30	1	1110	1	1100
31	1	1111	1	1101

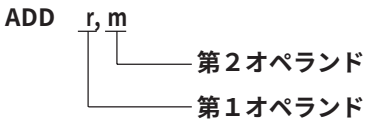
演算結果	2進4ビット減算		BCD減算	
	CY	演算結果	CY	演算結果
0	0	0000	0	0000
1	0	0001	0	0001
2	0	0010	0	0010
3	0	0011	0	0011
4	0	0100	0	0100
5	0	0101	0	0101
6	0	0110	0	0110
7	0	0111	0	0111
8	0	1000	0	1000
9	0	1001	0	1001
10	0	1010	1	1100
11	0	1011	1	1101
12	0	1100	1	1110
13	0	1101	1	1111
14	0	1110	1	1100
15	0	1111	1	1101
−16	1	0000	1	1110
−15	1	0001	1	1111
−14	1	0010	1	1100
−13	1	0011	1	1101
−12	1	0100	1	1110
−11	1	0101	1	1111
−10	1	0110	1	0000
−9	1	0111	1	0001
−8	1	1000	1	0010
−7	1	1001	1	0011
−6	1	1010	1	0100
−5	1	1011	1	0101
−4	1	1100	1	0110
−3	1	1101	1	0111
−2	1	1110	1	1000
−1	1	1111	1	1001

８.２.６　ALUブロック処理手順

プログラム上で算術演算命令，論理演算命令，ビット判断命令，比較判断命令および回転処理命令が実行されると，演算，判断または処理されるデータおよび処理するデータがそれぞれ一時記憶レジスタAおよびBに格納されます。

処理されるデータとは，各命令の第１オペランドでアドレス指定されるジェネラル・レジスタの内容またはデータ・メモリの内容であり，４ビットのデータです。処理するデータとは各命令の第２オペランドでアドレス指定されるデータ・メモリの内容または第２オペランドで直接指定されるイミーディエト・データで４ビットのデータです。

たとえば



命令において処理されるデータはrでアドレス指定されるジェネラル・レジスタの内容であり，処理するデータはmでアドレス指定されるデータ・メモリの内容となります。また

```
ADD m, #n4
```

命令は，処理されるデータはmでアドレス指定されるデータ・メモリの内容であり，処理するデータは#n4で指定されるイミーディエト・データになります。また回転処理命令である

```
RORC r
```

命令は，処理する方法が決まっているため処理されるデータのみ必要となり，rでアドレス指定されるジェネラル・レジスタの内容になります。

次に，一時記憶レジスタAおよびBに格納されたデータは，各命令に従いALUで算術演算，論理演算，ビット判断，比較判断および回転処理を実行します。実行された命令が算術演算，論理演算および回転処理のときは，ALUで処理されたデータを，命令の第１オペランドで指定されるジェネラル・レジスタまたはデータ・メモリに格納して動作を終了します。また，実行された命令がビット判断および比較判断であるときは，ALUで処理された結果によりプログラム上の次の命令をスキップ（次の命令はNOP命令として実行）して動作を終了します。

ALUブロック動作については次の点に注意が必要です。

（１）算術演算は，プログラム・ステータス・ワードのCMPフラグおよびBCDフラグの影響を受ける。

（２）論理演算は，プログラム・ステータス・ワードのCMPフラグおよびBCDフラグの影響は受けない。またZフラグ，CYフラグには影響を与えない。

（３）ビット判断はプログラム・ステータス・ワードのCMPフラグをリセットする。

（４）算術演算，論理演算，ビット判断，比較判断および回転処理は，プログラム・ステータス・ワードのIXEフラグがセット（１）されていると，インデクス・レジスタによる修飾を受ける。

８.３　算術演算（２進４ビット加減算およびBCD加減算）

表８－３に示すように，算術演算は，加算と減算に大別され，さらにキャリーとともに加算およびボローとともに減算とに分けられます。算術演算命令はこの４種類に分けられ，“ ADD ”，“ ADDC ”，“ SUB ”，“ SUBC ”命令を使用します。

“ ADD ”，“ ADDC ”，“ SUB ”，“ SUBC ”命令は，さらにジェネラル・レジスタとデータ・メモリの加減算およびデータ・メモリとイミーディエト・データの加減算に分けられます。これは各命令のオペランドに記述する値により決定されます。すなわちオペランドが“ r, m ”であればジェネラル・レジスタとデータ・メモリの加減算になり“ m, #n4 ”であればデータ・メモリとイミーディエト・データの加減算になります。

算術演算命令はステータス・フリップフロップすなわち，システム・レジスタのプログラム・ステータス・ワード（PSWORD）の影響を受けます。プログラム・ステータス・ワードのBCDフラグにより２進４ビット演算およびBCD演算を行い，CMPフラグにより，演算結果をどこにも格納しないことができます。

８.３.１-８.３.４に各算術演算命令とプログラム・ステータス・ワードについて説明します。

★ **表８－３　算術演算の種類**

算術演算	加算	キャリーは無視	ジェネラル・レジスタとデータ・メモリ	ADD r, m
		ADD	データ・メモリとイミーディエト・データ	ADD m, #n4
		キャリーとともに加算	ジェネラル・レジスタとデータ・メモリ	ADDC r, m
		ADDC	データ・メモリとイミーディエト・データ	ADDC m, #n4
	減算	ボローは無視	ジェネラル・レジスタとデータ・メモリ	SUB r, m
		SUB	データ・メモリとイミーディエト・データ	SUB m, #n4
		ボローとともに減算	ジェネラル・レジスタとデータ・メモリ	SUBC r, m
		SUBC	データ・メモリとイミーディエト・データ	SUBC m, #n4

８.３.１　CMPフラグ＝０，BCDフラグ＝０のときの加減算

２進４ビットの加減算を行い，結果をジェネラル・レジスタまたはデータ・メモリに格納します。

CYフラグは演算結果が1111Bを越えたとき（キャリーの発生）と，0000B未満（ボローの発生）になるとセット（１）され，それ以外ではリセット（０）されます。

演算結果が0000Bになるとキャリーおよびボローの発生に関係なくZフラグをセット（１）し，0000Bでなければリセット（０）します。

例１．

```
MOV   R1, #1111B   ;ジェネラル・レジスタR1に1111Bを転送
MOV   M1, #0001B   ;データ・メモリM1に0001Bを転送
ADD   R1, M1       ;R1とM1を加算
```

このとき，R1＋M1は次のように計算されます。

```
  1111B…R1の内容
＋0001B…M1の内容
―――――
 10000B
 ^
キャリー
```

したがって，加算結果の0000BをR1の内容に書き込み，CYフラグをセット（１）します。M1の内容は変化しません。

また，演算結果が0000Bであるためフラグをセット（１）します。

R1とM1の内容を加算した場合に，キャリーが出力されないと，CYフラグはリセット（０）されます。

２．

```
MOV   M1, #1010B   ;データ・メモリM1に1010Bを転送
ADD   M1, #0101B   ;M1にイミーディエト・データ0101Bを加算
```

このとき，M1＋0101Bは次のように計算されます。

```
  1010B…M1の内容
＋0101B…イミーディエト・データ
―――――
 01111B
 ^
キャリー
```

したがって1111BをM1の内容に書き込み，CYフラグおよびZフラグをリセットします。

例３．

```
MOV   R1, ＃1000B   ；ジェネラル・レジスタR1に1000Bを書き込む。
MOV   M1, ＃1111B   ；データ・メモリM1に1111Bを書き込む。
；①
ADD   M1, ＃0001B   ；M1にイミーディエト・データ0001Bを加算
；②
ADDC  R1, M1        ；R1とM1をキャリーとともに加算
```

①においては次のように計算します。

```
  1111B…M1の内容
＋0001B…イミーディエト・データ
 1 0000B
キャリー
```

したがって，R1に0000Bを書き込みCYフラグおよびZフラグをセット（１）します。②においては次のように計算します。

```
  1000B…R1の内容
  0000B…M1の内容
＋    1 …CYフラグの内容
 0 1001B
キャリー
```

すなわち，ADDC命令実行時はそのときのCYフラグの内容とともに加算を行い，その結果のキャリー出力によりCYフラグを書き換えます。

４．

```
MOV   R1, ＃0000B     ；ジェネラル・レジスタR1に0000Bを書き込む。
MOV   M1, ＃1000B     ；データ・メモリM1に1000Bを書き込む。
SUB   R1, M1          ；R1からM1を減算
```

このときR1-M1は次のように計算されます。

```
  0000B…R1の内容
－1000B…M1の内容
 1 1000B
ボロー
```

したがって，減算結果の1000BをR1に書き込みます。このとき，ボローが発生しているためCYフラグをセット（１）します。
加算命令によるキャリーと減算命令によるボローは，同一のCYフラグによって管理されていることになります。

例５．

```
MOV   R1, #0000B   ;
MOV   M1, #0000B   ;
;①
SUB   M1, #0001B   ;
;②
SUBC  R1, M1       ;
```

このとき①，②では次のように計算されます。

①

```
  0000B…M1の内容
－0001B…イミーディエト・データ
―――――
 1 1111B
ボロー
```

②

```
  0000B…R1の内容
  1111B…M1の内容
－    1 …CYフラグの内容
―――――
 1 0000B
ボロー
```

したがって演算結果はR1＝0000B，M1＝1111B，CYフラグ＝１，Zフラグ＝１となります。

8.3.2　CMPフラグ＝１，BCDフラグ＝０のときの加減算

２進４ビットの加減算を行います。

ただしCMPフラグがセット（１）されているため，演算結果がジェネラル・レジスタまたはデータ・メモリに格納されません。

演算結果によりキャリーまたはボローが発生するとCYフラグがセット（１）され，発生しなければリセット（０）されます。

Zフラグは，演算結果が0000Bであれば以前の状態を保持し，0000Bでなければリセット（０）されます。

例１．

```
MOV   PSW, #1000B ；CMPフラグをセットする（プログラム・ステータス・ワードに書き込む）。
MOV   R1, #1111B   ;
MOV   M1, #1111B   ;
;①
ADD   R1, M1       ;
;②
SUB   R1, M1       ;
MOV   PSW, #1010B ；CMPフラグとZフラグをセットする。
;③
SUB   R1, M1       ;
```

このとき①では次のように計算されます。

```
  1111B…R1の内容
＋1111B…M1の内容
―――――
 11110B
 ^^
キャリー
```

ただし，CMPフラグがセット（１）されているため，演算結果はR1に格納されません。
キャリーが発生するためCYフラグはセット（１）されます。
演算結果が0000Bでないためフラグはリセットされます。
②ではR1とM1の内容は①と同じであるため，CYフラグはリセット（０）されます。演算結果が0000Bであるためフラグは以前の状態である0を保持します。
③は②と同様の演算結果ですが，Zフラグがあらかじめセット（１）されているためZフラグは１を保持します。
CMPフラグがセット（１）されていると算術演算結果は格納されず，CYフラグとZフラグの状態のみが変化するため，５ビット以上のデータを比較するときなどに使用すると便利です。

例２．

```
 MOV   PSW, #1010B       ; CMPフラグとZフラグをセット（１）
;①
 SUB   M1, #0001B（1H）  ;
;②
 SUBC  M2, #0010B（2H）  ;
;③
 SUBC  M3, #0011B（3H）  ;
```

このとき，CMPフラグがセット（１）されているため，演算結果は格納されません。したがって①，②，③を実行してもM1，M2およびM3の内容は変化しません。
また，Zフラグを最初にセット（１）しているため，①，②，③のすべての演算結果が0000Bであればフラグはセット（１）されたままになり，演算結果の１つでも0000Bにならなければリセット（０）されます。
CYフラグはM3，M2およびM1の12ビットの内容が001100100001B（321H）より小さいときにセットされます。
したがって①，②，③を終了した時点でZフラグとCYフラグをテストすることによりM3，M2およびM1の12ビット・データと321Hの12ビット・データを比較できます。すなわち，次のようになります。

Z＝1，CY＝0ならM3，M2，M1＝321H
↑
必ず０になる
Z＝0，CY＝0ならM3，M2，M1＞321H
Z＝0，CY＝1ならM3，M2，M1＜321H

例２においてSUB r，mおよびSUBC r，m命令を用いることによりジェネラル・レジスタとデータ・メモリの内容を比較することもできます。

8.3.3　CMPフラグ＝０，BCDフラグ＝１のときの加減算

BCD演算を行います。

演算結果は，ジェネラル・レジスタまたはデータ・メモリに格納されます。CYフラグは演算結果が1001B（9D）を越えるか，0000B（0D）未満になるとセット（１）され，0000B（0D）～1001B（9D）であればリセット（０）されます。

Zフラグは，演算結果が0000B（0D）になるとセット（１）され，0000B（0D）以外になるとリセット（０）されます。

BCD演算は，一度２進で演算された結果を10進補正回路で10進に変換する方法を用いています。この２進-10進変換については**表８－２　２進４ビット演算結果とBCD演算結果**を参照してください。

したがって，BCD演算を正しく実行するためには，次のことに注意してください。

（１）加算の結果が0D～19Dであること
（２）減算の結果が0D～9Dまたは－10D～－1Dであること
0D～19Dとは，CYフラグを考慮した値であり，２進４ビットで表すと
0, 0000B～1, 0011Bのことです
CY　　CY
－10～－1Dとは同様に
1, 0110B～1, 1111Bのことです
CY　　CY

上記（１），（２）以外でBCD演算を行うとCYフラグがセット（１）され，演算結果として1010B（0AH）以上のデータが出力されます。

例１．

```
MOV   M1, #0111B（７）  ;
MOV   RPL,#0001B        ; BCDフラグをセット（BCDフラグはシステム・レジスタのRPLのb0に
                        ; 割り当てられている）
MOV   PSW, #0000B       ; CMPフラグ，CYフラグおよびZフラグをリセット
;①
ADD   M1, #1001B（９）  ; 7＋9
;②
SUB   M1, #0111B（７）  ; 6－7
```

このとき①は次のように計算します。

```
  0111B…M1の内容
＋1001B…イミーディエト・データ
―――――
 10000B…２進の加算結果
キャリー

    ↓２進10進補正（表８－２　２進４ビット演算結果とBCD演算結果を参照）
 10110B…M1格納データ
キャリー
```

すなわち，CYフラグがセットされ，M1に0110B（６）が格納されます。CYフラグを10の重みと考えれば7＋9＝16の10進演算を行ったことになります。
②においては，次のように計算します。

```
  0110B…M1の内容
－0111B…イミーディエト・データ
―――――
 11111B…２進の減算結果
ボロー

    ↓２進10進補正
 11001B…M1格納データ
```

すなわち，①でM1には６が格納されているため６－７を行ったことになり，結果は９となりCYフラグがセットされます。

例２．

```
    MOV   M1, #0101B（５）；
    MOV   M2, #0110B（６）；
    MOV   M3, #0111B（７）；
    MOV   RPL, #0001B      ; BCDフラグをセット（１）
    MOV   PSW, #0000B      ; CMPフラグ，CYフラグおよびZフラグをリセット（０）
  ;①
    SUB   M1, #0111B（７）；
  ;②
    SUBC  M2, #0110B（６）；
  ;③
    SUBC  M3, #0101B（５）；
```

このとき①，②および③では次のように計算します。

①

　0101B…M1の内容
－0111B…イミーディエト・データ
―――――
　1 1110B
　ボロー

　↓２進10進補正
　1 1000B（８）…M1格納データ
　ボロー

②

　0110B…M2の内容
－0110B…イミーディエト・データ
―――――
　1 1111B…CYフラグ
ボロー

　↓２進10進補正
　1 1001B（９）…M2格納データ
ボロー

③

　0111B…M3の内容
－0101B…イミーディエト・データ
―――――
　0 0001B…CYフラグ
ボロー

　↓２進10進補正
　0 0001B（１）…M3格納データ

すなわち，M3，M2およびM1に格納されていた765から，イミーディエト・データ567を減算したことになり，結果は198になっています。

例３．

```
        MOV   M1, #1001B        ;
        MOV   RPL, #0001B       ; BCDフラグをセット（１）
        MOV   PSW, #0000B       ; CMPフラグ，CYフラグおよびZフラグをリセット（０）
      ; ①
        ADDC  M1, #1010B        ;
      ; ②
        ADDC  M1, #1010B        ;
```

このとき①は次のように計算します。

```
  1001B（９）…M1の内容
＋1010B（10）…イミーディエト・データ
―――――
 10011B…CYフラグ
キャリー

    ↓２進10進変換

 1 1001B…演算結果
キャリー
```

すなわち，9＋10＝9となりCYフラグを考慮すれば9＋10＝19の10進演算を行ったことになります。ところが，②では次のように計算してしまいます。

```
  1001B（９）…M1の内容
＋1010B（10）…イミーディエト・データ
―――――
 10100B…CYフラグ
キャリー

    ↓２進10進変換

 1 1110B…演算結果
キャリー
```

すなわち，CYフラグがセット（１）されていたため演算結果が19を越えてしまい，正確な10進演算が行われなくなります。

８.３.４　CMPフラグ＝１，BCDフラグ＝１のときの加減算

BCD演算を行います。

演算結果はジェネラル・レジスタまたはデータ・メモリへ格納されません。

すなわち，CMPフラグ＝１のときと，BCDフラグ＝１のときの動作を同時に行います。

例１．

```
MOV    RPL,    #0001B；BCDフラグをセット（１）
MOV    PSW,    #1010B；CMPフラグとZフラグをセット（１），CYフラグをリセット（０）
SUB    M1,     #0001B；①
SUBC   M2,     #0010B；②
SUBC   M3,     #0011B；③
```

このとき，①②③によりM3, M2, M1の12ビットの内容とイミーディエト・データの321とを，10進数で比較することが可能となります。

２．

```
MOV    RPL,    ＃0001B；BCDフラグをセット（１）
MOV    PSW,    ＃1010B；CMPフラグとZフラグをセット（１），CYフラグをリセット（０）
；①
SUB    M1,     ＃0001B；
；②
SUBC   M2,     ＃0010B；
；③
SUBC   M3,     ＃0011B；
```

このとき，①，②および③によりM3，M2およびM1の12ビットの内容とイミーディエト・データの321とを10進数で比較することができます。

８.３.５　算術演算使用時の注意

プログラム・ステータス・ワード（PSWORD）に対して算術演算を行うときは，プログラム・ステータス・ワードには，算術演算の結果が格納される，という点に注意する必要があります。

すなわちプログラム・ステータス・ワードの中のCYフラグおよびZフラグは，通常，算術演算結果によりセット／リセットされますが，プログラム・ステータス・ワード自身に算術演算が行われると，算術演算結果が格納されてしまい，キャリー，ボローおよびゼロの判定ができないことになります。

ただし，CMPフラグがセット（１）されているときは，算術演算結果が格納されないため，CYフラグ，Zフラグは通常どおりセット／リセットされます。

例１．

```
MOV   PSW, #0110B
ADD   PSW, #1010B
```

このとき，次のように計算します。

```
  0110B…PSWの内容
＋1010B…イミーディエト・データ
 10000B
 ^
キャリー
```

したがって，CYフラグおよびZフラグがセットされるはずですがCMPフラグが“ 0 ”のため，PSWには演算結果である0000Bが格納されます。

２．

```
MOV   PSW, #1010B
ADD   PSW, #1000B
```

このとき，次のように計算します。

```
  1010B…PSWの内容
＋1000B…イミーディエト・データ
 10010B
 ^
キャリー
```

ただし，CMPフラグがセット（１）されているため，演算結果の0010BはPSWに格納しません。
したがって，PSWのCYフラグをセット（１）してZフラグをリセット（０）します。すなわちPSWには1100Bが格納されます。

８．４　論理演算

表８－４に示すように，論理演算は論理和（OR），論理積（AND）および排他的論理和（XOR）が使用できます。

論理演算命令はこの３種類に分けられ，それぞれ“ OR ”，“ AND ”および“ XOR ”命令を使用します。

“ OR ”，“ AND ”，“ XOR ”命令は，さらにジェネラル・レジスタとデータ・メモリの論理演算およびデータ・メモリとイミーディエト・データの論理演算に分けられます。これは算術演算と同様に命令のオペランドに記述された値“ r, m ”または“ m, #n4 ”により決定されます。

論理演算は，プログラム・ステータス・ワード（PSWORD）のBCDフラグおよびCMPフラグの影響は受けません。またCYフラグおよびZフラグには何の影響も与えません。ただし，インデクス・イネーブル・フラグ（IXEフラグ）がセット（１）されているときは，インデクス・レジスタにより修飾されます。

★

表８－４　論理演算

論理演算	論理和 OR	ジェネラル・レジスタとデータ・メモリ	OR r, m
		データ・メモリとイミーディエト・データ	OR m, #n4
	論理積 AND	ジェネラル・レジスタとデータ・メモリ	AND r, m
		データ・メモリとイミーディエト・データ	AND m, #n4
	排他的論理和 XOR	ジェネラル・レジスタとデータ・メモリ	XOR r, m
		データ・メモリとイミーディエト・データ	XOR m, #n4

表８－５　論理演算の真理値表

論理積 C = A AND B			論理和 C = A OR B			排他的論理和 C = A XOR B		
A	B	C	A	B	C	A	B	C
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

８.４.１　論理和（Logical OR）

論理和命令は表８－５の真理値に従い，４ビット・データのOR演算を行います。

例

```
    MOV   R1, #1010B      ;
    MOV   M1, #1001B      ;
  ;①
    OR    R1, M1          ;
  ;②
    OR    M1, #1100B      ;
```

このとき，①は次のように計算します。

```
    1010B…R1の内容
OR 1001B…M1の内容
    1011B…演算結果
```

したがって1011BがR1に格納されます。
②では次のように演算します。

```
    1001B…M1の内容
OR 1100B…イミーディエト・データ
    1101B
```

したがって，1101BがM1に格納されます。

論理和は，データ・メモリの内容を１ビット，２ビット，３ビットまたは４ビット単位でセット（１）するときに便利です。

8.4.2　論理積（Logical AND）

論理積命令は表８－５の真理値に従い，４ビット・データのAND演算を行います。

例

```
    MOV   R1, #1010B      ;
    MOV   M1, #1001B      ;
  ;①
    AND   R1, M1          ;
  ;②
    AND   M1, #1100B      ;
```

このとき，①は次のように演算します。

```
     1010B…R1の内容
AND 1001B…M1の内容
     1000B…
```

したがって1000BがR1に格納されます。
②は次のように演算します。

```
      1001B…M1の内容
AND 1100B…イミーディエト・データ
      1000B
```

したがって，1000BがM1に格納されます。

論理積は，データ・メモリの内容を１ビット，２ビット，３ビットまたは４ビット単位でリセット（０）するときに便利です。

８.４.３　排他的論理和（Logical Exclusive OR）

排他的論理和命令は表８－５の真理値に従い，４ビット・データのXOR演算を行います。

例

```
    MOV   R1, #1010B      ;
    MOV   M1, #1001B      ;
  ;①
    XOR   R1，M1          ;
  :②
    XOR   M1, #1100B      ;
```

このとき，①は次のように演算します。

```
      1010B…R1の内容
XOR 1001B…M1の内容
      0011B
```

したがって0011BがR1に格納されます。
②は次のように演算します。

```
      1001B…M1の内容
XOR 1100B…イミーディエト・データ
      0101B
```

したがって，0101BがM1に格納されます。

排他的論理和は，データ・メモリの内容を１ビット，２ビット，３ビットまたは４ビット単位で反転するときに便利です。

８.５　ビット判断

表８－６に示すように，ビット判断はTrueビット（１）判断およびFalseビット（０）判断に分けられます。

Trueビット（１）判断およびFalseビット（０）判断はそれぞれ“ SKT ”および“ SKF ”命令を使用します。

“ SKT ”，“ SKF ”命令はデータ・メモリに対してのみ行うことができます。

ビット判断は，プログラム・ステータス・ワード（PSWORD）のBCDフラグの影響を受けません。またCYフラグおよびZフラグには何の影響も与えません。ただし，CMPフラグは“ SKT ”および“ SKF ”命令が実行されるとリセット（０）されます。インデクス・イネーブル・フラグ（IXEフラグ）がセット（１）されているときは，インデクス・レジスタにより修飾されます。インデクス・レジスタによる修飾については**第６章　システム・レジスタ（SYSREG）**を参照してください。

８.５.１，８.５.２にTrueビット（１）判断およびFalseビット（０）判断について説明します。

★

表８－６　ビット判断命令

ビット判断	Trueビット（１）判断	SKT m, #n
	Falseビット（０）判断	SKF m, #n

８.５.１　Trueビット（１）判断

Trueビット（１）判断命令“ SKT m, #n ”は，データ・メモリの４ビットのうち，nで指定されたビットが“ True（１）”であるかを判断します。nで指定されたビットがすべて“ True（１）”であるとき，この命令の次の命令をスキップします。

```
例  MOV   M1,   #1011B
    SKT   M1,   #1011B ; ①
    BR    A
    BR    B
    SKT   M1,   #1101B ; ②
    BR    C
    BR    D
```

このとき，①ではM1のビット３，１，０を判断し，すべてTrue（１）であるからBへ分岐します。
②では，M1のビット３，２，０を判断しますが，M1のビット２はFalse（０）であるため，Cへ分岐します。

８.５.２　Falseビット（０）判断

Falseビット（０）判断命令“ SKF m, #n ”はデータ・メモリの４ビットのうちnで指定されたビットがFalse（０）であるかを判断します。nで指定されたビットがすべて“ False（０）”であるとき，この命令の次の命令をスキップします。

```
例  MOV   M1,   #1001B ;
    SKF   M1,   #0110B ; ①
    BR    A            ;
    BR    B            ;
    SKF   M1,   #1110B ; ②
    BR    C            ;
    BR    D            ;
```

このとき①では，M1のビット２，１を判断し，すべてFalse（０）であるためBへ分岐します。
②ではM1のビット３，２，１を判断しますが，M1のビット３はTrue（１）であるためCへ分岐します。

８.６　比較判断

表８－７に示すように，比較判断は“等しい”，“等しくない”，“以上”および“未満”の判断に分けられます。

“等しい”，“等しくない”，“以上”および“未満”の判断はそれぞれ“SKE”，“SKNE”，“SKGE”および“SKLT”命令を使用します。

“SKE”，“SKNE”，“SKGE”，“SKLT”命令は，データ・メモリとイミーディエト・データとの比較判断のみ行うことができます。ジェネラル・レジスタとデータ・メモリとの比較判断を行うときは，プログラム・ステータス・ワード（PSWORD）のCMPフラグおよびZフラグを用いて減算命令により行えます（**８.３　算術演算（２進４ビット加減算およびBCD加減算）**参照）。

比較判断は，プログラム・ステータス・ワードのBCDフラグおよびCMPフラグの影響を受けません。またCYフラグおよびZフラグには何の影響も与えません。

インデクス・イネーブル・フラグ（IXEフラグ）がセット（１）されているときは，インデクス・レジスタにより修飾されます。インデクス・レジスタによる修飾については**第６章　システム・レジスタ（SYSREG）**を参照してください。

８.６.１-**８.６.４**に“等しい”，“等しくない”，“以上”および“未満”の判断について説明します。

★ **表８－７　比較判断命令**

比較判断	等しい SKE m, #n4
	等しくない SKNE m, #n4
	以上 SKGE m, #n4
	未満 SKLT m, #n4

８.６.１　“等しい”の判断

“等しい”の判断命令“SKE m, #n4”はデータ・メモリとイミーディエト・データの内容が“等しい”かを判断します。

データ・メモリとイミーディエト・データの内容が“等しい”とき，この命令の次の命令をスキップします。

```
例  MOV   M1,   #1010B
    SKE   M1,   #1010B；①
    BR    A
    BR    B
    ；
    SKE   M1,   #1000B；②
    BR    C
    BR    D
```

このとき，①では，M1の内容とイミーディエト・データの1010Bが等しいためBへ分岐します。
②ではM1の内容とイミーディエト・データの1000Bが等しくないためCへ分岐します。

８.６.２　“等しくない”の判断

“等しくない”の判断命令“SKNE m, #n4”は，データ・メモリとイミーディエト・データの内容が“等しくない”かを判断します。

データ・メモリとイミーディエト・データの内容が“等しくない”とき，この命令の次の命令をスキップします。

```
例  MOV   M1,   #1010B
    SKNE  M1,   #1000B；①
    BR    A
    BR    B
    ；
    SKNE  M1,   #1010B；②
    BR    C
    BR    D
```

このとき，①では，M1の内容とイミーディエト・データの1000Bが等しくないためBへ分岐します。
②では，M1の内容とイミーディエト・データの1010Bが等しいためCへ分岐します。

８.６.３　“以上”の判断

“以上”の判断命令“SKGE m, #n4”はデータ・メモリとイミーディエト・データの内容を比較し，データ・メモリの内容が，イミーディエト・データより“大きい”か，または“等しい”ときに，この命令の次の命令をスキップします。

```
例  MOV   M1,   #1000B
    SKGE  M1,   #0111B；①
    BR    A
    BR    B
    ；
    SKGE  M1,   #1000B；②
    BR    C
    BR    D
    ；
    SKGE  M1,   #1001B；③
    BR    E
    BR    F
```

このとき，M1の内容は1000Bであるため①は“大きい”，②は“等しい”，③は“小さい”と判断され，それぞれB, D, Eに分岐します。

８.６.４　“未満”の判断

“未満”の判断命令“SKLT m, #n4”はデータ・メモリとイミーディエト・データの内容を比較し，データ・メモリの内容が，イミーディエト・データより“小さい”とき，この命令の次の命令をスキップします。

```
例  MOV   M1,   #1000B
    SKLT  M1,   #1001B；①
    BR    A
    BR    B
    ；
    SKLT  M1,   #1000B；②
    BR    C
    BR    D
    ；
    SKLT  M1,   #0111B；③
    BR    E
    BR    F
```

このとき，M1の内容は1000Bであるため，①は“小さい”，②は“等しい”，③は“大きい”と判断され，それぞれB, C, Eに分岐します。

８.７　回転処理

回転処理は，右回転処理と左回転処理に分けられます。

右回転処理には“ RORC ”命令を使用します。

“ RORC ”命令は，ジェネラル・レジスタに対してのみ行うことができます。

“ RORC ”命令による回転処理は，プログラム・ステータス・ワード（PSWORD）のBCDフラグおよびCMPフラグの影響を受けません。またZフラグには何の影響も与えません。

“ RORC ”命令は，インデクス・イネーブル・フラグ（IXEフラグ）がセット（１）されていても，インデクス・レジスタ（IX）によるアドレス修飾（増減）はしません。

左回転処理は，加算命令である“ ADDC ”命令により行うことができます。

８.７.１，８.７.２で，回転処理について説明します。

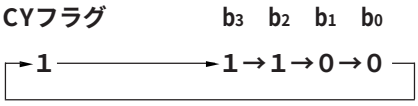
８.７.１　右回転処理

右回転処理命令“ RORC r ”はジェネラル・レジスタの内容を下位ビット方向に１ビット回転します。

このとき，ジェネラル・レジスタの内容の最上位ビット３にはCYフラグの内容が書き込まれ，最下位ビット０の内容をCYフラグに書き込みます。

例１.
```
MOV    PSW,    #0100B；CYフラグをセット（１）
MOV    R1,     #1100B
RORC   R1
```

このとき，次のように処理されます。



すなわち，CYフラグ→b_3, b_3→b_2, b_2→b_1, b_1→b_0, b_0→CYフラグのように右に回転を行います。

例２.
```
MOV    PSW,    #0000B；CYフラグをリセット（０）
MOV    R1,     #1000B；最上位
MOV    R2,     #0100B
MOV    R3,     #0010B；最下位
RORC   R1
RORC   R2
RORC   R3
```

このとき，上記プログラムはCY，R1, R2, R3の13ビット・データを右に回転します。

８.７.２　左回転処理

左回転処理は加算命令である “ ADDC r, m ” 命令を用いることにより行えます。

```
例  MOV   PSW,   #0000B；CYフラグをリセット（０）
    MOV   R1,    #1000B；最上位
    MOV   R2,    #0100B
    MOV   R3,    #0010B；最下位
    ADDC  R3, R3
    ADDC  R2, R2
    ADDC  R1, R1
    SKF   CY
    OR    R3,    #0001B
```

このとき，上記プログラムはCY，R1, R2, R3の13ビット・データを左に回転します。

第９章　レジスタ・ファイル（RF）

レジスタ・ファイルは“ PEEK ”および“ POKE ”命令で操作できるレジスタ領域です。
主として周辺ハードウエアの条件設定をするためのレジスタです。

９.１　レジスタ・ファイルの構成

レジスタ・ファイルは，図９－１のように，コントロール・レジスタとデータ・メモリとで構成されます。

この図のように，アドレス40Hから7FH番地までは，データ・メモリと重なっています。すなわち，レジスタ・ファイルの40H-7FH番地は，データ・メモリの，そのとき選択されているバンクのアドレス40Hから7FH番地そのものなのです。

したがって，たとえば現在選択しているバンクがBANK0ならば，レジスタ・ファイルのアドレス40H-7FH番地もBANK0で，40H-7FH番地はデータ・メモリとしてもレジスタ・ファイルとしても操作できます。

また，レジスタ・ファイルの00H-3FH番地はコントロール・レジスタで，主にここで周辺ハードウエアの条件を設定します。

以上の領域が合成されて，図９－２のように128ニブル（128ワード×４ビット）のレジスタを構成しています。

図９－１　レジスタ・ファイルとデータ・メモリの関係

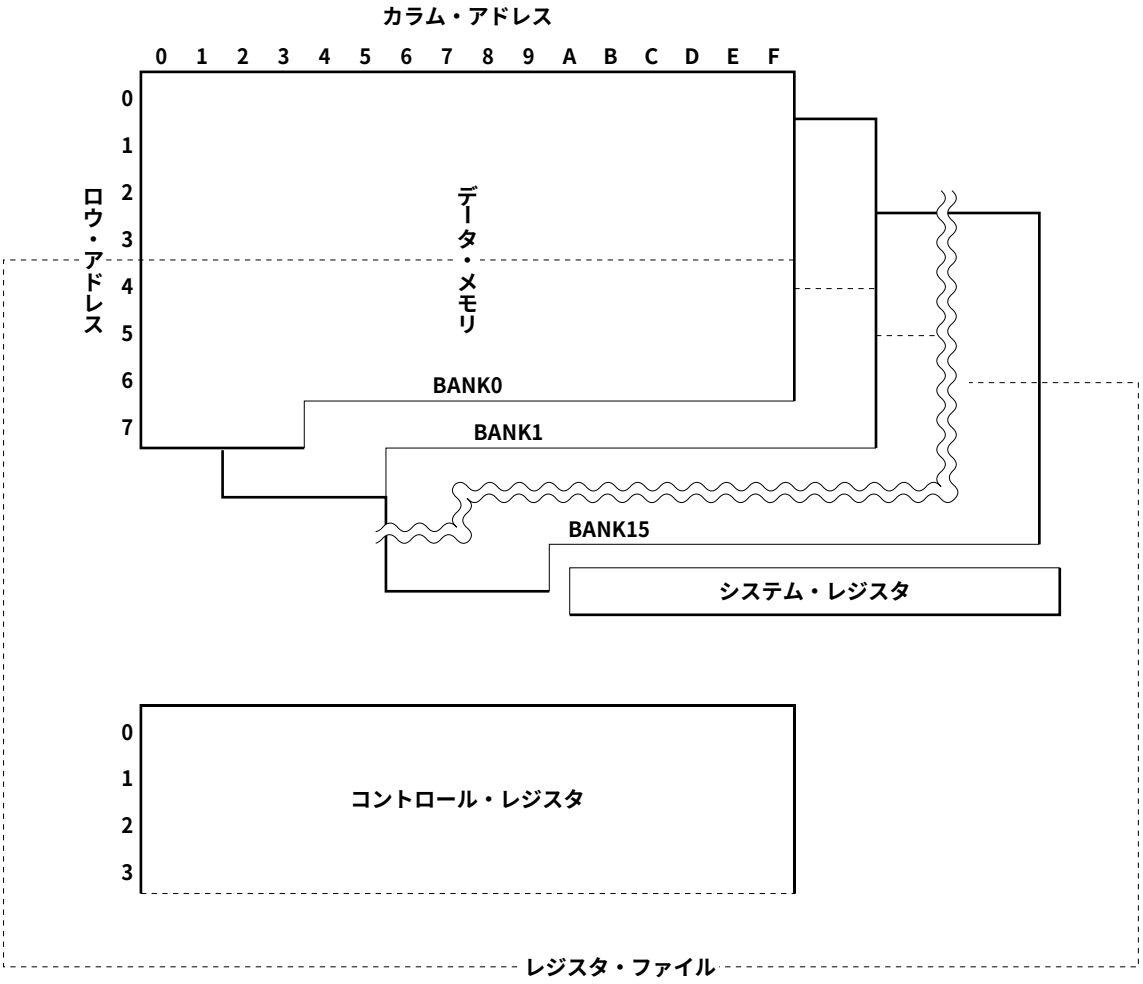


図９－２　レジスタ・ファイルの構成

９.２　レジスタ・ファイルの機能

９.２.１　レジスタ・ファイルの機能

コントロール・レジスタはレジスタ・ファイルの中にあり，主として周辺ハードウエアの条件設定を行う制御レジスタです。

残りのレジスタ・ファイル（40H-7FH番地）はデータ・メモリと重なっているので，レジスタ・ファイル操作命令の“ PEEK ”および“ POKE ”命令で操作できる点を除けば通常のデータ・メモリと何ら変わりはありません。

９.２.２　レジスタ・ファイルの操作命令

レジスタ・ファイルへのデータ書き込みおよびレジスタ・ファイルからのデータの読み込みは，システム・レジスタの中のウインドウ・レジスタ（WR：アドレス78H）を介して行います。

データの書き込みおよびデータ読み込みは次の専用命令を使用します。

PEEK　WR, rf：WRにrfでアドレス指定されるレジスタ・ファイルのデータを読み込む。
POKE　rf, WR：rfでアドレス指定されるレジスタ・ファイルにWRのデータを書き込む。

次に使用例を示します。

★ 例

```
M030    MEM   0.30H   ;データ・メモリの30H番地をWRの退避領域として使用
M032    MEM   0.32H   ;データ・メモリの32H番地をWRの操作領域として使用
RF11    MEM   0.91H   ;シンボル定義
RF33    MEM   0.B3H   ;レジスタ・ファイルの00H-3FH番地のシンボル定義はBANK0の80H-BFH
RF70    MEM   0.70H   ;として定義する必要があります。詳細については９.４　レジスタ・ファ
RF73    MEM   0.73H   ;イル使用時の注意を参照してください。
BANK0
①PEEK   WR, RF11     ;

CLR1    MPE           ;汎用データ・メモリ（00H-3FH番地）にWRの内容を退避する例を示しま
CLR1    IXE           ;す。一例として，アドレス修飾をせず，データ・メモリの30H番地に退避
OR      RPL, #0110B   ;する場合を示します。
②LD     M030, WR      ;

③POKE   RF73, WR      ;40H-7FH番地のデータ・メモリとコントロール・レジスタはWRとPEEK,
④PEEK   WR, RF70      ;POKE命令で直接データのやりとりが行えます。
⑤POKE   RF33, WR      ;

⑥ST     WR, M032      ;
```

図９－３に動作例を示します。

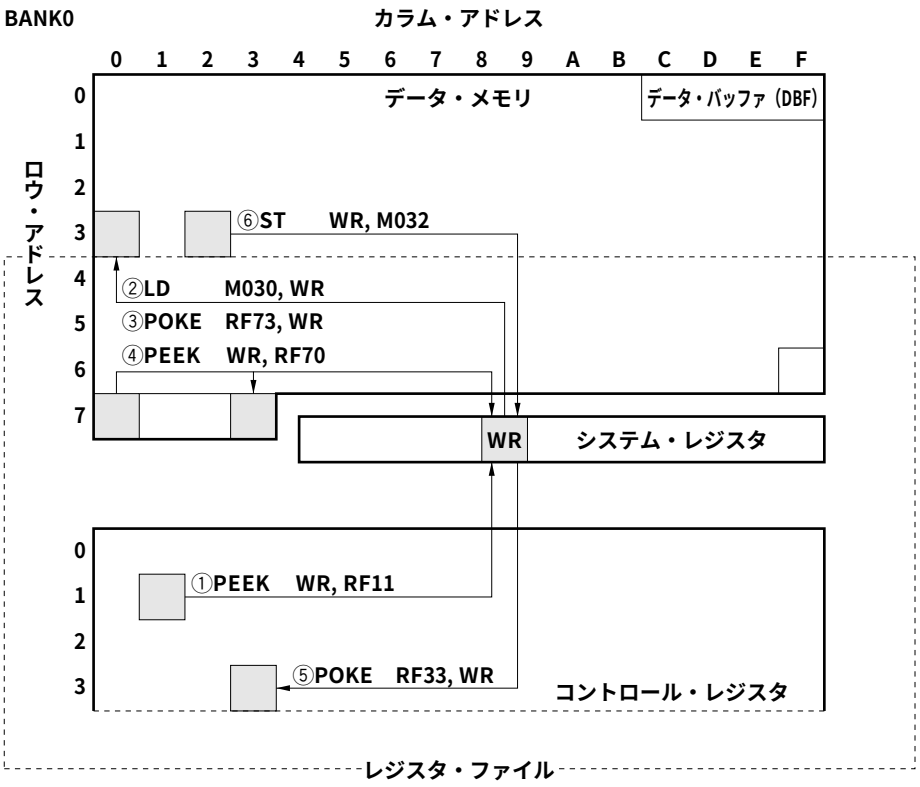
図９－３に示すように，コントロール・レジスタ（00H-3FH番地）は“ PEEK WR, rf ”および“ POKE rf, WR ”命令が実行されると，“ rf ”でアドレス指定されるレジスタ・ファイルの内容をウインドウ・レジスタに対してのみ読み込みまたは書き込みを行います。

レジスタ・ファイルの40H-7FH番地は，データ・メモリと重なっているため，“ PEEK WR, rf ”および“ POKE rf, WR ”命令を実行すると，そのとき認定されているバンクのデータ・メモリ・アドレス“ rf ”に対して命令を実行します。

また，レジスタ・ファイルの40H-7FH番地は通常のメモリ操作命令でも操作できます。

コントロール・レジスタは，組み込みマクロ命令を使用することにより１ビット単位で操作することができます（**9.4.2　レジスタ・ファイルのシンボル定義と予約語**参照）。

★ **図９－３　PEEK，POKE命令によるレジスタ・ファイルのアクセス例**



９.３　コントロール・レジスタ

９.３.１　コントロール・レジスタの構成

コントロール・レジスタは，周辺ハードウエアの条件設定をするためのレジスタです。

コントロール・レジスタはレジスタ・ファイルのアドレス00H-3FH番地の計64ワード×４ビットのレジスタです。

ただし，そのうち実際に使用しているのは品種によって異なります。

★ 各コントロール・レジスタは１ニブルずつ属性があって，それぞれ読み出し書き込み可能（R/W），読み出し専用（R），書き込み専用（W）および読み出したときリセットされる（R＆Reset）の４種類があります。ただし，読み出し書き込み可能（R/W）なフラグの中には，読み出し時，必ず“０”が読み出されるフラグがありますので，注意してください。

読み出し専用（RおよびR＆Reset）レジスタに書き込みを行っても何も変化しません。

書き込み専用（W）レジスタを読み出すと“不定”の値が読み出されます。

また，１ニブルの中の４ビット・データのうち，“０”に固定されているビットは，読み出したときは常に“０”となり，書き込みを行っても“０”を保持します。

未使用レジスタは，内容を読み出すと不定の値が読み出され，書き込みを行っても何も変化しません。

★ 未使用レジスタ，書き込み専用レジスタ（W）および読み込み専用レジスタ（R）を操作するときは，アセンブラ（RA17K）上注意が必要です。詳しくは**９.４　レジスタ・ファイル使用時の注意**を参照してください。

９.３.２　コントロール・レジスタの各周辺ハードウエア制御機能

周辺ハードウエアに対するコントロール・レジスタの役割は，各品種のデータ・シートに書いてあります。ご参照ください。

９.４　レジスタ・ファイル使用時の注意

９.４.１　コントロール・レジスタ（読み出し専用および未使用レジスタ）操作時の注意

コントロール・レジスタ（レジスタ・ファイルのアドレス00H-3FH番地）の読み出し専用レジスタ（R）および
★ 未使用レジスタを操作するときは次に示すようにデバイス動作上，17Kシリーズのアセンブラ（RA17K）および
インサーキット・エミュレータ（IE-17K, IE-17K-ET）使用時に注意が必要です。

（１）デバイス動作

読み出し専用レジスタに書き込みを行っても何も変化しません。

未使用部分を読み出すと“ 不定な値 ”が読み出され，書き込みを行っても何も変化しません。

★ **（２）アセンブラ（RA17K）使用時**

読み出し専用レジスタに書き込みを行う命令に“ エラー ”が発生します。

未使用部分を読み出したり，書き込みを行う命令に“ エラー ”が発生します。

★ **（３）インサーキット・エミュレータ（IE-17K, IE-17K-ET）使用時（バッチ処理等で操作したとき）**

読み出し専用レジスタに書き込みを行っても何も変化しません。“ エラー ”は発生しません。

未使用部分を読み出すと“ 不定な値 ”が読み出され，書き込みを行っても何も変化しません。“ エラー ”は発生しません。

９.４.２　レジスタ・ファイルのシンボル定義と予約語

★　17Kシリーズのアセンブラ（RA17K）を使用するうえでは，“ PEEK WR, rf ” および “ POKE rf, WR ” 命令のオペランド “ rf ” に直接数値でレジスタ・ファイル・アドレスを記述すると “ エラー ” が発生します。

したがって，例１に示すようにレジスタ・ファイルのアドレスをあらかじめシンボルとして定義する必要があります。

例１．エラーになる場合

```
PEEK    WR, 02H    ;
POKE    21H, WR    ;
```

エラーにならない場合

```
RF71    MEM0.71H   ; シンボル定義
PEEK    WR, RF71   ;
```

このとき次の点に注意してください。

●コントロール・レジスタを，データ・メモリ・アドレス型としてシンボル定義する場合，BANK0のアドレス80H-BFHとして定義する必要がある。

これは，コントロール・レジスタがウインドウ・レジスタを介して操作する仕組みになっているため，“ PEEK ” および “ POKE ” 以外の命令で操作した場合に，アセンブラで “ エラー ” を発生させる必要があるからです。

ただし，データ・メモリと重なっているレジスタ・ファイル（アドレス40H-7FH番地）は，そのままのアドレスでシンボル定義できます。

次にその例を示します。

★　**例２．**

```
RF71    MEM1.71H   ; データ・メモリと重なっているレジスタ・ファイル
RF02    MEM0.82H   ; コントロール・レジスタ

PEEK    WR, RF71   ; RF71は “ 71H番地 ” のデータ・メモリになる。
PEEK    WR, RF02   ; RF02はコントロール・レジスタの02H番地になる。
```

★　また，アセンブラ（RA17K）を使用する場合は，フラグ型シンボル操作命令として次のマクロ命令があらかじめアセンブラに組み込まれています。

SETn　　：フラグに“１”をセット
CLRn　　：フラグを“０”にリセット
SKTn　　：フラグがすべて“１”であればスキップ
SKFn　　：フラグがすべて“０”であればスキップ
NOTn　　：フラグを反転
INITFLG　：フラグをイニシャライズ
★　INITFLGX：フラグをイニシャライズ

したがって，これらの組み込みマクロ命令を使用することにより，次の例３に示すようにレジスタ・ファイルの内容を１ビット単位で操作することができます。

★　ここで，コントロール・レジスタは１ビット単位で操作するフラグが多いため，アセンブラ（RA17K）上であらかじめフラグ型シンボルとして“予約語”が定義されています。

ただし，スタック・ポインタにはフラグ型の予約語はありません。スタック・ポインタの予約語は唯一データ・メモリ型として“SP”で定義されています。このため，予約語によるフラグの操作命令は，使用できません。

★　**例３．**INITFLG　WDTRES　；イニシャライズ
（SET1　WDTRES　；フラグのセット）

マクロ展開

```
PEEK   WR, .MF.WDTRES SHR4
OR     WR, #.DF.WDTRES AND 0FH
POKE   .MF.WDTRES SHR4, WR
```

★　９.４.３　アセンブラ（RA17K）組み込みマクロ命令使用時の注意

コントロール・レジスタにアセンブラ組み込みマクロ命令を使用するときは次の（１）および（２）に示す点に注意が必要です。

（１）スタック・ポインタに対してフラグ操作組み込みマクロ命令は使えない

９.４.２で説明したようにスタック・ポインタにはフラグ型の予約語が定義されていないため，予約語によるフラグ操作命令は使えません。

（２）書き込み専用レジスタに対してフラグ操作組み込みマクロ命令は使えない

書き込み専用レジスタに対してはフラグ操作組み込みマクロ命令は使えません。

書き込み専用レジスタに“SETn”組み込みマクロ命令を用いると，一度レジスタ・ファイルの内容をウインドウ・レジスタに読み込む動作を行います。

このとき，ウインドウ・レジスタに読み込まれた値が不定となってしまうので（書き込み専用レジスタ・ファイルは，その内容を読み出すと不定の値となる），“SETn”命令で指定されてないビットに不定の値を書き込んでしまいます。

★　このときアセンブラ（RA17K）はエラーを発生します。

第10章　データ・バッファ（DBF）

データ・バッファは周辺ハードウエアとのデータ転送時およびテーブル参照のデータ読み込み時に使用するバッファです。

10.1　データ・バッファの構成

図10－１のように，データ・バッファは，データ・メモリ上のBANK0のアドレス0CH-0FHに割り当てられており，４ワード×４ビットの計16ビットから構成されています。

データ・バッファは，データ・メモリ上にあるので，すべてのデータ・メモリ操作命令が使えます。

図10－１　データ・バッファの配置

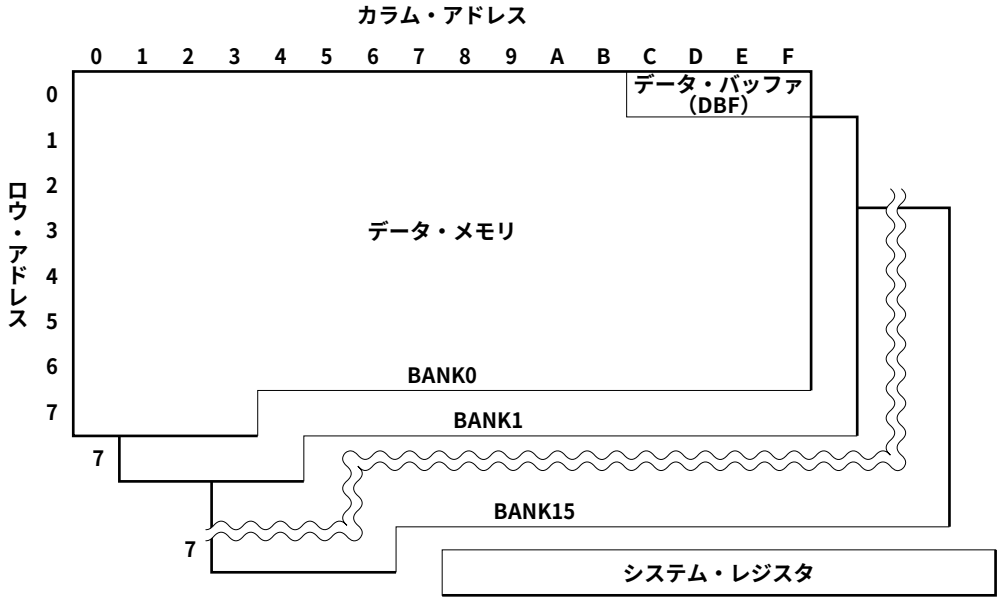


図10－２のようにデータ・バッファはデータ・メモリ・アドレス0FH番地のビットb_0をLSBとし，アドレス0CH番地のビットb_3をMSBとする計16ビットのバッファです。

図10－２　データ・バッファの構成

データ・メモリ	アドレス	0CH				0DH				0EH				0FH			
	ビット	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
データ・バッファ	ビット	b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
	記号	DBF3				DBF2				DBF1				DBF0			
	データ	〈MSB〉															〈LSB〉
		データ															

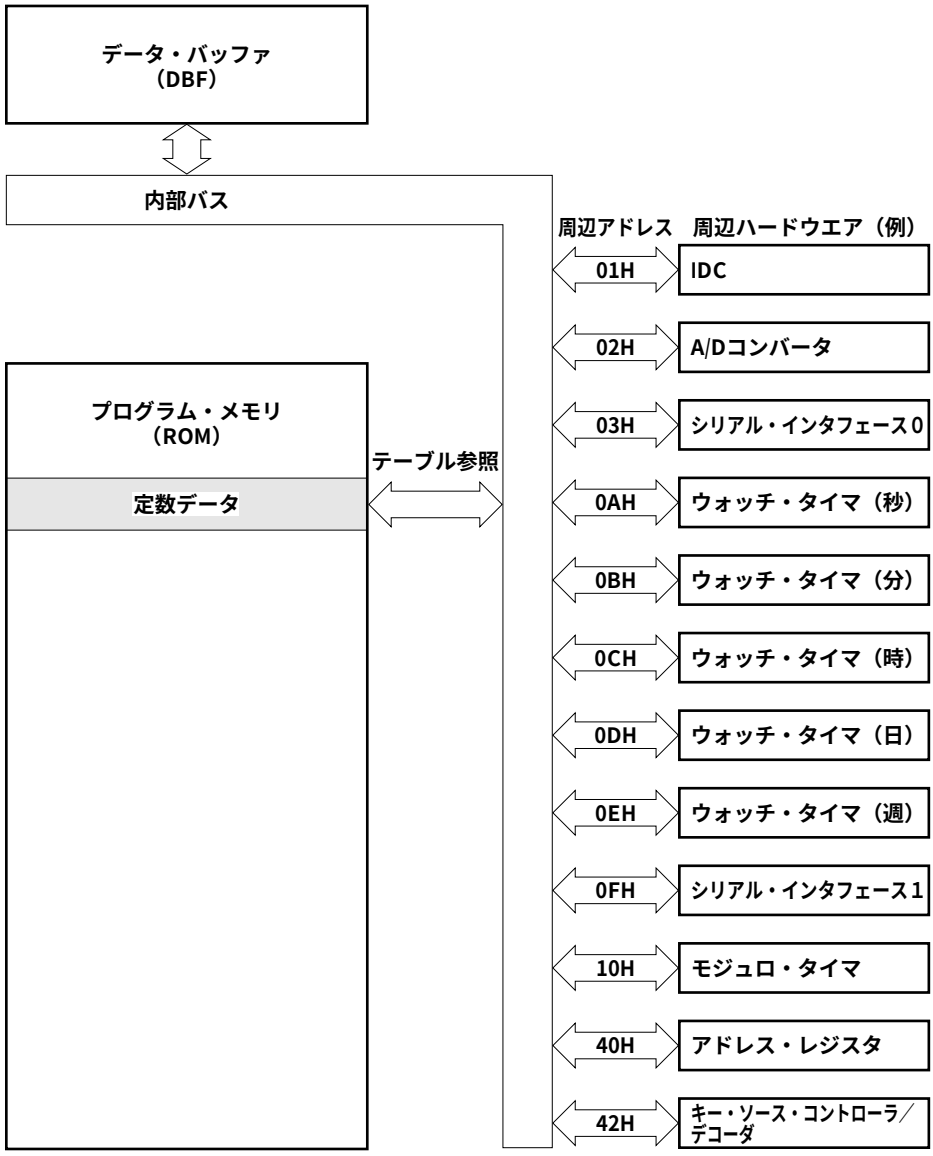
10.2　データ・バッファの機能

データ・バッファには，次の２つの機能があります。

（1）プログラム・メモリ上の定数データの読み込み（テーブル参照）機能
（2）周辺ハードウエアとのデータ転送機能

図10－３はデータ・バッファと周辺ハードウエアおよびテーブル参照の関係の例です。
テーブル参照については，**10.4　データ・バッファとテーブル参照**を，周辺ハードウエアとの関係については，**10.5　データ・バッファと周辺ハードウエア**をそれぞれご参照ください。

図10－３　データ・バッファと周辺ハードウエアおよびテーブル参照との関係（例）



10.3　データ・バッファ使用時の注意

10.3.1　書き込み専用，読み出し専用のレジスタおよび未使用アドレスの操作時について

データ・バッファを介して周辺ハードウエアとのデータ転送をするとき，未使用の周辺アドレスや書き込み専用周辺レジスタ（PUTのみ）および読み出し専用周辺レジスタ（GETのみ）については次の点にご注意ください。

（1）デバイス動作

書き込み専用レジスタを読み出すと“不定な値”が読み出されます。
読み出し専用レジスタに書き込みを行っても何も変化しません。
未使用アドレスを読み出すと“不定な値”が読み出され，書き込んでも何も変化しません。

★ **（2）アセンブラ（RA17K）使用時**

書き込み専用レジスタを読み出す命令に対して“エラー”が発生します。
読み出し専用レジスタに書き込む命令に対して“エラー”が発生します。
未使用アドレスを読み出す命令および書き込む命令に対して“エラー”が発生します。

★ **（3）エミュレータ（IE-17K, IE-17K-ET）使用時（バッチ処理等で命令を実行したとき）**

書き込み専用レジスタを読み出すと“不定な値”が読み出されます。“エラー”は発生しません。
読み出し専用レジスタに書き込みを行っても何も変化しません。“エラー”は発生しません。
未使用アドレスを読み出すと“不定な値”が読み出され，書き込んでも何も変化しません。“エラー”は発生しません。

★ 10.3.2　周辺レジスタのアドレス指定について

17Kシリーズのアセンブラ（RA17K）では，例１のように“PUT p, DBF”および“GET DBF, p”命令で，周辺アドレス“p”を直接（数値で）指定しても“エラー”は発生しません。
ただし，この方法はプログラムのバグ低減のためには好ましくありません。
例２のように，アセンブラ疑似命令であるシンボル定義疑似命令を用いて周辺アドレスをシンボル定義しておくことをおすすめします。

★ シンボル定義は，簡略化のため，アセンブラ（RA17K）内にあらかじめ周辺アドレスを“予約語”として定義しています。
したがって，予約語を用いれば，例３のようにシンボル定義をせずにプログラムを作成できます。
予約語については各品種のデータ・シートを参照してください。

```
例１．PUT     02H, DBF        ；周辺アドレスを02Hや03Hで直接指定しても，アセンブラはエラーとなら
      GET     DBF, 03H        ；ないが，プログラムのバグ低減のためには好ましくない。

   ２．SIO0DATA DAT 03H       ；シンボル定義疑似命令によりSIO0DATAを03Hに割り当てる。
       PUT     SIO0DATA, DBF ；

   ３．PUT     SIO0SFR, DBF    ；予約語である“SIO0SFR”を用いればシンボル定義を行う必要がない。
```

10.4　データ・バッファとテーブル参照

10.4.1　テーブル参照動作

テーブル参照は“ MOVT DBF, @AR ”命令を用いることにより，プログラム・メモリ上の定数データを，データ・バッファ上に読み込むことができます。

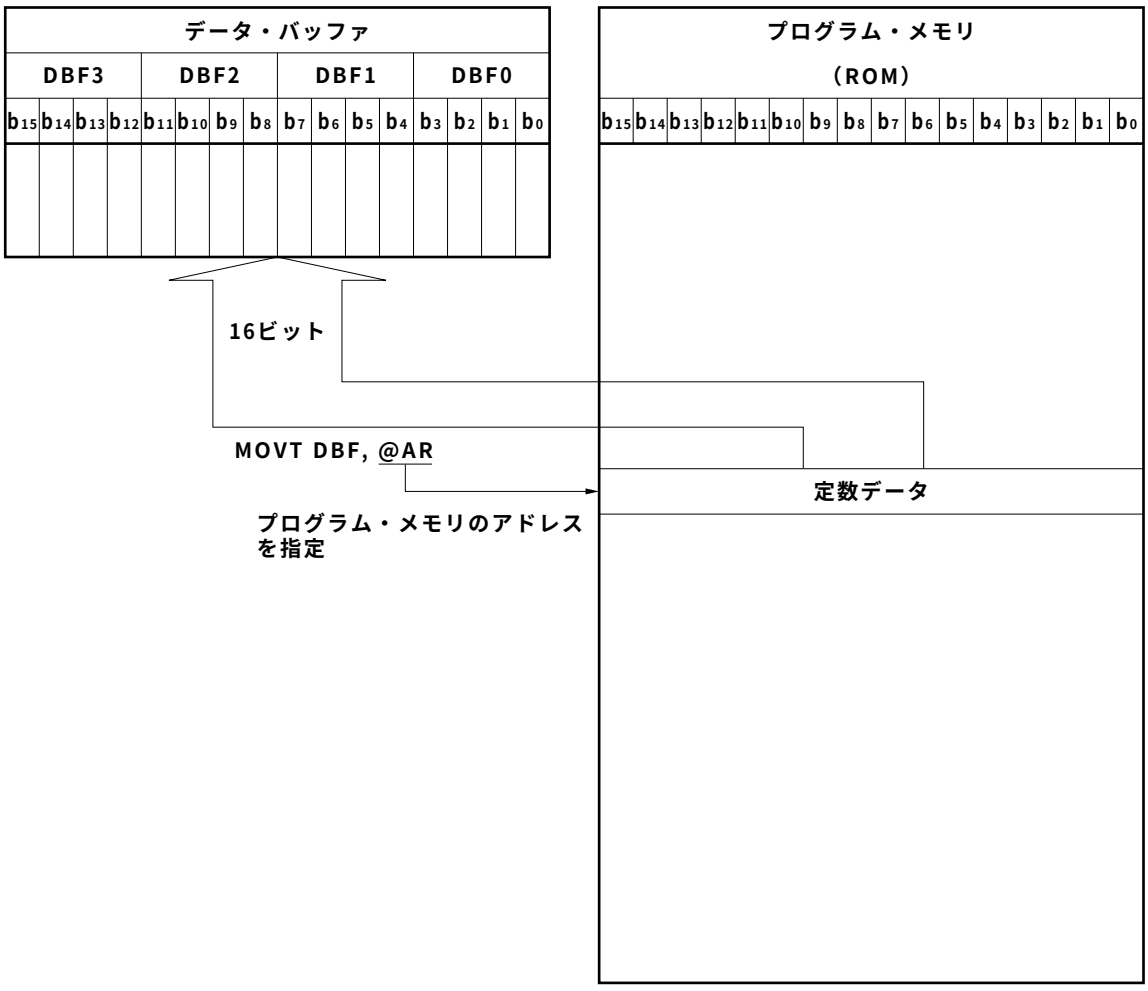
したがって，プログラム・メモリ上に表示データなどの定数データをあらかじめ書き込んでおき，必要なときにテーブル参照すれば，複雑なデータ変換プログラムは不要です。

次に“ MOVT ”命令について説明します。

例

```
MOVT　DBF, @AR　；図10－4のようにアドレス・レジスタの内容で指定するプログラム・メモリの内
                  容を，データ・バッファに読み出します。
```

図10－4　テーブル参照動作例



テーブル参照命令実行時はスタックが１レベル使用されます。

テーブル参照が可能なプログラム・メモリ・アドレスはアドレス・レジスタのビット数によって品種ごとに異なります。

第４章　アドレス・スタック（ASK）および６.３　アドレス・レジスタ（AR）も参照してください。

10.４.２　テーブル参照プログラム例

次の例１，例２および例３はテーブル参照のプログラム例です。

例１.

```
M000   MEM 0.00H    ;
P0A    MEM 0.70H    ;
P0B    MEM 0.71H    ;
P0C    MEM 0.72H    ;

START：                ;プログラム・アドレス0000H番地
    BR     MAIN

DATA：
    DW     0001H       ;定数データ
    DW     0002H       ;
    DW     0004H       ;
    DW     0008H       ;
    DW     0010H       ;
    DW     0020H       ;
    DW     0040H       ;
    DW     0080H       ;
    DW     0100H       ;
    DW     0200H       ;
    DW     0400H       ;
    DW     0800H       ;
MAIN：
    BANK0              ;組み込みマクロ
    SET4   P0ABIO3, P0ABIO2, P0ABIO1, P0ABIO0
    SET4   P0BBIO3, P0BBIO2, P0BBIO1, P0BBIO0
    SET1   P0CGIO
    MOV    RPH, #0000B ;ジェネラル・レジスタをBANK0のロウ・アドレス7Hに設定
    MOV    RPL, #0100B ;
    MOV    AR3, #（.DL.DATA SHR 12 AND 0FH）
    MOV    AR2, #（.DL.DATA SHR  8 AND 0FH）
    MOV    AR1, #（.DL.DATA SHR  4 AND 0FH）
    MOV    AR0, #（.DL.DATA SHR  0 AND 0FH）
                       ;アドレス・レジスタ（AR）を0001Hに設定
```

```
LOOP：
；①
    MOVT  DBF, @AR     ；データ・バッファにARの内容で指定されるROMの値を
                       ；転送
；②
    LD    P0A, DBF2    ；データ・バッファの値をPort0A（70H）,Port0B（71H）
    LD    P0B, DBF1    ；およびPort0C（72H）の各ポート・データ・レジスタへ転送
    LD    P0C, DBF0    ；
    ADD   M000, #1     ；アドレス・レジスタの内容を１だけインクリメント
    ADD   AR0, M000
    ADDC  AR1, #0
    ADDC  AR2, #0
    ADDC  AR3, #0
    SKNE  AR0, #0CH    ；AR0の値が0CHになったときAR0に０を書き込む。
    MOV   AR0, #0      ；
    BR    LOOP
```

上記プログラムは，プログラム・メモリのアドレス0001H-000CH番地の定数データを①で順次データ・バッファに読み込み，②でPort0A, Port0B, Port0Cに出力します。

このときこれらの定数データは，内容が“１”のビットが１ビットずつ順に左にシフトしているので，結果としてPort0A, Port0BおよびPort0Cの各端子に順次ハイ・レベルを出力していきます。

この例１では，定数データを格納するプログラム・メモリ上の開始アドレスを，“MOV”命令でアドレス・レジスタに設定しています。

このように“MOV”命令を使用すると，たとえば格納する定数データの種類が多い場合には，各定数データの開始アドレスを何度もアドレス・レジスタに設定しなければなりません。

したがって，“MOV”命令を何度も使用してステップ数が増加しそうな場合や，共通なルーチンで管理したいときなどには例２のようなプログラムが便利です。

例２．

```
    M000    MEM 0.00H     ;
START：
    BR      MAIN          ;
DATAFETCH：
    DI                    ;
    POP     AR            ;アドレス・スタック・レジスタの内容をアドレス・レジスタに読み込
                          ;む。このときスタック・ポインタはメイン・ルーチンの戻り番地を指す
    ADD     AR0, M000     ;M000の内容で指定される定数データ・アドレス分をシフトする。
    ADDC    AR1, #0       ;
    ADDC    AR2, #0       ;
    ADDC    AR3, #0       ;
    MOVT    DBF, @AR      ;定数データの読み込み
    EI
    RET                   ;メイン・ルーチンに戻る。
DATA1：
    CALL    DATAFETCH     ;共通処理ルーチンをコール
    DW      0123H         ;このとき，アドレス・スタック・レジスタにDATA1＋1のアドレスが退
    DW      4567H         ;避される。
            ⋮
    DW      89ABH         ;
DATA2：
    CALL    DATAFETCH     ;共通処理ルーチンをコール
    DW      1357H         ;このとき，アドレス・スタック・レジスタにDATA2＋1のアドレスが退
    DW      2468H         ;避される。
            ⋮
    DW      9BDFH         ;
MAIN：
    BANK0                 ;組み込みマクロ
    SET4    P0ABIO3, P0ABIO2, P0ABIO1, P0ABIO0
    SET4    P0BBIO3, P0BBIO2, P0BBIO1, P0BBIO0
    SET1    P0CGIO
    MOV     RPH, #0000B ;ジェネラル・レジスタをBANK0のロウ・アドレス7Hに設定
    MOV     RPL, #0100B ;
LOOP：
    CALL    DATA1         ;M000の内容で指定される定数データDATA1の値を読み込む。
    LD      P0A, DBF2     ;
    LD      P0B, DBF1     ;データ・バッファの値をPort0A（70H）,Port0B（71H）
    LD      P0C, DBF0     ;およびPort0C（72H）の各ポート・データ・レジスタへ転送
    CALL    DATA2         ;M000の内容で指定される定数データDATA2の値を読み込む。
    LD      P0A, DBF2     ;
    LD      P0B, DBF1     ;データ・バッファの値をPort0A（70H）,Port0B（71H）
    LD      P0C, DBF0     ;およびPort0C（72H）の各ポート・データ・レジスタへ転送
    ADD     M000, #1      ;
```

```
    SKNE   M000, #0CH  ; M000の内容が0CHになったときAR0に０を書き込む。
    MOV    M000, #0    ;
    BR     LOOP
```

例２では，“ CALL ”命令が２回と，“ POP ”，“ MOVT ”命令が実行されるため，スタックが２レベル必要になります。

また，例３のように“ CALL ”命令を１回のみで行うこともできますが，この場合も“ MOVT ”命令によりスタックが２レベル必要です。

例３． DATAFETCH：

```
    DI                 ;
    POP    AR          ; アドレス・スタック・レジスタの内容をアドレス・レジスタに読み込む。
    MOVT   DBF, @AR    ; データ・バッファに定数データの格納アドレスを転送
    INC    AR          ; メイン・ルーチンの戻り番地を格納
    PUSH   AR          ;
    PUT    AR, DBF     ; アドレス・レジスタに定数データ格納アドレスを転送
    ADD    AR0, M000   ; M000の内容で指定される定数データ・アドレス分をシフトする。
    ADDC   AR1, #0     ;
    ADDC   AR2, #0     ;
    ADDC   AR3, #0     ;
    MOVT   DBF, @AR    ; 定数データの読み込み
    EI
    RET                ; メイン・ルーチンに戻る。
DATA1：
    DW     0123H       ; 定数データ
                       ;
DATA2：
    DW     1357H       ; 定数データ
                       ;
MAIN：

LOOP：
    CALL   DATAFETCH   ;
    DW     .DL.DATA1   ;
    LD     P0A, DBF2   ;

    CALL   DATA2       ;
    DW     .DL.DATA2   ;
    LD     P0A, DBF2   ;

    BR     LOOP
```

10.5　データ・バッファと周辺ハードウエア

10.5.1　周辺ハードウエアの制御方法

中央演算ユニット（CPU）は，データ・バッファを介して，周辺ハードウエアにデータを設定したり，周辺ハードウエアからデータを読み込んだりして，周辺ハードウエアを制御します。

周辺ハードウエアはそれぞれデータ転送用のレジスタ（周辺レジスタと呼ぶ）を持ち，このレジスタにはそれぞれアドレス（周辺アドレスと呼ぶ）が割り付けられています。

この周辺レジスタに対して，専用命令“GET”および“PUT”により，データ・バッファと各周辺ハードウエアとのデータ転送ができます。

“GET”および“PUT”命令は次の動作をします。

```
GET   DBF, p ；データ・バッファにpでアドレスされる周辺レジスタのデータを読み込む。
PUT   p, DBF ；pでアドレスされる周辺レジスタにデータ・バッファのデータを設定する。
```

周辺レジスタには書き込み読み出し可能（PUT/GET），書き込み専用（PUT）および読み出し専用（GET）のものがあります。

このとき，書き込み専用（PUTのみ）に“GET”命令や，読み出し専用（GETのみ）に“PUT”命令を使うと，次のようになります。

- 書き込み専用（PUTのみ）周辺レジスタに対して読み出し（GET）命令実行時
  …不定の値が読み出されます。
- 読み出し専用（GETのみ）周辺レジスタに対して書き込み（PUT）命令実行時
  …何も変わりません。

★　これについては，17Kシリーズのアセンブラ（RA17K）やエミュレータ（IE-17K, IE-17K-ET）使用時に特に注意が必要です。必ず**10.3　データ・バッファ使用時の注意**を参照してください。

また，周辺レジスタについては，各品種のデータ・シートを参照してください。

10.5.2　周辺レジスタとのデータ転送時のデータ長の取り扱い

データ・バッファと各周辺レジスタとのデータ転送は８ビットまたは16ビット単位で行います。

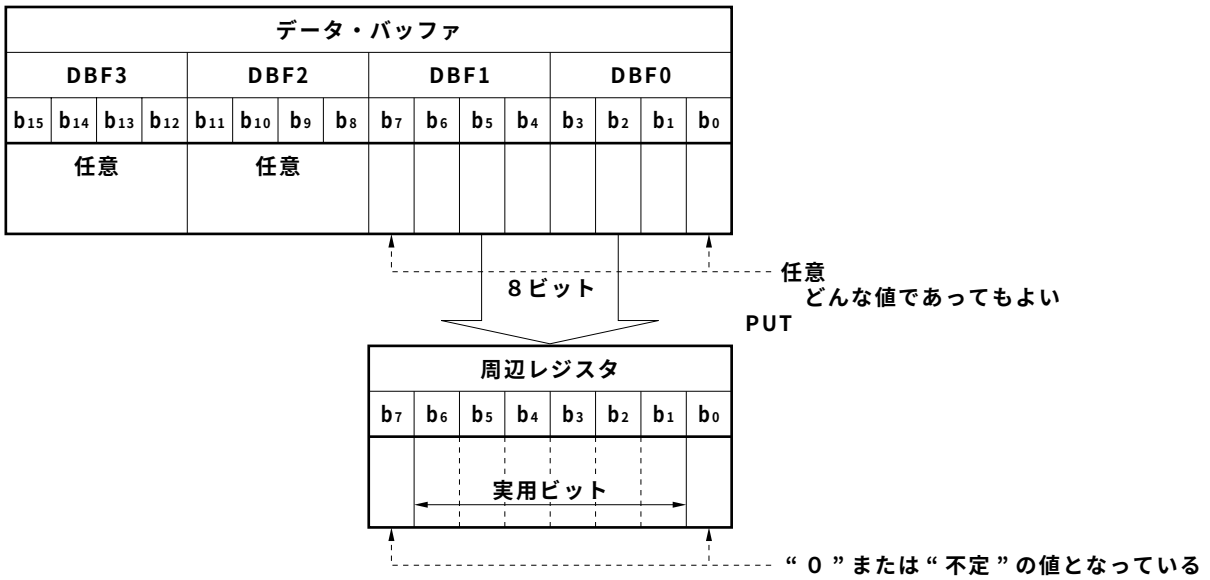
このとき，“ PUT ”および“ GET ”命令は，データ・ビットが16ビットであっても１命令実行時間で実行します。

また，たとえば周辺レジスタの実用データ・ビット長が７ビットであるときに８ビットのデータ転送を行うと１ビットは余分なデータとなります。

この余分なデータ・ビット分は例１，例２のように書き込み時は“ 任意（どんな値でもよい） ”，読み出し時には“ 不定な値 ”になります。

図10－５は，“ PUT ”命令時（周辺レジスタの実用ビットがビットb_1-ビットb_6の６ビットであるとき）の動作の例です。

図10－５　データ・バッファと周辺ハードウエアとの間のデータ転送例１

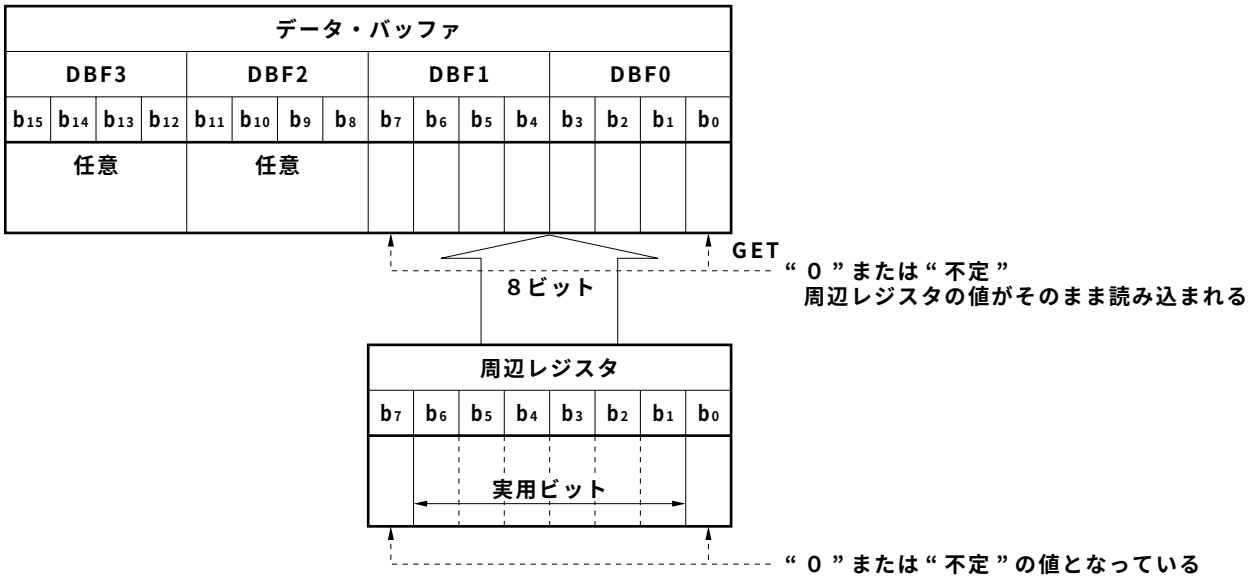


周辺レジスタに８ビット・データを書き込むときはデータ・バッファの上位８ビット（DBF3およびDBF2の内容）は“ 任意（どんな値であってもよい） ”です。

データ・バッファの８ビット・データのうち，周辺レジスタの実用ビットに対応しない各ビットは“ 任意 ”となります。

図10－６は，“ GET ”命令時の動作の例です。

図10－６　データ・バッファと周辺ハードウエアとの間のデータ転送例２



周辺レジスタの８ビット・データを読み出すときにはデータ・バッファの上位８ビット（DBF3およびDBF2）の値は変化しません。

データ・バッファの８ビット・データのうち，周辺レジスタの実用ビットでない各ビットは“ 0 ”または“ 不定 ”となります。“ 0 ”となるか“ 不定 ”となるかは，周辺レジスタによりあらかじめ決まっています。

第11章　汎用ポート

汎用ポートは，外部回路への信号の出力，外部回路からの信号の読み込みを行います。

11.1　汎用ポートの構成

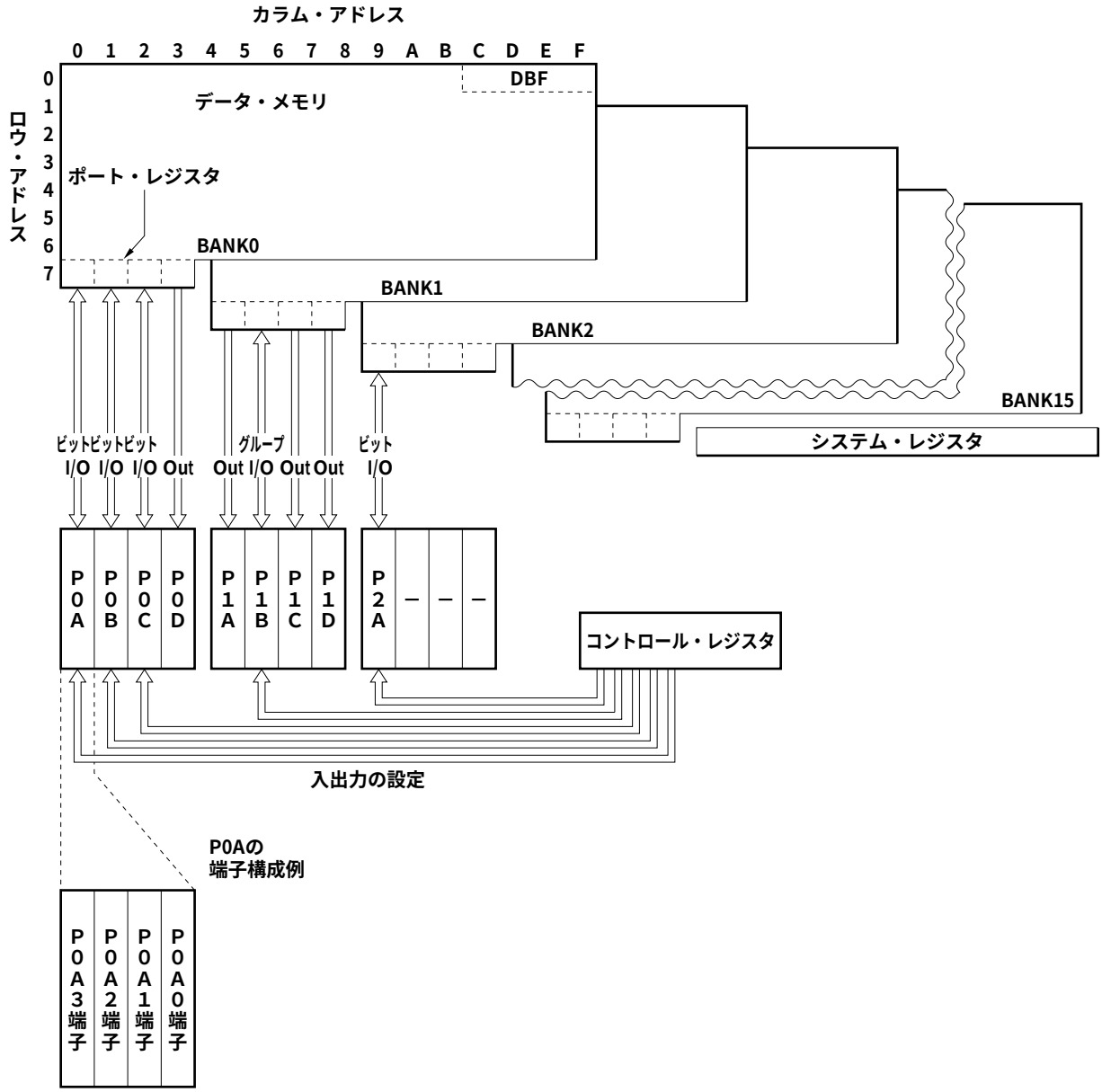
図11－１のように，汎用ポートは入力／出力するデータをデータ・メモリの各バンクのアドレス70H番地から73H番地（ポート・レジスタ）に書き込みます。

各ポートは，それぞれ端子（たとえばPort0A端子は$P0A_3$端子-$P0A_0$端子）を持ちます。

また，汎用ポート端子には，入出力ポート，入力ポートおよび出力ポートがあります。

入出力ポートには，１ビット単位（１端子単位）で入出力を指定できるビットI/Oと，４ビット単位（４端子単位）で入出力を指定できるグループI/Oポートとがあります。

図11－1　汎用ポートのブロック図



11.2　汎用ポートの機能

汎用出力ポートまたは出力ポートに設定されている汎用入出力ポートは，ポート・レジスタにデータを設定することにより，対応する各端子からハイ・レベルまたはロウ・レベルを出力します。

また，汎用入力ポートまたは入力ポートに設定されている汎用入出力ポートは，ポート・レジスタの内容を読み込むことにより，対応する各端子に加えられている入力信号レベルを検出できます。

汎用入出力ポートの入力ポートおよび出力ポートの切り替えは各ポートに対応するコントロール・レジスタにより行います。

すなわち，プログラムで入出力を切り替えることが可能です。

汎用入出力ポートは，パワーオン・リセット時に汎用入力ポートに設定されるため，他の周辺ハードウエアを兼用している端子は対応するコントロール・レジスタにより独立して指定します。

11.2.1　汎用ポート・データ・レジスタ（ポート・レジスタ）

ポート・レジスタは各汎用ポートの出力データの設定や入力データの読み込みを行います。

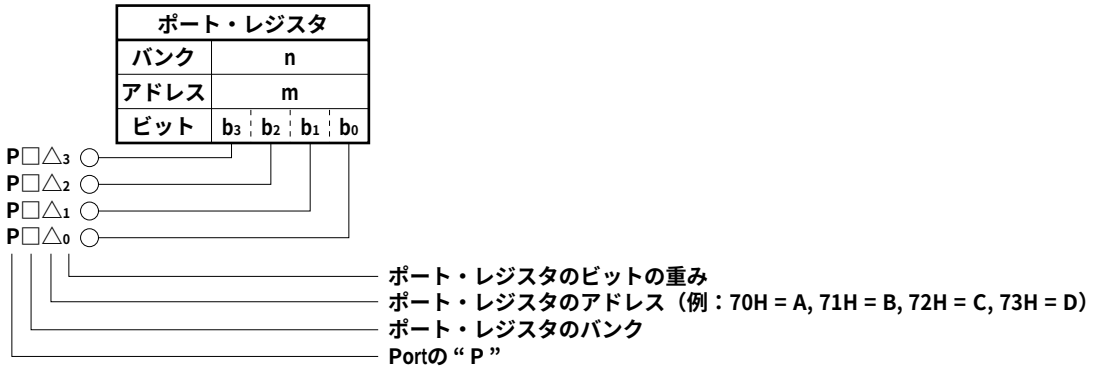
ポート・レジスタは，データ・メモリ上に配置されているため，すべてのデータ・メモリ操作命令で操作することができます。

図11－2にポート・レジスタと対応する各端子の関係を示します。

汎用出力ポートに設定されている端子に対応するポート・レジスタにデータを設定することにより各端子の出力を設定します。

また，汎用入力ポートに設定されている端子に対応するポート・レジスタを読み込むことにより各端子の入力状態を検出します。

図11－2　ポート・レジスタと各端子の関係



ポート・レジスタにはアセンブラ上で予約語が定義されています。

この予約語はフラグ（ビット）単位で定義されているため，アセンブラ組み込みマクロ命令を使用できます。

ポート・レジスタにはデータ・メモリ型の予約語は定義されていませんので注意してください。

〔メ　モ〕

第12章　割り込み機能

割り込み機能は，周辺ハードウエアの中から指定しておいたものが，あらかじめ決めておいた特定のデータを出力したら，速やかにいままでの処理を中断し，その特定のデータの発生に対して実行すべきプログラムを実行する機能です。

そのために，周辺ハードウエアからの要求の発生と，それによってプログラムの実行を中断し，あらかじめ指定した番地（ベクタ・アドレス）から始まるプログラムへの分岐とを行います。

12.1　割り込みブロックの構成

図12－１のように，割り込みブロックは割り込み要求を制御する各割り込み要求ブロックと，割り込み全体の許可をする割り込みイネーブル・フリップフロップ（INTE）と，割り込みが受け付けられたときに制御されるスタック・ポインタ，アドレス・スタック・レジスタ，プログラム・カウンタ，割り込みスタックとから構成されます。

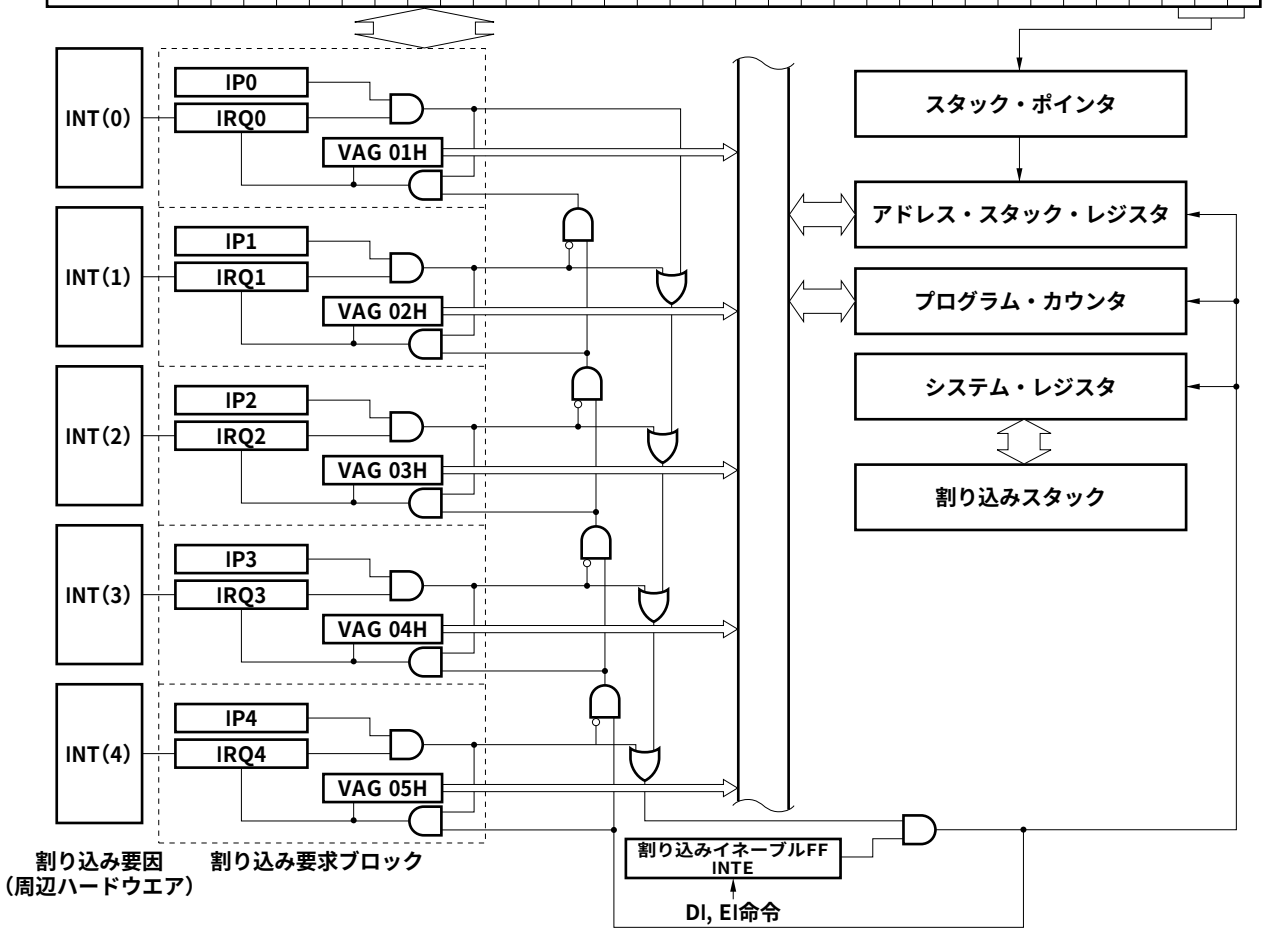
割り込み要求は，周辺ハードウエアが出します。

各周辺ハードウエアの割り込み要求処理ブロックは，割り込み要求を検出するインタラプト・リクエスト・フラグ（IRQ×××）と，各割り込みの許可を設定するインタラプト・パーミッション・フラグ（IP×××），および割り込み受け付け時にベクタ・アドレス（分岐先番地）を指定するベクタ・アドレス・ジェネレータ（VAG）とで構成されます。

インタラプト・リクエスト・フラグ（IRQ×××）およびインタラプト・パーミッション・フラグ（IP×××）は，図12－１では割り込み処理ブロック内に描かれていますが，実際はコントロール・レジスタの中のインタラプト・リクエスト・レジスタおよびインタラプト・パーミッション・レジスタ内にあります。

図12－1　割り込みブロックの構成例

コントロール・レジスタ																													
名称	インタラプトリクエスト1（INTREQ1）				インタラプトリクエスト2（INTREQ2）				インタラプトリクエスト3（INTREQ3）				インタラプトリクエスト4（INTREQ4）				インタラプトリクエスト5（INTREQ5）				インタラプトパーミッション1（INTPM1）				インタラプトパーミッション2（INTPM2）					スタック・ポインタ SP			
アドレス	3BH				3CH				3DH				3EH				3FH				2EH				2FH					01H			
ビット	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0		b_3	b_2	b_1	b_0
フラグ記号	0	0	0	IRQ4	0	0	0	IRQ3	0	0	0	IRQ2	0	0	0	IRQ1	0	0	0	IRQ0	0	0	0	IP4	IP3	IP2	IP1	IP0		〈SP3〉	〈SP2〉	〈SP1〉	〈SP0〉



12.2　割り込みの機能

割り込み機能とは，これらの周辺ハードウエアが，ある状態になったときに，そのとき実行していたプログラムを一時中断し，専用の処理プログラムを実行させる機能です。

このとき，周辺ハードウエアからの割り込み信号を“割り込み要求”と呼び，割り込み信号が発生することを“割り込み要求が発行される”といいます。また，割り込み専用の処理プログラムを“割り込み処理ルーチン”と呼びます。

割り込みが受け付けられると，各割り込み要因ごとに決められたプログラム・メモリ・アドレス（ベクタ・アドレス）の内容を読み出して実行します。これを分岐と呼びます。したがって，このベクタ・アドレスから，各割り込み処理ルーチンを開始させます。

割り込み機能は割り込みを受け付けるまでの処理と割り込みを受け付けたあとの処理に分かれます。すなわち，各周辺ハードウエアからの割り込み要求に対してその割り込みを受け付けるまでの機能と，割り込みを受け付けてからベクタ・アドレスへ分岐し，さらに割り込み前のプログラムに復帰する機能に分かれます。

12.2.1-12.2.5は図12－1の各ブロックの機能を示します。

12.2.1　周辺ハードウエア

各周辺ハードウエアには，割り込み要求を発行する条件を設定できます。

たとえば外部割り込み端子は，割り込み要求の発行を端子に加えられた信号の立ち上がり立ち下がりのいずれかに選択できます。

各周辺ハードウエアの割り込み要求発行条件の詳細については各品種ごとのデータ・シートを参照してください。

12.2.2　割り込み要求処理ブロック

割り込み要求処理ブロックは各周辺ハードウエアごとに設けられており，それぞれの割り込み要求の有無，割り込みの許可および割り込み受け付け時のベクタ・アドレスの生成を行います。

次の12.2.3，12.2.4は割り込み要求処理ブロックの各フラグについて説明します。

12.2.3　インタラプト・リクエスト・フラグ（IRQ×××）の構成と機能

各インタラプト・リクエスト・フラグ（IRQ×××）は，各周辺ハードウエアから割り込み要求が発行されると“１”にセットされ，割り込みが受け付けられると“０”にリセットされます。

割り込みが許可されていない場合などに，これらのインタラプト・リクエスト・フラグ（IRQ×××）を検出することにより，各割り込み要求の発行状態を検出することができます。

また，インタラプト・リクエスト・フラグに対して，ウインドウ・レジスタを介して直接“１”を書き込んだ場合も，割り込み要求が発行されたことと同等になります。

一度このフラグが“１”にセットされると，対応する割り込みが受け付けられるか，または割り込み要求リセット・マクロが実行されるまでリセットされません。

複数の割り込み要求が同時に発行された場合でも，受け付けられなかった割り込みに対応するインタラプト・リクエスト・フラグは，リセットされません。

次に，インタラプト・リクエスト・フラグの構成と機能を示します。

図12－２　インタラプト・リクエスト・フラグの構成例

レジスタ名	インタラプト・リクエスト・レジスタ１				インタラプト・リクエスト・レジスタ２			
アドレス	品種によって異なる							
R/W								
ビット	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
フラグ略号	0	0	0	IRQ4	0	0	0	IRQ3

0に固定　　0に固定

IRQ3	INT（3）の割り込み要求の発行状態を検出
0	割り込み要求なし
1	割り込み要求あり

IRQ4	INT（4）の割り込み要求の発行状態を検出
0	割り込み要求なし
1	割り込み要求あり

12.2.4　インタラプト・パーミッション・フラグ（IP×××）の構成と機能

各インタラプト・パーミッション・フラグは，周辺ハードウエアごとの割り込みの許可を設定します。

★　割り込みが受け付けられるためには，次の３つの条件をすべて満たす必要があります。

・各インタラプト・パーミッション・フラグにより割り込みが許可されている

・対応する各インタラプト・リクエスト・フラグにより割り込み要求が発行されている

・“ EI ”命令（すべての割り込み許可）が実行されている

インタラプト・パーミッション・フラグは，コントロール・レジスタ内のインタラプト・パーミッション・レジスタにあるのでウインドウ・レジスタ（WR）を介して読み書きできます。

一度このフラグをセットすると，ウインドウ・レジスタを介して“ 0 ”を書き込むまでリセットされません。

次にインタラプト・パーミッション・フラグの構成と機能を示します。

図12－３　インタラプト・パーミッション・フラグの構成例

レジスタ名	インタラプト・パーミッション・レジスタ１				インタラプト・パーミッション・レジスタ２			
アドレス	品種によって異なる							
R/W								
ビット	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
フラグ略号	0	0	0	IP4	IP3	IP2	IP1	IP0

0に固定

IP0	INT（0）の割り込みの許可を設定
0	許可しない
1	許可する

IP1	INT（1）の割り込みの許可を設定
0	許可しない
1	許可する

IP2	INT（2）の割り込みの許可を設定
0	許可しない
1	許可する

IP3	INT（3）の割り込みの許可を設定
0	許可しない
1	許可する

IP4	INT（4）の割り込みの許可を設定
0	許可しない
1	許可する

12.2.5　スタック・ポインタ，アドレス・スタック・レジスタとプログラム・カウンタ

アドレス・スタック・レジスタは，割り込み処理ルーチンからの復帰時の戻り番地を退避します。

スタック・ポインタは，アドレス・スタック・レジスタのアドレスを指定します。

つまり，割り込みが受け付けられると，スタック・ポインタの値を－１し，そのときのプログラム・カウンタの値をスタック・ポインタで指定されるアドレス・スタック・レジスタに退避します。

次に，割り込み処理ルーチンの処理を実行したあと，専用復帰命令である“ RETI ”命令が実行されると，スタック・ポインタで指定されるアドレス・スタック・レジスタの内容をプログラム・カウンタに復帰し，スタック・ポインタの値を＋１します。

第４章　アドレス・スタック（ASK）も参照してください。

12.2.6　割り込みイネーブル・フリップフロップ（INTE）

割り込みイネーブル・フリップフロップは，すべての割り込みの許可を設定します。

このフリップフロップがセットされていると，すべての割り込みが許可されます。また，リセットされていると，すべての割り込みが禁止されます。

このフリップフロップのセットおよびリセットは，専用命令である“ EI（セット）”および“ DI（リセット）”命令で行います。

“ EI ”命令は，“ EI ”命令の次の命令が実行された時点でこのフリップフロップをセットし，“ DI ”命令は，“ DI ”命令実行中にこのフリップフロップをリセットします。

割り込みが受け付けられると，このフリップフロップは自動的にリセットされます。

DI状態中に，“ DI ”命令を実行しても，またはEI状態中に“ EI ”命令を実行しても，何も影響は与えません。

★　パワーオン・リセット時，クロック・ストップ命令実行時，およびCEリセット時は，このフリップフロップはリセットされます。

12.2.7　ベクタ・アドレス・ジェネレータ（VAG）

各周辺ハードウエアの割り込みを受け付けたときに，受け付けた割り込み要求に対するプログラム・メモリの分岐先アドレス（ベクタ・アドレス）を発生します。

12.2.8　割り込みスタック

割り込みスタックは，図12－４のような構成で，割り込みを受け付けたときに，システム・レジスタの内容を退避します。

割り込みを受け付け，システム・レジスタの内容を退避すると，システム・レジスタはリセット（０）されます。

図12－４　割り込みスタックの構成例

スタック名		割り込みスタック（INTSK）								
		バンク・スタック（BANKSK）				ステータス・スタック（PSWSK）				
ビット		b_3	b_2	b_1	b_0	b_4	b_3	b_2	b_1	b_0
アドレス	0H	－		BANKSK0		BCD SK0	Z SK0	CY SK0	CMP SK0	IXE SK0
	1H	－		BANKSK1		BCD SK1	Z SK1	CY SK1	CMP SK1	IXE SK1

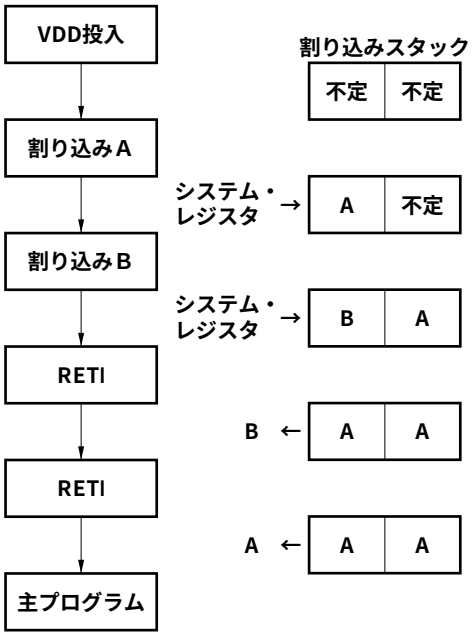
割り込みスタックは，各品種の持つ最大スタック・レベルまで退避することができます。したがって，ある割り込み処理ルーチンの中で別の割り込みを受け付けるような多重割り込みを，そのレベルまで行うことができます。

図12－５の（ａ）のように，割り込みが受け付けられるたびにデータを退避していき，割り込みリターン命令（RETI）が実行されるごとにデータをシステム・レジスタに復帰します。

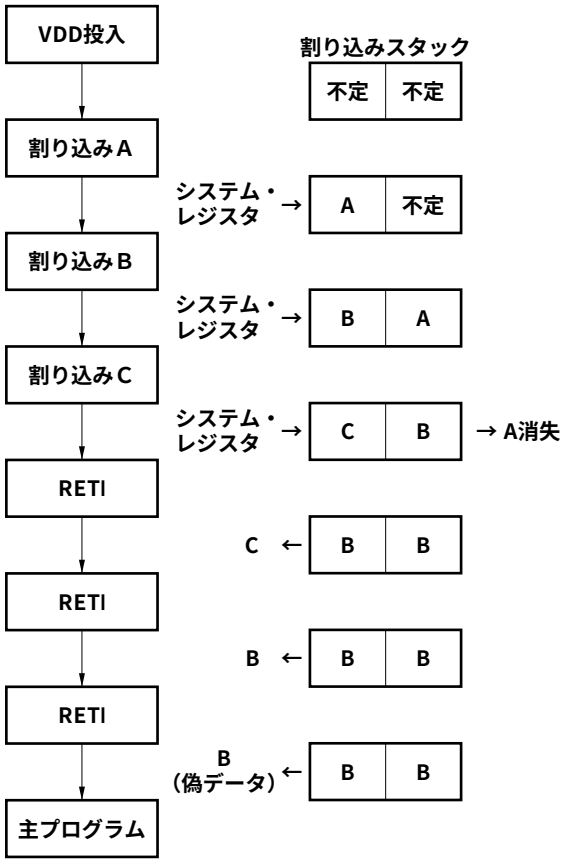
しかし，最大スタック・レベルを越える割り込みを受け付けると，図12－５の（ｂ）のように最初のデータは掃き捨てられてしまうので，その場合はプログラムで退避しておかなければなりません。

図12－5　割り込みスタックの動作例（最大スタック・レベル＝2の場合）

（a）割り込みレベルが2レベル以内のとき



（b）割り込みが2レベルを越えるとき



12.3　割り込み受け付け動作

12.3.1　割り込み受け付け動作と優先順位

割り込みを受け付けるまでの動作を次に示します。

（1）各周辺ハードウエアは，割り込み条件が満たされる（たとえば，外部割り込み端子に有効な信号が入力される）と，各割り込み制御ブロックに割り込み要求信号を出力します。

（2）各割り込み制御ブロックは，各周辺ハードウエアからの割り込み要求信号を受け付けると，対応するインタラプト・リクエスト・フラグ（IRQ×××）を“１”にセットします。

（3）各インタラプト・リクエスト・フラグが“１”にセットされたときに，各インタラプト・リクエスト・フラグに対応するインタラプト・パーミッション・フラグ（IP×××）が“１”にセットされていると，各割り込み制御ブロックから“１”が出力されます。

（4）各割り込み制御ブロックから出力された信号は，割り込みイネーブル・フリップフロップの出力とANDされ，割り込み受け付け信号を出力します。

この割り込みイネーブル・フリップフロップは，“EI”命令により，“１”にセットされ，“DI”命令により“０”にリセットされます。

割り込みイネーブル・フリップフロップが“１”にセットされているときに，各割り込み制御ブロックから“１”が出力されると，割り込みが受け付けられます。

図12－１に示したように，割り込みが受け付けられると割り込み受け付け信号は，各割り込み制御ブロックへ入力されます。

各割り込み制御ブロックへ入力された信号により，インタラプト・リクエスト・フラグが“０”にリセットされ，かつ各割り込みに対するベクタ・アドレスが出力されます。

このとき，割り込み制御ブロックから“１”が出力されていると，割り込み受け付け信号は次段に伝達されないため，同時に複数の割り込み要求が発行されたときは，回路で決められている優先順位で割り込みを受け付けます。この優先順位をハードウエア優先順位と呼びます。

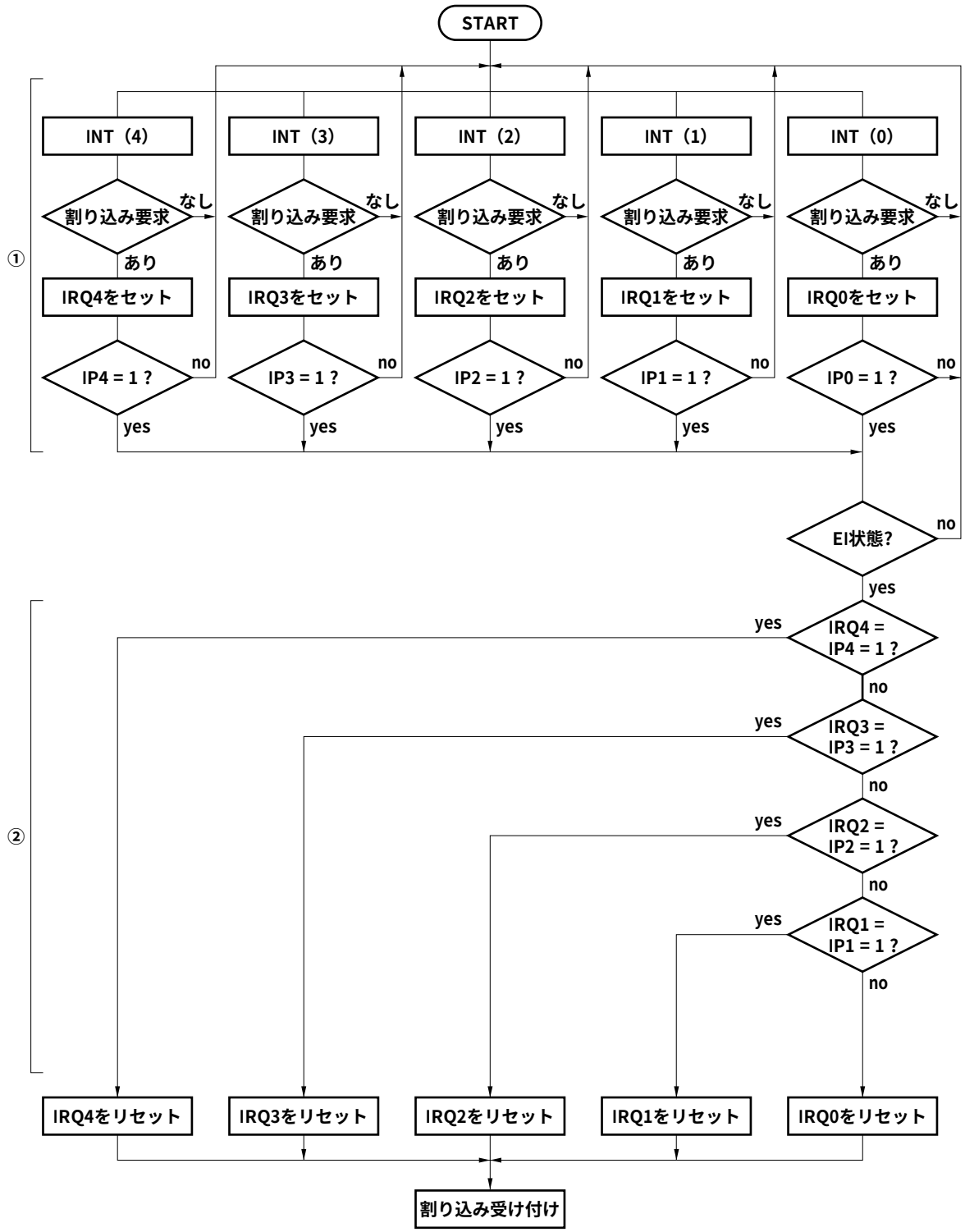
図12－６のフロー・チャートで，①の処理は常に並列に処理されるので，まったく同時に複数の割り込み要求が発生したときは，各インタラプト・リクエスト・フラグ（IRQ×××）は同時にセットされます。

しかし，②の処理は各インタラプト・パーミッション・フラグ（IP×××）をプログラムでセット，リセットすることによって任意の順序で割り込み処理ができます。

このインタラプト・パーミッション・フラグのある割り込みを“マスカブル割り込み”と呼びます。マスカブル割り込みは，ハードウエア優先順位の高い割り込みでも禁止できるので，その優先順位を“ソフトウエア優先順位”とも呼びます。

インタラプト・パーミッション・フラグについては，**12.2.4　インタラプト・パーミッション・フラグ（IP×××）の構成と機能**を参照してください。

図12－6　割り込み受け付け動作のフロー・チャート



12.3.2　割り込み受け付け時のタイミング・チャート

図12－7に割り込み受け付け時のタイミング・チャートを示します。

図12－7の（1）は1種類の割り込みによるタイミング・チャートです。

（1）の（a）はインタラプト・リクエスト・フラグが最後に“1”にセットされた場合であり，（1）の（b）はインタラプト・パーミッション・フラグが最後に“1”にセットされた場合のタイミング・チャートです。

どちらの場合も割り込みの受け付けは，インタラプト・リクエスト・フラグ，割り込みイネーブル・フリップフロップおよびインタラプト・パーミッション・フラグのすべてが“1”にセットされた時点で行われます。

最後に“1”にセットされたフラグまたはフリップフロップが，“MOVT DBF, @AR”命令の第1命令サイクルまたは，スキップ条件を満たした命令の場合は，それぞれ“MOVT DBF, @AR”命令の第2命令サイクルおよびスキップした命令（NOPになる）を実行後に，割り込みが受け付けられます。

割り込みイネーブル・フリップフロップは，“EI”命令が実行された次の命令サイクルでセットされます。

図12－7の（2）は複数の割り込みを使用するときのタイミング・チャートを示しています。

複数の割り込みを使用するときは，インタラプト・パーミッション・フラグがすべてセットされていれば，ハードウエアで優先されている割り込みから順に受け付けられますが，プログラムでインタラプト・パーミッション・フラグを操作することにより，ハードウエアの優先度を変えることが可能になります。

なお，図12－7に示した“割り込みサイクル”とは，割り込みが受け付けられてからインタラプト・リクエスト・フラグのリセット，ベクタ・アドレスの指定，プログラム・カウンタの退避などを行うための特別なサイクルであり，1命令実行時間分を必要とします。

インタラプト・リクエスト・フラグは，“EI”命令やインタラプト・パーミッション・フラグとは関係なく，周辺ハードウエアの割り込み要求により“1”にセットされるため，プログラムでインタラプト・リクエスト・フラグ（IRQ×××）を調べると割り込み要求の有無がわかります。

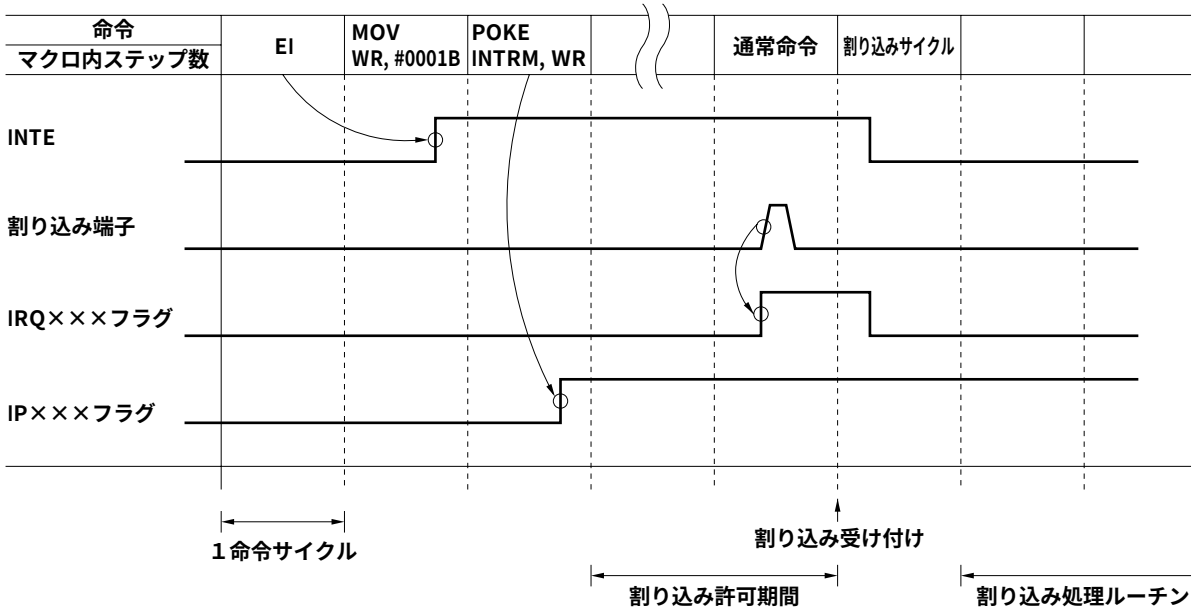
詳しくは**12.4　割り込み受け付け後の動作**を参照してください。

図12－7　割り込み受け付けタイミング・チャート（1/3）

（1）１種類の割り込み（例：外部割り込み端子の立ち上がり）を使用時

（a）インタラプト・パーミッション・フラグ（IP×××）による割り込みマスク時間がない場合

① 割り込み受け付け時が“ MOVT ”命令およびスキップ条件を満たした命令でない通常命令の場合



② 割り込み受け付け時が“ MOVT ”命令か，または“ スキップ条件を満たした命令 ”の場合

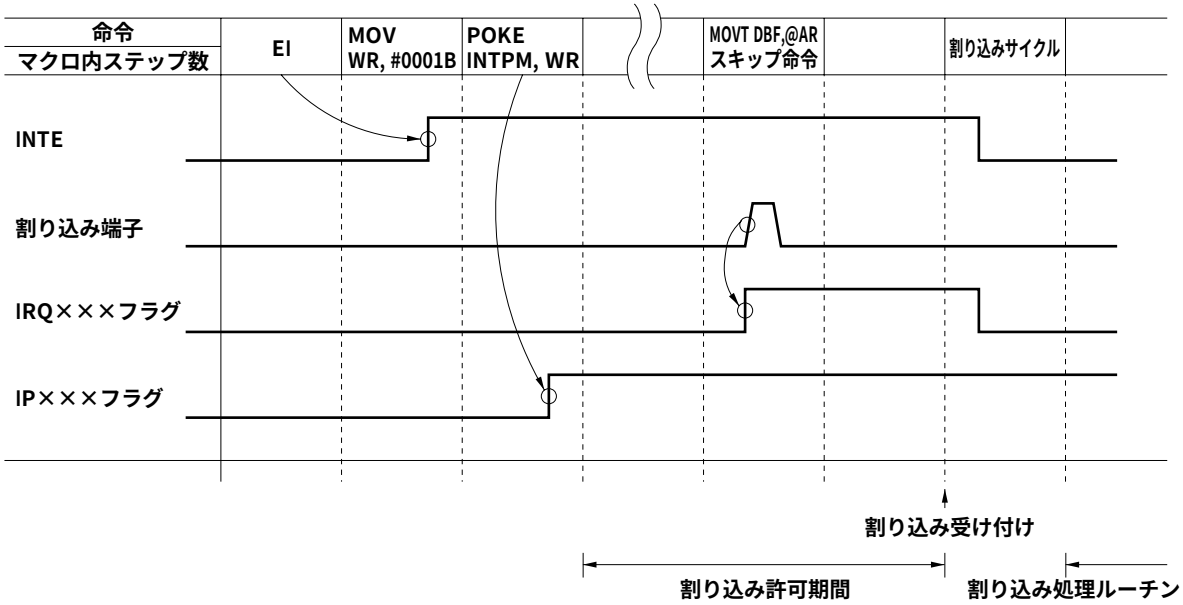


図12－7　割り込み受け付けタイミング・チャート（2/3）

（b）インタラプト・パーミッション・フラグによる割り込み保留期間がある場合

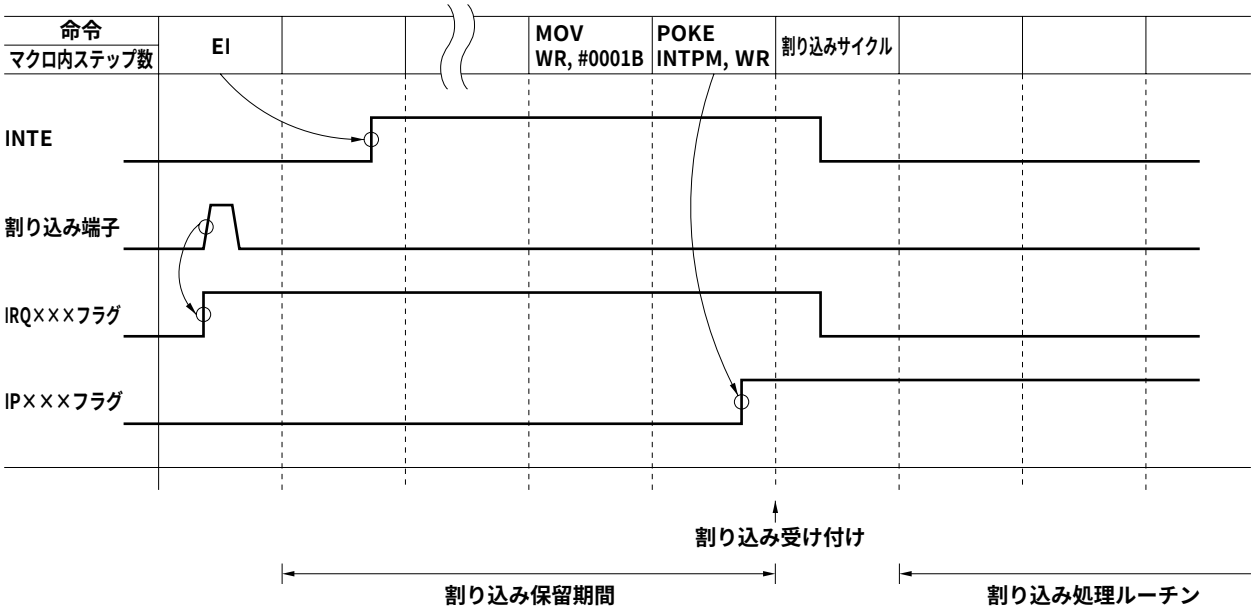
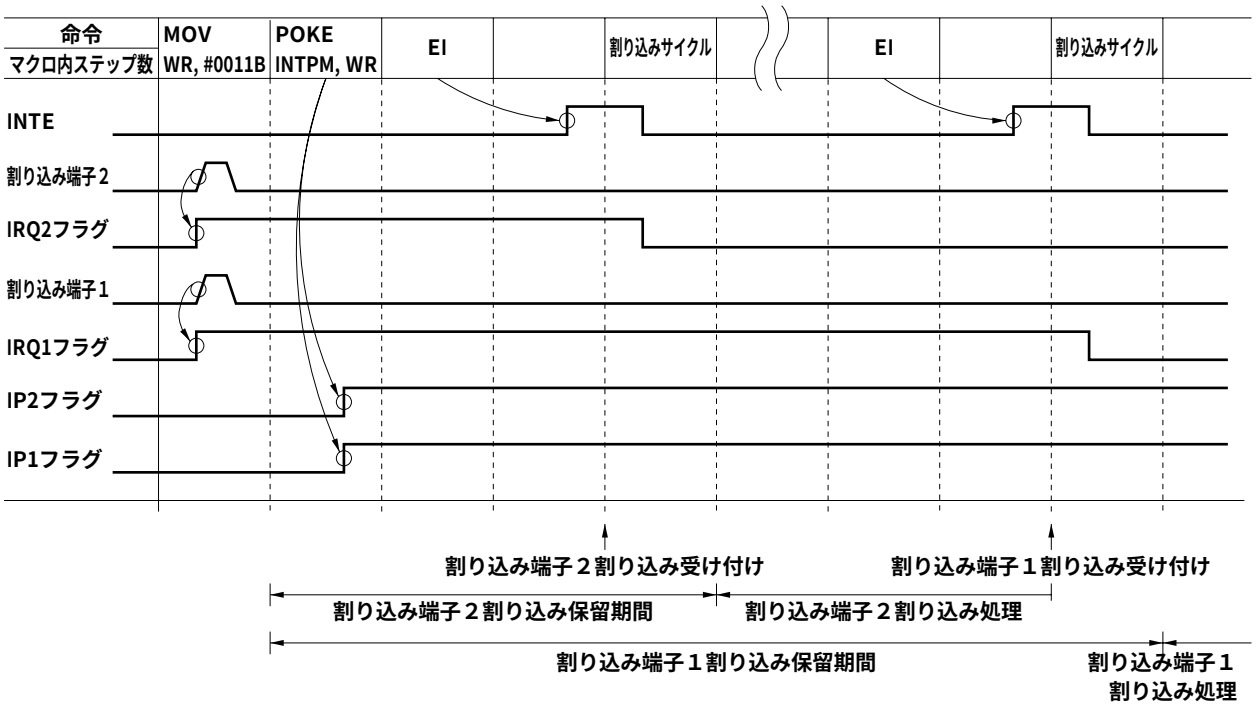


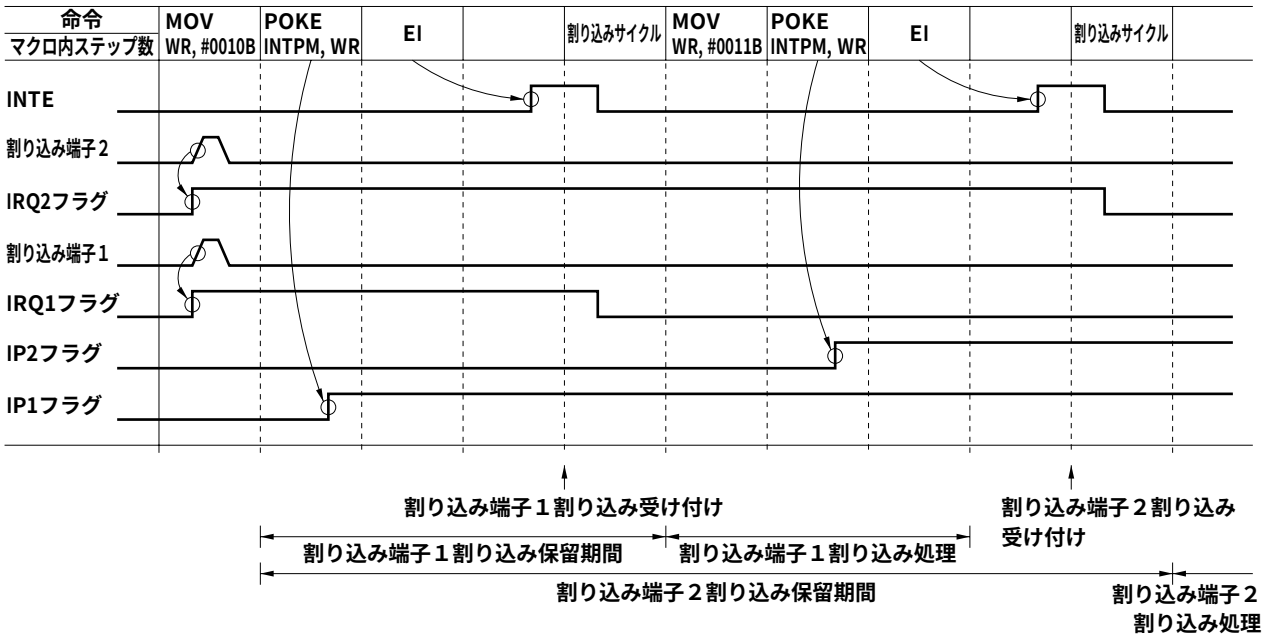
図12－7　割り込み受け付けタイミング・チャート（3/3）

（2）複数の割り込み（例：割り込み端子1，2（2がハードウエア優先）の2種類）を使用時

（a）ハードウエア優先



（b）ソフトウエア優先



12.4　割り込み受け付け後の動作

割り込みが受け付けられると，次に示す処理が順次自動的に実行されます。

（1）割り込みイネーブル・フリップフロップおよび受け付けられた割り込み要求に対応するインタラプト・リクエスト・フラグを“０”にリセットする。つまり割り込み禁止状態にする。

（2）スタック・ポインタの内容を－１する。

（3）プログラム・カウンタの内容をスタック・ポインタで指定されるアドレス・スタック・レジスタへ退避する。

　このときの退避される内容は，割り込みが受け付けられた時点の次のプログラム・メモリ・アドレスになる。

　たとえば分岐命令であれば分岐先のアドレスとなり，サブルーチン・コール命令であればコールしたアドレスになる。スキップ命令でスキップ条件が満たされたときは，次の命令を“NOP”命令として実行してから割り込みが受け付けられるため，退避される内容はスキップしたアドレスになる。

（4）バンク・レジスタ（BANK）の最下位ビットを割り込みスタックへ退避する。

（5）受け付けられた割り込みに対応するベクタ・アドレス・ジェネレータの内容をプログラム・カウンタに転送する。つまり割り込み処理ルーチンへ分岐する。

上記（1）-（5）の処理は通常の命令の実行を伴わない特別な１命令サイクルの間に実行されます。

この命令サイクルを“割り込みサイクル”と呼びます。

すなわち割り込みが受け付けられてから，対応するベクタ・アドレスへ分岐するまでに１命令サイクルの時間が必要になります。

12.5　割り込み処理ルーチン

割り込みは，割り込みが許可されているプログラム領域であれば，割り込み要求が発行された時点でそのとき実行しているプログラムに関係なく受け付けます。

したがって，割り込み処理を実行したのちにもとのプログラムに復帰させるときは，あたかも割り込み処理が実行されていないような状態に戻しておく必要があります。

たとえば割り込み処理中に算術演算等が実行されると，キャリー・フラグ（CY）などの内容は割り込みを受け付ける前と変化してしまう可能性があり，復帰後のプログラムは判断を誤ることになります。

このため，少なくとも割り込み処理ルーチン内で操作する可能性があるシステム・レジスタやコントロール・レジスタなどについては，割り込み処理ルーチンの中で退避および復帰操作を行う必要があります。

なお，割り込み処理中に他の割り込みを許可する場合（多重割り込み）の処理については**12.6　多重割り込み**の項も参照してください。

12.5.1　退避処理

システム・レジスタのうちで，ハードウエアにより自動的に退避されるものはバンク・レジスタ（BANK）とインデクス・イネーブル・フラグのみですから，必要であればプログラムで他のシステム・レジスタなどを退避します。

図12－８のように，システム・レジスタなどの退避や復帰処理には“ POKE ”命令と“ PEEK ”命令を使うと便利です。

“ PEEK ”および“ POKE ”命令以外にも転送命令（LD r, mやST m, rなど）を使う方法もありますが，割り込みを受け付けた時点でジェネラル・レジスタのロウ・アドレスが一定でない場合などは，転送命令による退避は退避するためのデータ・メモリ・アドレスの指定が困難です。

なぜなら，ジェネラル・レジスタ自身を退避するために転送命令を使うと，ジェネラル・レジスタのアドレスが一定でなければ，退避するアドレスも一定でなくなってしまうので，割り込み許可ルーチン中は少なくともジェネラル・レジスタを固定しなければならなくなるからです。

ところが“ PEEK ”命令や“ POKE ”命令で制御するレジスタ・ファイルはジェネラル・レジスタの内容とは無関係にアドレス指定でき，かつレジスタ・ファイルの40H-7FH番地は，その時選択しているバンクのデータ・メモリと重なっているため，バンクを指定するだけで各システム・レジスタなどを退避することができます。

図12－８ではウインドウ・レジスタとレジスタ・ポインタ（RPH, RPL）を“ PEEK ”および“ POKE ”命令で退避したのちに，ジェネラル・レジスタをBANK1のロウ・アドレス07Hに再指定し，その次に“ ST ”命令で他のシステム・レジスタなどを退避しています。

図12－９は，図12－８のプログラム例の退避動作のようすを図で表したものです。

12.5.2　復帰処理

復帰処理は，図12－８のように退避処理と反対の動作を行えばよいことになります。

割り込みを受け付けたときは当然割り込み許可状態（EI状態）であったはずですから，図12－８のように“ RETI ”命令を実行する前に“ EI ”命令を実行する必要があります。

“ EI ”命令は次の“ RETI ”命令が実行されたあとに割り込みイネーブル・フリップフロップをセット（１）します。したがって，割り込み受け付け前のプログラムに復帰したのちに割り込み許可状態になります。

最後に，専用命令である“ RETI ”命令を実行すると次の処理を順次自動的に実行し，割り込みを受け付けたときの処理に戻ります。

（１）スタック・ポインタが指定するアドレス・スタック・レジスタの内容をプログラム・カウンタに復帰します。

（２）割り込みスタックの内容をバンク・レジスタ（BANK：アドレス79H）の下位３ビットおよびプログラム・ステータス・ワード（IXE, BCD, CMP, CY, Zフラグ）に復帰します。

（３）スタック・ポインタの内容を＋1します。

上記（１）-（３）の処理は“ RETI ”命令を実行する１命令サイクル中に処理します。

“ RETI ”命令とサブルーチン・リターン命令“ RET ”および“ RETSK ”との違いは，上記（２）のバンク・レジスタおよびプログラム・ステータス・ワードの復帰動作の違いのみです。

図12－8　割り込み処理ルーチン内の退避と復帰

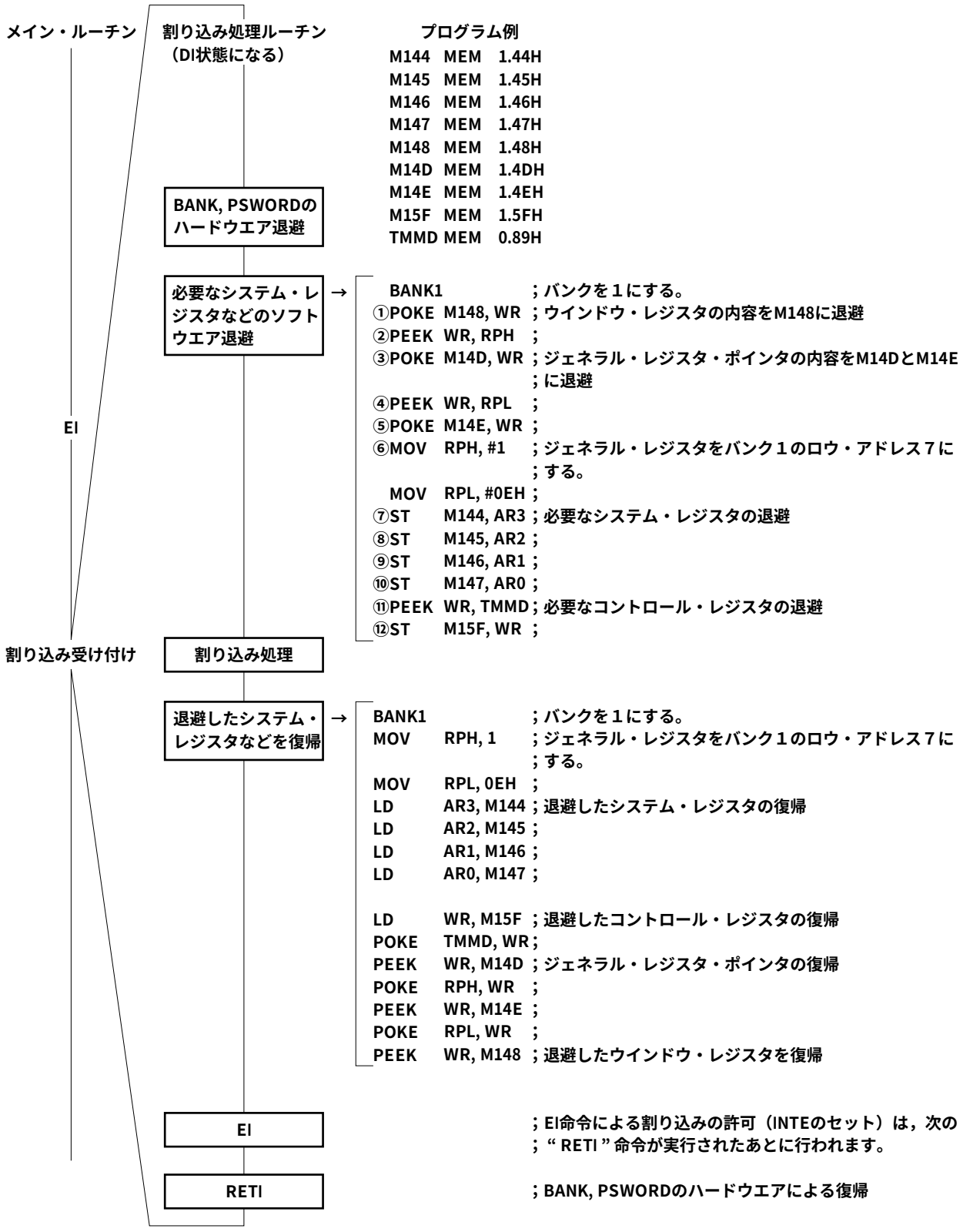
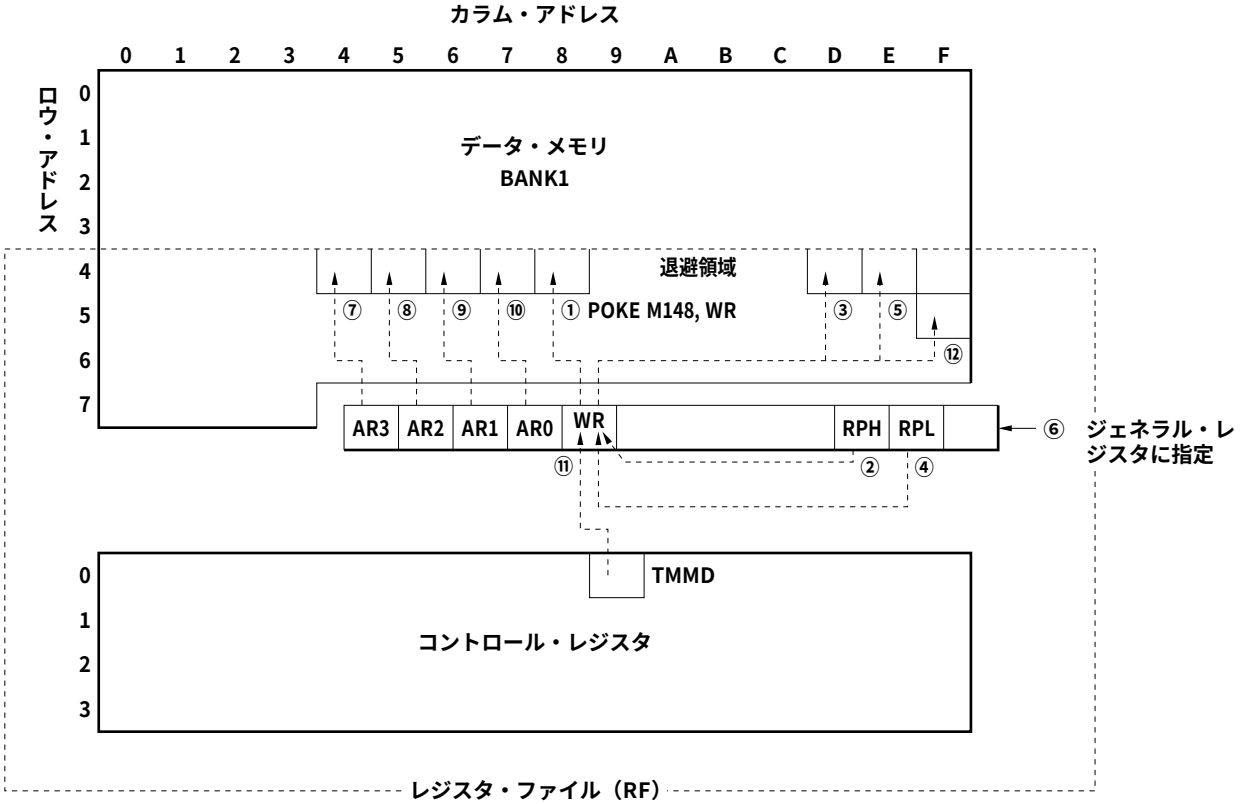


図12－９　“ PEEK ”および“ POKE ”命令を用いたときのシステム・レジスタやコントロール・レジスタの退避動作



備考　①-⑫は図12－８のプログラム例の中の番号です。

12.5.3　割り込み処理ルーチンでの注意

割り込み処理ルーチンでは次に示す点に注意してください。

（1）バンク・レジスタとプログラム・ステータス・ワードについて

バンク・レジスタおよびプログラム・ステータス・ワードは，割り込みスタックに退避されたあとすべて“ 0 ”にリセットされます。

（2）ソフトウエアにより退避した他のシステム・レジスタについて

ソフトウエアにより退避した他のシステム・レジスタのデータは，退避後でもリセットされません。

12.6　多重割り込み

多重割り込みとは図12－10のように，ある割り込み要因ＡやＢに対する割り込み処理中に，別の割り込み要因であるＣやＤの割り込みを処理する方法です。

このときの割り込みの深さを割り込みレベルと呼びます。

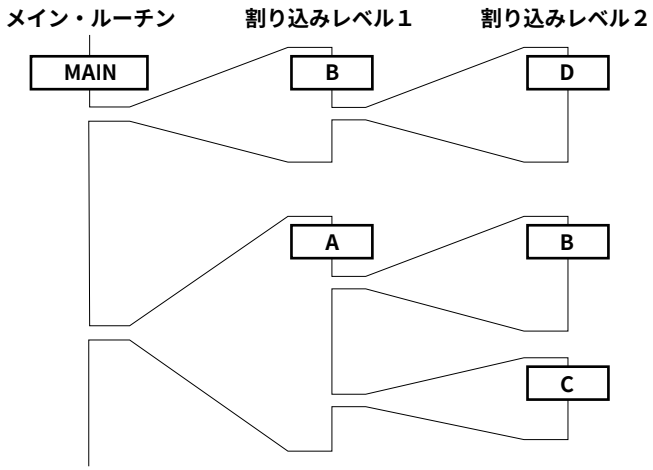
多重割り込みを使用するときには，次の点に注意が必要です。

（1）割り込み要因の優先順位

（2）割り込み用スタックによる割り込みレベルの制限

上記（1），（2）の詳細は**12.6.1**-**12.6.2**です。

図12－10　多重割り込み例



12.6.1　割り込み要因の優先順位

多重割り込みを使用する場合は割り込み要因の優先順位を設定しておく必要があります。

たとえば割り込み要因がＡ，Ｂ，ＣおよびＤであるときの優先順位は

A = B = C = D

でもかまいませんし，

A＜B＜C＜D

でもかまいません。

ただし，A = B = C = Dの場合には，メイン・ルーチンでは常にＡ，Ｂ，ＣおよびＤの割り込みが受け付けられますが，たとえばＣの割り込みが受け付けられれば他のＡ，ＢおよびＤの割り込みはすべて禁止されるので多重割り込みの意味はなくなってしまいます。

A＜B＜C＜Dの優先順位を持つときはＡまたはＢの割り込み処理実行中であってもＣの割り込みは優先的に処理しなければならず，Ｄの割り込みはさらにＣの割り込み処理実行中であっても優先しなければなりません。

優先順位は“12.3　割り込み受け付け動作”で説明したように，ハードウエアによる優先順位でもかまいませんし，インタラプト・パーミッション・フラグを用いたソフトウエアによる優先順位でもかまいません。

★　次のような場合に，多重割り込みにおいて優先順位を設定しておく必要があります。

・割り込み要因Ａ：割り込み要求10 msごとに発行
　　割り込み処理時間4 ms
・割り込み要因Ｂ：割り込み要求2 msごとに発行
　　割り込み処理時間1 ms

このときＡおよびＢに優先順位がなかったと仮定すると，たまたまＢの割り込み処理中にＡの割り込み要求によりＡの割り込み処理が実行されれば，Ｂの割り込み処理は数回分行われなくなってしまいます。

一般に割り込みは緊急性が高い処理に使うことが多いため，前述したような場合等はA＜Bの優先順位をつけてＢの割り込み処理中はＡの割り込み処理を禁止しておき，またＡの割り込み処理中であってもＢの割り込みは受け付けるようなプログラムが必要です。

緊急性のない目的で多重割り込みを使うときは，必ずしも優先順位をつけておく必要はありませんが，**12.6.2　割り込み用スタックによる割り込みレベルの制限**にあるような多重割り込みレベルの制限を越える割り込み要因数が存在する場合は，割り込みレベルを越えないように優先順位を決定しておく必要があります。

12.6.2　割り込み用スタックによる割り込みレベルの制限

割り込み用スタックには，システム・レジスタのバンク・レジスタとプログラム・ステータス・ワードの内容を自動的に退避します。

通常，割り込みスタックは図12－11の（ａ）のように動作します。

バンク・レジスタとプログラム・ステータス・ワードは割り込み用スタックに退避すると同時にすべてリセットされます。

割り込み用スタックの最大レベルを越える多重割り込みを行うとバンク・レジスタおよびプログラム・ステータス・ワードは図12－11の（ｂ）のように正常に復帰できなくなります。

ただし，図12－12のように，割り込みが許可されているメイン・ルーチンにおいて，必ずバンク・レジスタおよびプログラム・ステータス・ワードが固定されており，かつ割り込みの優先度が明確になっているような多重割り込みでは，サブルーチン・リターン命令（RET）を使うことにより最大スタック・レベル以上の多重割り込みができます。

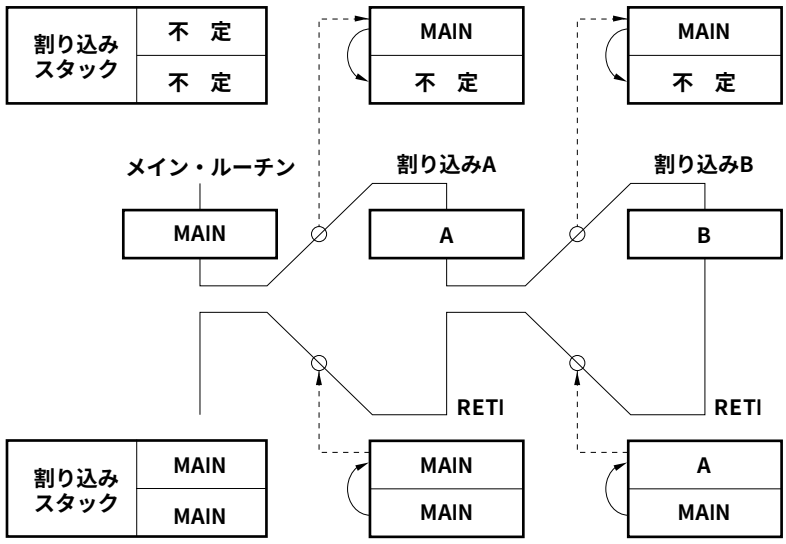
しかしこのとき，図12－12と図12－13とのように，デバイスとエミュレータとでは動作が違いますから注意が必要です。

★　すなわち，割り込みスタックのデバイス動作は“掃き捨て型”であり，エミュレータ（IE-17K, IE-17K-ET）の動作は“回転型”になっています。

なお，“RETI”命令と“RET”命令とでは割り込み用スタックの復帰処理が異なりますから，各品種の最大スタック・レベルを越える多重割り込みをするときの最終復帰命令には“RET”命令を使ってください。

図12－11　多重割り込み時の割り込みスタック

（a）通常の多重割り込み



（b）最大スタック・レベルを越える多重割り込み（割り込みスタックが２レベルのとき）

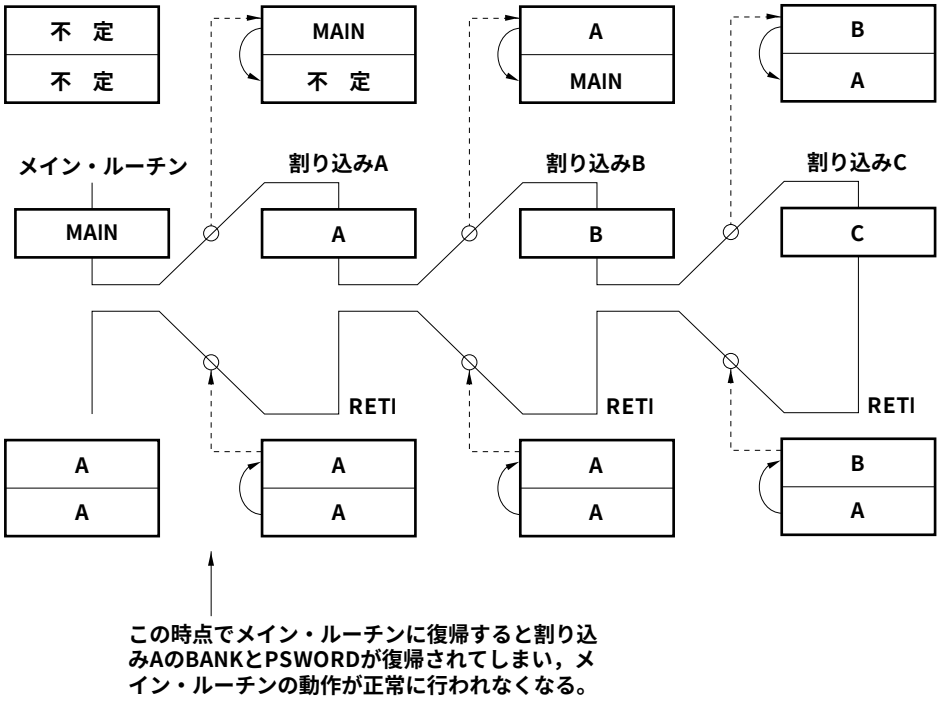


図12－12　最大スタック・レベル以上の多重割り込み使用例（割り込みスタックが２レベルのとき）

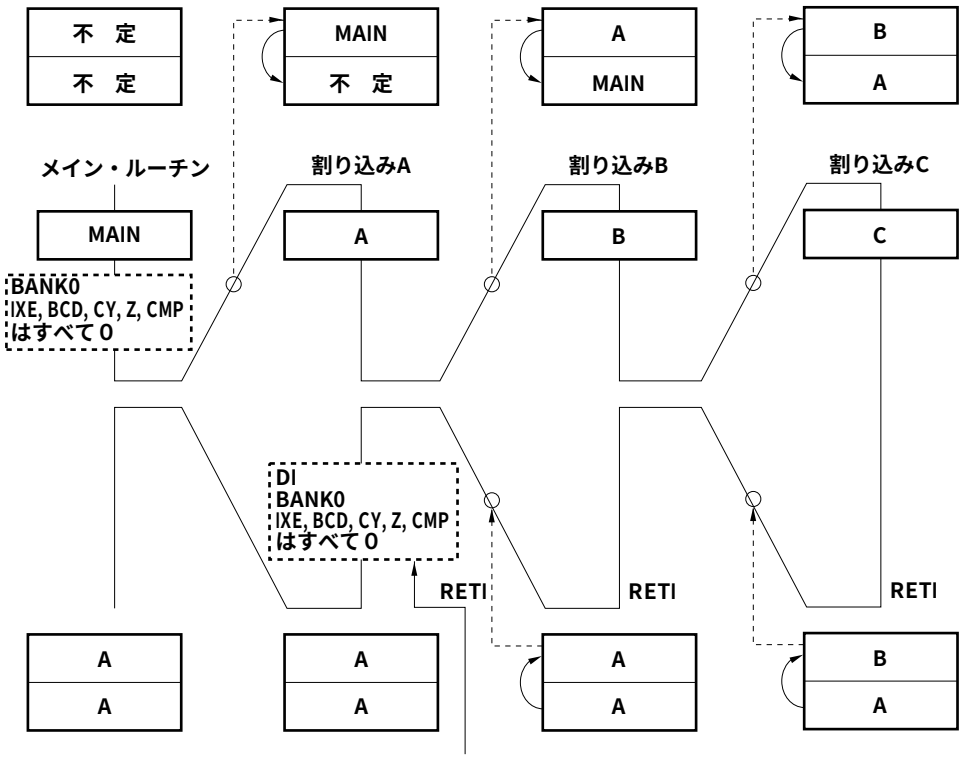
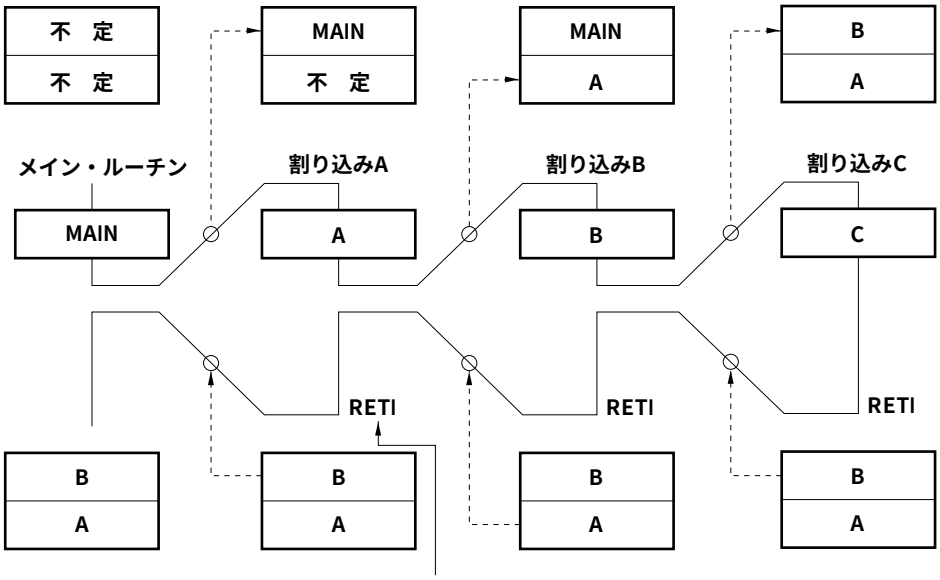


図12－12の例では，割り込みＡの優先順位は割り込みＢ，Ｃより必ず低くし，かつ割り込みＡを許可するメイン・ルーチンではバンク・レジスタとプログラム・ステータス・ワードを常に一定にしておけば，割り込みＡの処理終了時点でメイン・ルーチンのバンク・レジスタおよびプログラム・ステータス・ワードを指定したあとに“ RET ”命令を使うことにより３レベルの多重の割り込みが可能になります。

また，割り込みＡでのバンク・レジスタとプログラム・ステータス・ワードがメイン・ルーチンとまったく等しいときは“ RETI ”命令を使うことができますが，図12－13のように17Kシリーズのエミュレータ（IE-17K, IE-17K-ET）は実際の製品とは動作が異なるため“ RETI ”命令のディバグはできません。

★

図12－13　17Kシリーズのエミュレータ（IE-17K, IE-17K-ET）使用時の最大スタック・レベル以上の割り込みスタック動作



★　エミュレータ（IE-17K, IE-17K-ET）で“ RETI ”命令を使用すると，図12－13のように割り込みBのバンク・レジスタ（BANK）とインデクス・イネーブル・フラグ（IXE）の内容を復帰します。

第13章　スタンバイ機能

スタンバイ機能は，プログラム処理停止中のデバイスの消費電流を減少させるための機能です。

ホールト機能は，たとえば時計のみを動作させるときなどの消費電流を減らします。

また，クロック・ストップ機能はデータ・メモリのみ保持するときの消費電流を減らします。

しかし，これらの消費電流は，周辺ハードウエアや端子の状態によっては規格どおりに減らないことがありますので，あらかじめ各品種のデータ・シートの指示によってソフトウエア，ハードウエア上の処置対策をしてください。

13.1　スタンバイ・ブロックの構成

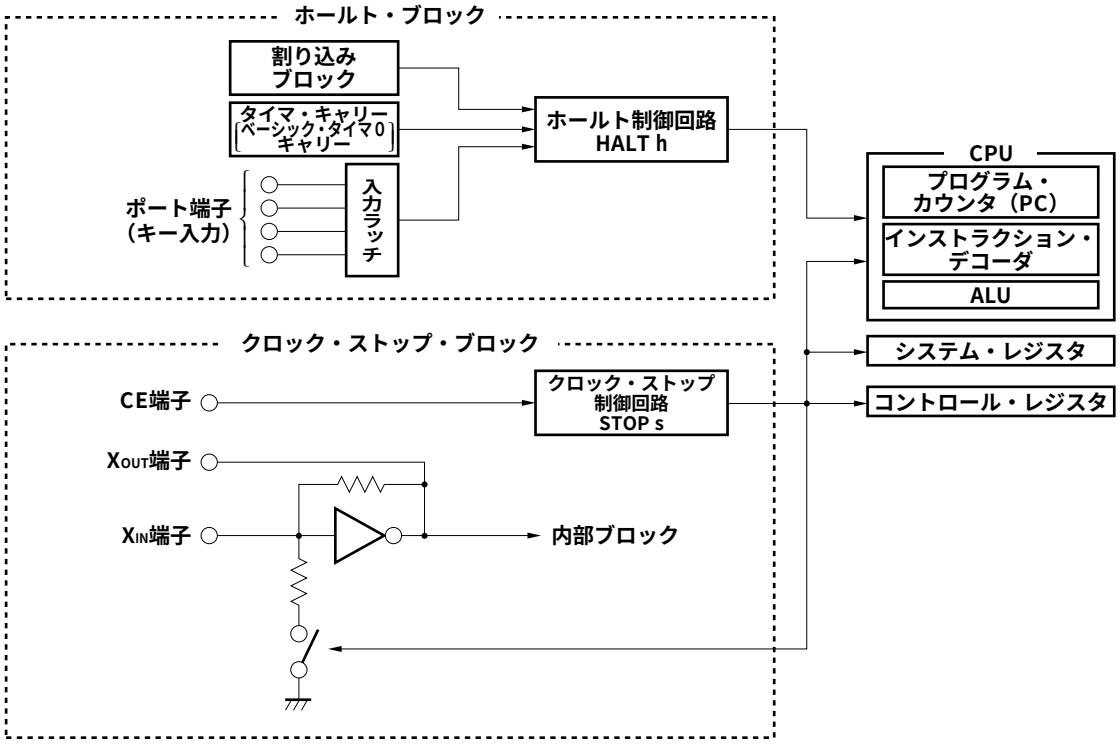
図13－１にスタンバイ・ブロックの構成を示します。

図13－１に示すようにスタンバイ・ブロックはホールト制御ブロックとクロック・ストップ制御ブロックに分けられます。

ホールト制御ブロックはホールト制御回路のほかに割り込み制御ブロック，タイマ・キャリー（ベーシック・タイマ０キャリー）およびキー入力端子であるポート端子から構成されており，CPU（プログラム・カウンタ，インストラクション・デコーダおよびALUブロック）の動作を制御します。

クロック・ストップ制御ブロックはクロック・ストップ制御回路により動作クロック発振回路，CPU, システム・レジスタおよびコントロール・レジスタを制御します。

図13－１　スタンバイ・ブロックの構成例



13.2　スタンバイ機能

スタンバイ機能は，デバイスの動作を一部またはすべて停止させることによりデバイスの消費電流を減少させます。

スタンバイ機能は，ホールト機能とクロック・ストップ機能に分けられます。

ホールト機能は，専用命令である“HALT h”命令を使用してCPUの動作を停止させることによりデバイスの消費電流を減少させます。

クロック・ストップ機能は，専用命令である“STOP s”命令を使用して水晶振動子の発振回路を停止させることによりデバイスの消費電流を減少させます。

ホールト機能とクロック・ストップ機能のほかに，CE端子によるデバイスの動作モードの設定があります。

CE端子はPLL周波数シンセサイザの動作制御およびデバイスのリセットに使用しますが，PLL周波数シンセサイザの動作制御を行う点ではスタンバイ機能の１つといえます。

13.3にCE端子によるデバイス動作モードの設定を説明します。

13.4および**13.5**にホールト機能およびクロック・ストップ機能を説明します。

13.3　CE端子によるデバイス動作モード

CE端子は，外部から入力される信号の入力レベルおよび立ち上がりエッジにより次の（1）-（3）に示す機能を制御します。

（1）内蔵する周辺ハードウエアの動作制御
（2）クロック・ストップ命令の有効，無効の制御
（3）デバイスのリセット

13.3.1　内蔵する周辺ハードウエアの動作制御

周辺ハードウエアは，CE端子がハイ・レベルならば動作し，ロウ・レベルならば停止します。

13.3.2　クロック・ストップ命令の有効，無効の制御

クロック・ストップ命令である“STOP s”命令は，CE端子がロウ・レベルのときのみ有効になります。

CE端子がハイ・レベルのときに実行された“STOP s”命令は，ノー・オペレーション命令（NOP）として実行されます。

13.3.3　デバイスのリセット

CE端子をロウ・レベルからハイ・レベルに立ち上げることによりデバイスにリセット（CEリセット）をかけることができます。

リセットにはCEリセットのほかに電源電圧V_{DD}を投入したときのパワーオン・リセットがあります。

詳しくは**第14章　リセット機能**を参照してください。

13.3.4　CE端子への信号入力

CE端子は，ノイズによる誤動作の防止のために，品種ごとに定める一定時間以上連続しないロウ・レベルまたはハイ・レベルは受け付けません。

また，CE端子の入力レベルは，コントロール・レジスタ内のCEフラグに反映されますから，CEフラグで入力レベルを検知できます。

13.4　ホールト機能

ホールト機能は“HALT h”命令を実行することによりCPUの動作クロックを停止します。

すなわち“HALT h”命令が実行されると，プログラムは“HALT h”命令で停止し，以後ホールト状態が解除されるまで“HALT h”命令で停止したままになります。

したがって，ホールト状態中のデバイス消費電流はCPUの動作電流分だけ減少します。

ホールト状態からの解除はキー入力，タイマ・キャリー（ベーシック・タイマ0キャリー），および割り込みにより行います。

キー入力，タイマ・キャリー，および割り込みの解除条件は“HALT h”命令のオペランドである“ h ”により指定します。“ h ”で指定する解除条件は品種によって異なるので，各品種のデータ・シートで確認してください。

“HALT h”命令はCE端子の入力レベルに関係なく有効です。

13.4.1-13.4.6にホールト状態，ホールト解除条件および各ホールト解除条件について説明します。

13.4.1　ホールト状態

ホールト状態中はCPUの動作がすべて停止します。

すなわちプログラムの実行が“HALT h”命令で停止している状態になります。

ただし，周辺ハードウエアは“HALT h”命令以前に設定されている動作をそのまま継続します。

詳しくは，各品種のデータ・シートを参照してください。

13.4.2　ホールト解除条件

図13－２にホールト解除条件を示します。

図13－２に示すようにホールト解除条件は，“ HALT h ”命令のオペランド“ h ”で指定する４ビット・データにより設定します。

ホールト状態は，オペランド“ h ”に“ 1 ”で指定された条件が満たされると解除されます。

ホールト状態が解除されると“ HALT h ”命令の次の命令から実行を始めます。

★　このとき一度に複数の解除条件が設定されていると，設定されている条件のどれか１つでも満たされれば，ホールト状態は解除されます。

また，デバイスにリセット（パワーオン・リセットまたはCEリセット）がかかったときは，ホールト状態は解除され各リセット動作を行います。

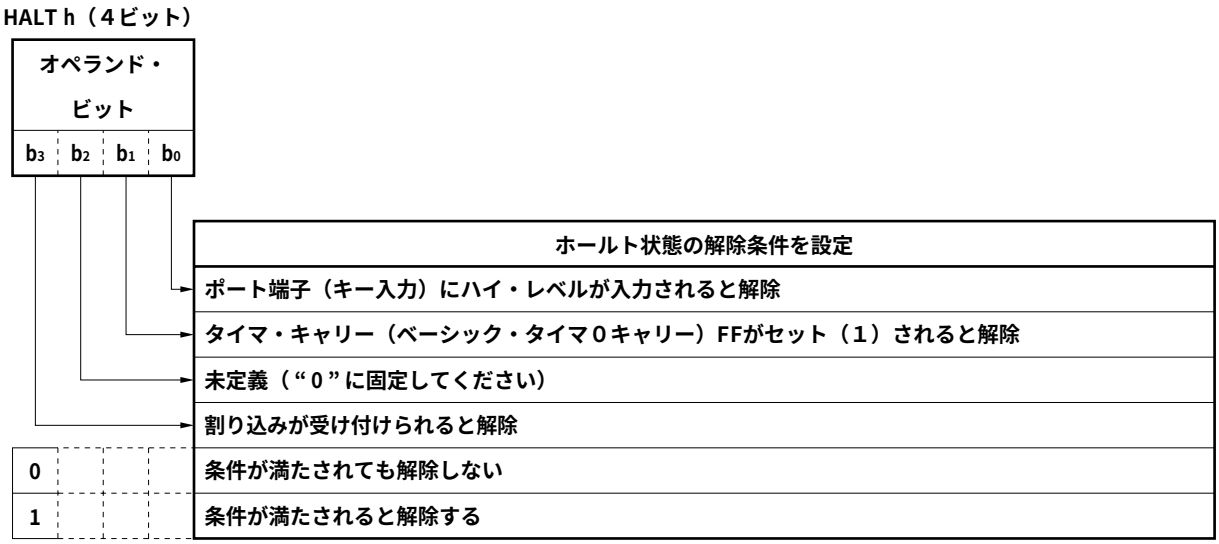
ホールト解除条件“ h ”に0000Bが設定されると，どの解除条件も設定されません。

このときはデバイスにリセット（パワーオン・リセットまたはCEリセット）がかかるとホールト状態が解除されます。

13.4.3-13.4.5にキー入力，タイマ・キャリー（ベーシック・タイマ０キャリー）および割り込みの各ホールト解除条件について説明します。

また，13.4.6に複数の解除条件を設定したときの使用例を示します。

図13－２　ホールト解除条件



13.4.3　キー入力によるホールト解除

キー入力によるホールト解除を“ HALT 0001B ”で指定します。

キー入力によるホールト解除を指定すると，ポート0D端子のうちどれか１本にハイ・レベル入力があれば，ホールトを解除します。

★　**備考**　ポート0Dの兼用端子は各品種の端子ごとに異なります。兼用機能を使用する場合，およびキー・ソース信号として汎用出力ポートを使用する場合は，各品種のデータ・シートに記載されている注意事項を参照してください。

13.4.4　タイマ・キャリー（ベーシック・タイマ０キャリー）によるホールト解除

タイマ・キャリー（ベーシック・タイマ０キャリー）によるホールト解除を“ HALT 0010B ”で指定します。

タイマ・キャリー（ベーシック・タイマ０キャリー）によるホールト解除を指定すると，タイマ・キャリー（ベーシック・タイマ０キャリー）FFが“ １ ”にセットされると同時にホールトを解除します。

コントロール・レジスタ内の専用フラグに対応しており，フラグでセットした一定時間ごとに“ １ ”にセットされます。

したがって，一定時間ごとにホールトを解除することができます。

★　**備考**　タイマ・キャリー（ベーシック・タイマ０キャリー）FFのセット時間は各品種によって異なります。各品種のデータ・シートを参照してください。

13.4.5　割り込みによるホールト解除

割り込みによるホールト解除を“ HALT 1000B ”で指定します。

割り込みによるホールト解除を指定すると，割り込みが受け付けられると同時にホールトを解除します。

割り込み要因が複数ある場合，どの割り込み要因でホールトを解除するかは，あらかじめプログラムで指定しておく必要があります。

また，割り込みが受け付けられるためには，各割り込み要因からの割り込み要求の発行以外に，すべての許可（EI命令）および各割り込みごとの許可（インタラプト・パーミッション・フラグがセットされている）がすべて満たされる必要があります。

したがって，割り込み要求が発行されてもその割り込みが許可されていなければ，割り込みは受け付けられず，ホールト状態も解除されません。

割り込みの受け付けによりホールト状態が解除されると，プログラムの流れは各割り込みのベクタ・アドレスへ移ります。

割り込みの処理後にRETI命令を実行すると，プログラムの流れはHALT命令の次の命令に復帰します。

★　**備考**　割り込み要因は各品種によって異なります。各品種のデータ・シートを参照してください。

★　**注意　インタラプト・パーミッション・フラグ（IP×××）がセットされているインタラプト・リクエスト・フラグ（IRQ×××）のセットを解除条件とするHALT命令を実行するときは，HALT命令の直前にNOP命令を記述してください。**

NOP命令をHALT命令の直前に記述すると，IRQ×××操作命令とHALT命令との間に１命令分の時間が生成されます。たとえば，CLR1 IRQ×××命令の場合は，IRQ×××のクリアがHALT命令に正しく反映されます（例１）。NOP命令をHALT命令の直前に記述しない場合，CLR1 IRQ×××命令はHALT命令に反映されず，HALTモードには入りません（例２）。

例１．HALT命令を正しく実行するプログラム例

```
      ⋮
      ⋮              ; IRQ×××のセット
      ⋮
CLR1  IRQ×××
NOP                  ; NOP命令をHALT命令の直前に記述
                     ; （IRQ×××のクリアがHALT命令に対して正しく反映される）
HALT  1000B          ; HALT命令を正しく実行する（HALTモードに入る）
      ⋮
      ⋮
```

２．HALTモードに入らないプログラム例

```
      ⋮
      ⋮              ; IRQ×××のセット
      ⋮
CLR1  IRQ×××         ; IRQ×××のクリアはHALT命令に対して反映されない
                     ; （反映されるのはHALT命令の次の命令）
HALT  1000B          ; HALT命令は無視される（HALTモードに入らない）
      ⋮
      ⋮
```

13.4.6　複数の解除条件が同時に設定された場合

複数のホールト解除条件が設定された場合は，設定された解除条件のうち１つでも条件が満たされるとホールト状態が解除します。

このとき複数の解除条件が同時に満たされたときの解除条件判別方法を次に示します。

★　例１．

```
        HLTINT    DAT 1000B
        HLTTMR    DAT 0010B
        HLTKEY    DAT 0001B
        INTPIN    DAT 0004H          ; INT端子割り込みベクタ・アドレス・シンボ
                                     ; ル定義
START：
        BR    MAIN
ORG   INTPIN

        処　理　A                    ; INT端子割り込み処理

        EI
        RETI
TMRUP：                              ; タイマ・キャリー処理

        処　理　B

        RET
KEYDEC：                             ; キー入力処理

        処　理　C

        RET
MAIN：
        SET2  TMMD1, TMMD0           ; 組み込みマクロ
                                     ; タイマ・キャリーFFのセット時間を1msに設定
        SET1  IP                     ; 組み込みマクロ
                                     ; INT端子の割り込みを許可
        EI
LOOP：
        HALT  HLTINT OR HLTTMR OR HLTKEY
                                     ; ホールト解除条件を割り込み，タイマ・
                                     ; キャリー
                                     ; およびキー入力に設定
        SKF1  TMCY                   ; 組み込みマクロ
                                     ; TMCYフラグを検出
        CALL  TMRUP                  ; セット（１）されていればタイマ・キャ
                                     ; リー処理
        SKF4  POD3, POD2, POD1, POD0 ; キー入力の有無を検出
                                     ; キー入力ラッチを検出
        CALL  KEYDEC                 ; ラッチされていればキー入力処理
        BR    LOOP
```

例１では，ホールト解除条件にINT端子の割り込み，１msタイマ・キャリーおよびキー入力の３条件を設定しています。

どの条件でホールト状態が解除されたかを検出するために，割り込みであればベクタ・アドレス，タイマ・キャリーであればTMCYフラグ，キー入力であればP0Dフラグを検出します。

複数の解除条件を使用するときは次に示す２点に注意が必要です。

（１）ホールト状態が解除されたときは設定されている解除条件のすべてを検出するようにする。
（２）優先順位の高い解除条件から検出する。

（１）および（２）については，たとえば，例１に示した“MAIN：”以降のプログラムが次に示す例２のようになっている場合に注意が必要です。例２に示すように，タイマ・キャリーによるタイマの優先順位が高いプログラムにおいては，次のようなプログラムは作成しないでください。

★　**備考**　製品によってフラグ名が異なる場合があります。各品種のデータ・シートを参照してください。

例２．

```
MAIN：
      SET4  P1C3, P1C2, P1C1, P1C0   ；キー・ソース信号に汎用出力ポートを使用する。
      SET2  TMMD1, TMMD0
      SET1  IP
      EI
LOOP：
      HALT  HLTINT OR HLTTMR OR HLTKEY
      SKF4  P0D3, P0D2, P0D1, P0D0   ；キー入力の有無を検出
      BR    KEYDEC
      SKF1  TMCY
      CALL  TMRUP
      BR    LOOP
KEYDEC：                             ；キー入力処理

      処　理　C

      BR    LOOP
```

例２のプログラムにおいて今，仮にキー入力によるホールト解除が行われたときに，そのすぐあとにタイマ・キャリーFFがセット（１）されたとします。

すると，プログラムはキー入力処理を実行したのちに再度“HALT”命令を実行します。

このときタイマ・キャリーFFはセットされたままであるためホールト状態は即座に解除されます。

ところが，キー入力は通常100 ms程度ハイ・レベルが入力されるため，さらにキー入力処理へ分岐してしまいます。

このため，タイマ・キャリーFFの検出が正確に行われなくなってしまいます。

★　**備考**　製品によってフラグ名が異なる場合があります。各品種のデータ・シートを参照してください。

13.5　クロック・ストップ機能

クロック・ストップ機能は“STOP s”命令を実行することにより水晶振動子の発振回路を停止します（クロック・ストップ状態）。

したがってデバイスの消費電流は最低限まで減少します。

消費電流については各品種によって異なります。各品種のデータ・シートを参照してください。

“STOP s”命令のオペランド“ s ”には“0000B”を指定します。

“STOP s”命令はCE端子がロウ・レベルのときのみ有効となり，CE端子がハイ・レベルの間に実行されてもノー・オペレーション命令（NOP）として実行されます。

すなわち，CE端子がロウ・レベルのときに“STOP s”命令を実行する必要があります。

クロック・ストップ状態の解除は，CE端子をロウ・レベルからハイ・レベルに立ち上げる（CEリセット）ことにより行います。

13.5.1-**13.5.3**にクロック・ストップ状態，クロック・ストップ状態の解除およびクロック・ストップ命令使用時の注意を説明します。

13.5.1　クロック・ストップ状態

クロック・ストップ状態は水晶振動子の発振回路が停止するため，CPUおよび周辺ハードウエアなどのデバイス動作はすべて停止します。

CPUおよび各周辺ハードウエアの動作は各品種のデータ・シートを参照してください。

クロック・ストップ状態中はデバイスの電源電圧V_{DD}を2.2 Vまで下げても停電検出回路は動作しません。したがって低電圧でのデータ・メモリのバックアップが可能になります。停電検出回路については**第14章　リセット機能**を参照してください。

13.5.2　クロック・ストップ状態の解除

クロック・ストップ状態の解除はCE端子をロウ・レベルからハイ・レベルに立ち上げる（CEリセット）か，デバイスの電源電圧V_{DD}を一度2.2 V以下に下げてから4.5 Vまで立ち上げる（パワーオン・リセット）ことにより行います。

図13－３および図13－４にそれぞれCEリセット時とパワーオン・リセット時の解除動作を示します。

パワーオン・リセットによるクロック・ストップ状態の解除を行うと停電検出回路が動作します。

パワーオン・リセットについては**14.4　パワーオン・リセット**を参照してください。

図13－3　CEリセットによるクロック・ストップの解除

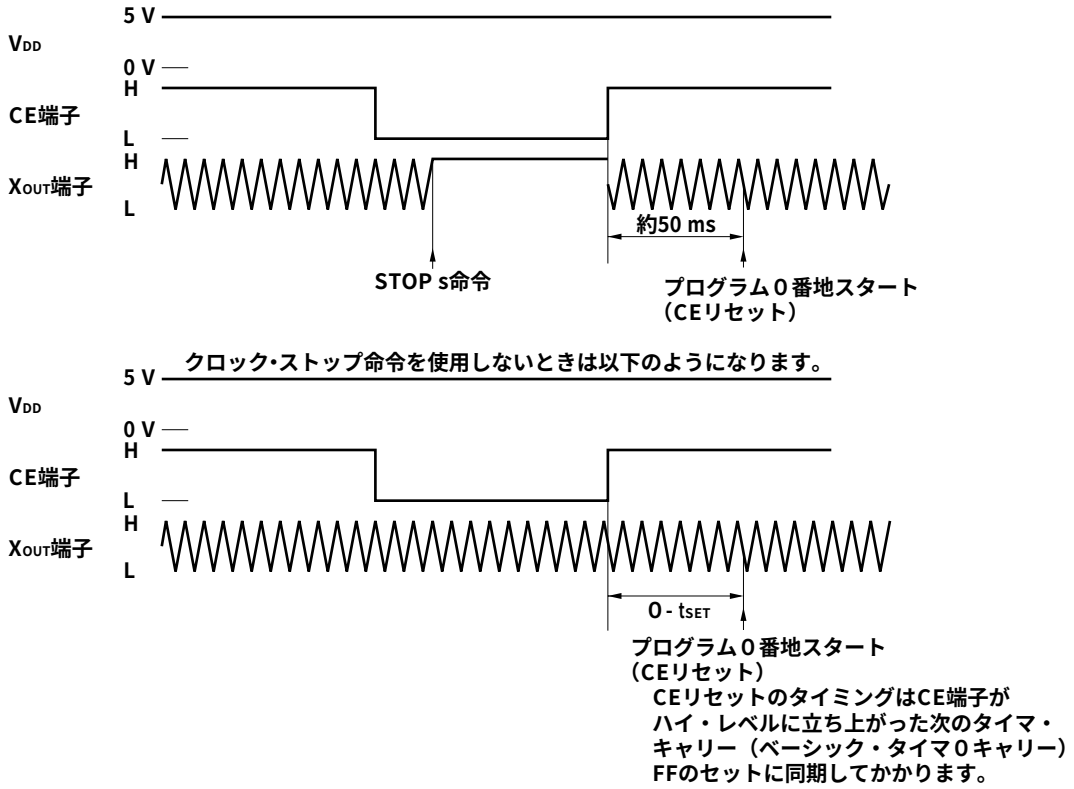
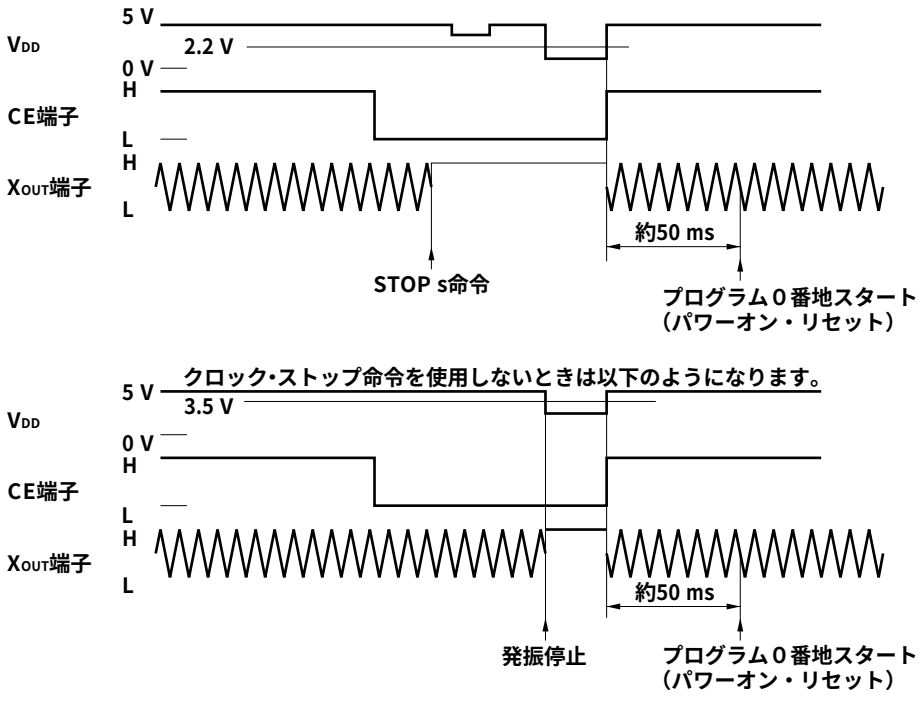


図13－4　パワーオン・リセット時のクロック・ストップ解除



13.5.3　CE端子がハイ・レベルの場合にクロック・ストップ命令によって起こる不具合と対策

クロック・ストップ命令（STOP s命令）はCE端子がロウ・レベルのときのみ有効です。

したがって，CE端子がたまたまハイ・レベルであったときの処理をプログラム上考慮しておく必要があります。

次のプログラム例では①でCE端子の状態を検出し，CE端子がロウ・レベルであれば処理Ａを実行後②のクロック・ストップ命令“ STOP XTAL ”を実行します。

ところが，図13－５のように②の“ STOP XTAL ”命令実行中にCE端子がハイ・レベルになってしまうと“ STOP XTAL ”命令はノー・オペレーション命令（NOP）として実行されてしまいます。

このとき，仮に③の分岐命令“ BR $-1 ”がなかったとすると，プログラムはメイン処理に移ってしまい，誤動作することが考えられます。

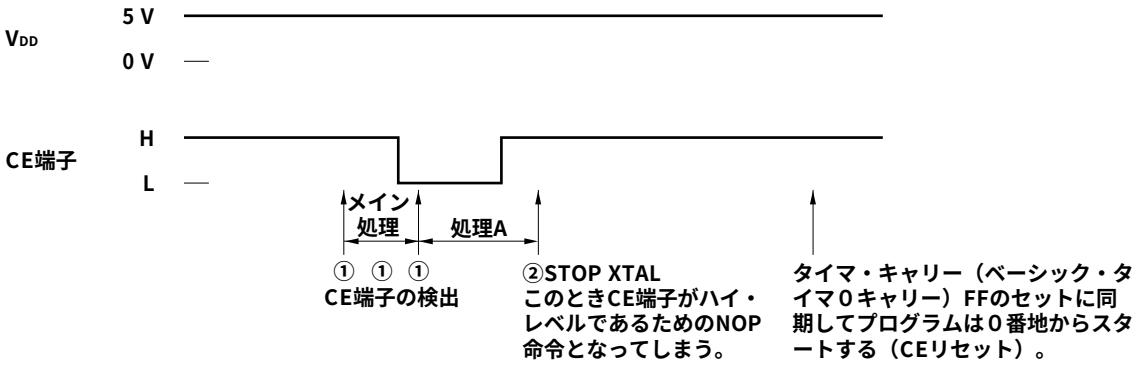
したがって，プログラム上は③のように分岐命令を挿入するか，またはメイン処理に移っても誤動作しないようなプログラムにしておく必要があります。

③のように分岐命令を用いたときは，図13－５のようにCE端子がハイ・レベルのままであっても，次のタイマ・キャリー（ベーシック・タイマ０キャリー）FFのセットと同期してCEリセットがかかります。

例

```
        XTAL    DAT     0000B   ;クロック・ストップ条件のシンボル定義
CEJDG：
  ;①
        SKF1    CE              ;組み込みマクロ
                                ;CE端子の入力レベルを検出
        BR      MAIN            ;CE=ハイ・レベルならメイン処理へ分岐

        処　理　A               ;CE=ロウ・レベルの処理

  ;②
        STOP    XTAL            ;クロック・ストップ
  ;③
        BR      $-1
MAIN：

        メイン処理

        BR      CEJDG
```

図13－５　CE端子入力によって起こるクロック・ストップ命令の誤動作と対策



第14章　リセット機能

リセット機能は，デバイス動作を初期化する機能で，パワーオン・リセットとCEリセットとの２種類があります。どちらであるのかは，動作を開始してからプログラムで判定し，それぞれに適した初期化プログラムを実行させます。

この判定用のハードウエアとプログラムとを合わせて停電検出と呼びます。

★　パワーオン・リセットは，デバイスの電源電圧V_{DD}を一定電圧以下から立ち下げることにより行います。内部の書き換え可能な部分はすべて初期化されます。

CEリセットは，クロック・ストップ状態で，規格で許される範囲の低電圧でデータ保持したあと，動作を再開させるためのものです。したがって，通常はデータ・メモリの内容を書き換えず，主なレジスタ等の初期化が済みましたら主作業の実行を開始させます。

★　**備考**　製品により，リセットで使用するタイマ名称とフラグ名は異なります。各品種のデータ・シートを参照してください（この章では，タイマ・キャリー，TMCYフラグ，TMMD0, １フラグと記述しています）。

・タイマ・キャリー←→ベーシック・タイマ０キャリー

・TMCYフラグ←→BTM0CYフラグ

・TMMD0, TMMD1, TMMD2, TMMD3フラグ←→BTM0CY0, BTM0CY1, BTM1CY0, BTM1CY1フラグ

14.1　リセット・ブロックの構成

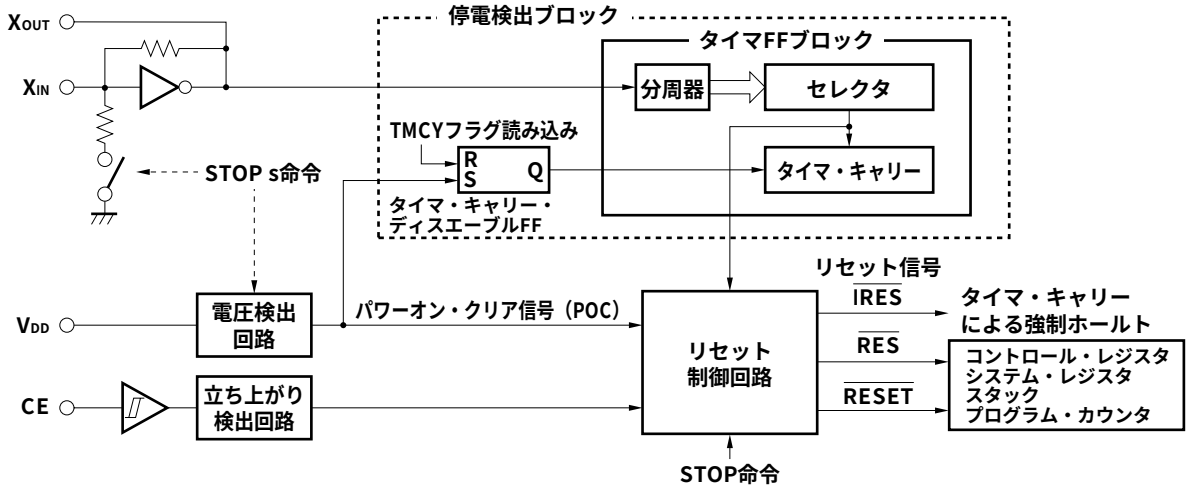
図14－１にリセット・ブロックの構成を示します。

デバイスのリセットは電源電圧V_{DD}の投入によるリセット（パワーオン・リセットまたはV_{DD}リセット）とCE端子によるリセット（CEリセット）に分けられます。

パワーオン・リセット・ブロックはV_{DD}端子に入力された電圧を検出する電圧検出回路，停電検出回路およびリセット制御回路から構成されています。

CEリセット・ブロックはCE端子に入力された信号の立ち上がりを検出する回路とリセット制御回路から構成されています。

図14－１　リセット・ブロックの構成例



14.2　リセットの機能

パワーオン・リセットは電源電圧V_{DD}が一定電圧以下から立ち上がったときにかかり，CEリセットはCE端子がロウ・レベルからハイ・レベルに立ち上がったときにかかります。

パワーオン・リセットは，プログラム・カウンタ，スタック，システム・レジスタおよびコントロール・レジスタなどの初期化を行い，プログラムを0000H番地から実行します。

CEリセットは，プログラム・カウンタ，スタック，システム・レジスタおよびコントロール・レジスタの一部などの初期化を行い，プログラムを0000H番地から実行します。

パワーオン・リセットとCEリセットの違いは主に，初期化されるコントロール・レジスタと14.6で説明する停電検出回路の動作です。

パワーオン・リセットとCEリセットは図14－１に示したリセット制御回路から出力されるリセット信号$\overline{IRES}$, $\overline{RES}$および$\overline{RESET}$信号により制御されます。

表14－１に$\overline{IRES}$, $\overline{RES}$および$\overline{RESET}$信号とパワーオン・リセットおよびCEリセットの関係を示します。

また**第13章　スタンバイ機能**で説明したクロック・ストップ命令（STOP s）実行時もリセット制御回路が動作します。

14.3および14.4にそれぞれCEリセットとパワーオン・リセットについて説明します。

14.5にCEリセットとパワーオン・リセットの関係を説明します。

表14－１　内部リセット信号と各リセットの関係

内部リセット信号	出力される信号			各リセット信号により制御される内容
	CEリセット時	パワーオン・リセット時	クロック・ストップ時	
$\overline{IRES}$	×	○	○	デバイスを強制的にホールト状態にする。 ホールトの解除はタイマ・キャリーFFのセットにより行う。
$\overline{RES}$	×	○	○	コントロール・レジスタの一部を初期化する。
$\overline{RESET}$	○	○	○	プログラム・カウンタ，スタック，システム・レジスタおよびコントロール・レジスタの一部を初期化する。

14.3　CEリセット

CEリセットは，CE端子をロウ・レベルからハイ・レベルに立ち上げることにより行います。

CE端子がハイ・レベルに立ち上がると，次のタイマ・キャリーFFセット用パルスの立ち上がりエッジに同期して$\overline{\text{RESET}}$信号が出力され，デバイスにリセットがかかります。

CEリセットがかかると，$\overline{\text{RESET}}$信号によりプログラム・カウンタ，スタック，システム・レジスタおよびコントロール・レジスタの一部を初期値にイニシャライズしてプログラムを0000H番地から実行します。

初期値については各品種のデータ・シートを参照してください。

CEリセットは，クロック・ストップを使用するときと使用しないときではリセット動作が異なります。

これを**14.3.1**および**14.3.2**にそれぞれ示します。

14.3.3にCEリセット時の注意を示します。

14.3.1　クロック・ストップ（STOP s命令）を使用しないときのCEリセット

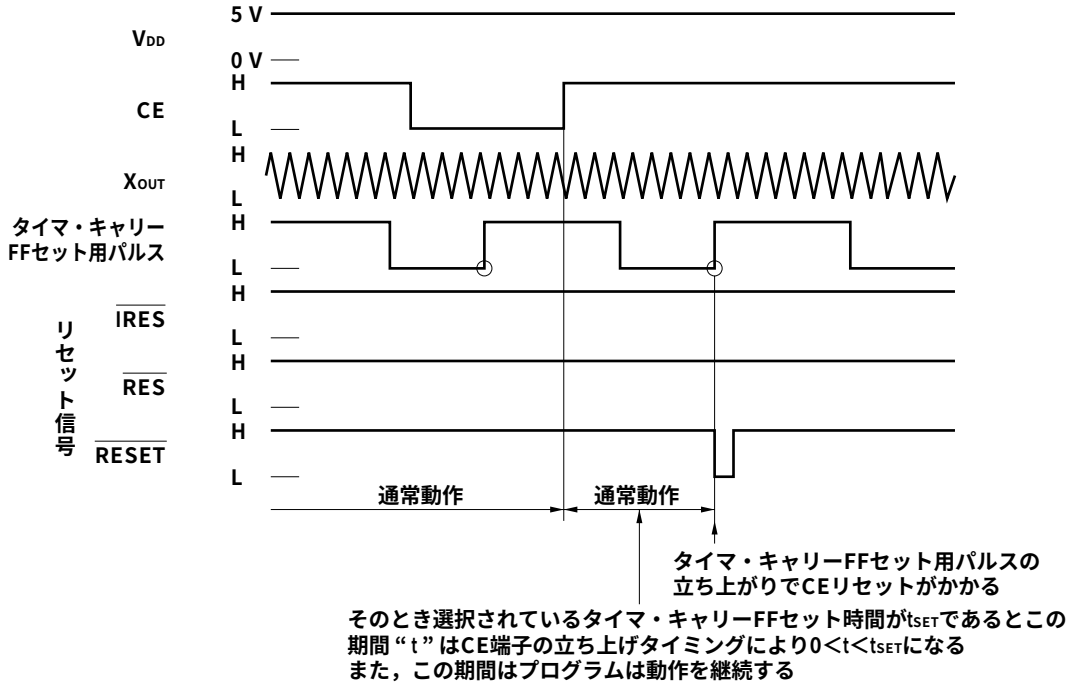
図14－2に動作を示します。

クロック・ストップ（STOP s命令）を使用しないときは，コントロール・レジスタのタイマ・モード・セレクト・レジスタはイニシャライズされません。

したがってCE端子がハイ・レベルになったあと，そのとき選択されているタイマ・キャリーFFセット用パルスの立ち上がりエッジにより$\overline{\text{RESET}}$信号が出力され，リセットがかかります。

★　**備考**　製品によって，タイマ・モード・セレクト・レジスタはベーシック・タイマ・クロック・セレクト・レジスタという名称になっています。各品種のデータ・シートを参照してください。

図14－2　クロック・ストップを使用しないときのCEリセット動作



14.3.2　クロック・ストップ（STOP s命令）を使用したときのCEリセット

図14－３に動作を示します。

クロック・ストップを使用するときは，“ STOP s ”命令が実行された時点で$\overline{\text{IRES}}$, $\overline{\text{RES}}$および$\overline{\text{RESET}}$信号が出力されます。

このとき$\overline{\text{RES}}$信号によりコントロール・レジスタのタイマ・モード・セレクト・レジスタが0000Bに初期化されるため，タイマ・キャリーFFセット用信号は100 msに指定されます。

また，CE端子がロウ・レベルの間は$\overline{\text{IRES}}$信号が出力され続けるため，タイマ・キャリーによる解除の強制ホールト状態になります。

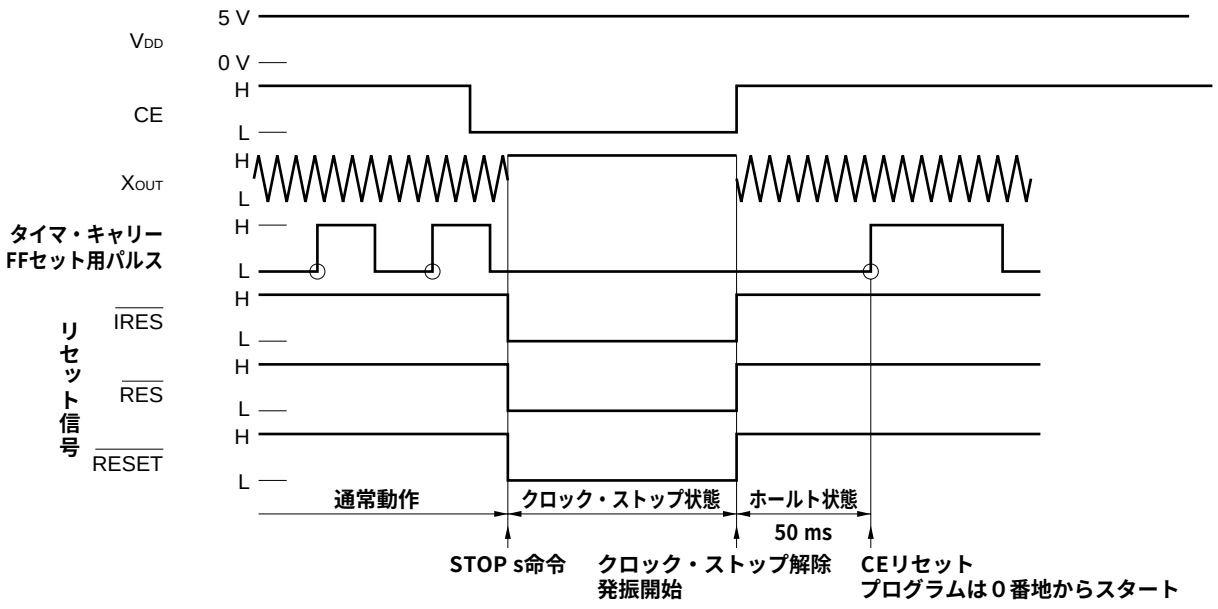
ただし，クロック自身が停止しているためデバイスは動作を停止しています。

CE端子がハイ・レベルに立ち上がるとクロック・ストップ状態が解除され発振が開始します。

このとき$\overline{\text{IRES}}$信号によりタイマ・キャリーによる解除のホールト状態になっているため，CE端子が立ち上がったあと，タイマ・キャリーFFセット用パルスが立ち上がるとホールト状態が解除されプログラムが０番地からスタートします。

タイマ・キャリーFFセット用パルスは100 msに初期化されているため，すなわちCE端子がハイ・レベルに立ち上がってから50 ms後にCEリセットがかかります。

図14－３　クロック・ストップを使用するときのCEリセット動作



14.3.3　CEリセット時の注意

CEリセットは，実行中の命令に関係なくかかるため次の（1）および（2）に示す点に注意が必要です。

（1）時計などのタイマ処理を実行する時間

タイマ・キャリーやタイマ割り込みを使用して時計のプログラムを作成する場合は，そのプログラムの処理時間を一定時間以内に終了させる必要があります。

詳しくは，各品種のデータ・シートを参照してください。

（2）プログラムに使用しているデータやフラグなどの処理

1命令で処理できないデータやフラグなどにおいて，CEリセットがかかっても内容が変化してはいけないようなもの，たとえばセキュリティ・コードなどはその内容を書き換えるときに注意が必要です。

次に例を用いて説明します。

★　**例１．**

```
R1    MEM    0.01H          ；セキュリティ・コードのキー入力データの１桁目
R2    MEM    0.02H          ；セキュリティ・コードのキー入力データの２桁目
R3    MEM    0.03H          ；セキュリティ・コード変更時の１桁目のデータ
R4    MEM    0.04H          ；セキュリティ・コード変更時の２桁目のデータ
M1    MEM    0.11H          ；現在のセキュリティ・コードの１桁目
M2    MEM    0.12H          ；現在のセキュリティ・コードの２桁目

START：

      キー入力処理
          R1←キーＡの内容   ；セキュリティ・コード入力待ちモード
          R2←キーＢの内容   ；押されたキー内容を，R1，R2に代入する

      SET2   CMP，Z      ；①；セキュリティ・コードと入力されたデータの比較
      SUB    R1，M1
      SUB    R2，M2
      SKT1   Z
      BR     ERROR          ；入力されたデータがセキュリティ・コードと異なる
MAIN：

      キー入力処理
          R3←キーＣの内容   ；セキュリティ・コードの書き換えモード
          R4←キーＤの内容   ；押されたキーの内容を，R3，R4に代入する

      ST     M1，R3      ；②；セキュリティ・コードの書き換え
      ST     M2，R4      ；③
      BR     MAIN
ERROR：

      動作しないようにする
```

例１のプログラムにおいて，いま仮にセキュリティ・コードが“12H”であったとすると，データ・メモリM1, M2の内容はそれぞれ“1H”，“2H”になります。

このとき，CEリセットがかかると，①でキー入力の内容とセキュリティ・コード“12H”を比較して，同一であれば通常処理を行います。

また，メイン処理でセキュリティ・コードが変更されるときは，変更されたコードを②，③でM1, M2に書き換えます。

ここで，仮にセキュリティ・コードが“34H”に変更されたとすると，②，③でM1, M2に“3H”,“4H”を書き込みます。

ところが，②を実行した時点でCEリセットがかかったとすると，プログラムは③を実行しないで0000H番地から実行してしまいます。

したがって，セキュリティ・コードは“32H”になってしまうため，セキュリティの解除ができなくなってしまいます。

このような場合は，例２に示すようなプログラムで対処してください。

★　**例２．**

```
R1          MEM     0.01H          ;セキュリティ・コードのキー入力データの１桁目
R2          MEM     0.02H          ;セキュリティ・コードのキー入力データの２桁目
R3          MEM     0.03H          ;セキュリティ・コード変更時の１桁目のデータ
R4          MEM     0.04H          ;セキュリティ・コード変更時の２桁目のデータ
M1          MEM     0.11H          ;現在のセキュリティ・コードの１桁目
M2          MEM     0.12H          ;現在のセキュリティ・コードの２桁目
CHANGE      FLG     0.13H.0        ;セキュリティ・コード変更中は“１”

START：

            キー入力処理
              R1←キーＡの内容  ;セキュリティ・コード入力待ちモード
              R2←キーＢの内容  ;押されたキーの内容を，R1，R2に代入する

            SKT1    CHANGE   ;④ ;CHANGEフラグが“１”であれば
            BR      SECURITY_CHK
            ST      M1，R3         ;M1，M2に再度書き込みを行う
            ST      M2，R4
            CLR1    CHANGE
SECURITY_CHK：
            SET2    CMP，Z   ;① ;セキュリティ・コードと入力されたデータの比較
            SUB     R1，M1
            SUB     R2，M2
            SKT1    Z
            BR      ERROR          ;入力されたデータがセキュリティ・コードと異なる
MAIN：

            キー入力処理
              R3←キーＣの内容  ;セキュリティ・コードの書き換えモード
              R4←キーＤの内容  ;押されたキーの内容を，R3，R4に代入する

            SET1    CHANGE   ;⑤ ;セキュリティ・コードの変更が終わるまで
                                ;CHANGEフラグを“１”に設定する
            ST      M1，R3   ;② ;セキュリティ・コードの書き換え
            ST      M2，R4   ;③
            CLR1    CHANGE         ;セキュリティ・コードの変更が終わったら
                                   ;CHANGEフラグを“０”に設定する
            BR      MAIN
ERROR：

            動作しないようにする
```

例２のプログラムでは，②，③でセキュリティ・コードの書き換えを行う前に，⑤でCHANGEフラグを“１”に設定しています。

したがって，③を実行する前にCEリセットがかかってしまった場合でも，④で再度書き込みを行います。

14.4　パワーオン・リセット

パワーオン・リセットは，デバイスの電源電圧V_{DD}を一定電圧（パワーオン・クリア電圧と呼ぶ）以下から立ち上げることにより行います。

電源電圧V_{DD}がパワーオン・クリア電圧以下のときは，図14－１に示した電圧検出回路からパワーオン・クリア信号（POC）が出力されます。

パワーオン・クリア信号が出力されると，水晶発振回路を停止しデバイス動作を停止します。

また，パワーオン・クリア信号出力中は$\overline{IRES}$, $\overline{RES}$および$\overline{RESET}$信号が出力されます。

電源電圧V_{DD}がパワーオン・クリア電圧を越えるとパワーオン・クリア信号が切れ，水晶発振を開始すると同時に$\overline{IRES}$, $\overline{RES}$および$\overline{RESET}$信号も切れます。

このとき，$\overline{IRES}$信号によりタイマ・キャリーによる解除のホールト状態になっているため，次のタイマ・キャリーFFセット用信号の立ち上がりエッジでパワーオン・リセットがかかります。

★　タイマ・キャリーFFセット用信号は，$\overline{RESET}$信号により100 msに初期化されているためすなわち，電源電圧V_{DD}がパワーオン・クリア電圧を越えてから50 ms後にリセットがかかりプログラムが０番地からスタートします。

この動作を図14－４に示します。

パワーオン・リセット時のプログラム・カウンタ，スタック，システム・レジスタおよびコントロール・レジスタの初期化は，パワーオン・クリア信号が出力された時点で行われます。

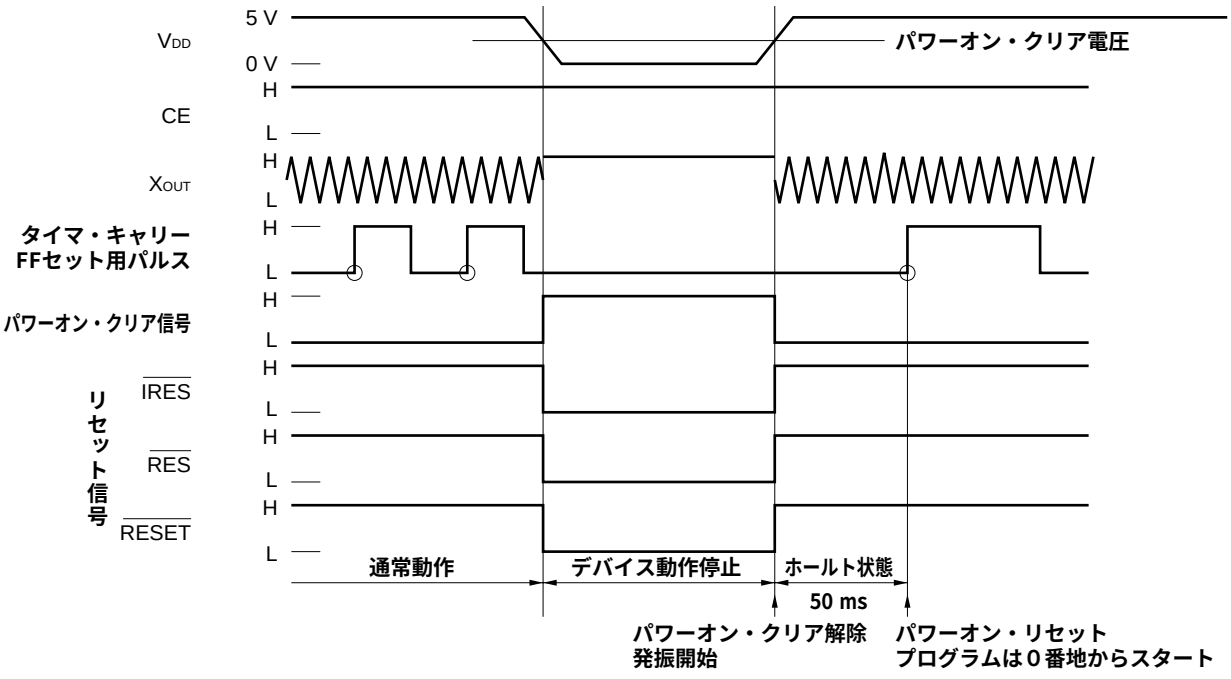
また，初期値については各品種のデータ・シートを参照してください。

パワーオン・クリア電圧は通常動作時は3.5 V（規格値）であり，クロック・ストップ状態中は2.2 V（規格値）になります。

このときの動作を**14.4.1**および**14.4.2**に示します。

また電源電圧V_{DD}を0 Vから立ち上げたときの動作を**14.4.3**に示します。

図14－４　パワーオン・リセットの動作



14.4.1　通常動作時のパワーオン・リセット

図14－5の（a）に動作を示します。

図14－5の（a）に示すようにCE端子の入力レベルに関係なく電源電圧V_{DD}が3.5 Vより下がるとパワーオン・クリア信号が出力されデバイスの動作が停止します。

次に電源電圧V_{DD}が再度3.5 V以上に立ち上がると，50 msのホールト後プログラムが0000H番地からスタートします。

通常動作時とは，クロック・ストップ命令を使用していないときで，ホールト命令によるホールト状態中も含みます。

14.4.2　クロック・ストップ状態中のパワーオン・リセット

図14－5の（b）に動作を示します。

図14－5の（b）に示すように電源電圧V_{DD}が2.2 Vより下がるとパワーオン・クリア信号が出力されデバイス動作が停止します。

ただし，クロック・ストップ状態であるため，見かけ上デバイス動作は変化しません。

次に電源電圧V_{DD}が3.5 V以上に立ち上がると50 msのホールト後プログラムが0000H番地からスタートします。

14.4.3　電源電圧V_{DD}が0 Vから立ち上がったときのパワーオン・リセット

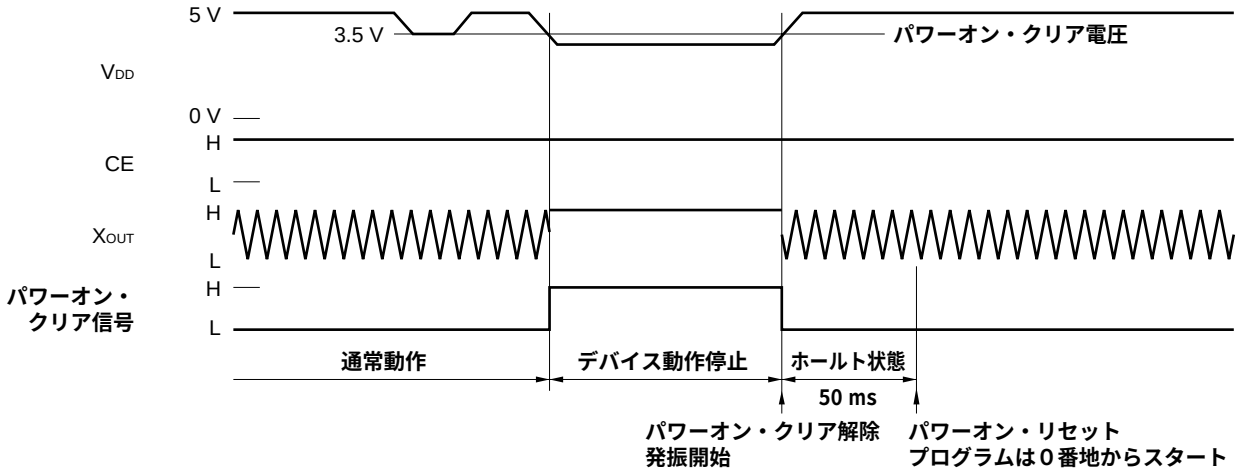
図14－5の（c）に動作を示します。

図14－5の（c）に示すように電源電圧V_{DD}が0 Vから3.5 Vに立ち上がるまでパワーオン・クリア信号が出力されます。

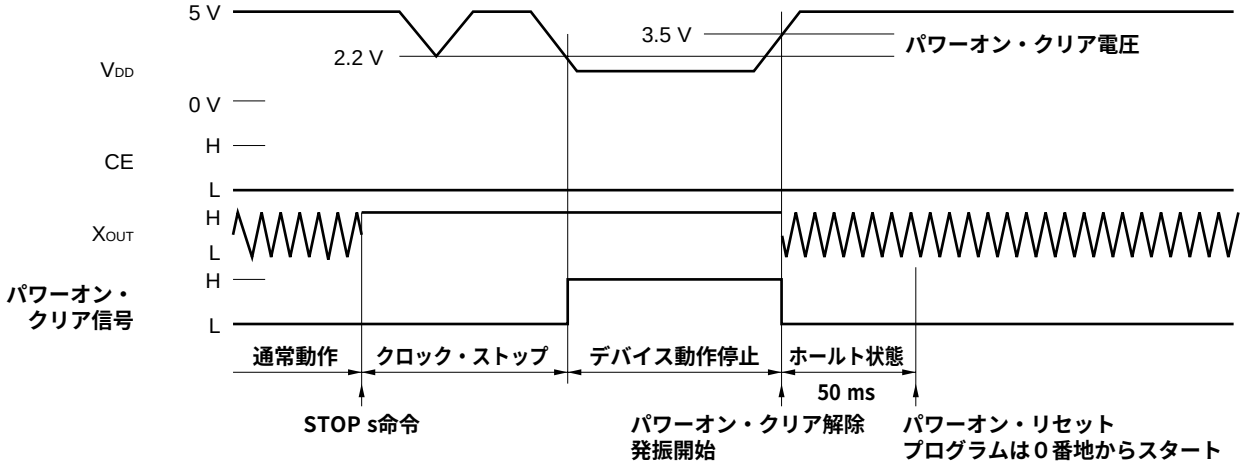
電源電圧V_{DD}がパワーオン・クリア電圧を越えると水晶発振回路が動作を開始し，50 msのホールト後にプログラムが0000H番地からスタートします。

図14－5　パワーオン・リセットと電源電圧V_{DD}

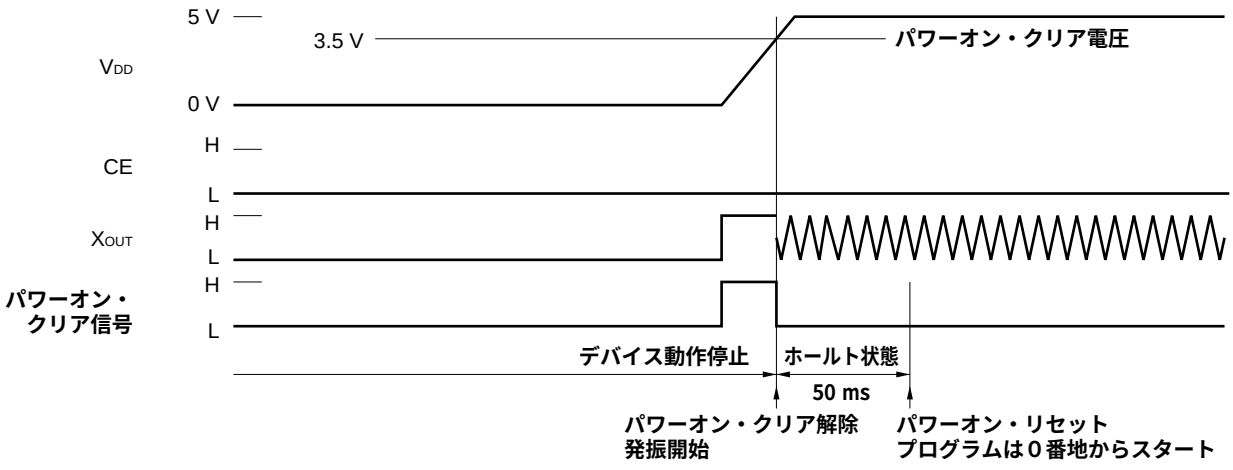
（a）通常動作時（ホールト状態中も含む）



（b）クロック・ストップ時



（c）電源電圧V_{DD}を0 Vから立ち上げたとき



14.5　CEリセットとパワーオン・リセットの関係

最初の電源電圧V_{DD}投入時などは，パワーオン・リセットとCEリセットが同時にかかる可能性があります。
このときのリセット動作を14.5.1-14.5.3に示します。
また，14.5.4に電源電圧V_{DD}立ち上げ時の注意を示します。

14.5.1　V_{DD}端子とCE端子が同時に立ち上がったとき

図14－6の（a）に動作を示します。
このときは，パワーオン・リセットによりプログラムが0000H番地からスタートします。

14.5.2　パワーオン・リセットの強制ホールト中にCE端子が立ち上がったとき

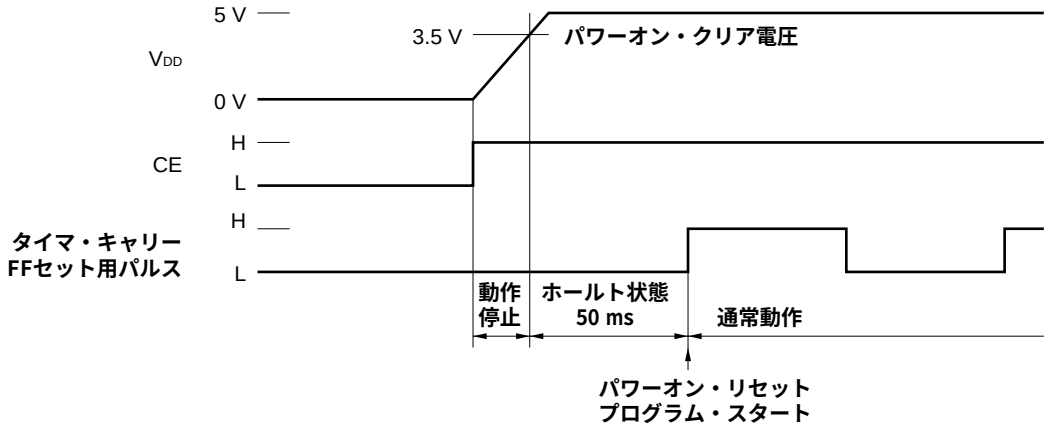
図14－6の（b）に動作を示します。
このときは，14.5.1と同様にパワーオン・リセットによりプログラムが0000H番地からスタートします。

14.5.3　パワーオン・リセット後にCE端子が立ち上がったとき

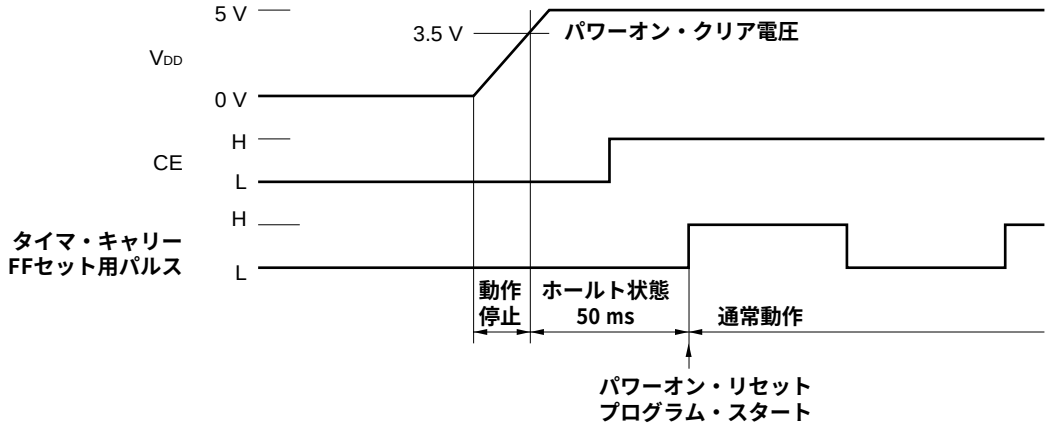
図14－6の（c）に動作を示します。
このときは，パワーオン・リセットによりプログラムが0000H番地からスタートし，さらにCEリセットにより次のタイマ・キャリーFFセット用信号の立ち上がりでプログラムが0000H番地から再度スタートします。

図14－6　パワーオン・リセットとCEリセットの関係

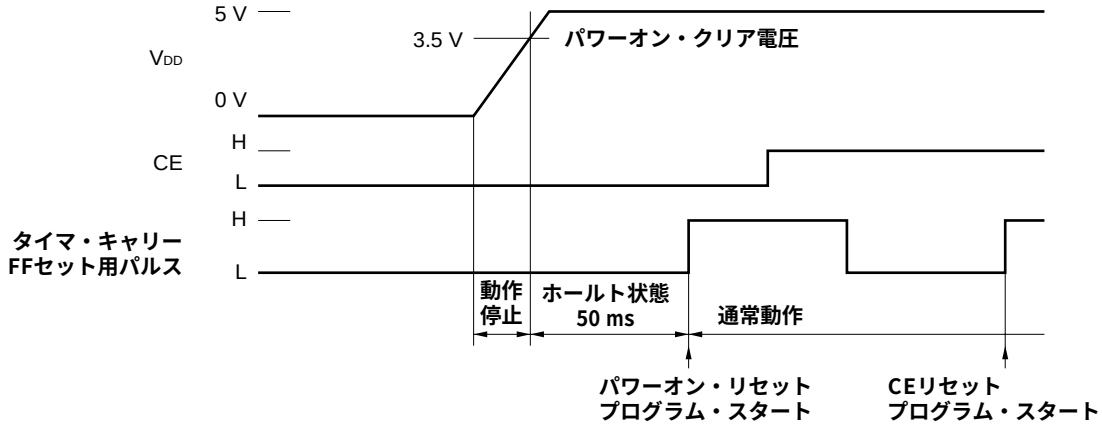
（a）V_{DD}とCE端子が同時に立ち上がったとき



（b）ホールト状態中にCE端子が立ち上がったとき



（c）パワーオン・リセット後にCE端子が立ち上がったとき



14.5.4　電源電圧VDD立ち上げ時の注意

電源電圧V_{DD}を立ち上げるときは，次の（1）および（2）に示す注意が必要です。

（1）電源電圧V_{DD}をパワーオン・クリア電圧以下から立ち上げるとき

電源電圧V_{DD}を立ち上げるときは，一度3.5 V以上に立ち上げる必要があります。

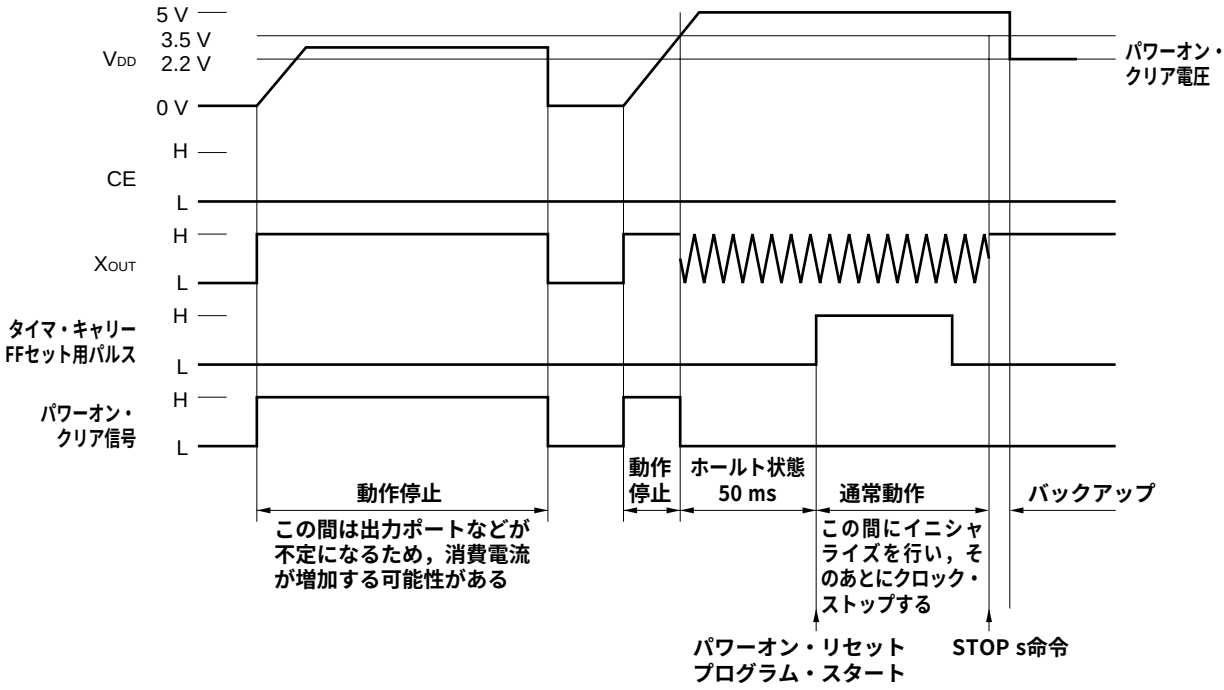
これを図14－7に示します。

図14－7に示すように，たとえばクロック・ストップを使用してV_{DD}を2.2 Vでバックアップするようなプログラムにおいて，V_{DD}投入時に3.5 V未満の電圧しか印加されなかったとすると，パワーオン・クリア信号が出力されたままになりプログラムは動作しません。

このとき，デバイスの出力ポート等は不定な値を出力するため，場合により消費電流が増加します。

すなわち，電池などによるバックアップ時はバックアップ時間が大幅に減少してしまいます。

図14－7　V_{DD}立ち上げ時の注意



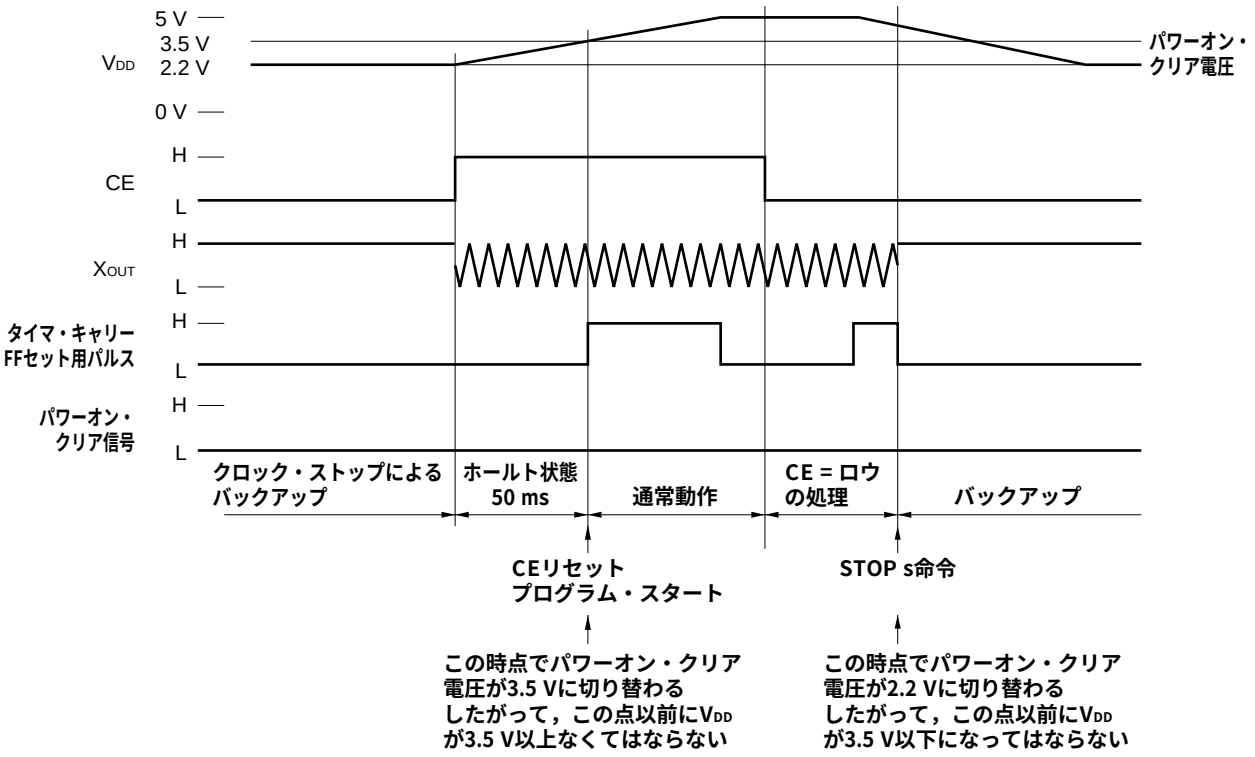
（2）クロック・ストップ状態からの復帰時

クロック・ストップを使用して電源電圧V_{DD}を2.2 Vでバックアップしているときに，バックアップ状態から復帰する場合は，CE端子がハイ・レベルになってから50 ms以内にV_{DD}を3.5 V以上に立ち上げる必要があります。

図14－8に示すようにクロック・ストップ状態からの復帰はCEリセットにより行いますが，CE端子が立ち上がってから50 ms後にパワーオン・クリア電圧が3.5 Vに切り替わるため，この時点にV_{DD}が3.5 V以上ないとパワーオン・リセットがかかってしまいます。

またV_{DD}を立ち下げるときも同様に注意が必要です。

図14－8　クロック・ストップ状態からの復帰



14.6　停電検出

停電検出は，図14－９に示すようにデバイスにリセットがかかったときに電源電圧V_{DD}の投入であったかもしくはCE端子によるリセットであったかを判断するために使用します。

電源電圧投入時は，データ・メモリや出力ポートなどの内容が“不定”になっているため，停電検出によりこれらの内容を初期化します。

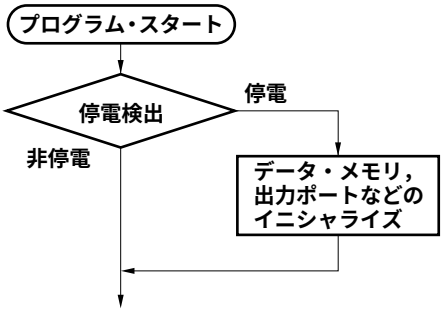
これに対し，CEリセットの場合は，データ・メモリや出力ポートに以前の値が残っていますから，これらの初期化は必要ありません。

停電検出方法には，停電検出回路を使用してTMCYフラグを検出する方法およびデータ・メモリの内容を検出する方法（RAMジャッジ）の２通りがあります。

14.6.1および**14.6.2**に停電検出回路およびTMCYフラグによる停電検出方法を説明します。

14.6.3および**14.6.4**にRAMジャッジ方法による停電検出を説明します。

図14－９　停電検出フロー・チャート



14.6.1　停電検出回路

停電検出回路は図14－１に示したように電圧検出回路と，電圧検出回路の出力（パワーオン・クリア信号）によりセットされるタイマ・キャリー・ディスエーブルFFおよびタイマ・キャリーで構成されます。

タイマ・キャリー・ディスエーブルFFは，パワーオン・クリア信号によりセット（１）され，TMCYフラグを読み込む命令が実行されるとリセット（０）されます。

タイマ・キャリー・ディスエーブルFFがセット（１）されているときは，TMCYフラグはセット（１）されません。

すなわち，パワーオン・クリア信号が出力される（パワーオン・リセット時）と，TMCYフラグはリセットされた状態でプログラムがスタートし，以後TMCYフラグを読み込む命令が実行されるまでセット禁止状態になります。

一度TMCYフラグを読み込む命令が実行されれば，以後タイマ・キャリーFFセット用パルスの立ち上がりごとにTMCYフラグがセットされます。したがってデバイスにリセットがかかったときにTMCYフラグの内容を検出し，リセット（０）されていればパワーオン・リセット（停電）であり，セット（１）されていればCEリセット（非停電）であることを判断できます。

停電を検出できる電圧はパワーオン・リセットのかかる電圧と同じになるため，水晶発振時でV_{DD} = 3.5 V，クロック・ストップ時でV_{DD} = 2.2 Vになります。

図14－10にTMCYフラグの状態遷移を示します。

また，図14－11に図14－10のタイミング・チャートおよびTMCYフラグの動作を示します。

図14－10　TMCYフラグの状態遷移

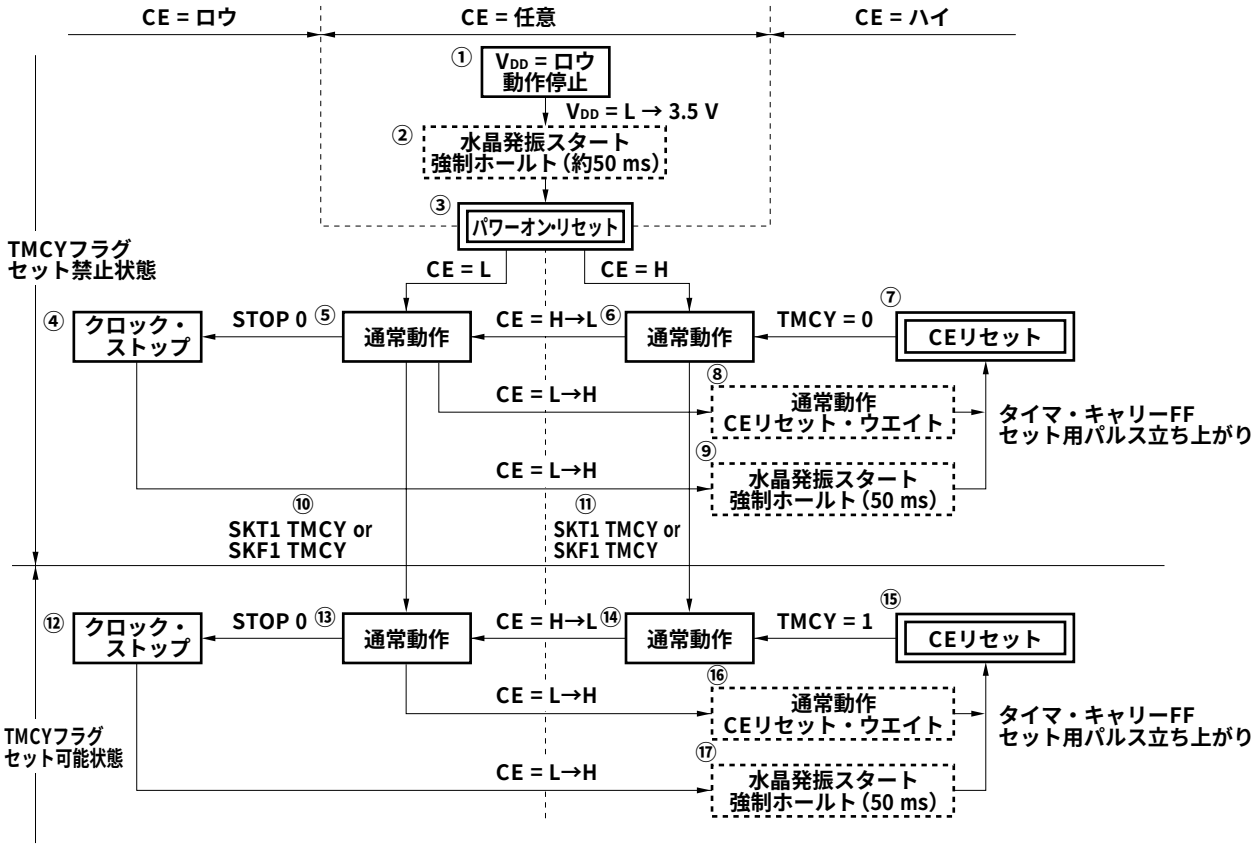
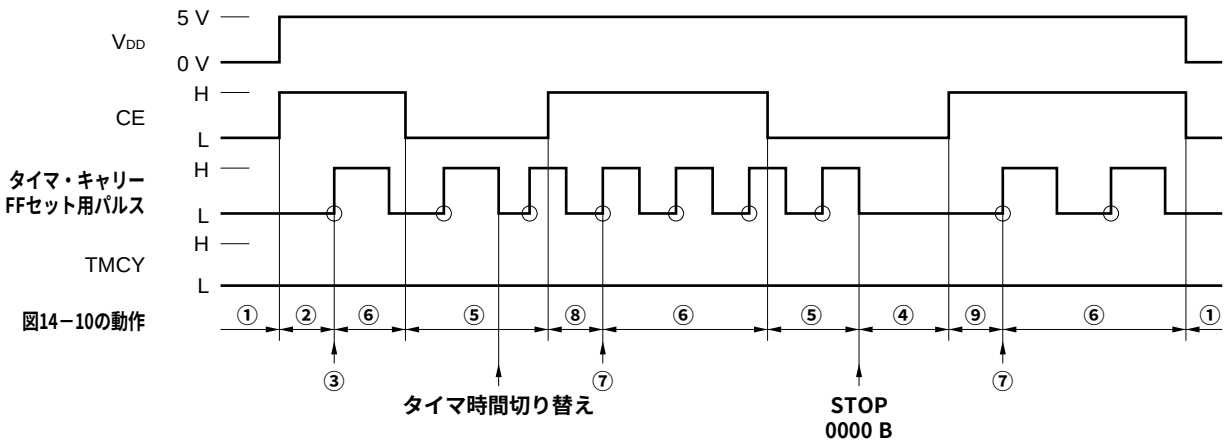
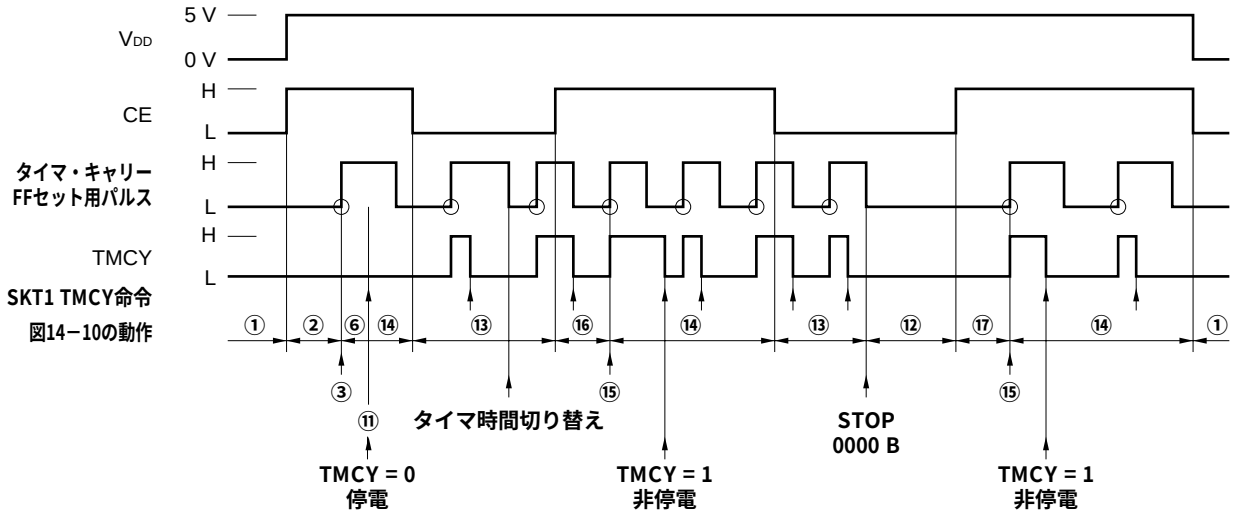


図14－11　TMCYフラグの動作

（a）TMCYフラグを一度も検出しないとき（SKT1 TMCYまたはSKF1 TMCYを実行しない）



（b）TMCYフラグによる停電検出を行うとき



14.6.2　TMCYフラグによる停電検出時の注意

TMCYフラグで時計のカウントなどを行うときは次の点に注意が必要です。

（1）時計の更新

タイマ・キャリーを使用して時計などのプログラムを作成するときは，停電検出後に時計を更新する必要があります。

これは停電検出時にTMCYフラグを読み込むためにTMCYフラグがリセット（0）され，1回分時計のカウントを取り逃がすためです。

（2）時計更新処理時間

時計の更新を行うときは，その処理を次のタイマ・キャリーFFセット用パルスが立ち上がるまでに終了させる必要があります。

これは，時計更新処理中にCE端子がハイ・レベルに立ち上がると時計更新処理を最後まで実行しないでCEリセットがかかってしまうためです。

また，停電時の処理を行うときは次の点に注意が必要です。

（3）停電検出を行うタイミング

TMCYフラグで時計のカウントなどを行うときは，停電検出のためのTMCYフラグの読み込みをプログラムが0000H番地からスタートしてから次のタイマ・キャリーFFセット用パルスが立ち上がる時間以内に行う必要があります。

これは，タイマ・キャリーFFセット時間をたとえば5 msに設定し，停電検出をプログラムがスタートしてから6ms後に行うとすると，1回分TMCYフラグを取り逃がしてしまうためです。

また，次の例に示すように，停電検出およびイニシャル処理は，タイマ・キャリーFFセット時間以内で行う必要があります。

これは停電処理およびイニシャル処理中にCE端子が立ち上がりCEリセットがかかると，これらの処理が途中で中断されてしまい，不具合が発生する可能性があるためです。

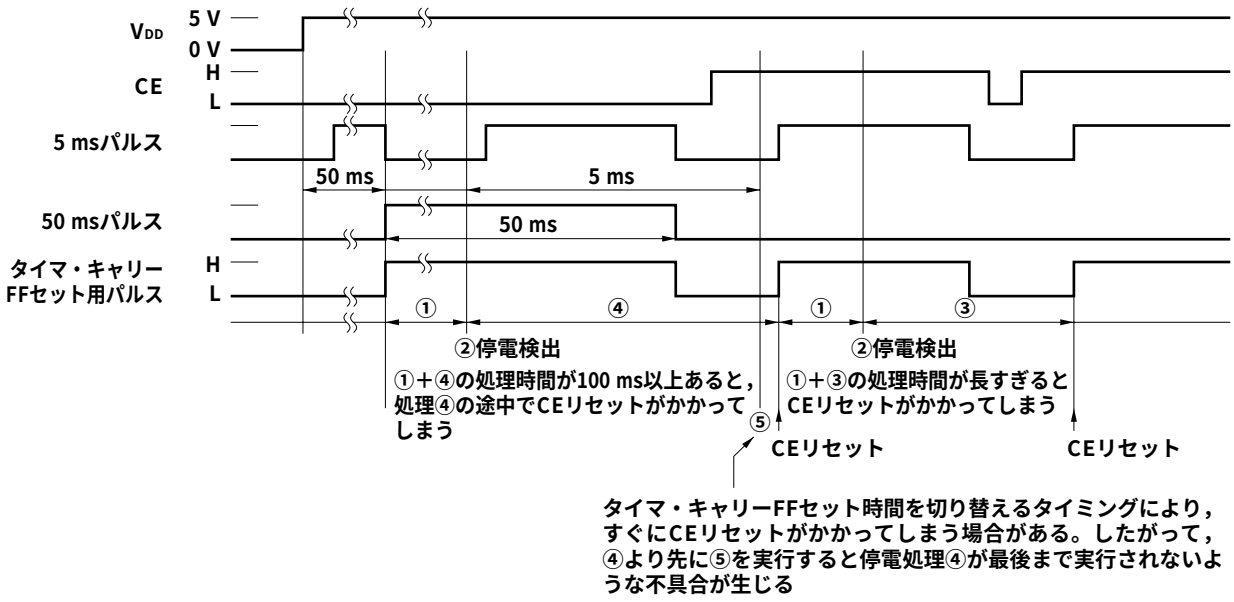
また，イニシャル処理でタイマ・キャリーFFセット時間を変更する場合，変更する命令はイニシャル処理の一番最後に1命令で行う必要があります。

これも，次の例に示すようにイニシャル処理よりさきにタイマ・キャリーFFのセット時間が切り替わると，CEリセットによりイニシャル処理が最後まで実行できない可能性があるためです。

例

```
START：                              ；プログラム・アドレス0000H
；①

  リセット時の処理

；②
  SKT1    TMCY                       ；停電検出
  BR      INITIAL
BACKUP：
；③

  時　計　更　新

  BR      MAIN
INITIAL：
；④

  イニシャル処理

；⑤
  INITFLG TMMD1, NOT TMMD0           ；組み込みマクロ
                                     ；タイマ・キャリーFFセット時間を5msに設定
MAIN：

  メ イ ン 処 理

  SKT1    TMCY
  BR      MAIN

  時　計　更　新

  BR      MAIN
```

動作例



14.6.3　RAMジャッジ方法による停電検出

RAMジャッジ方法は，デバイスにリセットがかかったときに，所定アドレスのデータ・メモリの内容が所定の値になっているかを判断することにより停電検出を行います。

次にRAMジャッジによる停電検出のプログラム例を示します。

★ RAMジャッジ方法は，電源電圧V_{DD}投入時のデータ・メモリの内容は“不定”になっており，“不定”の値と“所定”の値を比較することにより停電検出を行います。

したがって**14.6.4　RAMジャッジによる停電検出時の注意**に示すように停電検出を誤判断する可能性があります。

ただし，RAMジャッジ方法を用いる利点としては表14－2に示すように，停電検出回路による停電検出より低い電源電圧までバックアップが可能になる点です。

表14－2　停電検出回路とRAMジャッジによる停電検出の比較

	停電検出回路		RAMジャッジ	
データ保持電圧	実力値	規格	実力値	規格
（クロック・ストップ時）	1-2 V	2.2 V	0-1 V	2.0 V
動作状態	誤動作なし		誤動作の可能性あり	

★　例　RAMジャッジによる停電検出プログラム例

```
M012      MEM     0.12H
M034      MEM     0.34H
M056      MEM     0.56H
M107      MEM     1.07H
M128      MEM     1.28H
M16F      MEM     1.6FH
DATA0     DAT     1010B
DATA1     DAT     0101B
DATA2     DAT     0110B
DATA3     DAT     1001B
DATA4     DAT     1100B
DATA5     DAT     0011B

START：
          SET2    CMP, Z
          SUB     M012, #DATA0    ; M012=DATA0かつ
          SUB     M034, #DATA1    ; M034=DATA1かつ
          SUB     M056, #DATA2    ; M056=DATA2かつ
          BANK1
          SUB     M107, #DATA3    ; M107=DATA3かつ
          SUB     M128, #DATA4    ; M128=DATA4かつ
          SUB     M16F, #DATA5    ; M16F=DATA5であれば
          BANK0
          SKF1    Z
          BR      BACKUP          ; BACKUPへ分岐
；INITIAL：

          イニシャル処理

          MOV     M012, #DATA0
          MOV     M034, #DATA1
          MOV     M056, #DATA2
          BANK1
          MOV     M107, #DATA3
          MOV     M128, #DATA4
          MOV     M16F, #DATA5
          BR      MAIN
BACKUP：

          バックアップ処理

MAIN：

          メ イ ン 処 理
```

14.6.4　RAMジャッジによる停電検出時の注意

電源電圧V_{DD}投入時のデータ・メモリの値は基本的には“不定”の値であるため，次の（1）および（2）に示す点に注意が必要です。

（1）比較するデータ

RAMジャッジにより比較するデータ・メモリのビット数を“nビット”とすると，V_{DD}投入時にデータ・メモリの値がたまたま比較する値と一致する確率は$(1/2)^n$になります。

すなわち，RAMジャッジによる停電検出は$(1/2)^n$の確率でバックアップと判断されてしまうことになります。

したがってこの確率を小さくするためには，できる限り多くのビット数を比較するようにします。

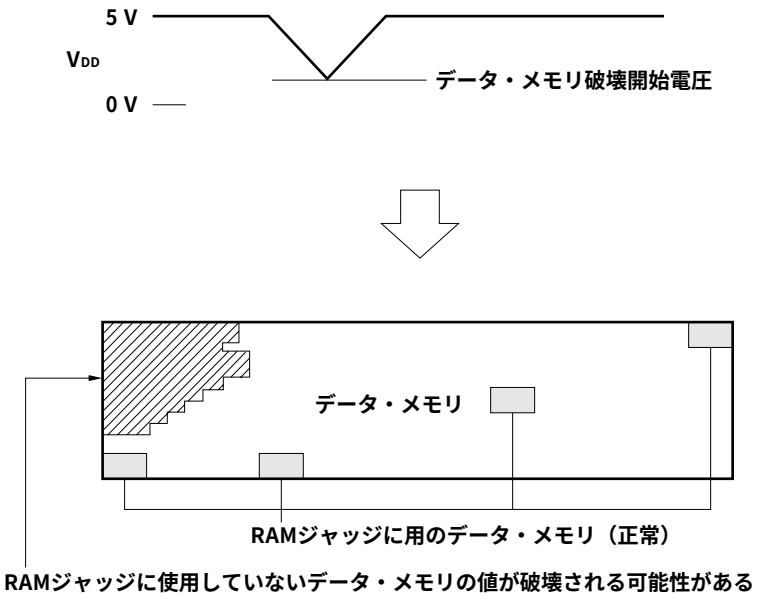
また，経験上，V_{DD}投入時のデータ・メモリの内容は“0000B”や“1111B”のような同一の値になりやすいため，比較するためのデータを“1010B”や“0110B”のように，“0”と“1”が混在するような値にした方が，判断を誤る可能性が少なくなります。

（2）プログラム上の注意

図14－12に示すようにV_{DD}がデータ・メモリが破壊され始める電圧から立ち上がったときは，比較するデータ・メモリの値が正常であっても，その他の部分が破壊される場合があります。

このときRAMジャッジによる停電検出ではバックアップと判断されるため，プログラム上，データ・メモリが破壊されていても暴走しないよう配慮する必要があります。

図14－12　V_{DD}とデータ・メモリの破壊



第15章　命令セット

★　デバイスによっては，サポートされていないニモニックがありますので，ターゲット・デバイスのデータ・シートやデバイス・ファイルのユーザーズ・マニュアルを参照してください。

15.1　命令セット概要

b_{14}-b_{11} \ b_{15}		0		1	
BIN	HEX				
0000	0	ADD	r, m	ADD	m, #n4
0001	1	SUB	r, m	SUB	m, #n4
0010	2	ADDC	r, m	ADDC	m, #n4
0011	3	SUBC	r, m	SUBC	m, #n4
0100	4	AND	r, m	AND	m, #n4
0101	5	XOR	r, m	XOR	m, #n4
0110	6	OR	r, m	OR	m, #n4
0111	7	INC	AR		
		INC	IX		
		MOVT	DBF, @AR		
		BR	@AR		
		CALL	@AR		
		RET			
★		SYSCAL	entry		
		RETSK			
		EI			
		DI			
		RETI			
		PUSH	AR		
		POP	AR		
		GET	DBF, p		
		PUT	p, DBF		
		PEEK	WR, rf		
		POKE	rf, WR		
		RORC	r		
		STOP	s		
		HALT	h		
		NOP			
1000	8	LD	r, m	ST	m, r
1001	9	SKE	m, #n4	SKGE	m, #n4
1010	A	MOV	@r, m	MOV	m, @r
1011	B	SKNE	m, #n4	SKLT	m, #n4
1100	C	BR	addr（ページ０）	CALL	addr
1101	D	BR	addr（ページ１）	MOV	m, #n4
1110	E	BR	addr（ページ２）	SKT	m, #n
1111	F	BR	addr（ページ３）	SKF	m, #n

15.2　凡　　例

AR　　：アドレス・レジスタ
ASR　　：スタック・ポインタで示されるアドレス・スタック・レジスタ
addr　　：プログラム・メモリ・アドレス（11ビット，上位１ビットは０固定）
BANK　　：バンク・レジスタ
CMP　　：コンペア・フラグ
CY　　：キャリー・フラグ
DBF　　：データ・バッファ
entry　　：システム・セグメントのエントリ・アドレス
h　　：ホールト解除条件
INTEF　　：インタラプト・イネーブル・フラグ
INTR　　：割り込み時スタックに自動退避されるレジスタ
INTSK　　：割り込みスタック・レジスタ
IX　　：インデクス・レジスタ
MP　　：データ・メモリ・ロウ・アドレス・ポインタ
　MPE　　：メモリ・ポインタ・イネーブル・フラグ
m　　：m_R，m_Cで示されるデータ・メモリ・アドレス
　m_R　　：データ・メモリ・ロウ・アドレス（上位）
　m_C　　：データ・メモリ・カラム・アドレス（下位）
n　　：ビット・ポジション（４ビット）
n4　　：イミーディエト・データ（４ビット）
PC　　：プログラム・カウンタ
p　　：周辺アドレス
　p_H　　：周辺アドレス（上位３ビット）
　p_L　　：周辺アドレス（下位４ビット）
r　　：ジェネラル・レジスタ・カラム・アドレス
rf　　：レジスタ・ファイル・アドレス
　rf_R　　：レジスタ・ファイル・ロウ・アドレス（上位３ビット）
　rf_C　　：レジスタ・ファイル・カラム・アドレス（下位４ビット）
SP　　：スタック・ポインタ
s　　：ストップ解除条件
WR　　：ウインドウ・レジスタ
（×）　　：×でアドレスされる内容

15.3　命令セット一覧

命令群	ニモニック	オペランド	オペレーション	マシン・コード			
				オペ・コード	オペランド		
加算	ADD	r, m	(r) ← (r) ＋ (m)	00000	m_R	m_C	r
		m, #n4	(m) ← (m) ＋n4	10000	m_R	m_C	n4
	ADDC	r, m	(r) ← (r) ＋ (m) ＋CY	00010	m_R	m_C	r
		m, #n4	(m) ← (m) ＋n4＋CY	10010	m_R	m_C	n4
	INC	AR	AR←AR＋1	00111	000	1001	0000
		IX	IX←IX＋1	00111	000	1000	0000
減算	SUB	r, m	(r) ← (r) － (m)	00001	m_R	m_C	r
		m, #n4	(m) ← (m) －n4	10001	m_R	m_C	n4
	SUBC	r, m	(r) ← (r) － (m) －CY	00011	m_R	m_C	r
		m, #n4	(m) ← (m) －n4－CY	10011	m_R	m_C	n4
論理演算	OR	r, m	(r) ← (r) ∨ (m)	00110	m_R	m_C	r
		m, #n4	(m) ← (m) ∨n4	10110	m_R	m_C	n4
	AND	r, m	(r) ← (r) ∧ (m)	00100	m_R	m_C	r
		m, #n4	(m) ← (m) ∧n4	10100	m_R	m_C	n4
	XOR	r, m	(r) ← (r) ⊻ (m)	00101	m_R	m_C	r
		m, #n4	(m) ← (m) ⊻ n4	10101	m_R	m_C	n4
判断	SKT	m, #n	CMP←0，if (m) ∧n = n，then skip	11110	m_R	m_C	n
	SKF	m, #n	CMP←0，if (m) ∧n = 0，then skip	11111	m_R	m_C	n
比較	SKE	m, #n4	(m) －n4，skip if zero	01001	m_R	m_C	n4
	SKNE	m, #n4	(m) －n4，skip if not zero	01011	m_R	m_C	n4
	SKGE	m, #n4	(m) －n4，skip if not borrow	11001	m_R	m_C	n4
	SKLT	m, #n4	(m) －n4，skip if borrow	11011	m_R	m_C	n4
回転	RORC	r	CY→$(r)_{b3}$→$(r)_{b2}$→$(r)_{b1}$→$(r)_{b0}$	00111	000	0111	r
転送	LD	r, m	(r) ← (m)	01000	m_R	m_C	r
	ST	m, r	(m) ← (r)	11000	m_R	m_C	r
	MOV	@r, m	if MPE = 1：(MP，(r)) ← (m) if MPE = 0：(BANK，m_R，(r)) ← (m)	01010	m_R	m_C	r
		m, @r	if MPE = 1：(m) ← (MP，(r)) if MPE = 0：(m) ← (BANK，m_R，(r))	11010	m_R	m_C	r
		m, #n4	(m) ←n4	11101	m_R	m_C	n4
	MOVT[注]	DBF, @AR	SP←SP－1，ASR←PC，PC←AR DBF← (PC) ，PC←ASR， SP←SP＋1	00111	000	0001	0000

注　MOVT命令の実行には例外的に２命令サイクルを必要とします。

命令群	ニモニック	オペランド	オペレーション	マシン・コード			
				オペ・コード	オペランド		
転送	PUSH	AR	SP←SP−1，ASR←AR	00111	000	1101	0000
	POP	AR	AR←ASR，SP←SP＋1	00111	000	1100	0000
	PEEK	WR, rf	WR←（rf）	00111	rf_R	0011	rf_C
	POKE	rf, WR	（rf）←WR	00111	rf_R	0010	rf_C
	GET	DBF, p	DBF←（p）	00111	p_H	1011	p_L
	PUT	p, DBF	（p）←DBF	00111	p_H	1010	p_L
分岐	BR	addr	PC_{10-0}←addr，PAGE←0	01100	addr		
			PC_{10-0}←addr，PAGE←1	01101			
			PC_{10-0}←addr，PAGE←2	01110			
			PC_{10-0}←addr，PAGE←3	01111			
		@AR	PC←AR	00111	000	0100	0000
サブルーチン	CALL	addr	SP←SP−1，ASR←PC， PC←addr	11100	addr		
		@AR	SP←SP−1，ASR←PC， PC←AR	00111	000	0101	0000
★	SYSCAL	entry	SP←SP−1，ASR←PC，SGR←1， $PC_{12,11}$←0，PC_{10-8}←$entry_H$，PC_{7-4}←0， PC_{3-0}←$entry_L$	00111	$entry_H$	0000	$entry_L$
	RET		PC←ASR，SP←SP＋1	00111	000	1110	0000
	RETSK		PC←ASR，SP←SP＋1 and skip	00111	001	1110	0000
	RETI		PC←ASR，INTR←INTSK，SP←SP＋1	00111	100	1110	0000
割り込み	EI		INTEF←1	00111	000	1111	0000
	DI		INTEF←0	00111	001	1111	0000
その他	STOP	s	STOP	00111	010	1111	s
	HALT	h	HALT	00111	011	1111	h
	NOP		No operation	00111	100	1111	0000

★ 15.4　アセンブラ（RA17K）組み込みマクロ命令

凡　例

flag n　：FLG型シンボル

〈　〉　：〈　〉内は省略可能

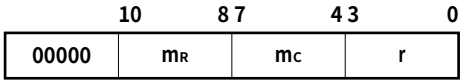
	ニモニック	オペランド	オペレーション	n
組み込みマクロ	SKTn	flag 1, …flag n	if（flag 1）～（flag n）= all “ 1 ”, then skip	1≦n≦4
	SKFn	flag 1, …flag n	if（flag 1）～（flag n）= all “ 0 ”, then skip	1≦n≦4
	SETn	flag 1, …flag n	（flag 1）～（flag n）←1	1≦n≦4
	CLRn	flag 1, …flag n	（flag 1）～（flag n）←0	1≦n≦4
	NOTn	flag 1, …flag n	if（flag n）= “ 0 ”, then（flag n）←1 if（flag n）= “ 1 ”, then（flag n）←0	1≦n≦4
	INITFLG	〈NOT〉flag 1, …〈〈NOT〉flag n〉	if description = NOT flag n, then（flag n）←0 if description = flag n, then（flag n）←1	1≦n≦4
	BANKn		BANK←n	0≦n≦2
拡張命令	BRX	Label	Jump Label	−
	CALLX	function-name	CALL sub-routine	−
	INITFLGX	〈NOT/INV〉flag 1, …〈NOT/INV〉flag n	if description = NOT（or INV）←0 flag,（flag）←0 if description = flag,（flag）←1	n≦4

15.5　命令の個別説明

15.5.1　加算命令

（1）ADD r, m　　Add data memory to general register

①　命令コード



②　機　　能

CMP = 0のとき　　(r) ← (r) ＋ (m)

ジェネラル・レジスタの内容にデータ・メモリの内容を加算し，結果をジェネラル・レジスタへ格納します。

CMP = 1のとき　　(r) ＋ (m)

結果はレジスタに格納されず，キャリー・フラグCYとゼロ・フラグZが結果によって変化します。

加算の結果，桁上げがあればキャリー・フラグCYをセットし，桁上げがなければキャリー・フラグCYをリセットします。

加算の結果がゼロ以外になったときは，コンペア・フラグCMPに関係なくゼロ・フラグZはリセットされます。

コンペア・フラグがリセットされた状態（CMP = 0）で加算の結果がゼロになったときは，ゼロ・フラグZがセットされます。

コンペア・フラグがセットされた状態（CMP = 1）で加算の結果がゼロになったときは，ゼロ・フラグZは変化しません。

加算には2進4ビット演算とBCD演算の2種類があり，どちらの演算を行うかをPSWORDのBCDフラグによって指定します。

③　例1

ジェネラル・レジスタとしてバンク0のロウ・アドレス0（0.00H-0.0FH）が指定されているとき（RPH = 0, RPL = 0）, 0.03H番地の内容に0.2FH番地の内容を加算した結果を0.03H番地に格納します。

（0.03H）←（0.03H）＋（0.2FH）

```
MEM003   MEM   0.03H
MEM02F   MEM   0.2FH
         MOV   BANK, #00H      ; データ・メモリのバンクを0
         MOV   RPH, #00H       ; ジェネラル・レジスタのバンクを0
         MOV   RPL, #00H       ; ジェネラル・レジスタのロウ・アドレスを0
         ADD   MEM003, ME02F
```

★ **例２**

ジェネラル・レジスタとしてバンク０のロウ・アドレス２（0.20H-0.2FH）が指定されているとき（RPH = 0, RPL = 4）, 0.23H番地の内容に0.2FH番地の内容を加算した結果を0.23H番地に格納します。

（0.23H）←（0.23H）＋（0.2FH）

```
MEM023   MEM   0.23H
MEM02F   MEM   0.2FH
         MOV   BANK, #00H      ; データ・メモリのバンクを０
         MOV   RPH, #00H       ; ジェネラル・レジスタのバンクを０注
         MOV   RPL, #04H       ; ジェネラル・レジスタのロウ・アドレスを２
         ADD   MEM023, MEM02F
```

注

レジスタ	RP							
	RPH				RPL			
ビット	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
データ	バンク							B
	0	0	0	0	ロウ・アドレス			C
								D

システム・レジスタ中のRP（ジェネラル・レジスタ・ポインタ）の割り当ては上の図のようになります。

したがって，ジェネラル・レジスタにバンク０とロウ・アドレス２を設定するためには，RPHに00Hを，RPLに04Hを格納しなければなりません。

この場合，BCDフラグをリセットしているので以降の算術演算は２進４ビット演算となります。

例３

0.03H番地の内容に0.6FH番地の内容を加算した結果を0.03H番地に格納します。このとき，IXE = 1, IXH = 0, IXM = 4, IXL = 0すなわちIX = 0.40Hなら，データ・メモリ・アドレスを2FHとすることでデータ・メモリの0.6FH番地を指定することができます。

（0.03H）←（0.03H）＋（0.6FH）

└ インデクス・レジスタの内容0.40Hとデータ・メモリのアドレス0.2FHをOR演算したアドレス

```
MEM003  MEM     0.03H
MEM02F  MEM     0.2FH
        MOV     RPH, #00H           ; ジェネラル・レジスタのバンクを0
        MOV     RPL, #00H           ; ジェネラル・レジスタのロウ・アドレスを0
        MOV     IXH, #00H           ; IX←00001000000B
        MOV     IXM, #04H           ;
        MOV     IXL, #00H           ;
        SET1    IXE                 ; IXEフラグ←1
        ADD     MEM003, MEM02F      ; IX                    00001000000B（0.40H）
                                    ; バンク・オペランド  OR）00000101111B（0.2FH）
                                    ; 指定アドレス          00001101111B（0.6FH）
```

★ **例4**

0.03H番地の内容に0.3FH番地の内容を加算した結果を0.03H番地に格納します。このとき，IXE = 1, IXH = 0, IXM = 1, IXL = 0すなわちIX = 0.10Hなら，データ・メモリ・アドレスを2FHとすることでデータ・メモリ0.3FHを指定することができます。

（0.03H）←（0.03H）＋（0.3FH）

└ インデクス・レジスタ0.10Hの内容とデータ・メモリのアドレス0.2FHをOR演算したアドレス

```
MEM003  MEM     0.03H
MEM02F  MEM     0.2FH
        MOV     BANK, #00H
        MOV     RPH, #00H           ; ジェネラル・レジスタのバンクを0
        MOV     RPL, #00H           ; ジェネラル・レジスタのロウ・アドレスを0
        MOV     IXH, #00H           ; IX←00000010000B（0.10H）注
        MOV     IXM, #01H
        MOV     IXL, #00H
        SET1    IXE                 ; IXEフラグ←1
        ADD     MEM003, MEM02F      ; IX                    00000010000B（0.10H）
                                    ; バンク・オペランド  OR）00000101111B（0.2FH）
                                    ; 指定アドレス          00100111111B（0.3FH）
```

注

レジスタ	IX											
	IXH				IXM				IXL			
ビット	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0	b_3	b_2	b_1	b_0
データ	MPE	0	0	0	0	ロウ・アドレス			カラム・アドレス			
		バンク										

システム・レジスタ中のIX（インデクス・レジスタ）の割り当ては前頁の図のようになります。したがって，IX = 0.10Hにするためには，IXHに00Hを，IXMに01Hを，IXLに00Hを格納しなければなりません。

この場合，MPE（メモリ・ポインタ・イネーブル）フラグをリセットしているので，ジェネラル・レジスタ間接転送のときのMP（メモリ・ポインタ）は無効になります。

④ 注　　意

ADD r, m命令の第１オペランドはジェネラル・レジスタのカラム・アドレスです。したがって，次のように書いた場合，ジェネラル・レジスタのカラム・アドレスは03Hになります。

```
MEM013   MEM    0.13H
MEM02F   MEM    0.2FH
         ADD    MEM013,  MEM02F
                  └ ジェネラル・レジスタのカラム・アドレスを意味し，下位４ビット（この場合は03H）が有効となります。
```

CMPフラグ = １のときは，加算結果が格納されません。

BCDフラグ = １のときは，BCD演算した結果が格納されます。

（２）　ADD m, #n4　　Add immediate data to data memory

① 命令コード

10	8 7	4 3	0
10000	m_R	m_C	n4

② 機　　能

CMP = 0のとき　　（m）←（m）＋n4

データ・メモリの内容にイミーディエト・データを加算し，結果をデータ・メモリへ格納します。

CMP = 1のとき　　（m）＋n4

結果はデータ・メモリに格納されず，キャリー・フラグCYとゼロ・フラグZが結果によって変化します。

加算の結果，桁上げがあればキャリー・フラグCYをセットし，桁上げがなければキャリー・フラグCYをリセットします。

加算の結果がゼロ以外になったときは，コンペア・フラグCMPに関係なくゼロ・フラグZはリセットされます。

コンペア・フラグがリセットされた状態（CMP = 0）で加算の結果がゼロになったときは，ゼロ・フラグZがセットされます。

コンペア・フラグがセットされた状態（CMP = 1）で加算の結果がゼロになったときは，ゼロ・フラグZは変化しません。

加算には２進４ビット演算とBCD演算の２種類があり，どちらの演算を行うかをPSWORDのBCDフラグによって指定します。

③ **例１**

0.2FH番地の内容に５を加算し，結果を0.2FH番地に格納します。

（0.2FH）←（0.2FH）＋5

```
MEM02F  MEM   0.2FH
        ADD   MEM02F, #05H
```

例２

0.6FH番地の内容に５を加算した結果を0.6FH番地に格納します。このとき，IXE = 1, IXH = 0, IXM = 4, IXL = 0すなわちIX = 0.40Hなら，データ・メモリ・アドレスを2FHとすることでデータ・メモリの0.6FH番地を指定することができます。

（0.6FH）←（0.6FH）＋05H

└インデクス・レジスタの内容0.40Hとデータ・メモリのアドレス0.2FHをOR演算したアドレス

```
MEM02F  MEM   0.2FH
        MOV   BANK, #00H      ; データ・メモリのバンクを０
        MOV   IXH, #00H       ; IX←00001000000B（0.40H）
        MOV   IXM, #04H
        MOV   IXL, #00H
        SET1  IXE             ; IXEフラグ←１
        ADD   MEM02F, #05H    ; IX                    00001000000B（0.40H）
                              ; バンク・オペランド  OR）00000101111B（0.2FH）
                              ; 指定アドレス          00001101111B（0.6FH）
```

★ **例３**

0.2FH番地の内容に５を加算した結果を0.2FH番地に格納します。このとき，IXE = 1, IXH = 0, IXM = 0, IXL = 0すなわちIX = 0.00Hなら，データ・メモリ・アドレスを2FHとすることでデータ・メモリの0.2FH番地を指定することができます。

（0.2FH）←（0.2FH）＋05H

└インデクス・レジスタの内容0.00Hとデータ・メモリのアドレス0.2FHをOR演算したアドレス

```
MEM02F  MEM   0.2FH
        MOV   BANK, #00H      ; データ・メモリのバンクを0
        MOV   IXH, #00H       ; IX←00000000000B
        MOV   IXM, #00H
        MOV   IXL, #00H
        SET1  IXE             ; IXEフラグ←1
        ADD   MEM02F, #05H    ; IX                    00000000000B（0.00H）
                              ; バンク・オペランド  OR）00000101111B（0.2FH）
                              ; 指定アドレス          00000101111B（0.2FH）
```

④ 注　　意

CMPフラグ＝1のときは，加算結果が格納されません。

BCDフラグ＝1のときは，BCD演算した結果が格納されます。

（3）ADDC r, m　　Add data memory to general register with carry flag

① 命令コード

10	8 7	4 3	0
00010	m_R	m_C	r

② 機　　能

CMP＝0のとき　　(r) ← (r) ＋ (m) ＋CY

ジェネラル・レジスタの内容にデータ・メモリの内容とキャリー・フラグCYの値を加算し，結果をrで示すジェネラル・レジスタへ格納します。

CMP＝1のとき　　(r) ＋ (m) ＋CY

結果はレジスタに格納されず，キャリー・フラグCYとゼロ・フラグZが結果によって変化します。

このADDC命令を使うことにより２ワード以上の加算が簡単にできます。

加算の結果，桁上げがあればキャリー・フラグCYをセットし，桁上げがなければキャリー・フラグCYをリセットします。

加算の結果がゼロ以外になったときは，コンペア・フラグCMPに関係なくゼロ・フラグZはリセットされます。

コンペア・フラグがリセットされた状態（CMP＝0）で加算の結果がゼロになったときは，ゼロ・フラグZがセットされます。

コンペア・フラグがセットされた状態（CMP＝1）で加算の結果がゼロになったときは，ゼロ・フラグZは変化しません。

加算には２進４ビット演算とBCD演算の２種類があり，どちらの演算を行うかをプログラム・ステータス・ワードPSWORDのBCDフラグによって指定します。

③ 例1

ジェネラル・レジスタとしてバンク0のロウ・アドレス0（0.00H-0.0FH）が指定されているとき，0.0DH番地から0.0FH番地の12ビットの内容に0.2DH番地から0.2FH番地の12ビットの内容を加算した結果を，0.0DH番地から0.0FH番地の12ビットに格納します。

（0.0FH）←（0.0FH）＋（0.2FH）
（0.0EH）←（0.0EH）＋（0.2EH）＋CY
（0.0DH）←（0.0DH）＋（0.2DH）＋CY

```
MEM00D  MEM   0.0DH
MEM00E  MEM   0.0EH
MEM00F  MEM   0.0FH
MEM02D  MEM   0.2DH
MEM02E  MEM   0.2EH
MEM02F  MEM   0.2FH
        MOV   BANK,   #00H     ; データ・メモリのバンクを0
        MOV   RPH,    #00H     ; ジェネラル・レジスタのバンクを0
        MOV   RPL,    #00H     ; ジェネラル・レジスタのロウ・アドレスを0
        ADD   MEM00F, MEM02F
        ADDC  MEM00E, MEM02E
        ADDC  MEM00D, MEM02D
```

★ 例2

ジェネラル・レジスタとしてバンク0のロウ・アドレス2（0.20H-0.2FH）が指定されているとき，0.2DH番地から0.2FH番地の12ビットの内容をキャリー・フラグも含めて1ビット左にシフトします。

```
MEM00D  MEM   0.0DH
MEM00E  MEM   0.0EH
MEM00F  MEM   0.0FH
MEM02D  MEM   0.2DH
MEM02E  MEM   0.2EH
MEM02F  MEM   0.2FH
        MOV   RPH, #00H        ; ジェネラル・レジスタのバンクを0
        MOV   RPL, #04H        ; ジェネラル・レジスタのロウ・アドレスを2
        MOV   BANK, #00H       ; データ・メモリのバンクを0
        ADDC  MEM00F, MEM02F
        ADDC  MEM00E, MEM02E
        ADDC  MEM00D, MEM02D
```

例3

0.0FH番地の内容と0.40H番地から0.4FH番地の内容を加算し，結果を0.0FH番地へ格納します。

（0.0FH）←（0.0FH）＋（0.40H）＋（0.41H）＋…＋（0.4FH）

```
MEM00F  MEM   0.0FH
MEM000  MEM   0.00H
        MOV   BANK, #00H      ; データ・メモリのバンクを0
        MOV   RPH, #00H       ; ジェネラル・レジスタのバンクを0
        MOV   RPL, #00H       ; ジェネラル・レジスタのロウ・アドレスを0
        MOV   IXH, #00H       ; IX←00001000000B（0.40H）
        MOV   IXM, #04H
        MOV   IXL, #00H
LOOP1：
        SET1  IXE             ; IXEフラグ←1
        ADD   MEM00F, MEM000
        CLR1  IXE             ; IXEフラグ←0
        INC   IX              ; IX←IX＋1
        SKE   IXL, #0
        JMP   LOOP1
```

★ **例4**

0.0DH番地から0.0FH番地の12ビットの内容に，0.40H番地から0.42H番地の12ビットの内容を加算した結果を，0.0DH番地から0.0FH番地の12ビットに格納します。

（0.0DH）←（0.0DH）＋（0.40H）

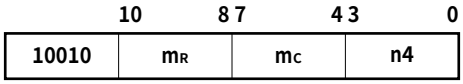
（0.0EH）←（0.0EH）＋（0.41H）＋CY

（0.0FH）←（0.0FH）＋（0.42H）＋CY

```
MEM000  MEM   0.00H
MEM001  MEM   0.01H
MEM002  MEM   0.02H
MEM00D  MEM   0.0DH
MEM00E  MEM   0.0EH
MEM00F  MEM   0.0FH
        MOV   BANK, #00H      ; データ・メモリのバンクを0
        MOV   RPH, #00H       ; ジェネラル・レジスタのバンクを0
        MOV   RPL, #00H       ; ジェネラル・レジスタのロウ・アドレスを0
        MOV   IXH, #00H       ; IX←00001000000（0.40H）
        MOV   IXM, #04H
        MOV   IXL, #00H
        SET1  IXE             ; IXEフラグ←1
        ADD   MEM00D, MEM000  ; （0.0DH）←（0.0DH）＋（0.40H）
        ADDC  MEM00E, MEM001  ; （0.0EH）←（0.0EH）＋（0.41H）
        ADDC  MEM00F, MEM002  ; （0.0FH）←（0.0FH）＋（0.42H）
```

（4）ADDC m, #n4　　Add immediate data to data memory with carry flag

①　命令コード



②　機　　能

CMP = 0のとき　　（m）←（m）＋n4＋CY

　データ・メモリの内容にイミーディエト・データとキャリー・フラグ（CY）の値を加算し，結果をデータ・メモリへ格納します。

CMP = 1のとき　　（m）＋n4＋CY

　結果はデータ・メモリに格納されず，キャリー・フラグCYとゼロ・フラグZが結果によって変化します。

　加算の結果，桁上げがあればキャリー・フラグCYをセットし，桁上げがなければキャリー・フラグCYをリセットします。

　加算の結果がゼロ以外になったときは，コンペア・フラグCMPに関係なくゼロ・フラグZはリセットされます。

　コンペア・フラグがリセットされた状態（CMP = 0）で加算の結果がゼロになったときは，ゼロ・フラグZがセットされます。

　コンペア・フラグがセットされた状態（CMP = 1）で加算の結果がゼロになったときは，ゼロ・フラグZは変化しません。

　加算には２進演算とBCD演算の２種類があり，どちらの演算を行うかをPSWORDのBCDフラグによって指定します。

③　例１

　0.0DH番地から0.0FH番地の12ビットの内容に５を加算した結果を0.0DH番地から0.0FH番地に格納します。

（0.0FH）←（0.0FH）＋05H
（0.0EH）←（0.0EH）＋CY
（0.0DH）←（0.0DH）＋CY

```
MEM00D  MEM   0.0DH
MEM00E  MEM   0.0EH
MEM00F  MEM   0.0FH
        MOV   BANK, #00H        ;データ・メモリのバンクを０
        ADD   MEM00F, #05H
        ADDC  MEM00E, #00H
        ADDC  MEM00D, #00H
```

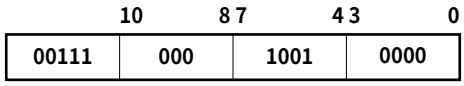
例２

0.4DH番地から0.4FH番地の12ビットの内容に５を加算した結果を0.4DH番地から0.4FH番地に格納します。

```
                （0.4FH）←（0.4FH）＋05H
                （0.4EH）←（0.4EH）＋CY
                （0.4DH）←（0.4DH）＋CY
MEM00D  MEM     0.0DH
MEM00E  MEM     0.0EH
MEM00F  MEM     0.0FH
        MOV     BANK, ＃00H         ; データ・メモリのバンクを０
        MOV     IXH, ＃00H          ; IX←00001000000B（0.40H）
        MOV     IXM, ＃04H
        MOV     IXL, ＃00H
        SET1    IXE                 ; IXEフラグ←１
        ADD     MEM00F, ＃5         ; （0.4FH）←（0.4FH）＋5H
        ADDC    MEM00E, ＃0         ; （0.4EH）←（0.4EH）＋CY
        ADDC    MEM00D, ＃0         ; （0.4DH）←（0.4DH）＋CY
```

（5）INC AR　　Increment address register

①　命令コード



②　機　　能

AR←AR＋1

アドレス・レジスタARの内容をインクリメントします。

③　例１

システム・レジスタ内のAR3からAR0（アドレス・レジスタ）の16ビットの内容に１を加算した結果をAR3からAR0に格納します。

```
AR0←AR0＋1
AR1←AR1＋CY
AR2←AR2＋CY
AR3←AR3＋CY
```

```
INC AR
```

また，この命令を加算命令を用いて行うと次のようになります。

```
ADD     AR0, ＃01H
ADDC    AR1, ＃00H
ADDC    AR2, ＃00H
ADDC    AR3, ＃00H
```

例２

テーブル参照命令（詳細については，**10.4　データ・バッファとテーブル参照**を参照してください）を用いてテーブル・データを16ビット（１アドレス）ごとにDBF（データ・バッファ）に転送します。

```
；アドレス              テーブル・データ
            ORG         10H
            DW          0F3FFH
            DW          0A123H
            DW          0FFF1H
            DW          0FFF5H
            DW          0FF11H
            ⋮
            ⋮
            ⋮
            MOV         AR3, ＃0H           ；テーブル・データのアドレス
```

```
                MOV     AR2, #0H        ; 0010Hをアドレス・レジスタに設
                MOV     AR1, #1H        ; 定します
                MOV     AR0, #0H
        LOOP：
                MOVT    DBF, @AR        ; テーブル・データがDBFに読み込
                ┆                       ; まれます
                                        ; テーブル・データを参照する処理
                INC     AR              ; アドレス・レジスタを１インクリ
                BR      LOOP            ; メント
```

④ 注　　意

アドレス・レジスタ（AR0-AR3）は，品種により使用できるビット数が異なりますので，使用する際は，各品種のデータ・シートを参照してください。

（6）INC IX　　Increment indexregister

① 命令コード

10	8 7	4 3	0
00111	000	1000	0000

② 機　　能

IX←IX＋1

インデクス・レジスタIXの内容をインクリメントします。

③ 例１

システム・レジスタ内のIXH, IXM, IXL（インデクス・レジスタ）の計12ビットの内容に１を加算した結果をIXH, IXM, IXLに格納します。

IXL←IXL＋1

IXM←IXM＋CY

IXH←IXH＋CY

INC IX

この演算を加算命令を用いて行うと次のようになります。

```
ADD     IXL, #01H
ADDC    IXM, #00H
ADDC    IXH, #00H
```

例２

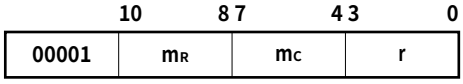
インデクス・レジスタを用いてデータ・メモリ0.00H-0.73Hの内容をすべて“０”にします。

```
            MOV    IXH, #00H        ; インデクス・レジスタの内容をバンク０の00Hに
                                    ; 設定します。
            MOV    IXM, #00H        ;
            MOV    IXL, #00H
  RAMクリア：
  MEM000    MEM    0.00H
            SET1   IXE              ; IXEフラグ←１
            MOV    MEM000, #00H     ; インデクス・レジスタで示されるデータ・メモリに
                                    ; ０を書き込みます
            CLR1   IXE              ; IXEフラグ←０
            INC    IX
            SET2   CMP, Z           ; CMPフラグ←１，Zフラグ←１
            SUB    IXL, #03H        ; インデクス・レジスタの内容がバンク０の73Hに
            SUBC   IXM, #07H        ; なったかどうかをチェックします
            SUBC   IXH, #00H        ;
            SKT1   Z                ; インデクス・レジスタの内容がバンク０の73Hにな
            BR     RAMクリア        ; るまでループします
```

15.5.2　減算命令

（1）SUB r, m　　**Subtract data memory from general register**

① 命令コード



② 機　　能

CMP = 0のとき　　（r）←（r）－（m）

ジェネラル・レジスタの内容からデータ・メモリの内容を減算し，結果をジェネラル・レジスタへ格納します。

CMP = 1のとき　　（r）－（m）

結果はレジスタに格納されず，キャリー・フラグCYとゼロ・フラグZが結果によって変化します。

減算の結果，ボローがあればキャリー・フラグCYをセットし，ボローがなければキャリー・フラグCYをリセットします。

減算の結果がゼロ以外になったときは，コンペア・フラグCMPに関係なくゼロ・フラグZはリセットされます。

コンペア・フラグがリセットされた状態（CMP = 0）で減算の結果がゼロになったときは，ゼロ・フラグZがセットされます。

コンペア・フラグがセットされた状態（CMP = 1）で減算の結果がゼロになったときは，ゼロ・フラグZは変化しません。

減算には２進４ビット演算とBCD演算の２種類があり，どちらの演算を行うかをプログラム・ステータス・ワードPSWORDのBCDフラグによって指定します。

③ 例１

ジェネラル・レジスタとしてバンク０のロウ・アドレス０（0.00H-0.0FH）が指定されているとき（RPH = 0, RPL = 0），0.03H番地の内容から0.2FH番地の内容を減算した結果を0.03H番地に格納します。

（0.03H）←（0.03H）＋（0.2FH）

```
MEM003   MEM    0.03H
MEM02F   MEM    0.2FH
         SUB    MEM003, MEM02F
```

★　**例２**

ジェネラル・レジスタとしてバンク０のロウ・アドレス２（0.20H-0.2FH）が指定されているとき（RPH = 0, RPL = 4），0.23H番地の内容から0.2FH番地の内容を減算した結果を0.23H番地に格納します。

（0.23H）←（0.23H）－（0.2FH）

```
MEM023   MEM    0.23H
MEM02F   MEM    0.2FH
         MOV    BANK, #00H        ; データ・メモリのバンクを０
         MOV    RPH, #00H         ; ジェネラル・レジスタのバンクを０
         MOV    RPL, #04H         ; ジェネラル・レジスタのロウ・アドレスを２
         SUB    MEM023, MEM02F
```

例３

0.03H番地の内容から0.6FH番地の内容を減算した結果を0.03H番地に格納します。このとき，IXE = 1, IXH = 0, IXM = 4, IXL = 0すなわちIX = 0.40Hなら，データ・メモリ・アドレスを2FHとすることでデータ・メモリの0.6FH番地を指定することができます。

（0.03H）←（0.03H）＋（0.6FH）

```
MEM003   MEM    0.03H
MEM02F   MEM    0.2FH
         MOV    BANK, #00H        ; データ・メモリのバンクを０
         MOV    RPH, #00H         ; ジェネラル・レジスタのバンクを０
         MOV    RPL, #00H         ; ジェネラル・レジスタのロウ・アドレスを０
         MOV    IXH, #00H         ; IX←00001000000B（0.40H）
         MOV    IXM, #04H         ;
         MOV    IXL, #00H         ;
         SET1   IXE               ; IXEフラグ←１
         SUB    MEM003, MEM02F    ; IX                    00001000000B（0.40H）
                                  ; バンク・オペランド　OR）00000101111B（0.2FH）
                                  ; 指定アドレス          00001101111B（0.6FH）
```

★　**例４**

0.03H番地の内容から0.3FH番地の内容を減算した結果を0.03H番地に格納します。このとき，IXE = 1, IXH = 0, IXM = 1, IXL = 0すなわちIX = 0.10Hなら，データ・メモリ・アドレスを2FHとすることでデータ・メモリの0.3FH番地を指定することができます。

（0.03H）←（0.03H）＋（0.3FH）

```
MEM003   MEM    0.03H
MEM02F   MEM    0.2FH
```

```
MOV    BANK, #00H        ; データ・メモリのバンクを0
MOV    RPH, #00H         ; ジェネラル・レジスタのバンクを0
MOV    RPL, #00H         ; ジェネラル・レジスタのロウ・アドレスを0
MOV    IXH, #00H         ; IX←00000010000B（0.10H）
MOV    IXM, #01H         ;
MOV    IXL, #00H         ;
SET1   IXE               ; IXEフラグ←1
SUB    MEM003, MEM02F    ; IX                    00000010000B（0.10H）
                         ; バンク・オペランド  OR）00000101111B（0.2FH）
                         ; 指定アドレス          00000111111B（0.3FH）
```

④ 注　意

SUB r, m命令の第１オペランドはジェネラル・レジスタ・アドレスでなければなりません。したがって，次のように書くと03H番地がレジスタとして指定されます。

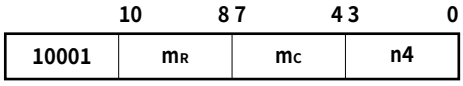
```
MEM013   MEM    0.13H
MEM02F   MEM    0.2FH
         SUB    MEM013, MEM02F
                  └── ジェネラル・レジスタ・アドレスは00H-0FHの範囲（レジスタ・ポインタをロウ・アドレス１以外に設定）でなければなりません。
```

CMPフラグ＝１のときは，減算結果が格納されません。

BCDフラグ＝１のときは，BCD演算した結果が格納されます。

（2）SUB m, #n4　　Subtract immediate data from data memory

① 命令コード



② 機　能

CMP＝0のとき　　（m）←（m）－n4

データ・メモリの内容からイミーディエト・データを減算し，結果をデータ・メモリへ格納します。

CMP＝1のとき　　（m）－n4

結果はデータ・メモリに格納されず，キャリー・フラグCYとゼロ・フラグZが結果によって変化します。

減算の結果，ボローがあればキャリー・フラグCYをセットし，ボローがなければキャリー・フラグCYをリセットします。

減算の結果がゼロ以外になったときは，コンペア・フラグCMPに関係なくゼロ・フラグZはリセットされます。

コンペア・フラグがリセットされた状態（CMP＝0）で減算の結果がゼロになったときは，ゼロ・フラグZがセットされます。

コンペア・フラグがセットされた状態（CMP＝1）で減算の結果がゼロになったときは，ゼロ・フラグZは変化しません。

減算には２進４ビット演算とBCD演算の２種類があり，どちらの演算を行うかをプログラム・ステータス・ワードPSWORDのBCDフラグによって指定します。

③ **例１**

0.2FH番地の内容から５を減算し，結果を0.2FH番地に格納します。

（0.2FH）←（0.2FH）－5

```
MEM02F  MEM   0.2FH
        SUB   MEM02F, #05H
```

例２

0.6FH番地の内容から５を減算した結果を0.6FH番地に格納します。このとき，IXE = 1, IXH = 0, IXM = 4, IXL = 0すなわちIX = 0.40Hなら，データ・メモリ・アドレスを2FHとすることでデータ・メモリの0.6FH番地を指定することができます。

（0.6FH）←（0.6FH）－5

└インデクス・レジスタの内容0.40Hとデータ・メモリのアドレス0.2FHをOR演算したアドレス

```
MEM02F  MEM   0.2FH
        MOV   BANK, #00H      ; データ・メモリのバンクを０
        MOV   IXH, #00H       ; IX←00001000000B（0.40H）
        MOV   IXM, #04H       ;
        MOV   IXL, #00H       ;
        SET1  IXE             ; IXEフラグ←１
        SUB   MEM02F, #05H    ; IX                    00001000000B（0.40H）
                              ; バンク・オペランド  OR）00000101111B（0.2FH）
                              ; 指定アドレス          00001101111B（0.6FH）
```

★ **例３**

0.2FH番地の内容から５を減算した結果を0.2FH番地に格納します。このとき，IXE = 1, IXH = 0, IXM = 0, IXL = 0すなわちIX = 0.00Hなら，データ・メモリ・アドレスを2FHとすることでデータ・メモリの0.2FH番地を指定することができます。

（0.2FH）←（0.2FH）－5

└インデクス・レジスタの内容0.00Hとデータ・メモリのアドレス0.2FHをOR演算したアドレス

```
MEM02F  MEM   0.2FH
        MOV   BANK0, #00H     ; データ・メモリのバンクを０
        MOV   IXH, #00H       ; IX←00000000000B（0.00H）
        MOV   IXM, #00H       ;
        MOV   IXL, #00H       ;
        SET1  IXE             ; IXEフラグ←１
        SUB   MEM02F, #05H    ; IX                    00000000000B（0.00H）
                              ; バンク・オペランド  OR）00000101111B（0.2FH）
                              ; 指定アドレス          00000101111B（0.2FH）
```

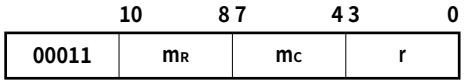
④ 注　　意

CMPフラグ＝1のときは，減算結果が格納されません。

BCDフラグ＝1のときは，BCD演算した結果が格納されます。

（3）SUBC r, m　　Subtract data memory from general register with carry flag

① 命令コード



② 機　　能

CMP＝0のとき　　(r) ← (r) － (m) －CY

ジェネラル・レジスタの内容からデータ・メモリの内容とキャリー・フラグCYの値を減算し，結果をジェネラル・レジスタへ格納します。このSUBC命令を使うことにより２ワード以上の減算が簡単にできます。

CMP＝1のとき　　(r) － (m) －CY

結果はレジスタに格納されず，キャリー・フラグCYとゼロ・フラグZが結果によって変化します。

減算の結果，ボローがあればキャリー・フラグCYをセットし，ボローがなければキャリー・フラグCYをリセットします。

減算の結果がゼロ以外になったときは，コンペア・フラグCMPに関係なくゼロ・フラグZはリセットされます。

コンペア・フラグがリセットされた状態（CMP＝0）で減算の結果がゼロになったときは，ゼロ・フラグZがセットされます。

コンペア・フラグがセットされた状態（CMP＝1）で減算の結果がゼロになったときは，ゼロ・フラグZは変化しません。

減算には２進４ビット演算とBCD演算の２種類があり，どちらの演算を行うかをプログラム・ステータス・ワードPSWORDのBCDフラグによって指定します。

③ 例1

ジェネラル・レジスタとしてバンク0のロウ・アドレス0（0.00H-0.0FH）が指定されているとき，0.0DH番地から0.0FH番地の12ビットの内容から，0.2DH番地から0.2FH番地の12ビットの内容を減算し，結果を0.0DH番地から0.0FH番地の12ビットに格納します。

（0.0FH）←（0.0FH）－（0.2FH）
（0.0EH）←（0.0EH）－（0.2EH）－CY
（0.0DH）←（0.0DH）＋（0.2DH）－CY

```
MEM00D  MEM   0.0DH
MEM00E  MEM   0.0EH
MEM00F  MEM   0.0FH
MEM02D  MEM   0.2DH
MEM02E  MEM   0.2EH
MEM02F  MEM   0.2FH
        SUB   MEM00F, MEM02F
        SUBC  MEM00E, MEM02E
        SUBC  MEM00D, MEM02D
```

例2

0.0DH番地から0.0FH番地の12ビットの内容から，0.40H番地から0.42H番地の12ビットの内容を減算した結果を0.0DH番地から0.0FH番地の12ビットに格納します。

（0.0DH）←（0.0DH）－（0.40H）
（0.0EH）←（0.0EH）－（0.41H）－CY
（0.0FH）←（0.0FH）＋（0.42H）－CY

```
MEM000  MEM   0.00H
MEM001  MEM   0.01H
MEM002  MEM   0.02H
MEM00D  MEM   0.0DH
MEM00E  MEM   0.0EH
MEM00F  MEM   0.0FH
        MOV   BANK, #00H        ; データ・メモリのバンクを0
        MOV   RPH, #00H         ; ジェネラル・レジスタのバンクを0
        MOV   RPL, #00H         ; ジェネラル・レジスタのロウ・アドレスを0
        MOV   IXH, #00H         ; IX←00001000000B（0.40H）
        MOV   IXM, #04H         ;
        MOV   IXL, #00H         ;
        SET1  IXE               ; IXEフラグ←1
        SUB   MEM00D, MEM000    ; （0.0DH）←（0.0DH）－（0.40H）
        SUBC  MEM00E, MEM001    ; （0.0EH）←（0.0EH）－（0.41H）
        SUBC  MEM00F, MEM002    ; （0.0FH）←（0.0FH）－（0.42H）
```

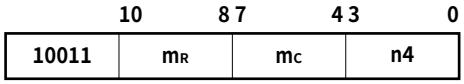
例３

0.00H番地から0.03H番地の12ビットの内容と0.0CH番地から0.0FH番地の12ビットの内容を比較し，同一であればLAB1へジャンプし，異なればLAB2へジャンプします。

```
MEM000   MEM    0.00H
MEM001   MEM    0.01H
MEM002   MEM    0.02H
MEM003   MEM    0.03H
MEM00C   MEM    0.0CH
MEM00D   MEM    0.0DH
MEM00E   MEM    0.0EH
MEM00F   MEM    0.0FH
         SET2   CMP, Z              ; CMPフラグ←１，Zフラグ←１
         SUB    MEM000, MEM00C      ; CMPフラグがセットされているため0.00H-0.03H
         SUBC   MEM001, MEM00D      ; 番地の内容は変化しません
         SUBC   MEM002, MEM00E      ;
         SUBC   MEM003, MEM00F      ;
         SKF1   Z                   ; 比較した結果，同一であればZフラグ＝１，異なっ
         BR     LAB1                ; ていればZフラグ＝０となります
         BR     LAB2
                :
LAB1：   :
LAB2：   :
                :
```

（4）SUBC m, #n4　　Subtract immediate data from data memory with carry flag

① 命令コード



② 機　　能

CMP = 0のとき　　（m）←（m）－n4－CY

　データ・メモリの内容からイミーディエト・データとキャリー・フラグCYの値を減算し，結果をデータ・メモリへ格納します。

CMP = 1のとき　　（m）－n4－CY

　結果はデータ・メモリに格納されず，キャリー・フラグCYとゼロ・フラグZが結果によって変化します。

　減算の結果，ボローがあればキャリー・フラグCYをセットし，ボローがなければキャリー・フラグCYをリセットします。

　減算の結果がゼロ以外になったときは，コンペア・フラグCMPに関係なくゼロ・フラグZはリセットされます。

　コンペア・フラグがリセットされた状態（CMP = 0）で減算の結果がゼロになったときは，ゼロ・フラグZがセットされます。

　コンペア・フラグがセットされた状態（CMP = 1）で減算の結果がゼロになったときは，ゼロ・フラグZは変化しません。

　減算には２進演算とBCD演算の２種類があり，どちらの演算を行うかをプログラム・ステータス・ワードPSWORDのBCDフラグによって指定します。

③ 例１

　0.0DH番地から0.0FH番地の12ビットの内容から５を減算した結果を，0.0DH番地から0.0FH番地に格納します。

```
                （0.0FH）←（0.0FH）－05H
                （0.0EH）←（0.0EH）－CY
                （0.0DH）←（0.0DH）－CY
MEM00D  MEM     0.0DH
MEM00E  MEM     0.0EH
MEM00F  MEM     0.0FH
        SUB     MEM00F, #05H
        SUBC    MEM00E, #00H
        SUBC    MEM00D, #00H
```

例２

0.4DH番地から0.4FH番地の12ビットの内容から５を減算した結果を0.4DH番地から0.4FH番地に格納します。

（0.4FH）←（0.4FH）－05H
（0.4EH）←（0.4EH）－CY
（0.4DH）←（0.4DH）－CY

```
MEM00D  MEM   0.0DH
MEM00E  MEM   0.0EH
MEM00F  MEM   0.0FH
        MOV   BANK, #00H      ; データ・メモリのバンクを０
        MOV   IXH, #00H       ; IX←00001000000B（0.40H）
        MOV   IXM, #04H       ;
        MOV   IXL, #00H       ;
        SET1  IXE             ; IXEフラグ←１
        SUB   MEM00F, #5      ; （0.4FH）←（0.4FH）－5
        SUBC  MEM00E, #0      ; （0.4EH）←（0.4EH）－CY
        SUBC  MEM00D, #0      ; （0.4DH）←（0.4DH）－CY
```

例３

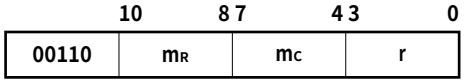
0.00H番地から0.03H番地の12ビットの内容とイミーディエト・データの0A3FHを比較し，同一であればLAB1へジャンプし，異なればLAB2へジャンプします。

```
MEM000  MEM   0.00H
MEM001  MEM   0.01H
MEM002  MEM   0.02H
MEM003  MEM   0.03H
        SET2  CMP, Z          ; CMPフラグ←１，Zフラグ←１
        SUB   MEM000, #0H     ; CMPフラグがセットされているため0.00H-0.03H
        SUBC  MEM001, #0AH    ; 番地の内容は変化しません
        SUBC  MEM002, #3H     ;
        SUBC  MEM003, #0FH    ;
        SKF1  Z               ; 比較した結果，同一であればZフラグ＝１，異なっ
        BR    LAB1            ; ていればZフラグ＝０となります
        BR    LAB2
                :
LAB1：          :
                :
LAB2：          :
                :
```

15.5.3　論理演算命令

（1）OR r, m　　**OR between general register and data memory**

① 命令コード



② 機　　能

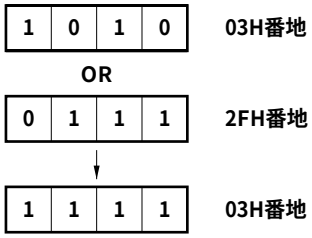
（r）←（r）∨（m）

ジェネラル・レジスタの内容とデータ・メモリの内容との論理和（OR）の結果をジェネラル・レジスタへ格納します。

③ 例1

0.03H番地の内容（1010B）と0.2FH番地の内容（0111B）をOR演算した結果（1111B）を0.03H番地へ格納します。

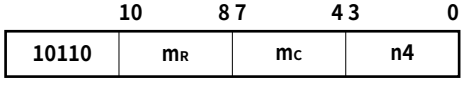
（0.03H）←（0.03H）∨（0.2FH）




```
MEM003  MEM   0.03H
MEM02F  MEM   0.2FH
        MOV   MEM003, #1010B
        MOV   MEM02F, #0111B
        OR    MEM003, MEM02F
```

（2）OR m, #n4　　**OR between data memory and immediate data**

① 命令コード



② 機　　能

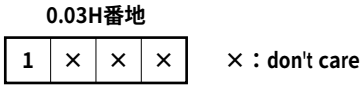
（m）←（m）∨n4

データ・メモリの内容とイミーディエト・データとの論理和（OR）の結果をデータ・メモリへ格納します。

③ 例１

0.03H番地のビット３（MSB）をセットします。

（0.03H）←（0.03H）∨1000B




```
MEM003   MEM    0.03H
         OR     MEM003, #1000B
```

例２

0.03H番地のすべてのビットをセットします。

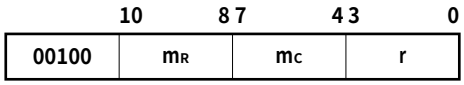
```
MEM003   MEM    0.03H
         OR     MEM003, #1111B
```

または

```
MEM003   MEM    0.03H
         MOV    MEM003, #0FH
```

（３）AND r, m　　AND between general register and data memory

① 命令コード



② 機　　能

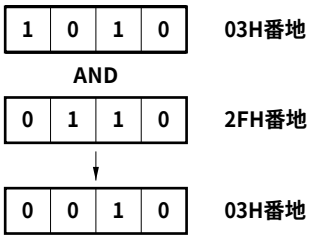
（r）←（r）∧（m）

ジェネラル・レジスタの内容とデータ・メモリの内容との論理積（AND）の結果をジェネラル・レジスタへ格納します。

③ 例１

0.03H番地の内容（1010B）と0.2FH番地の内容（0110B）をAND演算した結果（0010B）を0.03H番地へ格納します。

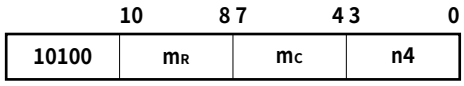
（0.03H）←（0.03H）∧（0.2FH）




```
MEM003   MEM    0.03H
MEM02F   MEM    0.2FH
         MOV    MEM003, #1010B
         MOV    MEM02F, #0110B
         AND    MEM003, MEM02F
```

（4）AND m, #n4　　AND between data memory and immediate data

① 命令コード



② 機　　能

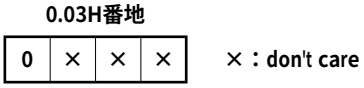
（m）←（m）∧n4

データ・メモリの内容とイミーディエト・データとの論理積（AND）の結果をデータ・メモリへ格納します。

③ 例1

0.03H番地のビット3（MSB）をリセットします。

（0.03H）←（0.03H）∧0111B




```
MEM003   MEM    0.03H
         AND    MEM003, #0111B
```

例2

0.03H番地のすべてのビットをリセットします。

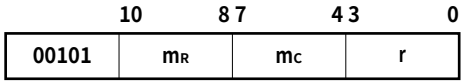
```
MEM003   MEM    0.03H
AND      MEM003, #0000B
```

または

```
MEM003   MEM    0.03H
         MOV    MEM003, #00H
```

（5）XOR r, m　　Exclusive OR between general register and data memory

① 命令コード



② 機　　能

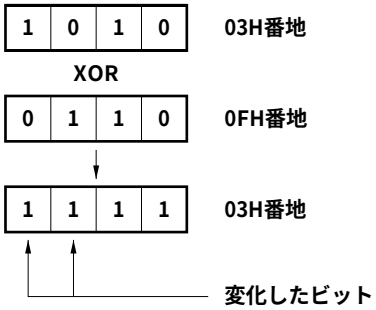
(r) ← (r)　　(m)

ジェネラル・レジスタの内容とデータ・メモリの内容との排他的論理和（XOR）の結果をジェネラル・レジスタへ格納します。

③ 例1

0.03H番地の内容と0.0FH番地の内容を比較した結果，異なるビットをセットして0.03H番地へ格納し，0.03H番地がすべてリセット（0.03H番地と0.0FH番地の内容が同じ）されていればLBL1へジャンプし，それ以外であればLBL2へジャンプします。

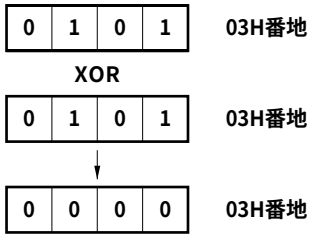
この例はオルタネート・スイッチの状態（0.03H番地の内容）と内部状態（0.0FH番地の内容）を比較し，変化したスイッチの処理へと分岐するときなどの例です。




```
MEM003  MEM   0.03H
MEM00F  MEM   0.0FH
        XOR   MEM003, MEM00F
        SKNE  MEM003, #00H
        BR    LBL1
        BR    LBL2
```

例２

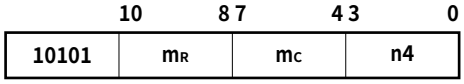
0.03H番地の内容をクリアします。




```
MEM003   MEM    0.03H
         XOR    MEM003, MEM003
```

（6）XOR m, #n4　　Exclusive OR between data memory and immediate data

① 命令コード



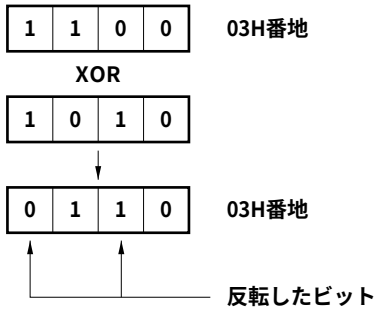
② 機　　能

（m）←（m）∀ n4

データ・メモリの内容とイミーディエト・データとの排他的論理和（XOR）の結果をデータ・メモリへ格納します。

③ 例

0.03H番地のビット１とビット３を反転して03H番地へ格納します。




```
MEM003   MEM    0.03H
         XOR    MEM003, #1010B
```

15.5.4　判断命令

（1）SKT m, #n　　Skip next instruction if data memory bits are true

①　命令コード

	10	8 7	4 3	0
11110	m_R	m_C	n	

②　機　　能

CMP←0, if（m）∧n = n, then skip

★　データ・メモリの内容とイミーディエト・データnの論理積の結果がnと等しければ次の１命令をスキップします（NOP命令として実行します）。

③　例１

03H番地のビット０が“１”ならばAAAへジャンプし，“０”ならばBBBへジャンプします。

```
SKT     03H, #0001B
BR      BBB
BR      AAA
```

例２

03H番地のビット０とビット１がともに“１”ならば次の命令をスキップします。

```
SKT     03H, #0011B
```

	b_3	b_2	b_1	b_0	
スキップ条件　03H	×	×	1	1	×：don't care

例３

次の２つの命令の実行結果は同じです。

```
SKT     13H, #1111B
SKE     13H, #0FH
```

（2）SKF m, #n　　Skip next instruction if data memory bits are false

① 命令コード

10	8 7	4 3	0
11111	m_R	m_C	n

② 機　　能

CMP←0, if（m）∧n = 0, then skip

データ・メモリの内容とイミーディエト・データnの論理積の結果が0のとき，次の1命令をスキップします（NOP命令として実行します）。

③ 例1

13H番地のビット2が“0”ならばデータ・メモリの0FH番地の内容にイミーディエト・データの00Hを格納し，“1”ならばABCへジャンプします。

```
MEM013   MEM    0.13H
MEM00F   MEM    0.0FH
         SKF    MEM013, #0100B
         BR     ABC
         MOV    MEM00F, #00H
```

例2

29H番地のビット3とビット0がともに“0”ならば次の命令をスキップします。

```
SKF      29H, #1001B
```

	b_3	b_2	b_1	b_0	
スキップ条件　29H	0	×	×	0	×：don't care

例3

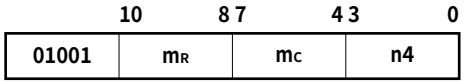
次の2つの命令の実行結果は同じです。

```
SKF      34H, #1111B
SKE      34H, #00H
```

15.5.5　比較命令

（1）SKE m, #n4　　Skip if data memory equal to immediate data

① 命令コード



★ ② 機　　能

（m）－n4, skip if zero

データ・メモリの内容がイミーディエト・データの値と等しいとき，次に続く１命令をスキップします（NOP命令として実行します）。

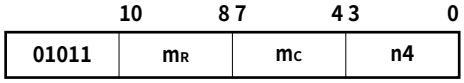
③ 例

24H番地の内容が０ならば24H番地に0FHを転送し，０でなければOPE1へジャンプします。

```
MEM024  MEM   0.24H
        SKE   MEM024, #00H
        BR    OPE1
        MOV   MEM024, #0FH
OPE1：
```

（2）SKNE m, #n4　　Skip if data memory not equal to immediate data

① 命令コード



★ ② 機　　能

（m）－n4, skip if not zero

データ・メモリの内容がイミーディエト・データの値と異なるとき，次に続く１命令をスキップします（NOP命令として実行します）。

③ 例

1FH番地の内容が１で，かつ1EH番地の内容が３ならばXYZへジャンプし，そうでなければABCへジャンプします。

８ビットの比較をする場合は次のように組み合わせて使います。

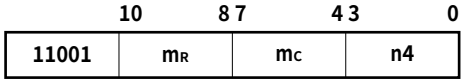
```
MEM01E  MEM   0.1EH
MEM01F  MEM   0.1FH
        SKNE  MEM01F, #01H
        SKE   MEM01E, #03H
        BR    ABC
        BR    XYZ
```

また，コンペア・フラグ，ゼロ・フラグを用いて上記の内容と同じことを行う場合は，次のようになります。

```
MEM01E  MEM   0.1EH
MEM01F  MEM   0.1FH
        SET2  CMP, Z            ; CMPフラグ←1，Zフラグ←1
        SUB   MEM01F, #01H
        SUB   MEM01E, #03H
        SKT1  Z
        BR    ABC
        BR    XYZ
```

（3）SKGE m, #n4　　Skip if data memory greater than or equal to immediate data

① 命令コード



② 機　能

（m）－n4, skip if not borrow

データ・メモリの内容がイミーディエト・データの値以上のとき，次に続く1命令をスキップします（NOP命令として実行します）。

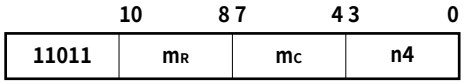
③ 例

1FH番地（上位）および2FH番地（下位）に格納されている8ビット・データがイミーディエト・データ17Hより大きければRETし，そうでなければRETSKするプログラム例。

```
MEM01F  MEM   0.1FH
MEM02F  MEM   0.2FH
        SKGE  MEM01F, #1
        RETSK
        SKNE  MEM01F, #1
        SKLT  MEM02F, #8        ; 7+1
        RET
        RETSK
```

（4）SKLT m, #n4　　Skip if data memory less than immediate data

①　命令コード



★　②　機　　能

（m）－n4, skip if borrow

データ・メモリの内容がイミーディエト・データの値未満のとき，次に続く１命令をスキップします（NOP命令として実行します）。

③　例

10H番地の内容がイミーディエト・データ“６”以上であれば，0FH番地の内容に01Hを格納し，小さければ0FH番地の内容に02Hを格納するプログラムです。

```
MEM00F   MEM    0.0FH
MEM010   MEM    0.10H
         MOV    MEM00F, #02H
         SKLT   MEM010, #06H
         MOV    MEM00F, #01H
```

15.5.6　回転命令

（1）RORC r　　Rotate right general register with carry flag

①　命令コード

			3	0
00111	000	0111	r	

②　機　　能

CY → (r)$_{b3}$ → (r)$_{b2}$ → (r)$_{b1}$ → (r)$_{b0}$

rで示すジェネラル・レジスタの内容をキャリー・フラグも含めて１ビット右に回転します。

③　例１

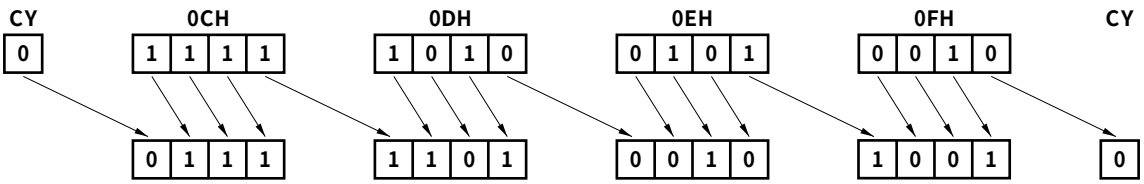
ジェネラル・レジスタとしてバンク０のロウ・アドレス０（0.00H-0.0FH）が指定されているとき（RPH = 0, RPL = 0），0.00H番地の値（1000B）を右に１ビット回転し，0100Bにします。

（0.00H）←（0.00H）÷2

```
MEM000   MEM    0.00H
         MOV    RPH, #00H        ; ジェネラル・レジスタのバンクを０
         MOV    RPL, #00H        ; ジェネラル・レジスタのロウ・アドレスを０
         CLR1   CY               ; CYフラグ←０
         RORC   MEM000
```

例２

ジェネラル・レジスタとしてバンク０のロウ・アドレス０（0.00H-0.0FH）が指定されているとき（RPH = 0, RPL = 0），データ・バッファDBFの内容0FA52Hを右に１ビット回転し，DBFの内容を7D29Hにします。



```
MEM00C  MEM   0.0CH
MEM00D  MEM   0.0DH
MEM00E  MEM   0.0EH
MEM00F  MEM   0.0FH
        MOV   RPH, #00H       ; ジェネラル・レジスタのバンクを0
        MOV   RPL, #00H       ; ジェネラル・レジスタのロウ・アドレスを0
        CLR1  CY              ; CYフラグ←0
        RORC  MEM00C
        RORC  MEM00D
        RORC  MEM00E
        RORC  MEM00F
```

15.5.7　転送命令

（1）LD r, m　　**Load data memory to general register**

① 命令コード

10	8 7	4 3	0
01000	m_R	m_C	r

② 機　　能

（r）←（m）

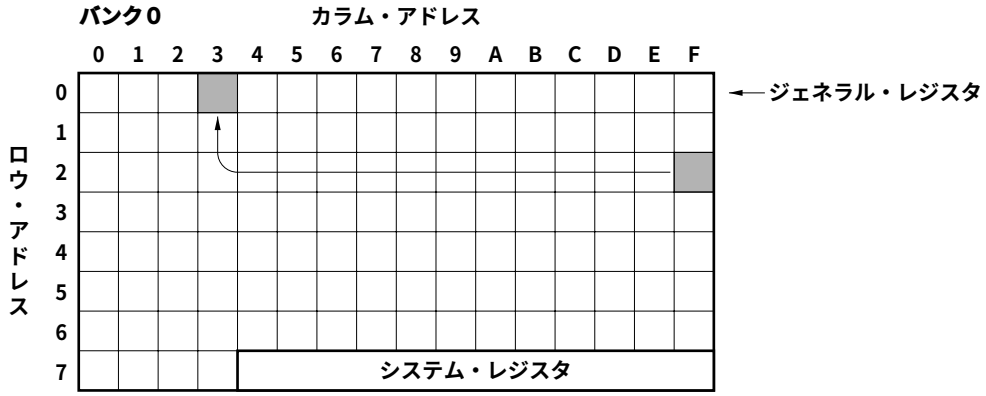
データ・メモリの内容をジェネラル・レジスタへ格納します

★ **③ 例1**

0.03H番地に0.2FH番地の内容を格納します。

（0.03H）←（0.2FH）

```
MEM003   MEM    0.03H
MEM02F   MEM    0.2FH
         MOV    RPH, #00H        ; ジェネラル・レジスタのバンクを0
         MOV    RPL, #00H        ; ジェネラル・レジスタのロウ・アドレスを0
         LD     MEM003, MEM02F
```

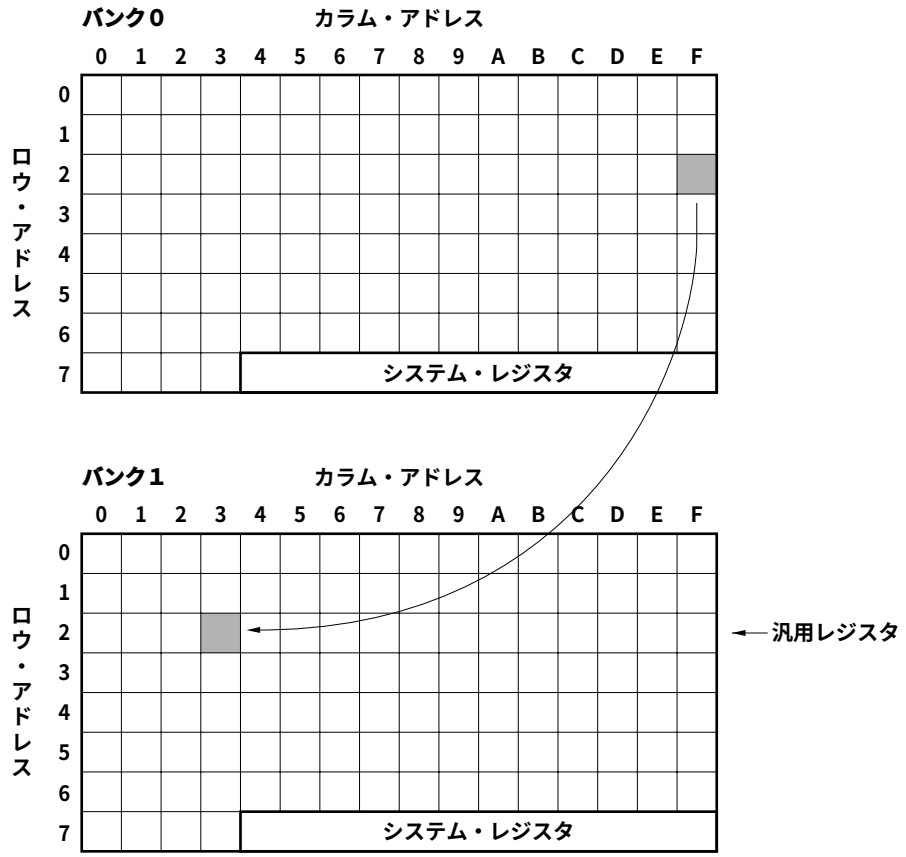


例2

ジェネラル・レジスタとしてバンク1のロウ・アドレス2（1.20H～1.2FH）が指定されているとき（RPH = 1, RPL = 4），0.2FH番地の内容を1.23H番地に格納します。

（1.23H）←（0.2FH）

```
MEM123   MEM    1.23H
MEM02F   MEM    0.2FH
         MOV    RPH, #01H        ; 汎用レジスタのバンクを1
         MOV    RPL, #04H        ; 汎用レジスタのロウ・アドレスを2
         LD     MEM123, MEM02F
```

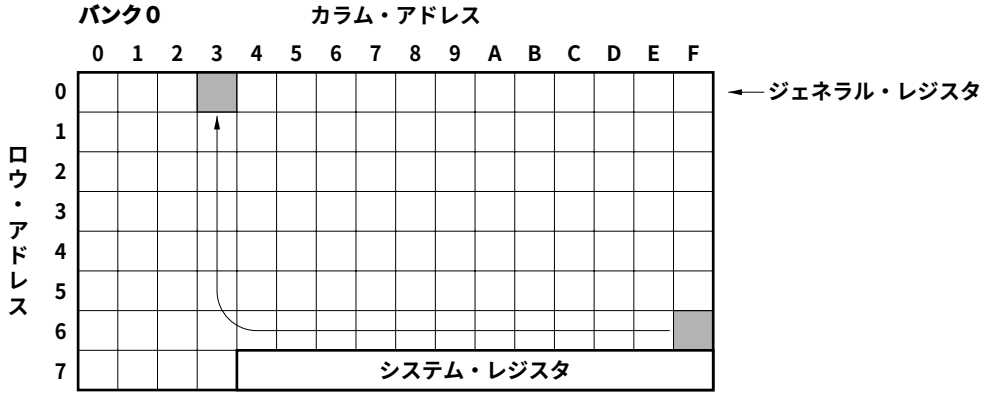


例3

0.6FH番地の内容を0.03H番地に格納します。このときIXE = 1, IXH = 0, IXM = 4, IXL = 0すなわちIX = 0.40Hなら，データ・メモリ・アドレスを2FHとすることでデータ・メモリ0.6FH番地を指定することができます。

IXH←00H
IXM←04H
IXL←00H
IXEフラグ←1
（0.03H）←（0.6FH）
　　インデクス・レジスタの内容040Hとデータ・メモリの0.2FHをOR演算したアドレス

```
MEM003   MEM    0.03H
MEM02F   MEM    0.2FH
         MOV    IXH, #00H          ; IX←00001000000B（0.40H）
         MOV    IXM, #04H
         MOV    IXL, #00H
         SET1   IXE                ; IXEフラグ←1
         LD     MEM003, MEM02F
```

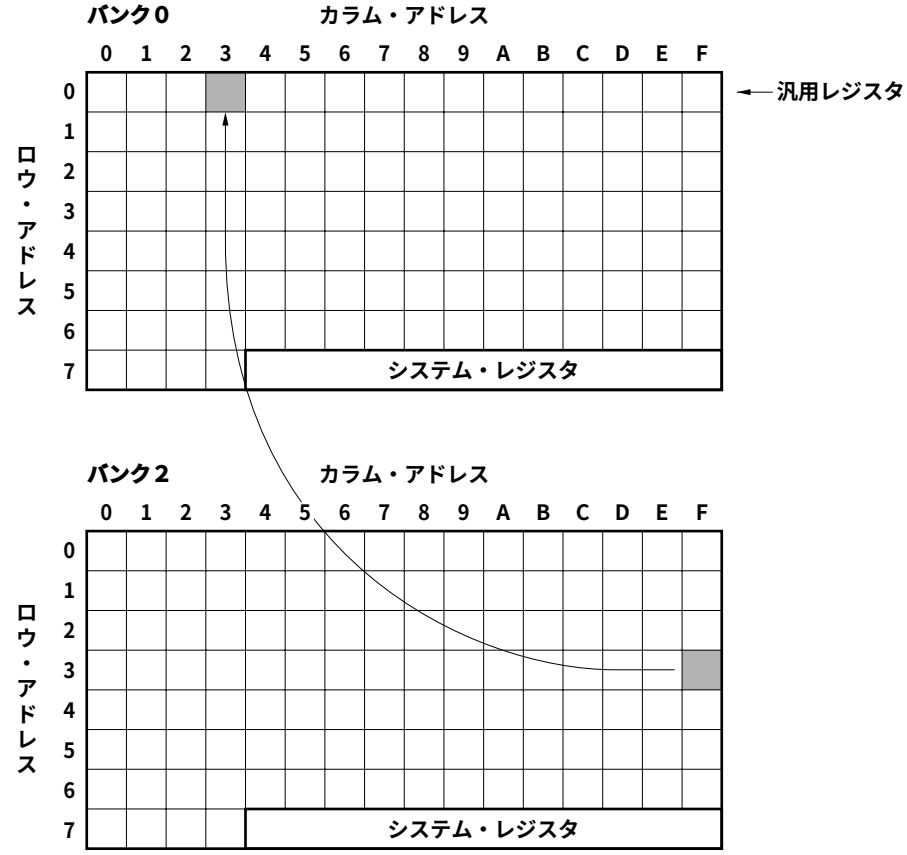


例4

2.3FH番地の内容を0.03H番地に格納します。このときIXE = 1，IXH = 1，IXM = 1，IXL = 0すなわちIX = 2.10Hなら，データ・メモリ・アドレスを2FHとすることでデータ・メモリ2.3FH番地を指定することができます。

（0.03H）←（2.3FH）
　　└インデクス・レジスタの内容を2.10Hとデータ・メモリの0.2FHをOR演算したアドレス

```
MEM003   MEM    0.03H
MEM02F   MEM    0.2FH
         MOV    IXH, #01H          ; IX←00100010000B（2.10H）
         MOV    IXM, #01H
         MOV    IXL, #00H
         SET1   IXE                ; IXEフラグ←1
         LD     MEM003, MEM02F
```



④ 注　　意

LD r, m命令の第１オペランドは汎用レジスタのカラム・アドレスです。したがって，次のように書いた場合，汎用レジスタのカラム・アドレスは03Hになります。

```
MEM013   MEM    0.13H
MEM02F   MEM    0.2FH
         LD     MEM013, MEM02F
                └── 汎用レジスタのカラム・アドレスを意味し，下位４ビットが有効となります。
                    この場合，汎用レジスタとしてバンク０のロウ・アドレス０が指定されていれ
                    ば03H番地が指定されます。
```

（2）ST m, r　　Store general register to data memory

① 命令コード

10	8 7	4 3	0
11000	m_R	m_C	r

② 機　　能

（m）←（r）

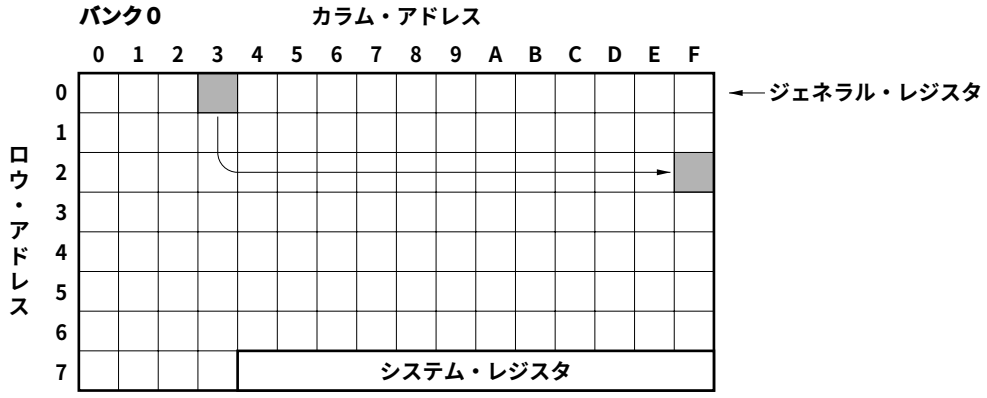
ジェネラル・レジスタの内容をデータ・メモリへ格納します。

★ ③ 例１

0.2FH番地に0.03H番地の内容を格納します。

（0.2FH）←（0.03H）

```
MOV     RPH, #00H          ; ジェネラル・レジスタのバンクを０
MOV     RPL, #00H          ; ジェネラル・レジスタのロウ・アドレスを０
ST      2FH, 03H           ; データ・メモリにジェネラル・レジスタの内容を転送
```

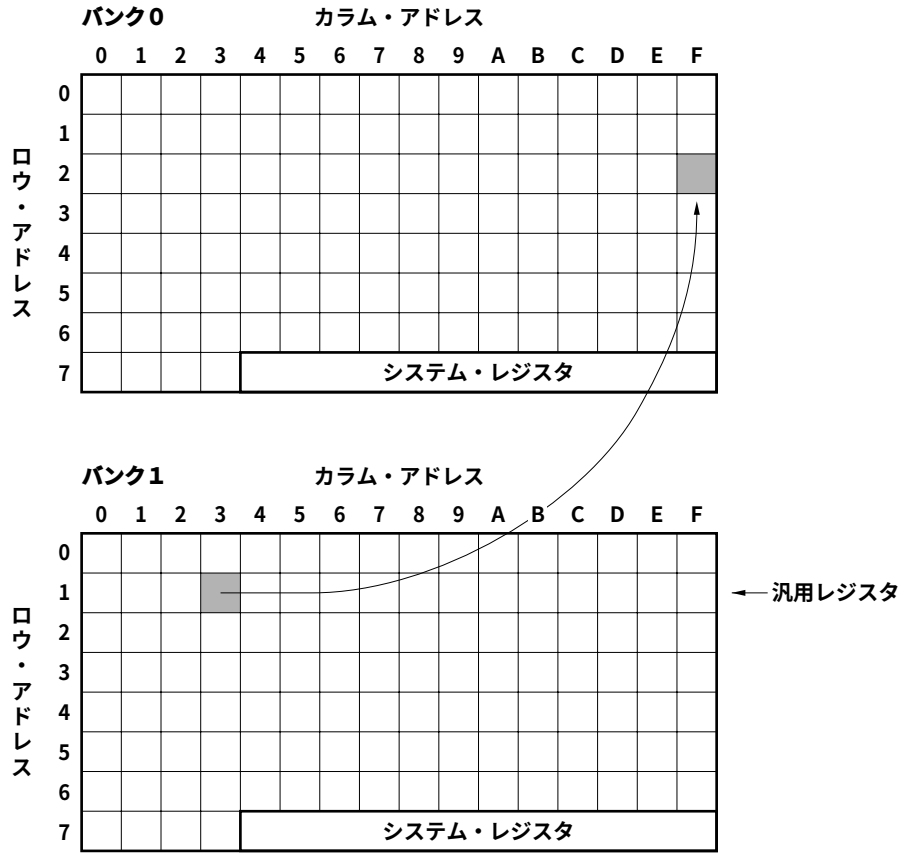


例２

0.2FH番地に1.13H番地の内容を格納します。レジスタ・ポインタでジェネラル・レジスタをバンク１のロウ・アドレス１（1.10H-1.1FH）に指定します。

（0.2FH）←（1.13H）

```
MEM02F   MEM    0.2FH
MEM113   MEM    1.13H
         MOV    RPH, #01H          ;汎用レジスタをバンク１
         MOV    RPL, #02H          ;汎用レジスタをロウ・アドレス１
         ST     MEM02F, MEM113     ;データ・メモリに汎用レジスタの内容を転送
```



例３

0.18H番地から0.1FH番地に0.00H番地の内容を格納します。インデクス・レジスタでデータ・メモリ（18H-1FH番地）を指定します。

（0.18H）←（0.00H）

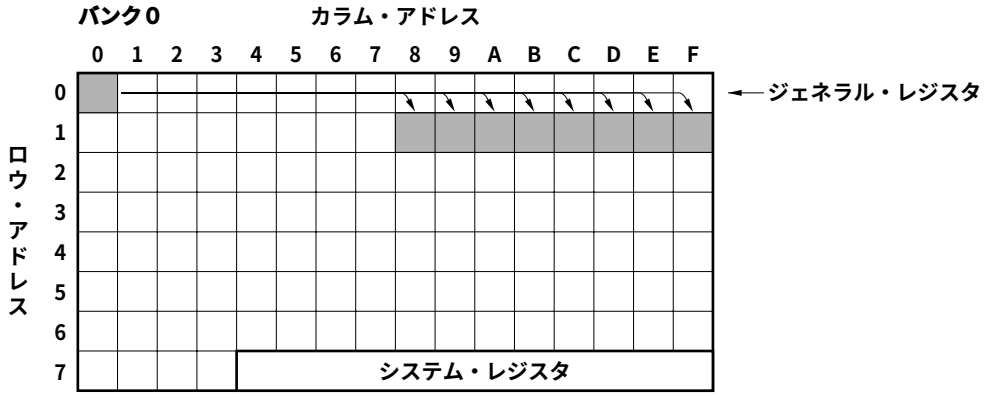
（0.19H）←（0.00H）

⋮

（0.1FH）←（0.00H）

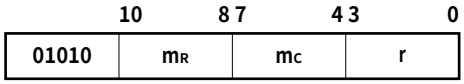
```
         MOV    IXH, #00H      ;IX←00000000000B（0.00H）
         MOV    IXM, #00H
         MOV    IXL, #00H      ;データ・メモリに0.00H番地を指定
MEM018   MEM    0.18H
MEM000   MEM    0.00H
```

```
LOOP1：
        SET1    IXE             ; IXEフラグ←１
        ST      MEM018, MEM000  ; （0.1×H）←（0.00H）
        CLR1    IXE             ; IXEフラグ←０
        INC     IX              ; IX←IX＋１
        SKGE    IXL, #08H
        BR      LOOP1
```



（3）MOV @r, m　　Move data memory to destination indirect

① 命令コード



② 機　　能

MPE = 1のとき

（MP, （r））←（m）

MPE = 0のとき

（BANK, mR, （r））←（m）

データ・メモリの内容をジェネラル・レジスタの内容でアドレスするデータ・メモリへ格納します。

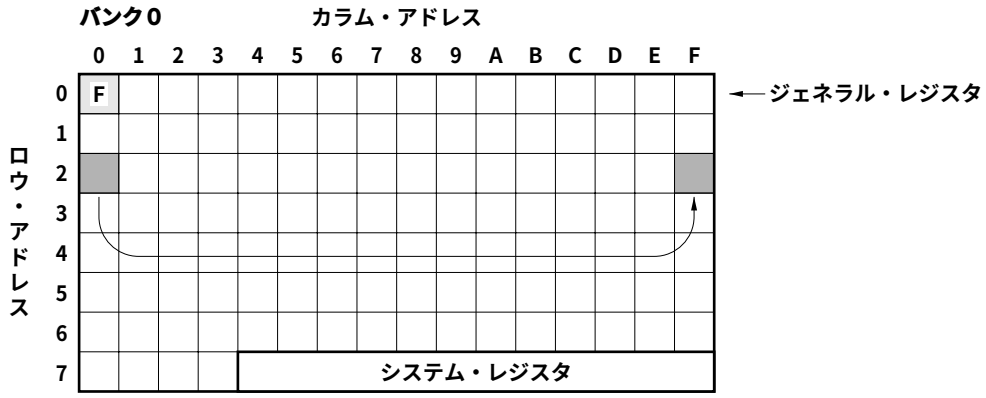
MPE = 0のときには，同一バンクの同一ロウ・アドレス内での転送となります。

③ 例1

MPEフラグを0にして，0.2FH番地に0.20H番地の内容を格納します。格納先のデータ・メモリは，転送元と同一のロウ・アドレスで，ジェネラル・レジスタの0.00H番地の内容がカラム・アドレスです。

（0.2FH）←（0.20H）

```
MEM000   MEM    0.00H
MEM020   MEM    0.20H
         CLR1   MPE                ; MPEフラグ←0
         MOV    MEM000, #0FH       ; ジェネラル・レジスタにカラム・アドレス設定
         MOV    @MEM000, MEM020    ; 格納
```



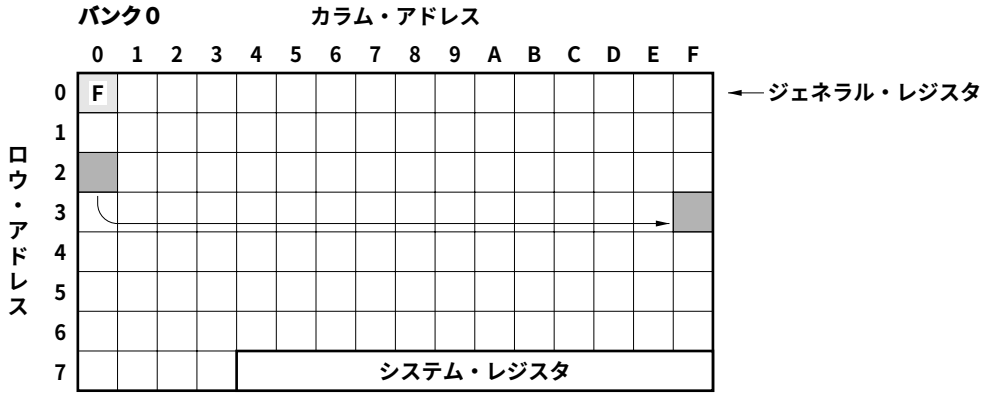
例2

MPEフラグを1にして，0.3FH番地に0.20H番地の内容を格納します。格納先のデータ・メモリは，メモリ・ポインタMPの内容がロウ・アドレスで，ジェネラル・レジスタの0.00H番地の内容がカラム・アドレスです。

（0.3FH）←（0.20H）

```
MEM000   MEM    0.00H
MEM020   MEM    0.20H
```

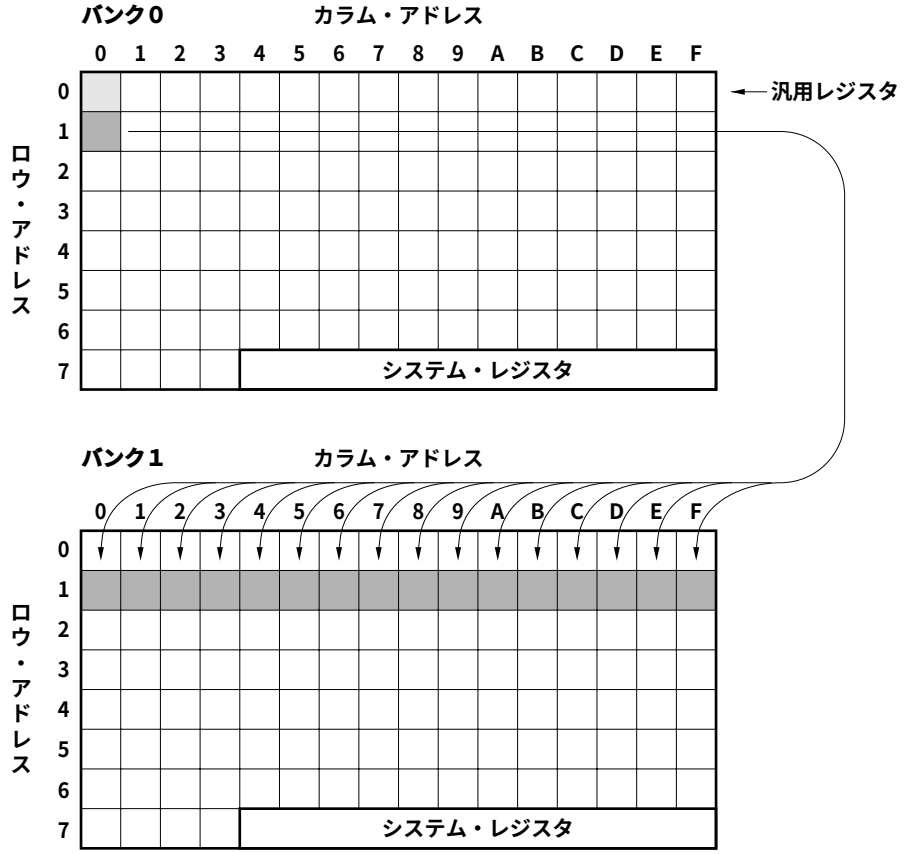
```
        MOV     RPH, #00H           ; ジェネラル・レジスタのバンクを0
        MOV     RPL, #00H           ; ジェネラル・レジスタのロウ・アドレスを0
        MOV     MEM000, #0FH        ; ジェネラル・レジスタにカラム・アドレス設定
        MOV     MPH, #00H           ; メモリ・ポインタにロウ・アドレスを設定
        MOV     MPL, #03H           ;
        SET1    MPE                 ; MPEフラグ←1
        MOV     @MEM000, MEM020     ; 格納
```



例3

1.10H番地から1.1FH番地に0.10H番地の内容を格納します。

```
                (1.10H) ← (0.10H)
                (1.11H) ← (0.10H)
                          ⋮
                (1.1FH) ← (0.10H)
MEM000  MEM     0.00H
MEM010  MEM     0.10H
        MOV     RPH, #00H           ; ジェネラル・レジスタのバンクを0
        MOV     RPL, #00H           ; ジェネラル・レジスタのロウ・アドレスを0
        MOV     MEM000, #00H        ; ジェネラル・レジスタにカラム・アドレス設定
        MOV     MPH, #00H           ; メモリ・ポインタにバンク1，ロウ・アドレス1
        MOV     MPL, #09H           ; を設定
        SET1    MPE                 ; MPEフラグ←1
LOOP1：
        MOV     @MEM000, MEM010     ; ((MP),(00H)) ← (10H)
        ADD     MEM000, #01H        ; カラム・アドレス＋1
        SKT1    CY                  ; バンク1の1FH番地まで終了したか
        BR      LOOP1
```



（4）MOV m, @r　　Move data memory to destination indirect

①　命令コード

10	8 7	4 3	0
11010	m_R	m_C	r

②　機　　能

MPE = 1のとき

（m）←（MP,（r））

MPE = 0のとき

（m）←（BANK, m_R,（r））

ジェネラル・レジスタの内容でアドレスするデータ・メモリの内容をデータ・メモリへ格納します。

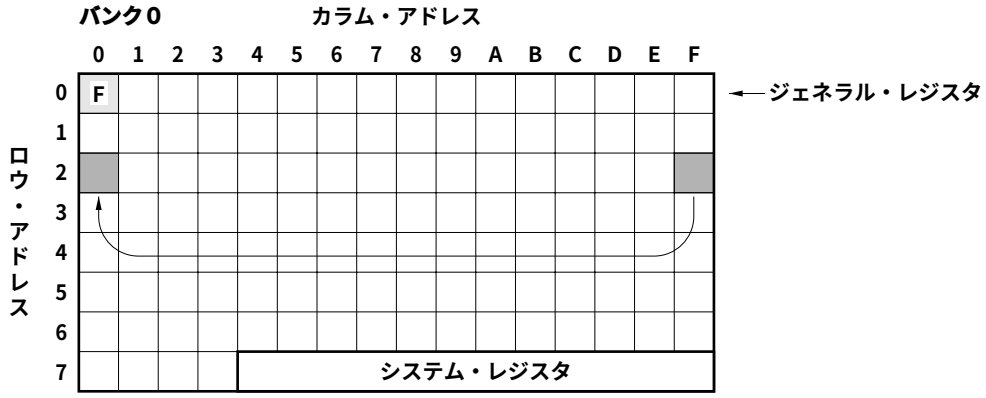
MPE = 0のときには，同一バンクの同一ロウ・アドレス内での転送となります。

③　例1

MPEフラグを0にして，0.20H番地に0.2FH番地の内容を格納します。転送元のデータ・メモリは，格納先と同一のロウ・アドレスで，ジェネラル・レジスタの0.00H番地の内容がカラム・アドレスです。

（0.20H）←（0.2FH）

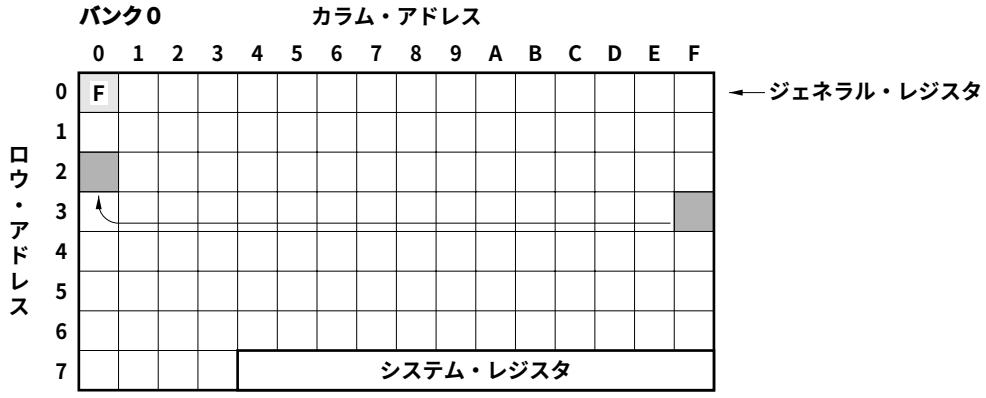
```
MEM000   MEM    0.00H
MEM020   MEM    0.20H
         CLR1   MPE                ; MPEフラグ←0
         MOV    MEM000, #0FH       ; ジェネラル・レジスタにカラム・アドレス設定
         MOV    MEM020, @MEM000    ; 格納
```



例2

MPEフラグを１にして，0.20H番地に0.3FH番地の内容を格納します。転送元のデータ・メモリは，メモリ・ポインタMPの内容がロウ・アドレスで，ジェネラル・レジスタの0.00番地の内容がカラム・アドレスです。

```
                （0.20H）←（0.3FH）
MEM000   MEM    0.00H
MEM020   MEM    0.20H
         MOV    MEM000, #0FH        ; ジェネラル・レジスタにカラム・アドレス設定
         MOV    MPH, #00H           ; メモリ・ポインタにロウ・アドレスを設定
         MOV    MPL, #03H           ;
         SET1   MPE                 ; MPEフラグ←１
         MOV    MEM020, @MEM000     ; 格納
```

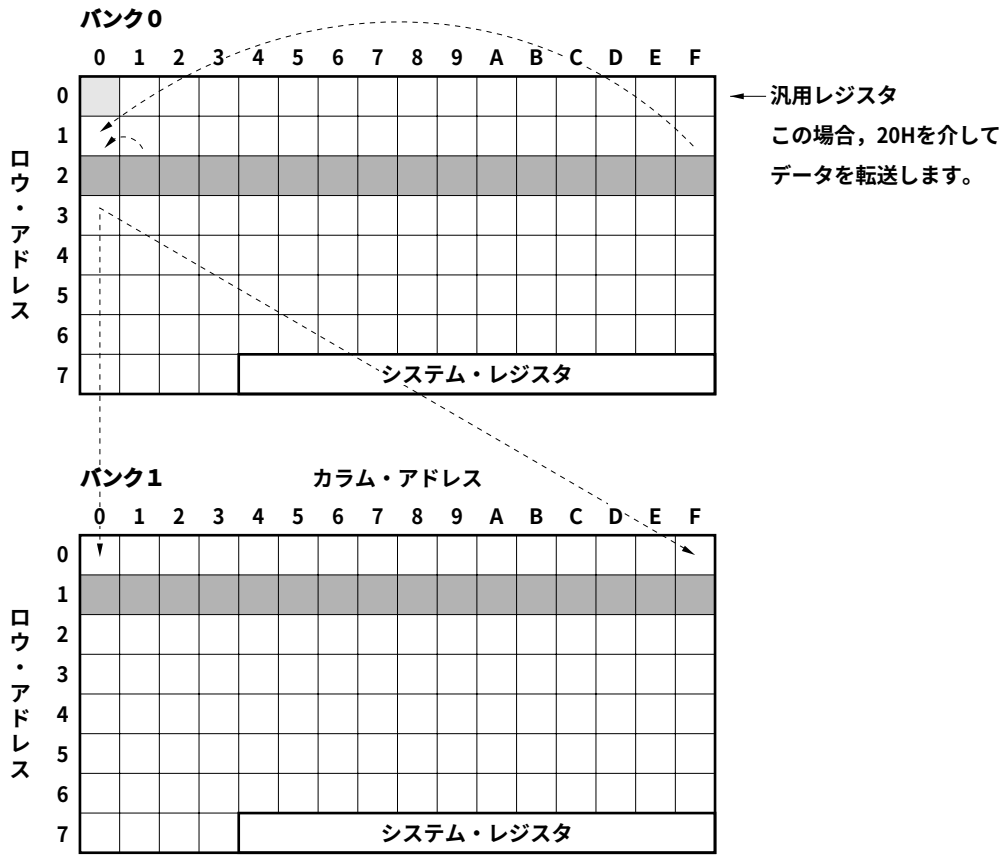


例3

0.20H番地から0.2FH番地の内容を1.10番地から1.1FH番地に格納します。転送元のデータ・メモリは，中継に使うデータ・メモリと同じロウ・アドレスで，ジェネラル・レジスタの0.00H番地の内容がカラム・アドレスです。

格納先のデータ・メモリは，メモリ・ポインタMPの内容がロウ・アドレスで，ジェネラル・レジスタの0.00H番地の内容がカラム・アドレスです。

```
                (1.10H) ← (0.20H)
                (1.11H) ← (0.21H)
                (1.12H) ← (0.22H)
                    ⋮
                (1.1FH) ← (0.2FH)
MEM000  MEM     0.00H
MEM020  MEM     0.20H
        CLR1    MPE             ; MPEフラグ←0
        MOV     MEM000, #00H    ; ジェネラル・レジスタにカラム・アドレス設定
        MOV     MPH, #00H       ; メモリ・ポインタ設定
        MOV     MPL, #09H       ; バンク1，ロウ・アドレス1
LOOP1：
        MOV     MEM020, @MEM000 ; (20H) ← (2, (00H))
        SET1    MPE             ; MPEフラグ←1
        MOV     @MEM000, MEM020 ; ((MP), (00H)) ← (20H)
        CLR1    MPE             ; MPEフラグ←0
        ADD     MEM000, #01H    ; カラム・アドレス＋1
        SKT1    CY              ; バンク1の1FHまで終了
        BR      LOOP1
```



（5）MOV m, ＃n4　　Move immediate data to data memory

① 命令コード

10	8 7	4 3	0
11101	m_R	m_C	n4

② 機　　能

（m）←n4

データ・メモリにイミーディエト・データを格納します。

③ 例１

データ・メモリの0.50H番地にイミーディエト・データの0AHを格納します。

```
          （0.50H）←0AH
MEM050    MEM    0.50H
          MOV    MEM050, ＃0AH
```

例２

データ・メモリとして0.00H番地が指定されているとき，IXH = 0, IXM = 3, IXL = 2, IXEフラグ = 1が設定されていると，0.32H番地にイミーディエト・データの07Hを格納します。

```
          （0.32H）←07H
MEM000    MEM    0.00H
          MOV    IXH, ＃00H          ; IX←00000110010B（0.32H）
          MOV    IXM, ＃03H
          MOV    IXL, ＃02H
          SET1   IXE                 ; IXEフラグ←1
          MOV    MEM000, ＃07H
```

（6）MOVT DBF, @AR　　Move program memory data specified by AR to DBF

① 命令コード

10	8 7	4 3	0
00111	000	0001	0000

② 機　　能

SP←SP－1, ASR←PC, PC←AR,

DBF←（PC）, PC←ASR, SP←SP＋1

アドレス・レジスタARでアドレスされるプログラム・メモリの内容をデータ・バッファDBFに格納します。

この命令は一時的にスタックを１レベル使用するため，サブルーチン，割り込みなどのネスティングに注意してください。

③ 例1

システム・レジスタ内のアドレス・レジスタAR3, AR2, AR1, AR0の値によりテーブル・データの16ビットをデータ・バッファDBF3, DBF2, DBF1, DBF0に転送します。

```
                ;＊
                ;＊＊　テーブル・データ
                ;＊
  アドレス      ORG   0010H
  0010H         DW    0000000000000000B  ;（0000H）
  0011H         DW    1010101111001101B  ;（0ABCDH）
                              ⋮
                ;＊
                ;＊＊　テーブル参照プログラム
                ;＊
                MOV   AR3, ＃00H         ;AR3←00H     アドレス・レジスタに0011Hを設定
                MOV   AR2, ＃00H         ;AR2←00H
                MOV   AR1, ＃01H         ;AR1←01H
                MOV   AR0, ＃01H         ;AR0←01H
                MOVT  DBF, @AR           ;アドレス0011H番地のデータをDBFに転送
```

この場合，DBFには次のようにデータが格納されます。

DBF3 = 0AH

DBF2 = 0BH

DBF1 = 0CH

DBF0 = 0DH

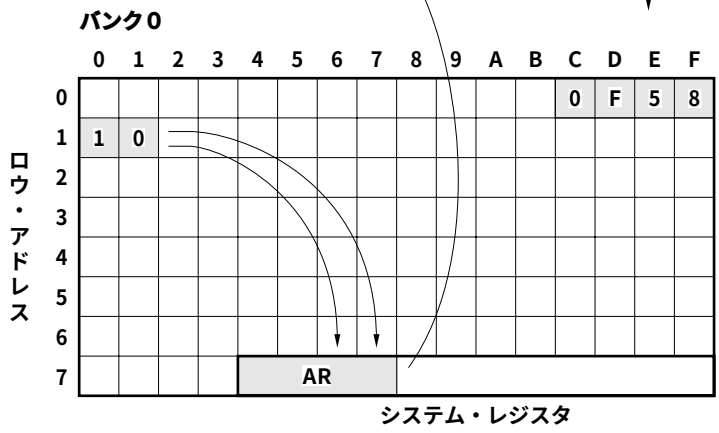
例2

データ・メモリとして0.10H番地と0.11H番地にチャンネル番号を設定しておき，その内容によりPLL分周値（N値）を求め，PLLレジスタに転送する。

```
                ;＊
                ;＊＊　N値用テーブル・データ
                ;＊
  アドレス      ORG   0010H
  0010H         DW    0F58H              ;87.5 MHz（最低周波数　00チャンネル）
  0011H         DW    0F5CH              ;87.6 MHz
  0012H         DW    0F60H              ;87.7 MHz
  0013H         DW    0F64H              ;87.8 MHz
  0014H         DW    0F68H              ;87.9 MHz
  0015H         DW    0F6CH              ;88.0 MHz
  0016H         DW    0F70H              ;88.1 MHz
  0017H         DW    0F74H              ;88.2 MHz
                              ⋮
                ;＊
                ;＊＊　N値設定プログラム
                ;＊
```

```
MEM010    MEM    0.10H
MEM011    MEM    0.11H
          MOV    RPH, #00H          ; RPH←00H　ジェネラル・レジスタとしてロウ・アドレス
          MOV    RPL, #0EH          ; RPL←0EH　7（0.70H-0.7FH）を設定
          MOV    AR3, #00H          ; AR3←0
          MOV    AR2, #00H          ; AR2←0
          LD     AR1, MEM010        ; AR1←10H　チャンネル・データ上位
          LD     AR0, MEM011        ; AR0←11H　チャンネル・データ下位
          ADD    AR1, #01H          ; テーブル・データのスタート・アドレスが0010H番地か
          ADDC   AR2, #00H          ; らであるため，アドレス・レジスタに0010Hを加算
          ADDC   AR3, #00H          ;
          MOVT   DBF, @AR           ; テーブル・データをDBFに格納
          PUT    PLLR, DBF          ; PLLレジスタ（PLLR）にN値を転送
                 ⋮
```

```
                ;＊
                ;＊＊　テーブル・データ
                ;＊
0010H           DW     0F58H       ; 87.5 MHz
0011H           DW     0F5CH       ; 87.6 MHz
0012H           DW     0F60H       ; 87.7 MHz
0013H           DW     0F64H       ; 87.8 MHz
0014H           DW     0F68H       ; 87.9 MHz
```



④ 注意１

アドレス・レジスタAR3, AR2, AR1, AR0は，品種により使用できるビット数が異なりますので，使用する際は，各品種ごとのデータ・シートを参照してください。

注意２

MOVT命令を実行する場合，スタックとして１レベルが使用されます。したがってサブルーチン内や割り込み処理ルーチン内で使用する場合はスタック・レベルについて十分注意する必要があります。

（7）PUSH AR　　Push address register

① 命令コード

10	8 7	4 3	0
00111	000	1101	0000

② 機　　能

SP←SP－1,

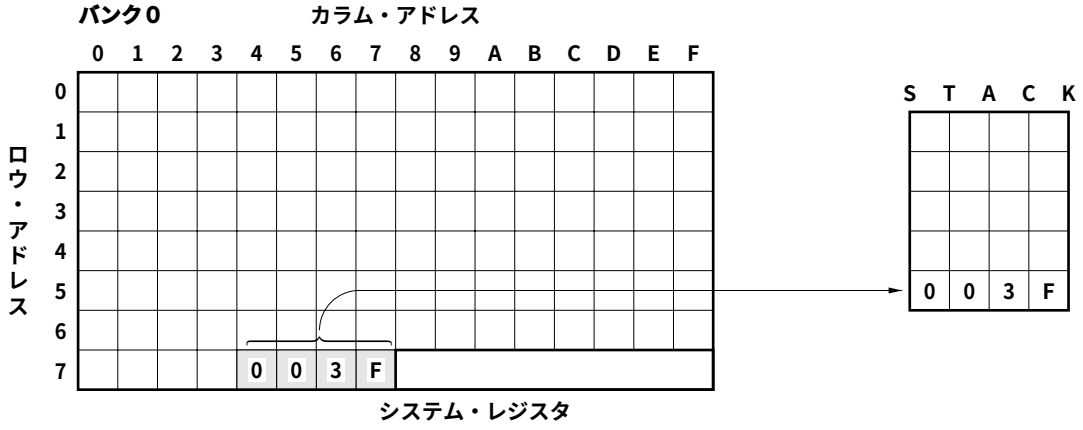
ASR←AR

スタック・ポインタSPをデクリメントしたあと，スタック・ポインタで指定されるアドレス・スタック・レジスタにアドレス・レジスタARの値を格納します。

③ 例１

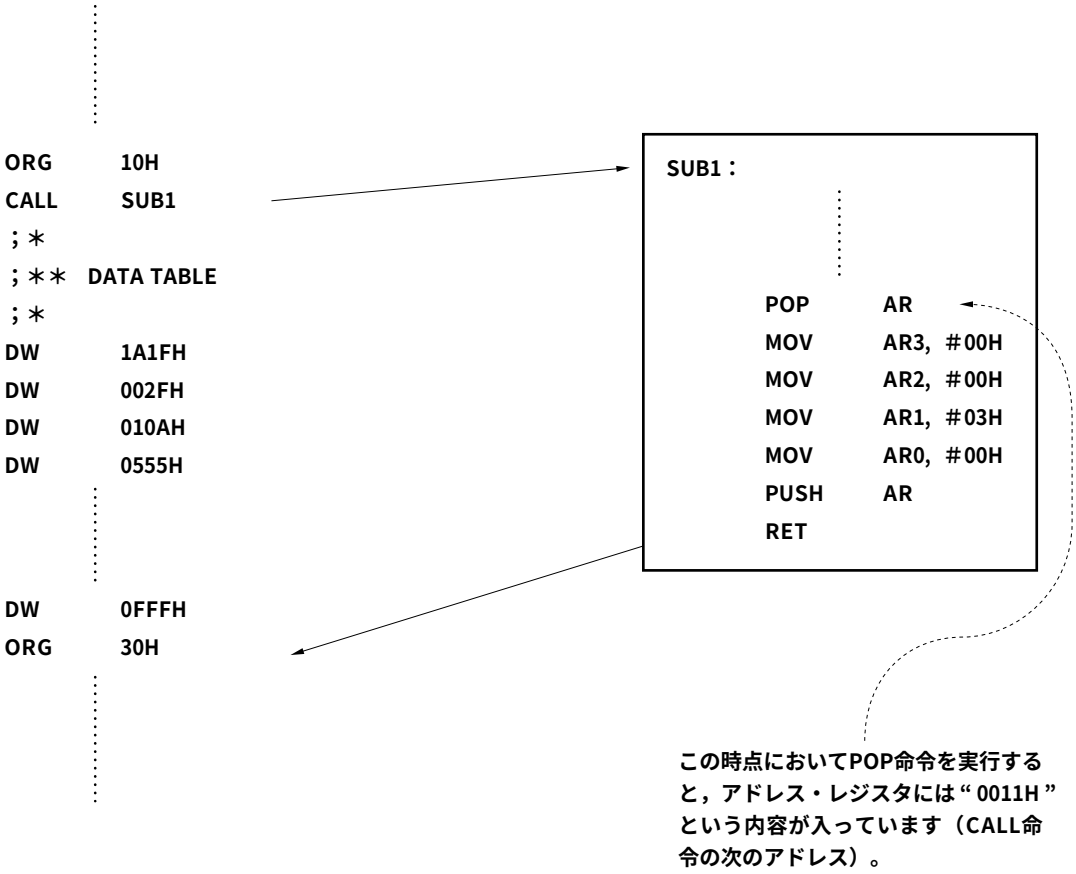
アドレス・レジスタに003FHを設定し，スタックに格納します。

```
MOV     AR3, #00H
MOV     AR2, #00H
MOV     AR1, #03H
MOV     AR0, #0FH
PUSH    AR
```



例２

CALL命令に続いてデータ・テーブルがある場合にサブルーチンからの戻り番地（データ・テーブルの次の番地）をアドレス・レジスタに設定しリターンします。



（8）POP AR　　Pop address register

①　命令コード

00111	000	1100	0000

②　機　　能

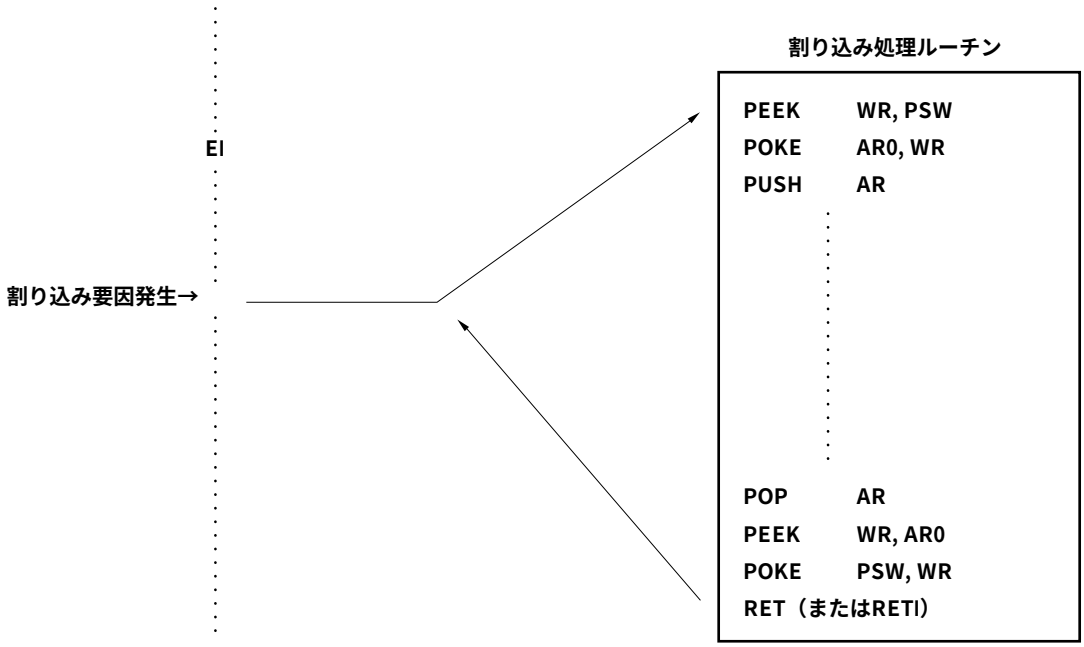
AR←ASR,

SP←SP＋１

スタック・ポインタで示されるアドレス・スタック・レジスタの内容をアドレス・レジスタARに取り出したあと，スタック・ポインタSPをインクリメントします。

③　例

割り込み処理を行う場合，割り込み処理ルーチン内にてPSWが変化してしまう場合に，割り込み処理の始めでPSWの内容をWRを介してアドレス・レジスタに転送し，PUSH命令によりアドレス・スタック・レジスタに退避し，リターンする前にPOP命令によりアドレス・レジスタに復帰させWRを介してPSWに転送します。



（9）PEEK WR, rf　　Peek register file to window register

① 命令コード

00111	rf_R	0011	rf_C

② 機　　能

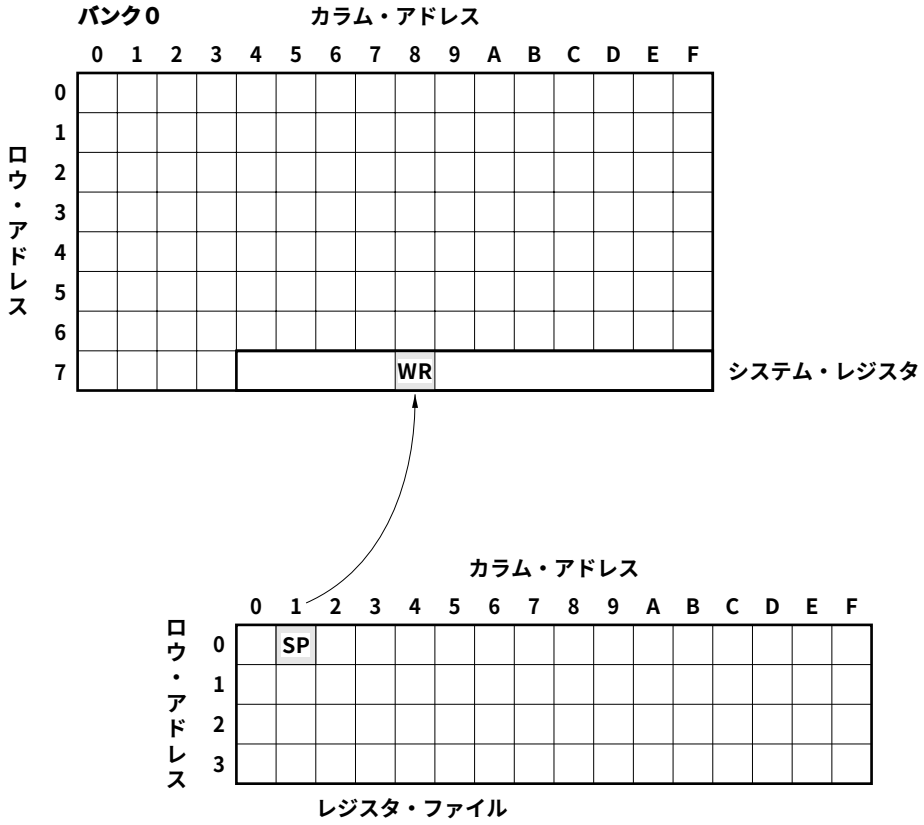
WR←（rf）

レジスタ・ファイルの内容をウインドウ・レジスタWRに格納します。

③ 例

レジスタ・ファイル内の01H番地のスタック・ポインタSPの内容をウインドウ・レジスタに格納します。

```
PEEK　WR, SP
```



(10) POKE rf, WR　　Poke window register to register file

① 命令コード

10	8 7	4 3	0
00111	rf_R	0010	rf_C

② 機　　能

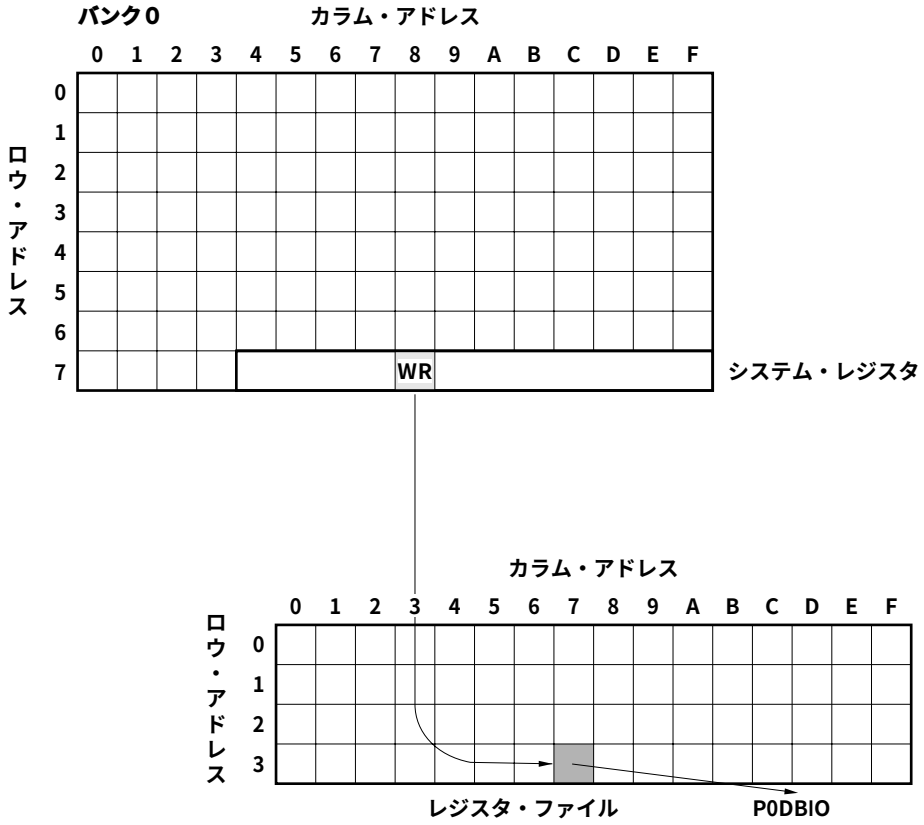
(rf) ←WR

ウインドウ・レジスタWRの内容をレジスタ・ファイルに格納します。

③ 例1

ウインドウ・レジスタを介してレジスタ・ファイルのP0DBIOにイミーディエト・データの0FHを格納します。

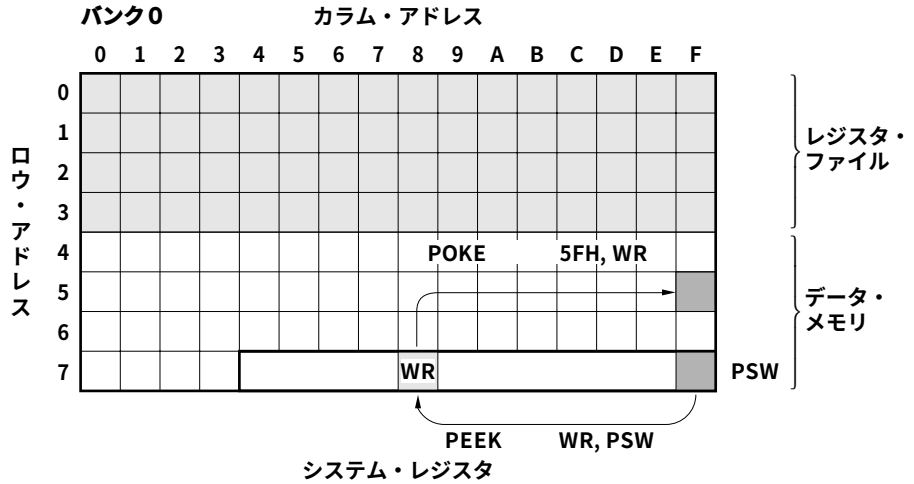
```
MOV     WR, #0FH
POKE    P0DBIO, WR          ; P0D0, P0D1, P0D2, P0D3をすべて出力モードに設定
```



④ 注　意

プログラム上は，レジスタ・ファイルの40Hから7FH番地に，データ・メモリのアドレス40Hから7FH番地と同じメモリがあるように見えます。したがって，PEEK, POKE命令ではレジスタ・ファイル以外にデータ・メモリの各バンクの40H番地から7FH番地までをアクセスすることができます。たとえば次のような使い方も可能です。

```
MEM05F   MEM    0.5FH
         PEEK   WR, PSW       ；システム・レジスタ内のPSW（7FH）の内容をWRに格納
         POKE   MEM05F, WR    ；WRの内容をデータ・メモリの5FH番地に格納
```



（11）GET DBF, p　　Get peripheral data to data buffer

① 命令コード

10	8 7	4 3	0
00111	p_H	1011	p_L

② 機　　能

DBF←（p）

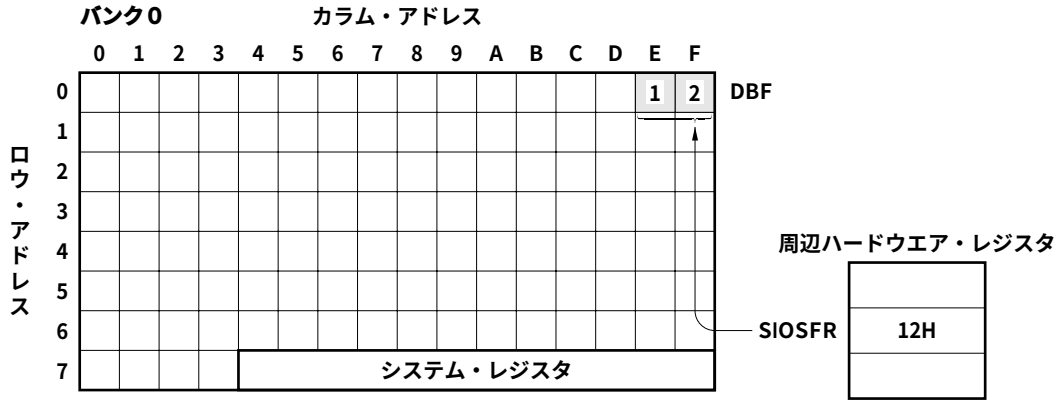
周辺レジスタの内容をデータ・バッファDBFに格納します。

DBFはバンク・レジスタの値に関係なく，データ・メモリのBANK0の0CHから0FH番地の16ビットの領域となります。

③ 例

シリアル・インタフェースのシフト・レジスタSIOSFRの内容（８ビット）をデータ・バッファのDBF0, DBF1に格納します。

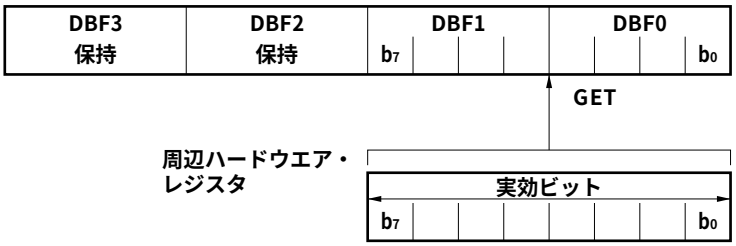
GET DBF, SIOSFR



④ 注　意

データ・バッファは16ビットで構成されています。ただし，周辺ハードウエアによってアクセスされるビット数は異なります。たとえば実効ビット長が８ビットの周辺ハードウエア・レジスタに対してGET命令を実行した場合は，データ・バッファDBFの下位８ビット（DBF1, DBF0）に周辺ハードウエア・レジスタの内容が格納されます。

★



（12）PUT p, DBF　　Put data buffer to peripheral

① 命令コード

10	8 7	4 3	0
00111	p_H	1010	p_L

② 機　　能

（p）←DBF

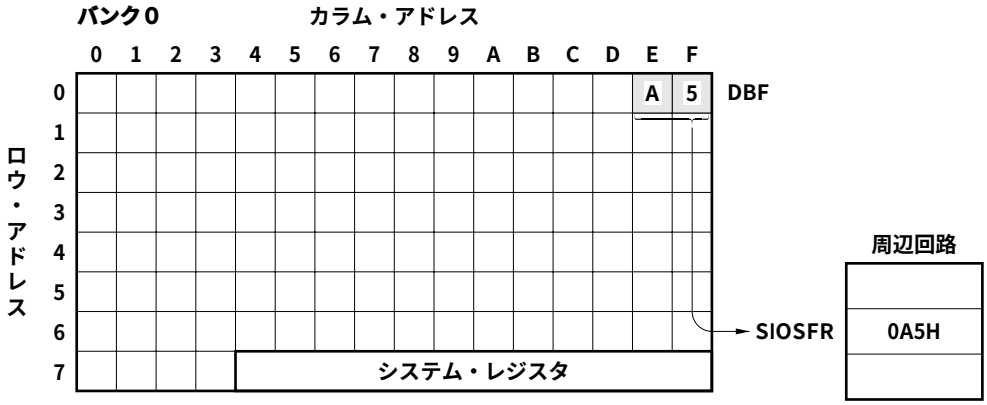
データ・バッファDBFの内容を周辺ハードウエア・レジスタに格納します。

★ DBFはバンク・レジスタの値に関係なく，データ・メモリのBANK0の0CHから0FH番地の番地の16ビットの領域となります。

③ 例

データ・バッファのDBF1とDBF0にそれぞれ0AH, 05Hを設定し，周辺レジスタの１つであるシリアル・インタフェース用のシフト・レジスタ（SIOSFR）に転送します。

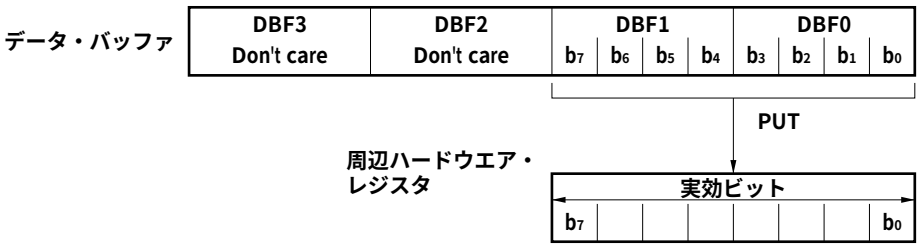
```
MOV     BANK, #00H          ；データ・メモリのバンクを０
MOV     DBF0, #05H
MOV     DBF1, #0AH
PUT     SIOSFR, DBF
```



④ 注　　意

データ・バッファは，16ビットで構成されています。ただし，周辺ハードウエアによってアクセスされるビット数は異なります。たとえば実効ビット長が８ビットの周辺ハードウエア・レジスタに対してPUT命令を実行した場合は，データ・バッファDBFの下位８ビット（DBF1, DBF0）の内容を周辺ハードウエア・レジスタに格納します（DBF3, DBF2の内容は無効です）。

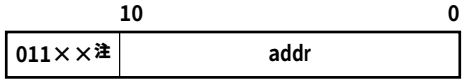
★



15.5.8　分岐命令

（1）BR addr　　Branch to the address

①　命令コード



注　④注意参照。

②　機　　能

PC_{10-0}←addr

addrで指定するアドレスに分岐します。

この命令で直接分岐できるアドレス範囲は0000H-1FFFH番地までの８Kステップ内です。

2000H番地以降のアドレスに分岐する場合は，次のBR @AR命令を使用してください。

③　例

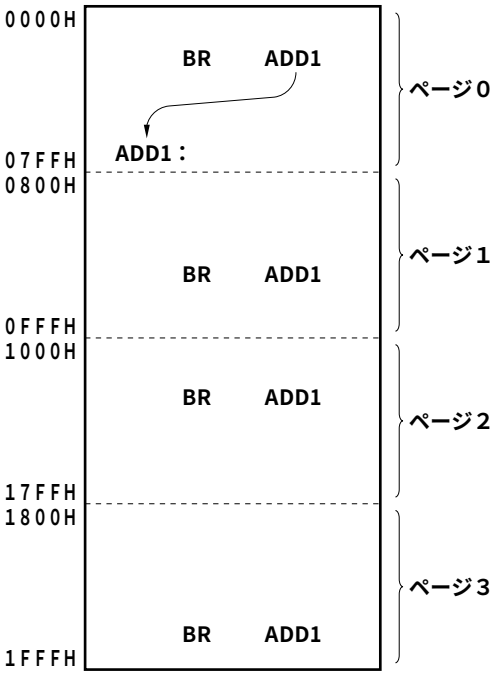
```
FLY     LAB   0FH          ; FLY = 0FHを定義
          ⋮
        BR    FLY          ; 0FH番地にジャンプします
          ⋮
        BR    LOOP1        ; LOOP1にジャンプします
          ⋮
        BR    $+2          ; 現在番地より２つ下の番地へジャンプします
          ⋮
        BR    $-3          ; 現在番地より３つ上の番地へジャンプします
          ⋮
LOOP1：
```

④　注　意

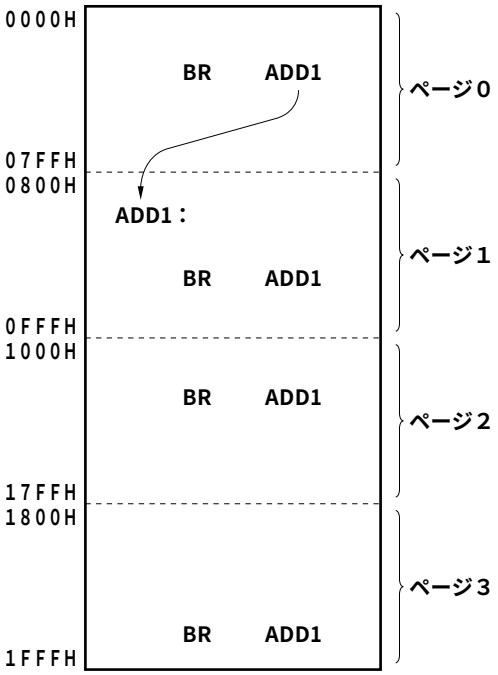
BR命令はアセンブラで記述する上ではページの概念はなく，ROMアドレス0000H-1FFFH番地の間で，同一記述で使用することができます。ただし，ページ０内（0000H-07FFH番地）へのBR命令と，ページ１内（07FFH-0FFFH番地）へのBR命令と，ページ２（1000H-17FFH番地）へのBR命令と，ページ３（17FFH-1FFFH番地）へのBR命令とでは，オペレーション・コードがそれぞれ異なります。

ページ０内でのオペレーション・コードは‘0C’，ページ１内でのオペレーション・コードは‘0D’，ページ２内でのオペレーション・コードは‘0E’，ページ３内でのオペレーション・コードは‘0F’です。これらはアセンブルする際，17Kシリーズのアセンブラを使用すればアセンブラがジャンプ先を参照して自動的に変換します。

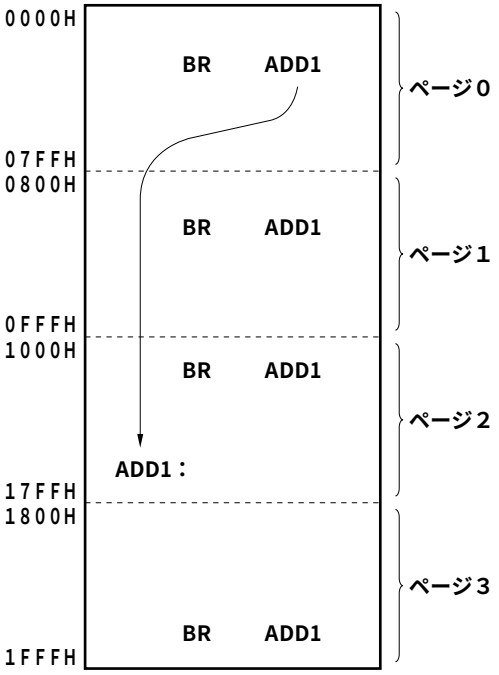
オペレーション・コードが0Cとなる場合
（ジャンプ先のアドレスがページ０内に
ある場合）



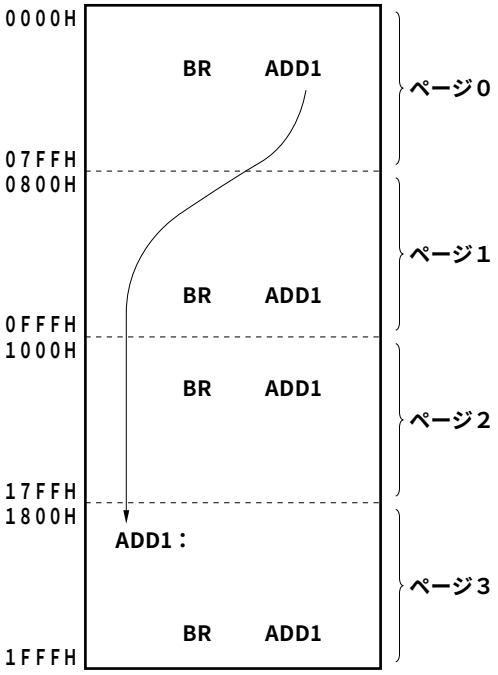
オペレーション・コードが0Dとなる場合
（ジャンプ先のアドレスがページ１内に
ある場合）



オペレーション・コードが0Eとなる場合
（ジャンプ先のアドレスがページ２内に
ある場合）

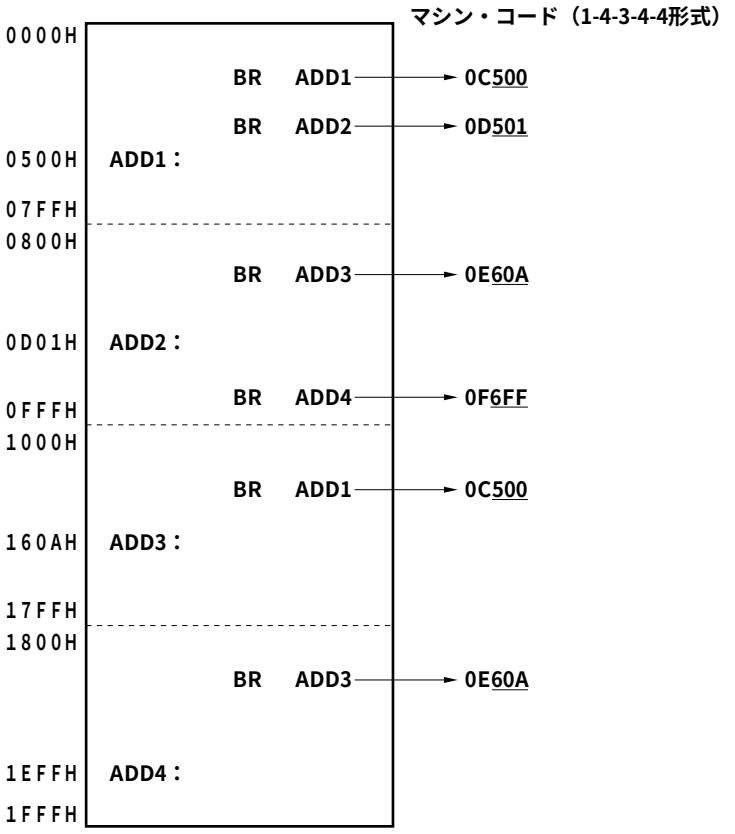


オペレーション・コードが0Fとなる場合
（ジャンプ先のアドレスがページ３内に
ある場合）



ディバグ時，パッチ修正を行う場合は，プログラム自身で‘0C’,‘0D’,‘0E’,‘0F’のそれぞれの変換を行う必要があります。

また，BR命令のジャンプ先がそれぞれ0000H-07FFH番地，0800H-0FFFH番地，1000H-17FFH番地，1800H-1FFFH番地の場合に対してアドレスの変換も必要です。つまり，0000H番地，0800H番地，1000H番地，1800H番地を000H番地として以降１番地ずつインクリメントされたアドレスとなります。



注意　μPD170××シリーズの各品種ごとでページ数が異なりますので，使用する各品種ごとのデータ・シートを参照してください。

（2）BR @AR　　Branch to the address specified by address register

①　命令コード

00111	000	0100	0000

②　機　　能

PC←AR

アドレス・レジスタARが指定するプログラム・アドレスに分岐します。

③　例1

アドレス・レジスタAR（AR0-AR3）に003FHを設定し，BR @AR命令により003FH番地へジャンプします。

```
MOV     AR3, #00H           ; AR3←00H
MOV     AR2, #00H           ; AR2←00H
MOV     AR1, #03H           ; AR1←03H
MOV     AR0, #0FH           ; AR0←0FH
BR      @AR                 ; 003FH番地へジャンプ
```

例2

データ・メモリの0.10H番地の内容によって次のように分岐先を変えます。

```
0.10Hの内容          分岐先のレーベル
  00H          →    AAA
  01H          →    BBB
  02H          →    CCC
  03H          →    DDD
  04H          →    EEE
  05H          →    FFF
  06H          →    GGG
  07H          →    HHH
08H-0FH        →    ZZZ
```

```
;＊
;＊＊ジャンプ・テーブル
;＊
ORG         10H
BR          AAA
BR          BBB
BR          CCC
BR          DDD
BR          EEE
BR          FFF
```

```
        BR      GGG
        BR      HHH
        BR      ZZZ
                 ⋮
MEM010  MEM     0.10H
        MOV     AR3, #00H    ; AR3←00H ARを001×Hにする
        MOV     AR2, #00H    ; AR2←00H
        MOV     AR1, #01H    ; AR1←01H
        MOV     RPH, #00H    ; ジェネラル・レジスタのバンクを0
        MOV     RPL, #02H    ; ジェネラル・レジスタのロウ・アドレスを1
        ST      AR0, MEM010  ; AR0←0.10H
        SKLT    AR0, #08H
        MOV     AR0, #08H    ; AR0の内容が08Hよりも大きい場合は，AR0の内容を08Hにする
        BR      @AR
```

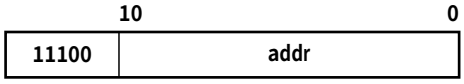
④ 注　意

アドレス・レジスタ（AR0-AR3）は品種により使用できるビット数が異なりますので，使用する際は，各品種ごとのデータ・シートを参照してください。

15.5.9　サブルーチン命令

（1）CALL addr　　Call subroutine

①　命令コード



②　機　　能

SP←SP－1, ASR←PC,

PC_{10-0}←addr，PAGE←0

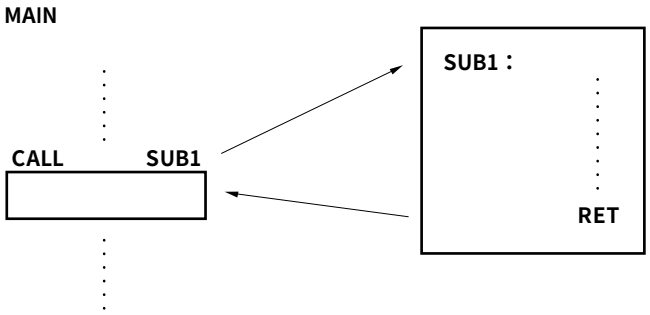
プログラム・カウンタPCの値をインクリメントし，スタックに格納したのち，addrで指定するサブルーチンに分岐します。

この命令で呼び出せるサブルーチンは0000H-07FFH番地までの２Kステップ内のみです。

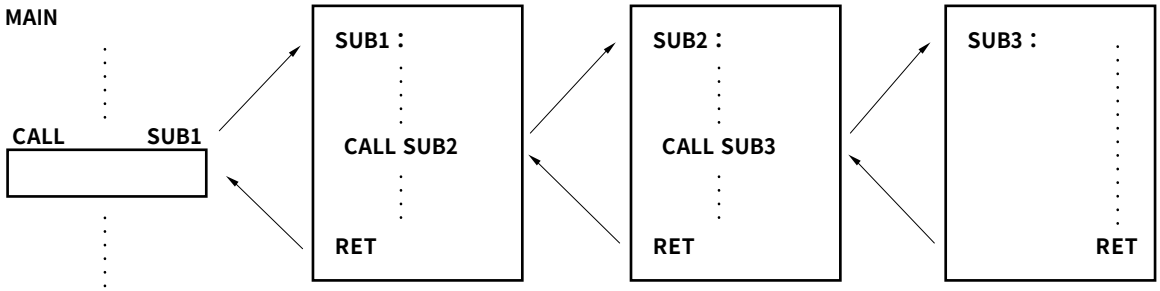
したがって，頻繁に使用するサブルーチンは0000H-07FFH番地までの範囲に配置しておくと便利です。

また，0800H番地以降に配置したサブルーチンを呼び出す場合は，次のCALL @AR命令を使用してください。

③　例１



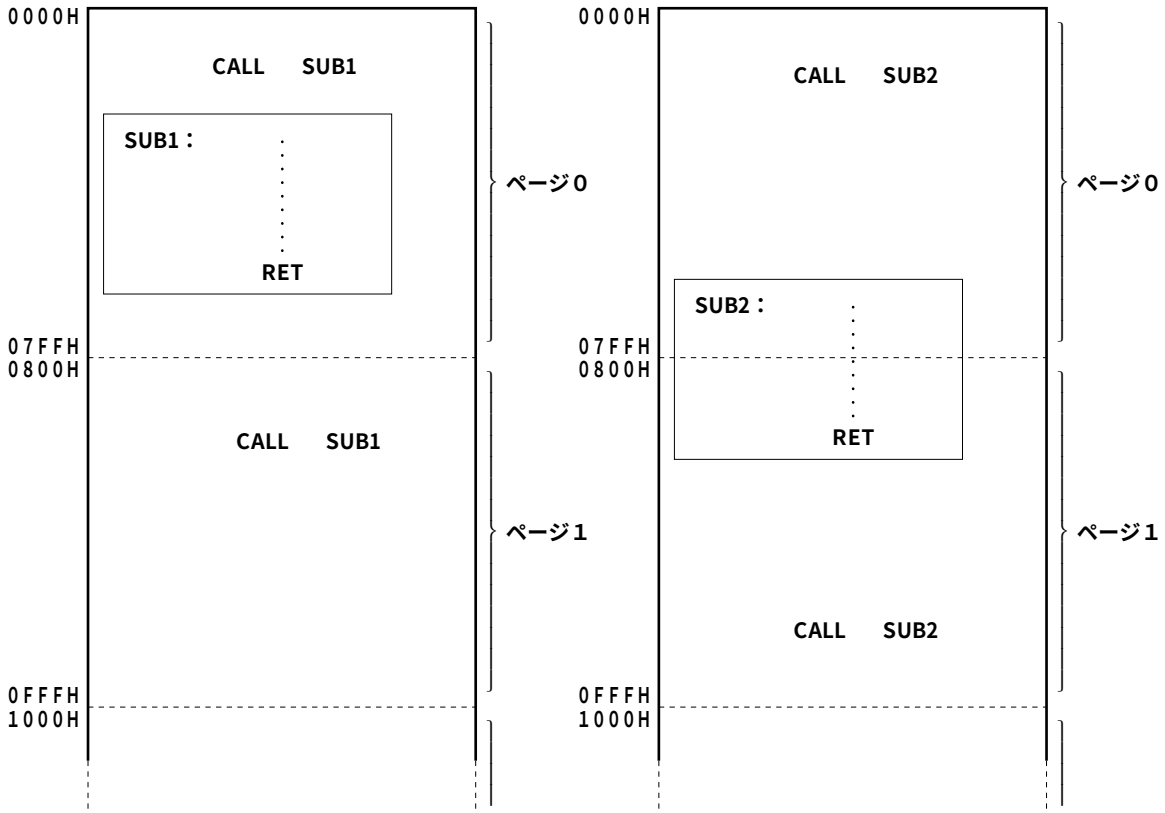
例２



④ 注　　意

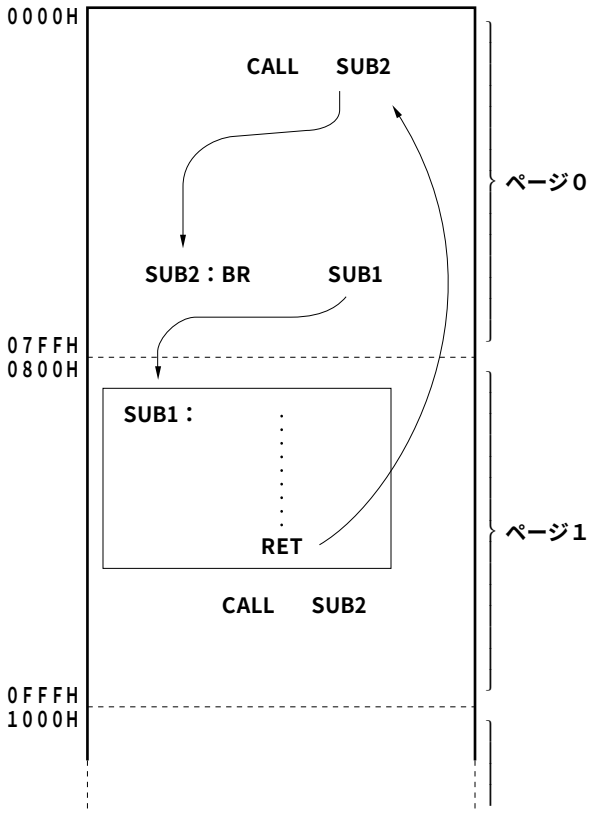
CALL命令を使用する際，その呼び出し番地，つまりサブルーチンの先頭番地はページ０内（0000H-07FFH）に置く必要があります。ページ０以外に先頭番地のあるサブルーチンを呼び出す場合にはCALL @AR命令を使用してください。

サブルーチンの先頭番地がページ０内にある場合



以上のように，サブルーチンの先頭番地がページ０内にあれば，サブルーチンの終了番地（RET, RETSK命令）はページ０以外であってもかまいません。

サブルーチンの先頭番地がページ０内にある限り，CALL命令はページの概念なく使用することができます。しかし，プログラム作成上，サブルーチンの先頭番地をページ０内に置くことができない場合は，次の手法が有効となります。



つまり，ページ0内にBR命令を設け，このBR命令を介して実際のサブルーチンに分岐する手法です。

（2）CALL @AR　　Call subroutine specified by address register

① 命令コード

00111	000	0101	0000

② 機　　能

SP←SP－1,

ASR←PC,

PC←AR

プログラム・カウンタPCの値をスタックに格納したのち，アドレス・レジスタARが指定するアドレスから始まるサブルーチンに分岐します。

③ 例1

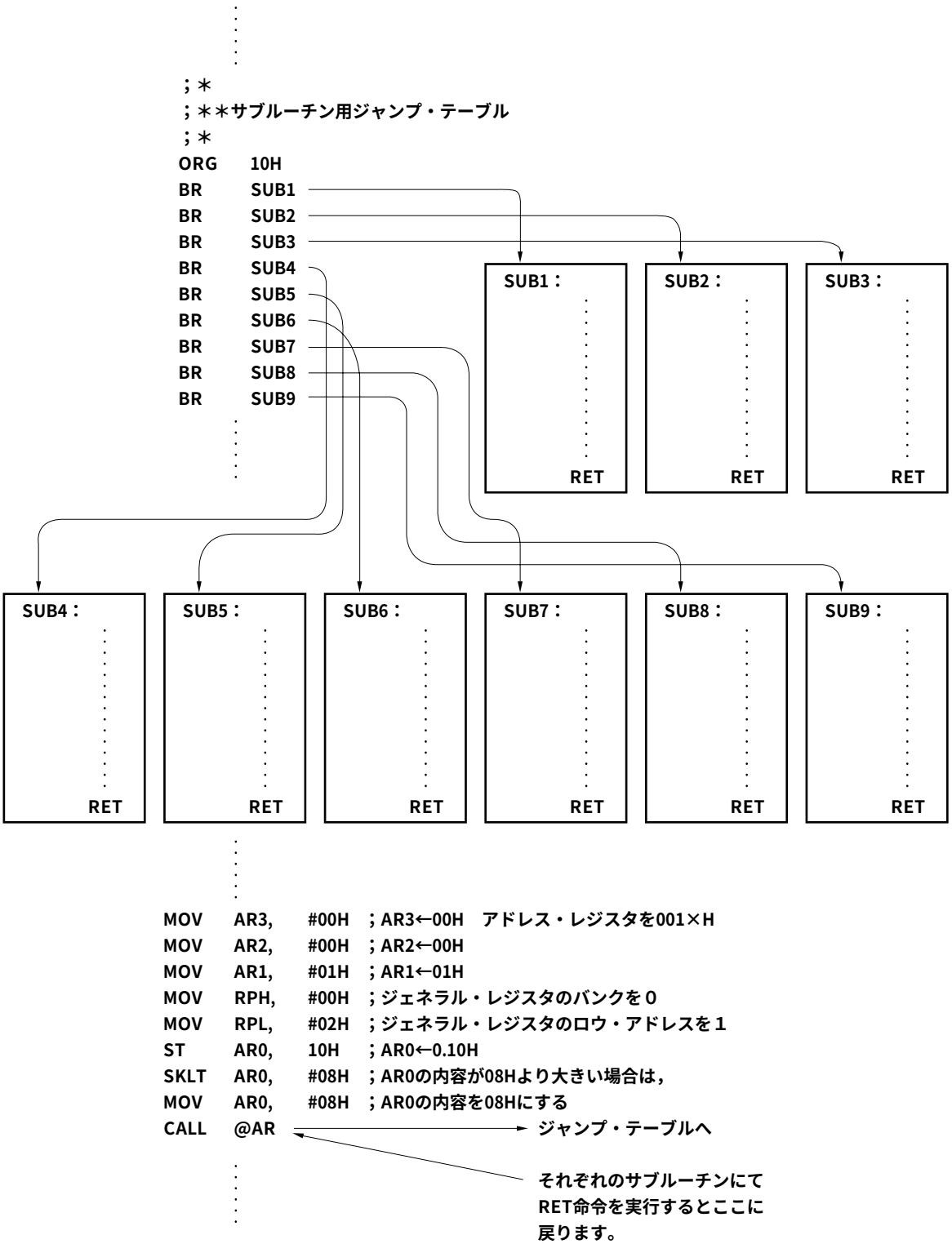
アドレス・レジスタAR（AR0-AR3）に0020Hを設定し，CALL @AR命令により0020H番地のサブルーチンをコールします。

```
MOV     AR3, #00H          ; AR3←00H
MOV     AR2, #00H          ; AR2←00H
MOV     AR1, #02H          ; AR1←02H
MOV     AR0, #00H          ; AR0←00H
CALL    @AR                ; 0020H番地のサブルーチンをコール
```

例2

データ・メモリの0.10H番地の内容によって次のサブルーチンをコールします。

0.10Hの内容		サブルーチン名
00H	→	SUB1
01H	→	SUB2
02H	→	SUB3
03H	→	SUB4
04H	→	SUB5
05H	→	SUB6
06H	→	SUB7
07H	→	SUB8
08H-0FH	→	SUB9



④ 注　　意

アドレス・レジスタ（AR0-AR3）は品種により使用できるビット数が異なりますので，使用する際は，各品種ごとのデータ・シートを参照してください。

★　（3）SYSCAL entry　　Call system segment entry address

① 命令コード

00111	entry$_H$	0000	entry$_L$

② 機　　能

SP←（SP）－1, ASR←PC＋1,

SGR←SYSSEG, PAGE←0, PC（10-8）←entry$_H$,

PC（7-4）←0, PC（3-0）←entry$_L$

プログラム・カウンタ（PC）の値をインクリメントし，スタックに格納したのち，システム・セグメントのページ０にあるentryで示されたサブルーチンに分岐します。

この命令で呼び出せるサブルーチンは，システム・セグメントのページ０にあるシステム・セグメント・エントリ・アドレスの256ステップになります。

③ 例１

```
MAIN：
        ⋮
        SYSCAL 34H
        ⋮
CSEG    n
ORG     304H
        ⋮
        RET
```

（n：システム・セグメント）

例２

```
MAIN：
        ⋮
        SYSCAL.D.L.（（ENTRY SHR 4 AND 0070H）OR（ENTRY AND 000FH））
        ⋮
ENTRY：
        ⋮
        RET
```

④ 注　　意

SYSCAL命令のオペランドには，レーベル型のシンボルではなく，データ型のシンボルを記述します。また，オペランドの値が７ビットを超えると，アセンブラ（RA17K）はエラーを発生します。

なお，上記③の例２では，“ ENTRY ”のアドレスがシステム・セグメント・エントリ・アドレス内にない場合でも，アセンブラ（RA17K）はエラーを発生しません。この場合，SYSCAL命令による分岐先はユーザが意図したアドレスと異なるアドレスになりますので，充分に注意してディバグを行う必要があります。

（4）RET　　Return to the main program from subroutine

① 命令コード

10	8 7	4 3	0
00111	000	1110	0000

② 機　　能

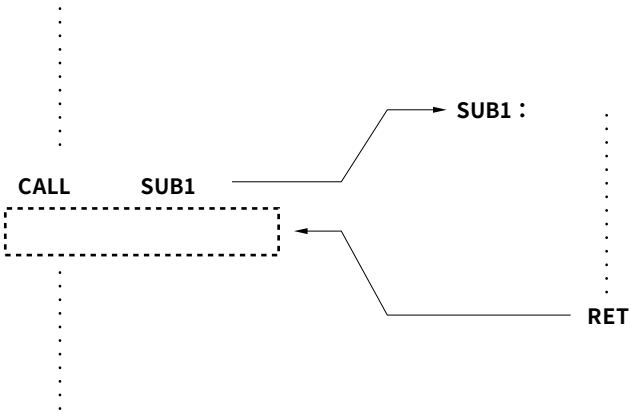
PC←ASR,

SP←SP＋１

サブルーチンからメイン・プログラムへ戻るための命令です。

CALL命令でスタックに退避した戻り番地を，プログラム・カウンタに復帰させます。

③ 例



（5）RETSK　　Return to the main program then skip next instruction

① 命令コード

00111	001	1110	0000

② 機　　能

PC←ASR, SP←SP＋１and skip

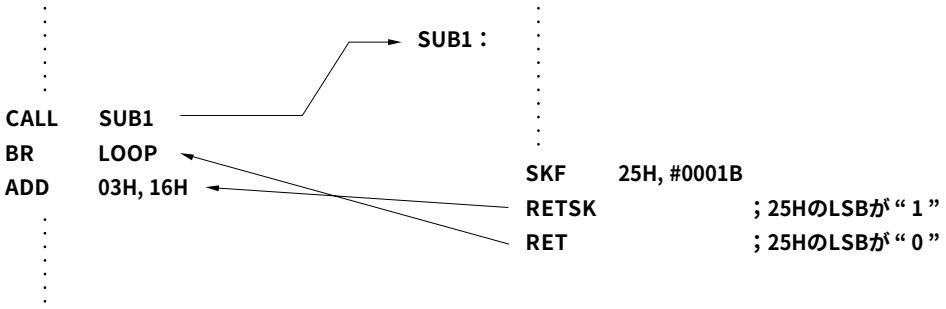
サブルーチンからメイン・プログラムへ戻るための命令です。

★ CALL命令の次の命令をスキップします（NOP命令として実行します）。

すなわち，CALL命令でスタックに退避した戻り番地をプログラム・カウンタPCに復帰させたのちプログラム・カウンタをインクリメントします。

③ 例

データ・メモリ（RAM）の25H番地のLSB（最下位ビット）の内容が“０”のときはRET命令を実行し，CALL命令の次の命令に戻り，“１”のときはRETSK命令を実行し，CALL命令の次の次の命令（ここではADD 03H, 16H）に戻ります。



（６）RETI　　Return to the main program from interrupt service routine

① 命令コード

00111	100	1110	0000

② 機　　能

PC←ASR, INTR←INTSK, SP←SP＋１

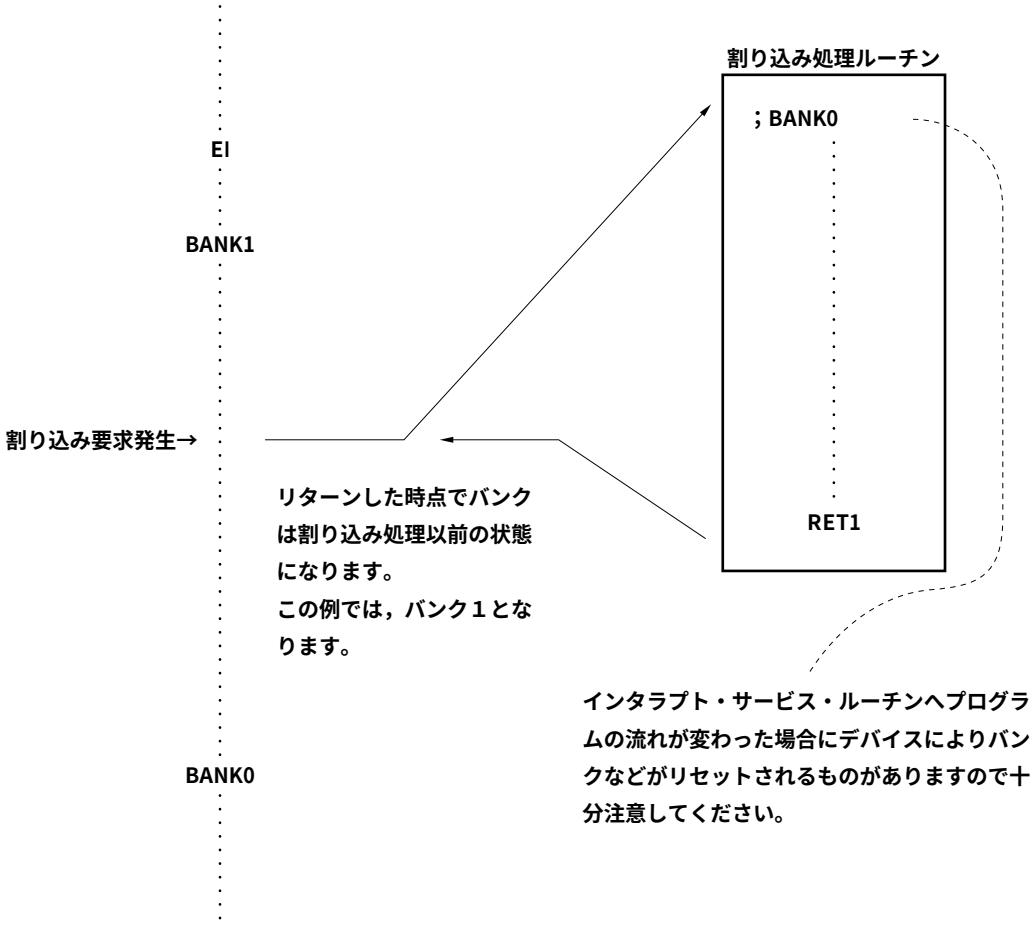
割り込み処理プログラムからメイン・プログラムへ戻るための命令です。

ベクタ割り込みでスタックに退避した戻り番地をプログラム・カウンタに復帰させます。

また，システム・レジスタの一部もベクタ割り込み発生以前の状態に戻ります。

③ 例

データ・メモリがバンク１にある場合にベクタ割り込みが発生し，割り込み処理でデータ・メモリのバンク０を使うので，現在のバンク・アドレスを退避しておく必要がある場合。



④　**注意１**

割り込みにより自動的に退避される（RETI命令で復帰できる）システム・レジスタの内容は品種によりそれぞれ異なりますので，各品種ごとのデータ・シートを参照してください。

注意２

通常のサブルーチンにおいてRET命令の代わりにRETI命令を使用した場合，戻り番地に戻った時点でバンクなど（割り込みにより退避されるもの）が割り込みスタックの内容に書き換わってしまうため，どのような状態になるか不明の場合がありますので，サブルーチンからリターンする場合は必ずRET（またはRETSK）命令を使用してください。

15.5.10　割り込み命令

（1）EI　　Enable Interrupt

① 命令コード

00111	000	1111	0000

② 機　　能

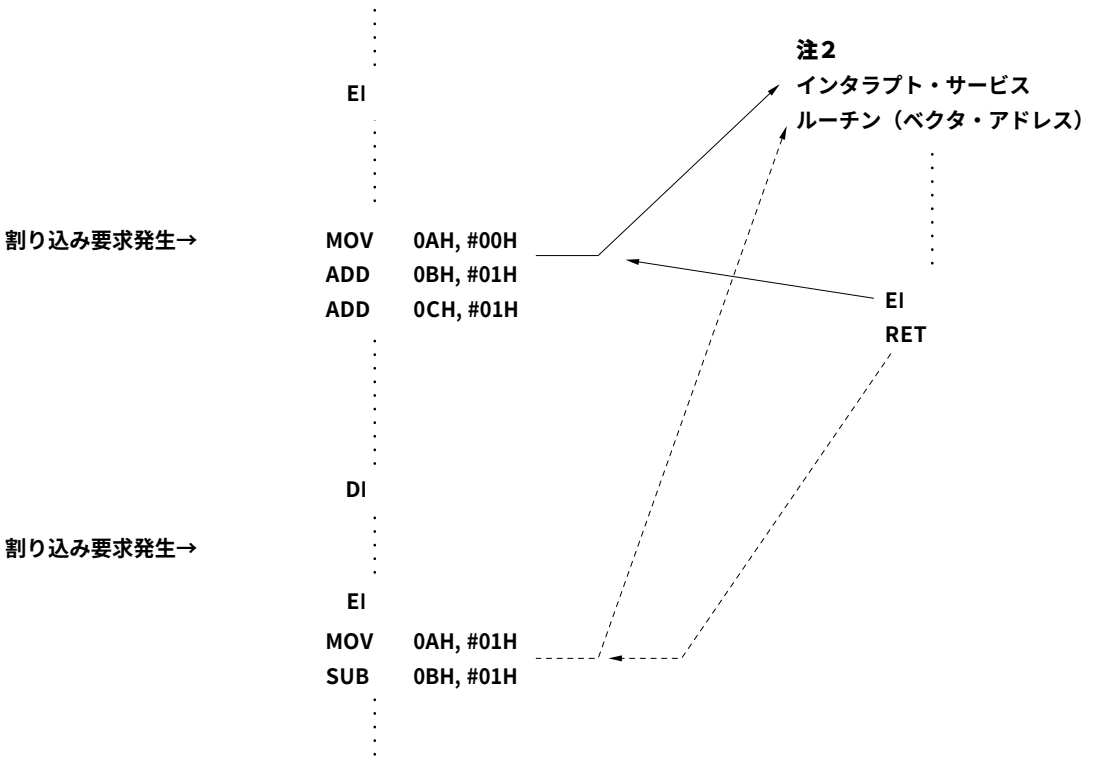
INTEF←1

ベクタ割り込みを許可する命令です。

割り込みはEI命令の次の命令を実行したあとに許可されます。

③ 例1

次の例で分かるように割り込み要求が受け付けられると，受け付け後の次の命令（プログラム・カウンタを操作する命令を除く）の実行が完了してからベクタ・アドレスに流れが移ります。[注1]

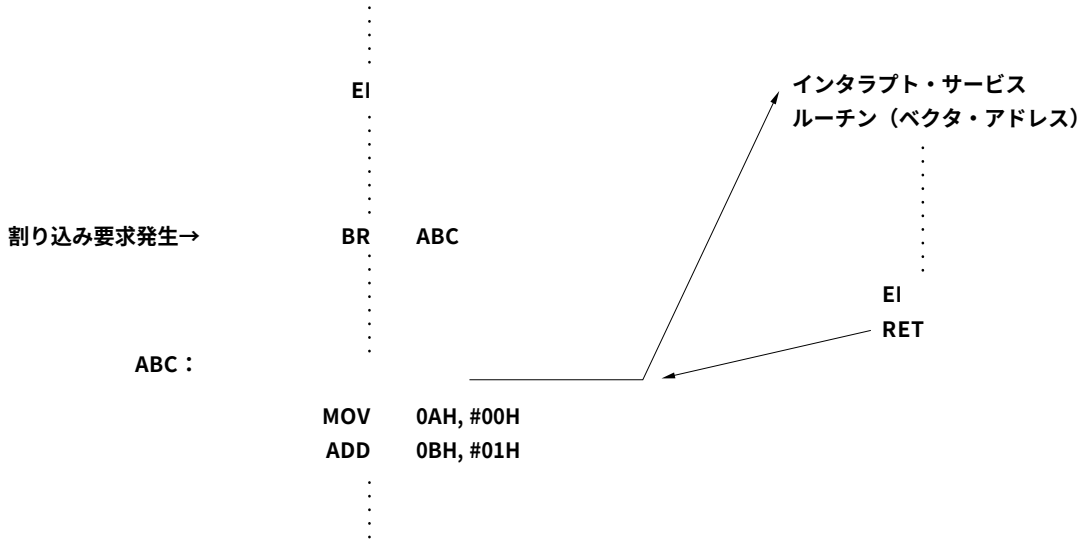


注1．ベクタ・アドレスは，受け付けられる割り込みにより異なりますので，詳細については，使用する製品のデータ・シートを参照してください。

2．ここで受け付けられる割り込み（EI命令実行後割り込み要求が発生しインタラプト・サービス・ルーチンに流れが移る）とはその割り込みに対する割り込み許可フラグ（IP×××）がセットされているものとします。各割り込み許可フラグがセットされていない状態で，EI命令実行後，割り込み要求が発生してもプログラムの流れは変化しません（割り込みは受け付けられません）。ただし，割り込み要求フラグ（IRQ×××）はセットされますので，割り込み許可フラグがセットされた時点で受け付けられます。詳細については，使用する製品のデータ・シートを参照してください。

例2

プログラム・カウンタPCを操作する命令実行中に受け付けられた割り込み要求で発生する割り込みの例を次に示します。



（2）DI　　Disable Interrupt

① 命令コード

00111	001	1111	0000

② 機　　能

★　　INTEF←0

ベクタ割り込みを禁止する命令です。

③ 例

（1）EIの例1を参照。

15.5.11　その他の命令

（1）STOP s　　Stop CPU and release by condition s

① 命令コード

			3	0
00111	010	1111	s	

② 機　　能

システム・クロックを停止させ，デバイスをSTOPモードにします。

デバイスをSTOPモードにすることにより消費電流を最小限に抑えることができます。

STOPモードを解除する条件をオペランド（ｓ）で指定します。

ストップ解除条件（ｓ）は，品種により異なりますので，使用する際は，各品種ごとのデータ・シートを参照してください。

（2）HALT h　　Halt CPU and release by condition h

① 命令コード

			3	0
00111	011	1111	h	

② 機　　能

デバイスをHALTモードにします。

デバイスをHALTモードにすることにより消費電流を低減させることができます。

HALTモードを解除する条件をオペランド（ｈ）で指定します。

ホールト解除条件（ｈ）は，品種により異なりますので，使用する際は，各品種ごとのデータ・シートを参照してください。

（3）NOP　　No operation

① 命令コード

00111	100	1111	0000

② 機　　能

何もせず１マシン・サイクル費やす命令です。

〔メ　モ〕

付録A　開発ツール

μPD170××シリーズのプログラム開発するために，次の開発ツールを用意しています。

★ A.1　ハードウエア

名　　称	概　　要
インサーキット・エミュレータ （IE-17K IE-17K-ET注1 EMU-17K注2）	IE-17K, IE-17K-ET, EMU-17Kは，17Kシリーズ共通のインサーキット・エミュレータです。IE-17K, IE-17K-ETは，ホスト・マシンであるPC-9800シリーズまたはIBM PC/AT™とRS-232-Cを介して接続して使用します。 各品種専用のシステム・エバリュエーション・ボード（SEボード）と組み合わせて使用することにより，その品種に対応したエミュレータとして動作します。マン・マシン・インタフェース・ソフトウエアである*SIMPLEHOST*を使用すると，さらに高度なディバグ環境を実現できます。 なお，EMU-17Kは，データ・メモリの内容をリアルタイムで確認できるという機能を備えています。
SEボード	単体でシステム評価に，インサーキット・エミュレータと組み合わせてディバグに使用します。各品種ごとに異なります。
エミュレーション・プローブ	変換ソケットとともに使用し，SEボードとターゲット・システムを接続します。各品種ごとに異なります。
交換ソケット	エミュレーション・プローブとターゲット・システムを接続するために使用します。デバイスのパッケージによって異なります。
PROMプログラマ （AF-9703注3，4 AF-9704注3，4 AF-9705注4 AF-9706注4）	プログラムアダプタを接続することにより，PROM製品をプログラミングすることができます。
プログラムアダプタ	PROMプログラマと組み合わせて使用します。各品種ごとに異なります。

注1．廉価版：電源外付けタイプ

2．株式会社内藤電誠町田製作所の製品です。詳細につきましては，株式会社内藤電誠町田製作所（川崎（044）822-3813）までお問い合わせください。

3．製造中止。

4．安藤電気株式会社の製品です。詳細につきましては，安藤電気株式会社（東京（03）3733-1151）までお問い合わせください。

★ A.2　ソフトウエア

名　称	概　要	ホスト・マシン	OS	供給媒体	オーダ名称
17Kアセンブラ（RA17K）	RA17Kは17Kシリーズ共通に使用できるアセンブラです。デバイスのプログラム開発には，このRA17Kとデバイス・ファイルを組み合わせて使用します。	PC-9800シリーズ	日本語版 Windows™	3.5インチ 2HD	μSAA13RA17K
		IBM PC/AT およびその互換機	日本語版 Windows	3.5インチ 2HC	μSAB13RA17K
			英語版 Windows		μSBB13RA17K
デバイス・ファイル（AS170××）	17Kシリーズ共通のアセンブラ（RA17K）と組み合わせて使用します。各品種ごとに異なります。	PC-9800シリーズ	日本語版 Windows	3.5インチ 2HD	μSAA13AS170××
		IBM PC/AT およびその互換機	日本語版 Windows	3.5インチ 2HC	μSAB13AS170××
			英語版 Windows		μSBB13AS170××
サポート・ソフトウエア（*SIMPLEHOST*）	*SIMPLEHOST*はインサーキット・エミュレータとパーソナル・コンピュータを用いてプログラム開発を行うときにWindows上でマン・マシン・インタフェースを行うソフトウエアです。	PC-9800シリーズ	日本語版 Windows	3.5インチ 2HD	μSAA13ID17K
		IBM PC/AT およびその互換機	日本語版 Windows	3.5インチ 2HC	μSAB13ID17K
			英語版 Windows		μSBB13ID17K

備考　××：各品種ごとに異なります。

付録B　マスクROMの発注方法

プログラム開発が完了しましたら，次の手順でマスクROM品を発注してください。

（1）マスクROM発注の予約

当社特約店あるいは当社販売部門に，マスクROM発注の予定を連絡してください。あらかじめ連絡をいただかないと納品が遅れる場合があります。

（2）発注媒体の作成

マスクROM発注用の媒体はUV-EPROMです。

★ まず，アセンブラ（RA17K）のアセンブル・オプションに/PROMを追加し，マスクROM発注用ヘキサ・ファイル（**拡張子が .PROになります**）を作成してください。

次に，マスクROM発注用ヘキサ・ファイルをUV-EPROMに書き込んでください。

なお，UV-EPROMで発注する場合には同じ内容のUV-EPROMを３個作成してください。

注意　マスクROM発注用ヘキサ・ファイルの拡張子が.ICEでは発注できません。

（3）必要書類の作成

マスクROM発注にあたって，下記の書類を記入してください。

・マスク式ROM発注書

・マスク式ROM発注チェック・シート

（4）発　　注

（2）で作成した媒体と（3）で記入した書類とをまとめて，発注予約日までに当社特約店または当社販売部門に渡してください。

★ **備考**　詳しくはインフォメーション資料「ROMコードの発注方法」（資料番号 C10302J）をご覧ください。

〔メ　モ〕

付録C　命令索引

★ C.1　命令索引（機能別）

【加算命令】

ADD　r, m　…　218
ADD　m, ＃n4　…　221
ADDC　r, m　…　223
ADDC　m, ＃n4　…　226
INC　AR　…　228
INC　IX　…　229

【減算命令】

SUB　r, m　…　231
SUB　m, ＃n4　…　233
SUBC　r, m　…　235
SUBC　m, ＃n4　…　238

【論理演算命令】

OR　r, m　…　240
OR　m, ＃n4　…　240
AND　r, m　…　241
AND　m, ＃n4　…　242
XOR　r, m　…　243
XOR　m, ＃n4　…　244

【判断命令】

SKT　m, ＃n　…　245
SKF　m, ＃n　…　246

【比較命令】

SKE　m, ＃n4　…　247
SKNE　m, ＃n4　…　247
SKGE　m, ＃n4　…　248
SKLT　m, ＃n4　…　249

【回転命令】

RORC　r　…　250

【転送命令】

LD　r, m　…　252
ST　m, r　…　255
MOV　@r, m　…　258
MOV　m, @r　…　260
MOV　m, ＃n4　…　263
MOVT　DBF, @AR　…　263
PUSH　AR　…　266
POP　AR　…　267
PEEK　WR, rf　…　268
POKE　rf, WR　…　269
GET　DBF, p　…　271
PUT　p, DBF　…　272

【分岐命令】

BR　addr　…　273
BR　@AR　…　276

【サブルーチン命令】

CALL　addr　…　278
CALL　@AR　…　281
SYSCAL　entry　…　283
RET　…　284
RETSK　…　284
RETI　…　285

【割り込み命令】

EI　…　287
DI　…　288

【その他の命令】

STOP　s　…　289
HALT　h　…　289
NOP　…　289

C.2　命令索引（アルファベット順）

【A】

ADD　m, ＃n4 … 221
ADD　r, m … 218
ADDC　m, ＃n4 … 226
ADDC　r, m … 223
AND　m, ＃n4 … 242
AND　r, m … 241

【B】

BR　addr … 273
BR　@AR … 276

【C】

CALL　addr … 278
CALL　@AR … 281

【D】

DI … 288

【E】

EI … 287

【G】

GET　DBF, p … 271

【H】

HALT　h … 289

【I】

INC　AR … 228
INC　IX … 229

【L】

LD　r, m … 252

【M】

MOV　m, ＃n4 … 263
MOV　m, @r … 260
MOV　@r, m … 258
MOVT　DBF, @AR … 263

【N】

NOP … 289

【O】

OR　m, ＃n4 … 240
OR　r, m … 240

【P】

PEEK　WR, rf … 268
POKE　rf, WR … 269
POP　AR … 267
PUSH　AR … 266
PUT　p, DBF … 272

【R】

RET … 284
RETI … 285
RETSK … 284
RORC　r … 250

【S】

SKE　m, ＃n4 … 247
SKF　m, ＃n … 246
SKGE　m, ＃n4 … 248
SKLT　m, ＃n4 … 249
SKNE　m, ＃n4 … 247
SKT　m, ＃n … 245
ST　m, r … 255
STOP　s … 289
SUB　m, ＃n4 … 233
SUB　r, m … 231
SUBC　m, ＃n4 … 238
SUBC　r, m … 235
SYSCAL　entry … 283

【X】

XOR　m, ＃n4 … 244
XOR　r, m … 243

付録D　改版履歴

これまでの改版履歴を次に示します。なお，適用箇所は各版での章を示します。

版　数	内　容	適用箇所
第2版	アセンブラを変更（AS17K→RA17K）	全　般
	インサーキット・エミュレータ　IE-17K-ETを追加	全　般
	旧版第1編　第14章　ワン・タイムPROM品を削除	旧版第14章　ワン・タイムPROM品
	はじめにの関連資料の表を変更	はじめに
	第1章　概　　説から μPD170××の製品展開の図と機能一覧表を削除	第1章　概　　説
	6.7.3　ジェネラル・レジスタ・ポインタ使用時の注意を追加	第6章　システム・レジスタ（SYSREG）
	図8－1　ALUブロックの構成にプログラム・ステータス・ワード（PSWORD）とステータス・フリップフロップの関係の図を追加	第8章　演算論理ユニット（ALU）
	表8－1　ALU処理命令一覧に命令の動作と説明を追加	
	8.2.3　ステータス・フリップフロップの機能に次の説明を追加 （1）Zフラグ （2）CYフラグ （3）CMPフラグ （4）BCDフラグ	
	表8－2　2進4ビット演算結果とBCD演算結果を変更	
	表8－3　算術演算の種類を追加	
	表8－4　論理演算を追加	
	表8－6　ビット判断命令を追加	
	表8－7　比較判断命令を追加	
	9.4.2　レジスタ・ファイルのシンボル定義と予約語に例3を追加	第9章　レジスタ・ファイル（RF）
	12.2.6　割り込みイネーブル・フリップフロップ（INTE）に説明を追加	第12章　割り込み機能
	次の項に備考を追加 13.4.3　キー入力によるホールト解除 13.4.4　タイマ・キャリー（ベーシック・タイマ0キャリー）によるホールト解除	第13章　スタンバイ機能
	13.4.5　割り込みによるホールト解除に備考と注意を追加	
	旧版　12.6　ホールト状態とクロックストップ状態時の消費電流を削除	
	14.4　パワーオン・リセットに説明を追加	第14章　リセット機能
	15.5.9（3）SYSCAL entryを追加	第15章　命令セット
	A.1　ハードウエアとA.2　ソフトウエアを変更	付録A　開発ツール
	C.1　命令索引（機能別）を追加	付録C　命令索引
	付録D　改版履歴を追加	付録D　改版履歴

お問い合わせ先

【技術的なお問い合わせ先】

ＮＥＣ半導体テクニカルホットライン（インフォメーションセンター）

電　話　：044−548−8899
FAX　：044−548−7900
E-mail ：s-info@saed.tmg.nec.co.jp

【営業関係お問い合わせ先】

半導体第一販売事業部 半導体第二販売事業部 半導体第三販売事業部		〒108-8001	東京都港区芝５－７－１　（日本電気本社ビル）	(03)3454−1111
中部支社	半導体第一販売部 半導体第二販売部	〒460-8525	愛知県名古屋市中区錦１－17－１　（日本電気中部ビル）	(052)222−2170 (052)222−2190
関西支社	半導体第一販売部 半導体第二販売部 半導体第三販売部	〒540-8551	大阪府大阪市中央区城見１－４－24　（日本電気関西ビル）	(06) 945−3178 (06) 945−3200 (06) 945−3208

北海道支社	札　幌	(011)231−0161	宇都宮支店	宇都宮	(028)621−2281	北陸支社	金　沢	(076)232−7303
東北支社	仙　台	(022)267−8740	小山支店	小　山	(0285)24−5011	富山支店	富　山	(0764)31−8461
岩手支店	盛　岡	(019)651−4344	甲府支店	甲　府	(0552)24−4141	福井支店	福　井	(0776)22−1866
郡山支店	郡　山	(0249)23−5511	長野支社	松　本	(0263)35−1662	京都支社	京　都	(075)344−7824
いわき支店	いわき	(0246)21−5511	静岡支社	静　岡	(054)254−4794	神戸支社	神　戸	(078)333−3854
長岡支店	長　岡	(0258)36−2155	立川支社	立　川	(042)526−5981,6167	中国支社	広　島	(082)242−5504
水戸支店	水　戸	(029)226−1717	埼玉支社	大　宮	(048)649−1415	鳥取支店	鳥　取	(0857)27−5311
土浦支店	土　浦	(0298)23−6161	千葉支社	千　葉	(043)238−8116	岡山支店	岡　山	(086)225−4455
群馬支店	高　崎	(027)326−1255	神奈川支社	横　浜	(045)682−4524	松山支店	松　山	(089)945−4149
太田支店	太　田	(0276)46−4011	三重支店	津	(059)225−7341	九州支社	福　岡	(092)261−2806

アンケート記入のお願い

お手数ですが，このドキュメントに対するご意見をお寄せください。今後のドキュメント作成の参考にさせていただきます。

［ドキュメント名］μPD170××シリーズ　ユーザーズ・マニュアル
（U13262JJ2V0UM00（第２版））

［お名前など］（さしつかえのない範囲で）

御社名（学校名，その他）　（　　）
ご住所　（　　）
お電話番号　（　　）
お仕事の内容　（　　）
お名前　（　　）

１．ご評価（各欄に○をご記入ください）

項　　目	大変良い	良　い	普　通	悪　い	大変悪い
全体の構成					
説明内容					
用語解説					
調べやすさ					
デザイン，字の大きさなど					
その他（　　）					
（　　）					

２．わかりやすい所（第　章，第　章，第　章，第　章，その他　　）
理由［　　］

３．わかりにくい所（第　章，第　章，第　章，第　章，その他　　）
理由［　　］

４．ご意見，ご要望

５．このドキュメントをお届けしたのは
NEC販売員，特約店販売員，NEC半導体ソリューション技術本部員，
その他（　　）

ご協力ありがとうございました。
下記あてにFAXで送信いただくか，最寄りの販売員にコピーをお渡しください。

NEC半導体インフォメーションセンター
FAX：（044）548-7900

キリトリ