組み込みマイコンの仕組みを理解しよう

第2回

FRマイコンの命令セットとメモリ・アーキテクチャ

平石 郁雄

本連載では，富士通の32ビット・マイコン「FRファミリ」を例に，マイコンの仕組みや使い方について解説している．今回は，命令セットとメモリ・アーキテクチャを中心に説明する．（編集部）

高級言語（C言語など）に慣れ親しんでいても，アセンブリ言語を使いこなしている人は少ないのではないでしょうか．アセンブリ言語は，機械語とほとんど等価な命令の集まり（命令セット）で，CPUの種類に依存しています．最近では，アセンブリ言語で作成したプログラムを読解できる技術者がいないという話も聞こえてきます．今日では高級言語を機械語に変換するコンパイラ（**図1**）と呼ばれるツールの性能が向上したこともあり，組み込みソフトウェア開発者のほとんどが，アセンブリ言語ではなく，CPUの種類を意識する必要がない高級言語でプログラム開発を行っています．

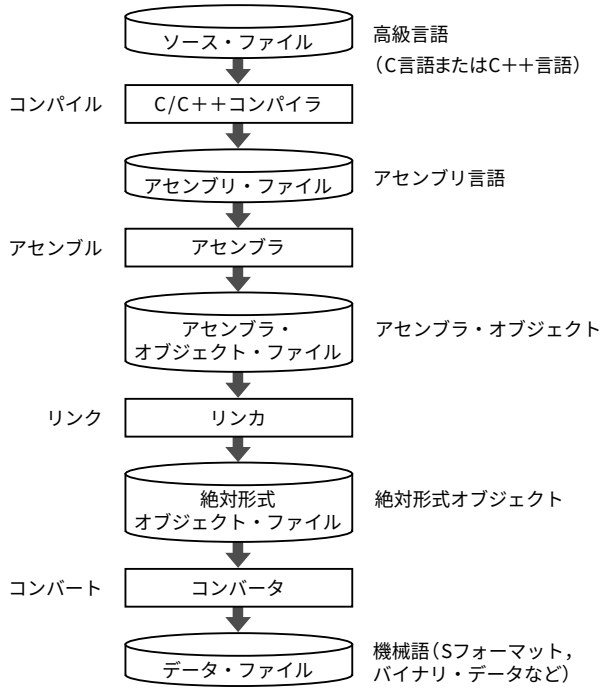


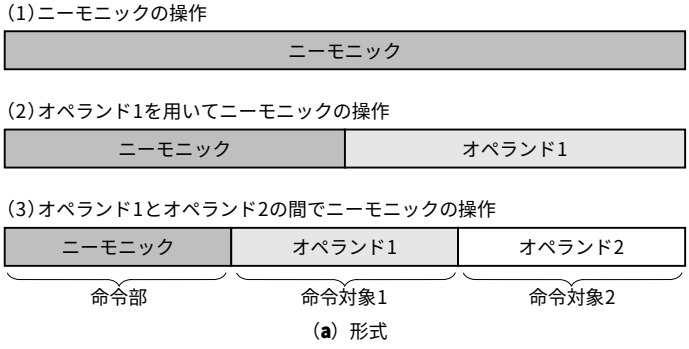
図1　C/C++コンパイルの例

C/C++コンパイラは，高級言語（C言語またはC++言語）ファイルをコンパイルしてアセンブリ言語ファイルを生成する．アセンブラは，アセンブリ言語ファイルをアセンブルしてアセンブラ・オブジェクトを生成する．リンカは，アセンブラが生成した複数のアセンブラ・オブジェクト・ファイルをリンクして，絶対形式オブジェクト・ファイルを生成する．コンバータは，絶対形式オブジェクト・ファイルを変換して，データ・ファイル（機械語）を生成する．

それではアセンブリ言語を学習する必要はないかというと，組み込み系のソフトウェア開発はそう簡単にはいきません．アプリケーションによっては，使用しているマイコンの最高性能を引き出すために，アセンブリ言語でゴリゴリと記述することを強いられます．また，ハードウェアの動きを正確に知るためには，アセンブリ言語を読み取る能力が要求されます．そのため，アセンブリ言語を知ることは，効率的な開発を行う上で必要なことと言えます．

● ニーモニック＋オペランドから構成される

アセンブリ言語は，「ニーモニック」と「オペランド」か



（**a**）形式

(1)ニーモニックの操作

```
<ニーモニック>
  RET
```

オペレーション：RPの値をPC（プログラム・カウンタ）へ格納する

(2)オペランド1を用いてニーモニックの操作

```
<ニーモニック><オペランド1>
  JMP          @R1
```

オペレーション：R1(オペランド1)の値をPCへ格納する

(3)オペランド1とオペランド2の間でニーモニックの操作

```
<ニーモニック><オペランド1><オペランド2>
  ADD          R1,          R2
```

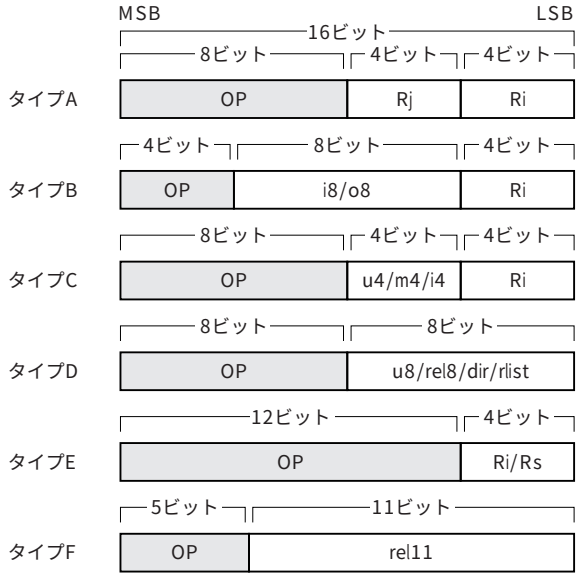
オペレーション：　R1（オペランド1）とR2（オペランド2）の値を加算して，R2へ格納する

（**b**）記述例

図2　命令の記述形式

FRの命令セットは，ニーモニックとオペランドを組み合わせた3種類の命令記述形式で構成している．

KEYWORD――マイコン，FRファミリ，アセンブリ言語，命令セット，加減算命令，乗除算命令，即値データ転送命令，シフト演算命令，比較命令，レジスタ間転送命令，専用レジスタ転送命令，メモリ・アーキテクチャ



（**a**）フォーマット

Ri/Rj	レジスタ
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	R12
1101	R13
1110	R14
1111	R15

RS	レジスタ
0000	TBR
0001	RP
0010	SSP
0011	USP
0100	MDH
0101	MDL
0110	予約
0111	予約
1000	予約
1001	予約
1010	予約
1011	予約
1100	予約
1101	予約
1110	予約
1111	予約

（**b**）汎用/専用レジスタのフィールド・ビット・パターン

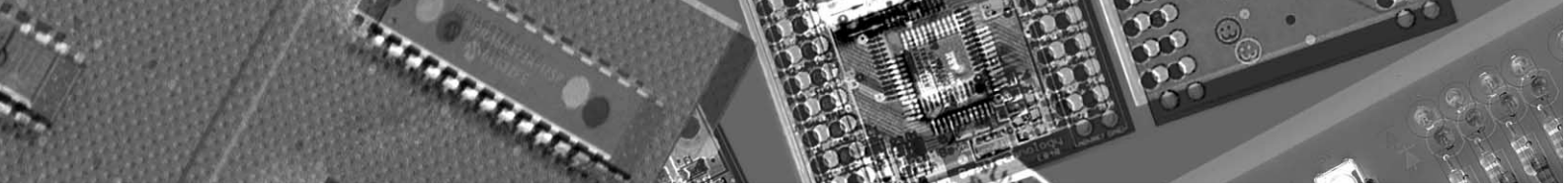
上位4ビット

下位4ビット	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	LD @(R13, Rj),Ri	ST Ri, @(R13,Rj)	LD@(R14,disp10),R	STRi,@(R14,disp10)	LDUH@(R14,disp9),Ri	STHRi,@(R14,disp9)	LDUB@(R14,disp8),Ri	STBRi,@(R14,disp8)	BANDL #u4,@Ri	BORL #u4,@Ri	ADDN #i4,Ri	LSR #u4,Ri	LDI:8 #i8,Ri	CALL label12	BRA label9	BRA:D label9
1	LDUH @ (R13,Rj),Ri	STH Ri, @(R13,Rj)							BANDH #u4,@Ri	BORH #u4,@Ri	ADDN2 #i4,Ri	LSR2 #u4,Ri			BNO label9	BNO:D label9
2	LDUB @ (R13,Rj),Ri	STB Ri, @(R13,Rj)							AND Rj,Ri	OR Rj,Ri	ADDN Rj,Ri	LSR Rj,Ri			BEQ label9	BEQ:D label9
3	LD @(R15, udisp6),Ri	ST Ri, @(R15,ud6)							ANDCCR #u8	ORCCR #u8	ADDSP #s10	MOV Ri,Rs			BNE label9	BNE:D label9
4	LD @Rj,Ri	ST Ri,@Rj							AND Rj, @Ri	OR Rj, @Ri	ADD #i4,Ri	LSL #u4,Ri			BC label9	BC:D label9
5	LDUH @Rj,Ri	STH Ri, @Rj							ANDH Rj,@Ri	ORH Rj, @Ri	ADD2 #i4,Ri	LSL2 #u4,Ri			BNC label9	BNC:D label9
6	LDUB @Rj,Ri	STB Ri, @Rj							ANDB Rj,@Ri	ORB Rj, @Ri	ADD Rj,Ri	LSL Rj,Ri			BN label9	BN:D label9
7	E format	E format							STILM #u8	E format	ADDC Rj,Ri	MOV Rs,Ri			BP label9	BP:D label9
8	DMOV @ d10,R13	DMOV R13,@d10							BTSTL #u4,@Ri	BEORL #u4,@Ri	CMP #i4,Ri	ASR #u4,Ri		CALL:D label12	BV label9	BV:D label9
9	DMOVH @d9,R13	DMOVH R13,@d9							BTSTH #u4,@Ri	BEORH #u4,@Ri	CMP2 #i4,Ri	ASR2 #u4,Ri			BNV label9	BNV:D label9
A	DMOVB @d8,R13	DMOVB R13,@d8							XCHB @Rj,Ri	EOR Rj,Ri	CMP Rj,Ri	ASR Rj,Ri			BLT label9	BLT:D label9
B	DMOV @ d10,@－R15	DMOV @ R15＋,@d10							MOV Rj,Ri	LD:20 #i20,Ri	MULU Rj,Ri	MULUH Rj,Ri			BGE label9	BGE:D label9
C	DMOV @ d10,@R13＋	DMOV @ R13＋,@d10							LDM0 (reglist)	EOR Rj,@Ri	SUB Rj,Ri	LDRES @Ri＋,#u4			BLE label9	BLE:D label9
D	DMOVH @d9,@R13＋	DMOVH @R13＋,@d9							LDM1 (reglist)	EORH Rj,@Ri	SUBC Rj,Ri	STRES #u4,@Ri＋			BGT label9	BGT:D label9
E	DMOVB @d8,@R13＋	DMOVB @R13＋,@d8							STM0 (reglist)	EORB Rj,@Ri	SUBN Rj,Ri				BLS label9	BLS:D label9
F	ENTER #u10	INT #u8							STM1 (reglist)	E format	MUL Rj,Ri	MULH Rj,Ri			BHI label9	BHI:D label9

（**c**）命令マップ

図3　命令フォーマット

命令コードの基本長が16ビットなので，限られた資源を有効に活用するため，メモリマップがぎっしり詰まっている．



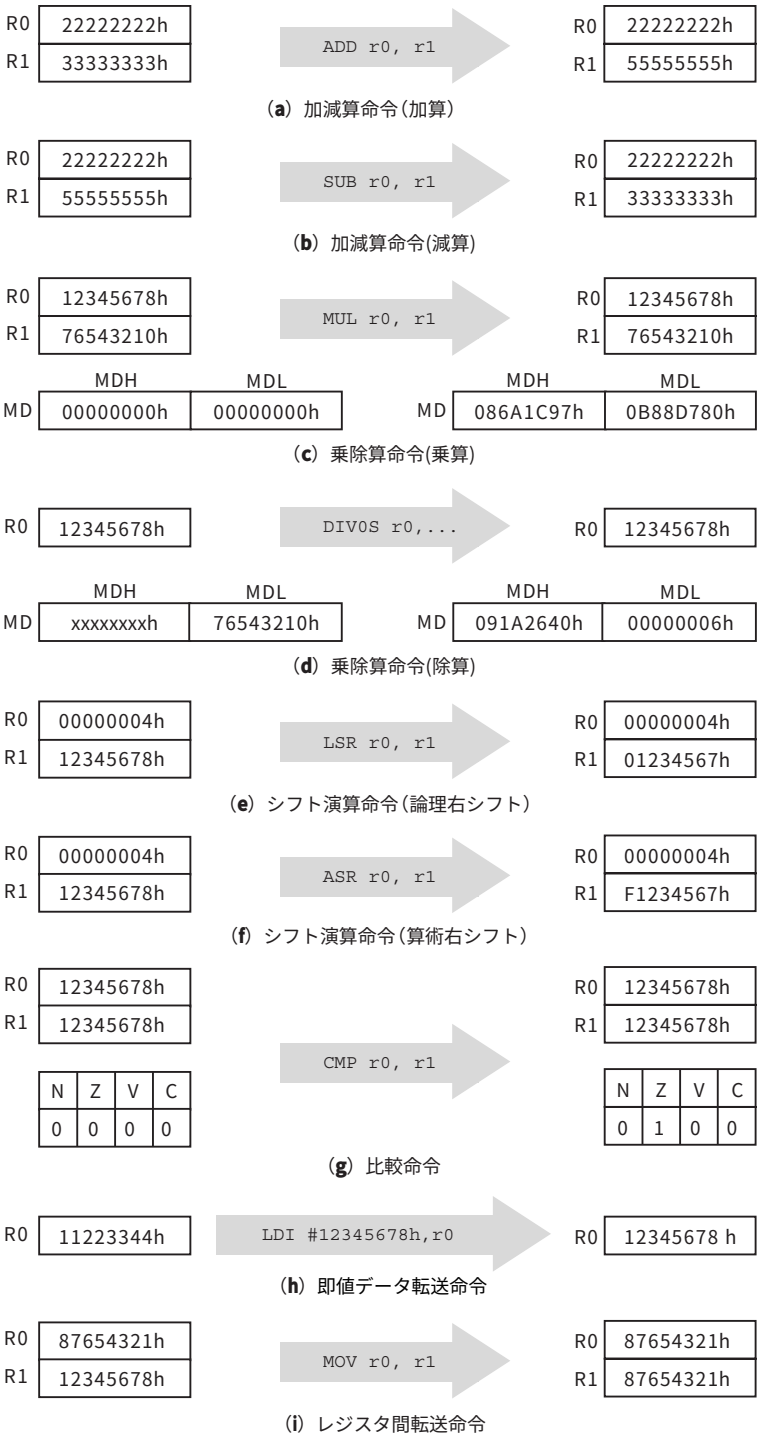


図4　算術演算命令

(a)のADD(ニーモニック)は，オペランド1(R0 ： 0x22222222)とオペランド2(R1 ： 0x33333333)を加算し，結果をオペランド2(R1 ： 0x55555555)に格納する．(c)のMULは，オペランド1(R0 ： 12345678)とオペランド2(R1 ： 0x76543210)を乗算し，結果をMDH，MDL(0x86A1C970B88D780)に格納する．(e)のLSRは，オペランド2(R1 ： 0x12345678)の値をオペランド1(R0 ： 0x4)の数だけ右へシフトし，結果をオペランド2(R1 ： 0x01234567)に格納す．(f)のASRは，データを補数として扱い，値の正・負によってビット・インする値が変わる．(g)のCMPは，オペランド2(R1 ： 0x12345678)からオペランド1(R0 ： 0x12345678)を減算し，PS(プログラム・ステータス・レジスタ)のZフラグが，セット(Z=1)される．(h)のLDIは，オペランド1(32ビット即値：0x12345678)をオペランド2(R0)へ格納する．(i)のMOVは，オペランド1(R0 ： 0x87654321)の値をオペランド2(R1)へ格納する．

ら構成されます．ニーモニックは，CPUがどのような操作を行うのかを示す命令部で，オペランドはニーモニック(操作)の対象になります．例えばFRの命令セットは，ニーモニックとオペランドを組み合わせた3種類の命令記述形式で構成しています(**図2**)．

命令実行サイクル数については，

- 一般的なRISC系のレジスタ-レジスタ間命令
- I/O制御に便利なレジスタ-メモリ間命令

が混在しているので，同じ種類の命令であっても異なります．例えば論理演算命令には，実行サイクル数1のレジスタ-レジスタ間命令と，実行サイクル数$1+2\times a$のレジスタ-メモリ間命令が混在しています(aはアクセス・サイクル数)．

命令サイズは，16ビット以上の即値データ転送命令を除いて，すべて16ビット固定長になっています(**図3**)．これは，コード・サイズの削減と，処理および回路の簡素化のためです．16ビットの限られた命令コードを，組み込み制御向け命令に絞って配置しています．

RISC系プロセッサは，CISC系と比べて命令の数が削減されています．しかし，組み込みマイコンは高速処理に加えて高速I/O制御が要求されるので，基本的なRISC系命令だけでは不十分です．そのため，組み込み制御向け命令が加えられ，一般のRISC系プロセッサと比べて命令の数が豊富になっています(FRの場合は165命令)．

命令セットは，操作の内容により，算術演算，ストアとロード，分岐，論理演算とビット操作，ダイレクト・アドレッシング，そのほかの機能グループに分類できます．

● 算術演算のための命令を用意

算術演算の命令には，以下の6種類があります．

1) 加減算命令

加減算命令は，オペランド1とオペランド2の間で加算または減算(ニーモニックの演算)を行い，結果をオペランド2へ格納します〔**図4**(**a**)，(**b**)〕．命令フォーマットは，**図3**(**a**)のタイプA，Cの形式になります．レジスタ-レジスタ間または即値-レジスタ間の演算なので，すべて1

サイクルで実行します．

2）乗除算命令

乗除算命令は，オペランド1とオペランド2の間で演算し，結果を乗除算演算レジスタ（MDH，MDL）へ格納します〔**図4**（**c**），（**d**）〕．命令フォーマットは，タイプAの形式になります．乗算の場合，32ビット×32ビットの処理を行うのに，8ビット×32ビットの乗算回路を4回繰り返して結果を格納するので，5サイクル実行になります．16ビット×16ビットも同じように8ビット×16ビットの演算を2回繰り返すので，3サイクル実行になります．センサなどからの入力結果を積和演算（乗算＋加算）する場合，乗算命令を繰り返し使用することになり，この2サイクルの差が処理速度に大きな影響を与えます．必要な実行結果の精度に合わせて変数の型を指定することで，処理速度が向上します．

3）シフト演算命令

シフト演算命令は，オペランド1で指定したビット数だけオペランド2のデータをビット・シフトし，結果をオペランド2へ格納します〔**図4**（**e**），（**f**）〕．命令フォーマットは，タイプA，Cの形式になります．バレル・シフタ回路により，論理シフトや算術シフトを1サイクルで実行します．

4）比較命令

比較命令は，オペランド1とオペランド2の間で演算し，PSレジスタのCCRフィールドのフラグをセットします〔**図4**（**g**）〕．命令フォーマットは，タイプA，Cの形式になります．分岐命令とセットで使用します．レジスタ-レジスタ間，即値-レジスタ間の演算なので，1サイクルで実行します．

5）即値データ転送命令

即値データ転送命令は，即値をレジスタへ格納します〔**図4**（**h**）〕．命令フォーマットは，タイプB，C，Eの形式になります．本命令のみ，コード・サイズが可変長になっています．20ビットの即値データの場合は32ビットのコード・サイズになり，32ビットの即値データの場合は48ビットのコード・サイズになります．即値のサイズによって実行サイクル数が違います．

6）レジスタ間転送命令，専用レジスタ転送命令

レジスタ間転送命令と専用レジスタ命令は，オペランド1のデータをオペランド2へ格納します〔**図4**（**i**）〕．命令フォーマットは，タイプA，Eの形式になります．レジスタ-レジスタ間の転送なので，1サイクルで実行します．

● メモリに対してデータを読み書きする命令を用意

次にメモリ・ストア命令とメモリ・ロード命令について説明します．

1）メモリ・ストア命令

メモリ・ストア命令は，オペランド1のデータをオペランド2で指定したアドレスに格納します〔**図5**（**a**）〕．関数処理に便利な命令を加えているため，命令フォーマットはタイプA，B，C，Eの複数の形式になっています．メモリ領域への書き込みに使用するため，実行サイクルはメモリへのアクセス・サイクル（aサイクル）になります．しかし実際には，パイプラインのメモリ・ステージでライト・バッファへ書き込むため，見かけ上は1サイクルで実行しているように見えます．

2）メモリ・ロード命令

メモリ・ロード命令は，オペランド1で指定したアドレスからデータを読み出してオペランド2へ格納します〔**図5**（**b**）〕．関数処理に便利な命令を加えているため，命令フォーマットはタイプA，B，C，Eの複数の形式になっています．メモリ領域から読み出しているため，実行サイクルはメモリへのアクセス・サイクルになります．

● 条件分岐のための命令を用意

分岐命令には，以下の二つの命令があります．

1）遅延なし分岐命令

遅延なし分岐命令は通常の（遅延のない）分岐を行いま

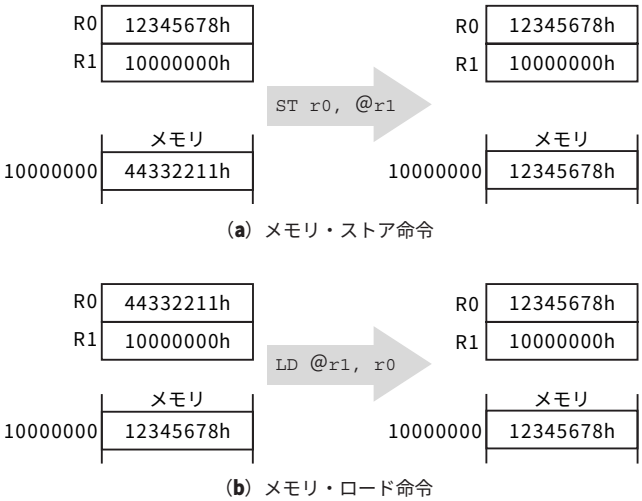
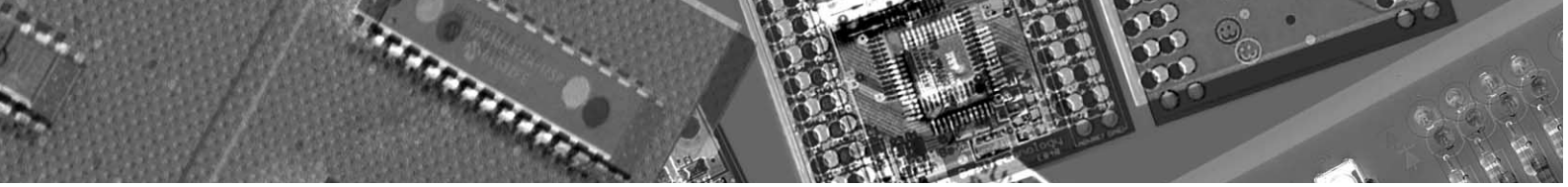


図5　ロード命令とストア命令

（a）のSTは，オペランド1（R0：0x12345678）の値をオペランド2（R1：0x10000000）のアドレスに格納する．（b）のLDは，オペランド1（R1：0x10000000）のアドレスに格納された値をオペランド2（R0）に格納する．



す〔**図6**(**a**)〕．命令フォーマットは，タイプD，E，Fの形式になります．PSレジスタのCCRフィールドの値によって，条件分岐します．デコードする前に次アドレスの命令をフェッチしているため，プリフェッチがキャンセルされてパイプラインに空きができます．

2）遅延分岐命令

遅延分岐命令は遅延分岐を行います〔**図6**(**b**)〕．命令フォーマットは，タイプD，E，Fの形式になります．PSレジスタのCCRフィールドの値によって，条件分岐します．分岐・非分岐にかかわりなく実行される命令を遅延スロットにすることで，プリフェッチがキャンセルされることなく実行されます．

● 論理値を取り扱う命令を用意

次に，論理演算命令とビット操作演算命令を説明します．

1）論理演算命令

論理演算命令は，オペランド1とオペランド2の間で論理演算を行い，結果をオペランド2へ格納します〔**図7**(**a**)〕．命令フォーマットはタイプAの形式になります．一般に，RISC系プロセッサは1サイクル1命令実行を基本としているので，レジスタにロード/ストアして演算します．ただし，高速制御が必要な組み込みマイコンは，レジスタに格納されたデータのみではなく，メモリや周辺機能I/Oレジスタに格納されたデータに直接アクセスして演算します．そのため，実行サイクルがメモリへのアクセス・サイクルに依存しています．

2）ビット操作演算命令

ビット操作演算命令は，オペランド1とオペランド2の間で論理演算を行い，結果をオペランド2へ格納します〔**図7**(**b**)〕．命令フォーマットはタイプCの形式になります．ほとんどの周辺機能I/Oレジスタは，バイト単位やワード単位ではなくビット単位で意味を持っているため，周辺機能の制御にはビット操作演算命令を多用します．

● アドレス指定専用の命令を用意

ダイレクト・アドレス指定命令は，オペランド1のデータをオペランド2へ格納します(**図8**)．命令フォーマットは，タイプDの形式になります．即値で指定したアドレスとレジスタの間の転送なので，メモリへのアクセス速度に依存した実行サイクル数になります．通常，即値データ転送命令とメモリ・ロード/メモリ・ストアの組み合わせによってメモリへデータを転送しますが，ダイレクト・アドレス指定命令は1命令で実行できるので，コード・サイズや処理速度の改善に有効です．周辺機能I/Oレジスタにアクセスする際に有効な命令です．

● そのほかにも多様な命令を用意

これ以外の命令として，リソース命令，コプロセッサ制御命令，高級言語用命令などがあります．

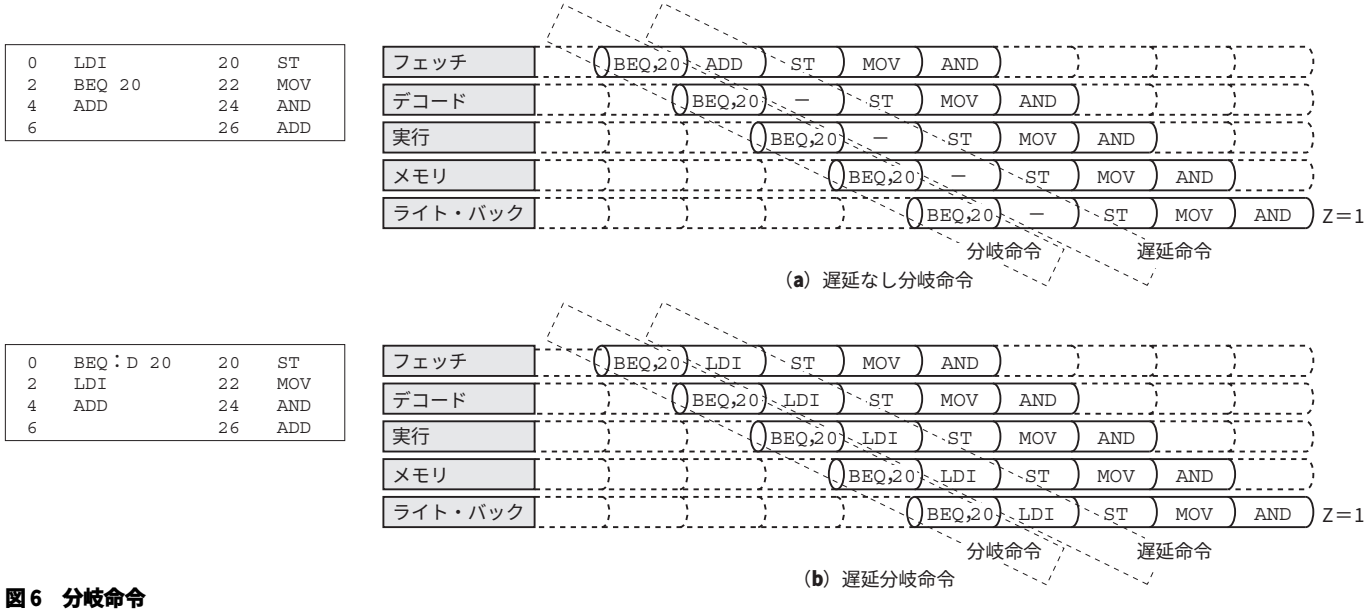


図6　分岐命令

(a)のBEQは，Z=1の場合に指定アドレス(0x20)へ分岐する．分岐命令がデコードされる前にフェッチした次の命令(ADD)は，プリフェッチ・キャンセルされる．(b)のBEQ：Dも同じように，Z=1の場合に指定アドレス(0x20)へ分岐する．分岐・非分岐に関係なく，デコードされる前にフェッチした次の命令(LDI)を実行する．

1）リソース命令

リソース命令は，メモリと指定した周辺機能の間でロードまたはストアします．命令フォーマットは，タイプCの形式になります．連続したアドレスに対して，読み出しまたは書き込みを行うのに有効な命令です．

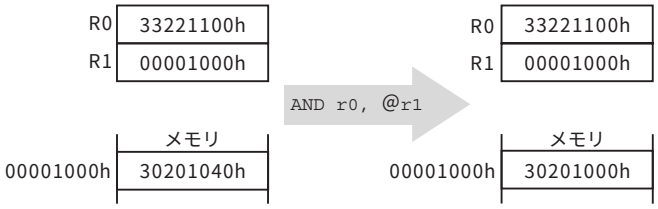
2）コプロセッサ制御命令

コプロセッサ制御命令は，コプロセッサに対する制御命令です．命令フォーマットは，タイプEの形式になります．

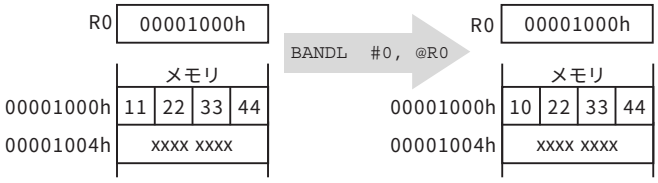
3）高級言語用命令

高級言語の関数呼び出し処理用に，マルチストア命令（STM）やスタック・フレーム生成命令などがあります（**図9**）．C言語などの記述を1命令で実行できます．

このほか，PSレジスタのCCRフィールドやILMフィールドへのアクセス命令や符号・ゼロ拡張命令なども用意されています．



（**a**）論理演算命令



（**b**）メモリ・ロード命令

図7　論理演算命令とビット操作演算命令

(a)のANDは，オペランド1（R0 ： 0x33221100）とオペランド2（R1 ： 0x1000）のアドレスに格納されたデータ（0x30201040）の論理積を行い，結果（0x30201000）をオペランド2（R1 ： 0x1000）のアドレスへ格納する．(b)のBANDLは，オペランド1（4ビット即値： 0x0）とオペランド2（R1 ： 0x1000）のアドレスに格納された下位4ビット・データ（0x11）の論理積を行い，結果（0x10）をオペランド2（R1 ： 0x1000）のアドレスへ格納する．

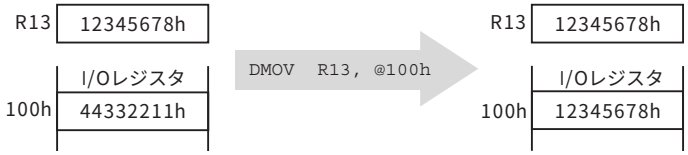


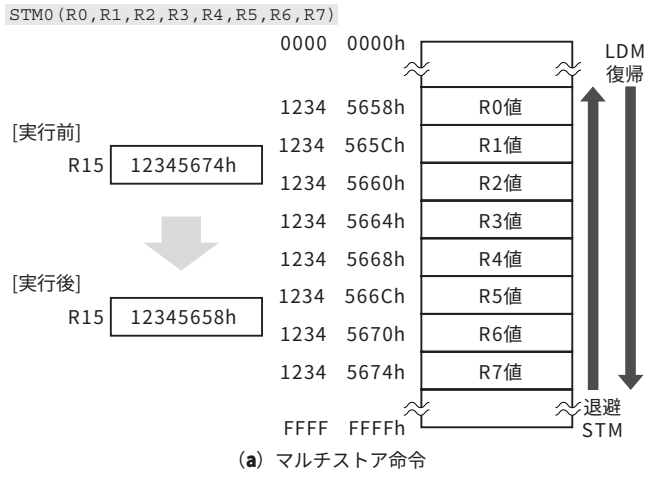
図8　ダイレクト・アドレス指定命令

DMOVは，オペランド1（R13 ： 0x12345678）の値をオペランド2（ダイレクト・アドレス： 0x100）へ格納する．

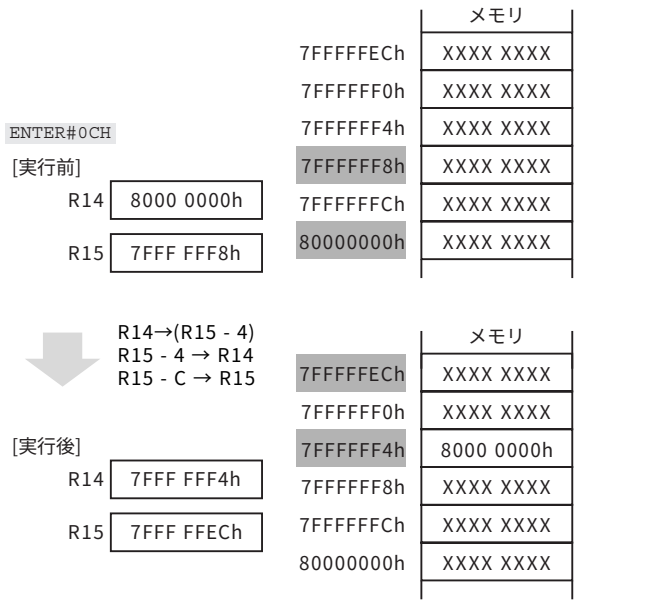
● 命令で指定できるアドレス範囲には制約がある

マイコンの中心には，演算処理を行うCPUが存在します（中核の回路という意味で，「CPUコア」と呼ぶこともある）．CPUには，バスを経由してプログラムを格納する内蔵ROMやデータを配置するRAM，周辺機能のI/Oレジスタなどが接続されています．

CPUは，アクセス先を区別するためにそれぞれの機能にア



（**a**）マルチストア命令



（**b**）スタック・フレーム生成命令

図9　そのほかの命令

(a)のSTMは，指定したレジスタ（R0, R1, R2, R3, R4, R5, R6, R7）の値をR15で指定したスタック領域（0x12345674）へ，アドレスをデクリメントしながら格納する．読み出す関数にて汎用レジスタを使用する場合，使用する汎用レジスタの値を退避および復帰する．一つの命令で，最大8個の汎用レジスタ値をバックアップすることができる．(b)のENTERは，オペランド1（フレーム・サイズ： 0x0C）で指定した値をスタック・フレームとして生成する．関数を読み出すときに，スタックおよびフレーム・ポインタを一つの命令で生成できる（スタック・フレーム構成は，本連載の第1回を参照）．

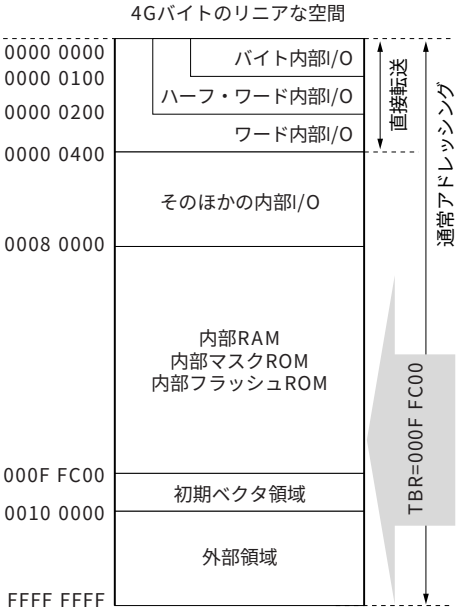


図10　CPUのメモリ空間

4Gバイトのリニアな領域になっている．

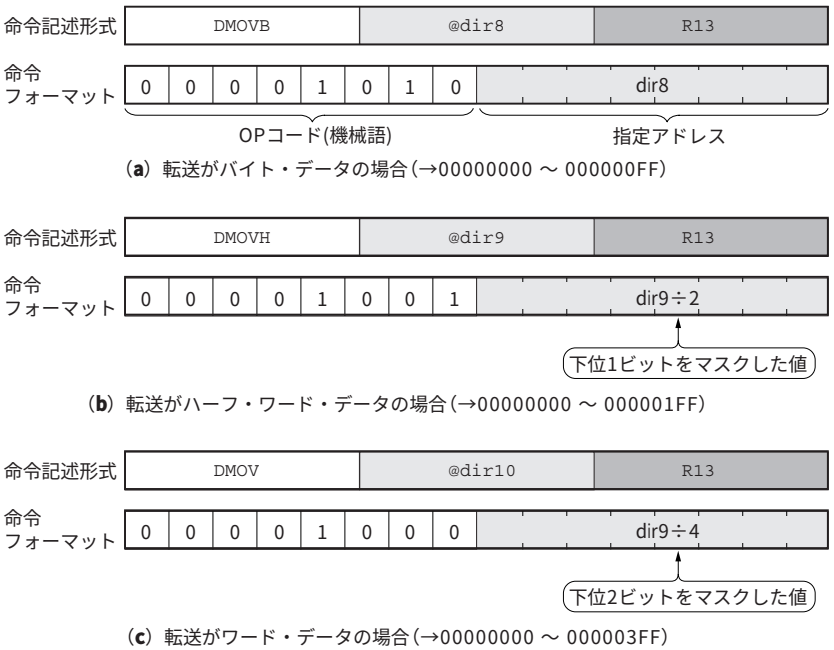


図11　ダイレクト・アドレス領域

ダイレクト・アドレス領域は，8ビットで指定できる領域に限定される．データ・アクセスはミス・アラインに対応していないので，ハーフ・ワード・アクセスには下位1ビット目を‘0’にマスクし，ワード・アクセスには下位1，2ビット目を‘0’にマスクする．また，使用するレジスタごとに機械語命令を用意しているので，対応レジスタを限定している．

ドレスを振り分けています．例えば，内蔵ROMやRAMのアドレスが重ならないように配置しています．アドレスが重なる場合は，動作モードや命令によって切り替えることになります．CPUがアクセスするメモリ空間は，CPUが取り扱うデータ・サイズと同じになる傾向があります．32ビット・マイコンの場合は，4Gバイトの連続したアドレス空間(0x00000000～0xFFFFFFFF)になります(**図10**)．

命令で直接指定できるアドレス範囲は命令長に依存するため，有限の範囲に限定されます．例えば命令長が16ビットの場合，OPコードが8ビット，指定アドレスが8ビットになり，8ビットで指定できる範囲に限定されます(**図11**)．8ビットで指定できないアドレスへは，ロード/ストア命令を用いて間接的にアクセスします．

割り込みのベクタ・テーブル初期領域は0xFFC00～0xFFFFFに配置してありますが，プログラムによって任意のアドレスへ変更することができます．データ構造はCPUの種類によって違うため，ビット・オーダリングやバイト・オーダリング(「オーダリング」とは「順序」の意味)を把握することは重要です．FRはビット・オーダリングがリトル・エンディアンで，バイト・オーダリングがビッグ・エンディアンになっています(**図12**)．

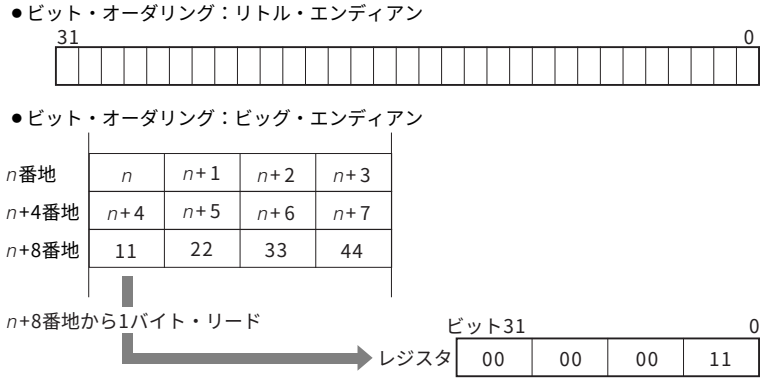


図12　データ構造

ビット・オーダリングがリトル・エンディアン(MSBに近いほど番号が大)で，バイト・オーダリングがビッグ・エンディアン(上位データをメモリ・アドレスの小さい方に配置し，下位データを大きい方に配置)になっている．n＋8番地に0x11223344のワード・データが格納されている場合，n＋8からバイト・アクセスすると，0x11がレジスタのビット0～7へ格納される．

データやプログラムにおけるアラインメントについては，CPUの回路構成を単純にするために，ミス・アラインに対応していません．そのため，アクセス・サイズによってメモリ・アドレスに制約があります．

ひらいし・いくお
富士通(株) 電子デバイス事業本部 システムマイクロ事業部