第4章

SH-3コアのレジスタや例外処理

SH-3コアの詳細

藤澤 幸穂

1　SH-3のCPU内部レジスタ

● レジスタ構成

SH-3のCPU内部レジスタを示します．CPU動作モードによって利用可能なレジスタやアクセスする命令が異なります．**図1**(p.36)にSH-3のレジスタを示します．まずはリセット直後の特権モードでのレジスタについて解説します．

CPU内部レジスタは三つのグループで構成されます．汎用レジスタ，コントロール・レジスタ，システム・レジスタの三つです．

● 汎用レジスタ

汎用レジスタは32ビット(SuperHではロング・ワード)長です．演算結果は32ビットで保存されるため，1バイトや2バイト(SuperHではワード)の区切りはありません．16本の汎用レジスタはデータ演算，ポインタ機能で利用できます(**図2**)．

汎用レジスタはどの番号の位置でも同じ機能です．ただし，R0は特定の命令で演算結果を格納する専用レジスタとして利用するほか，ベース・アドレスからの相対距離(インデックス)として利用されることがあります．

● コントロール・レジスタ

▶SRレジスタ

CPUのステータス・レジスタです(**図3**)．

Tビットは条件付き分岐のフラグでTrueの意味です．割り込みマスク・ビット(I3～I0)は優先度の低い割り込み要求をマスクし，受け付けを保留する機能です．

Sは積和演算のオーバフロー(加算時に桁あふれで符号が反転すること)を防ぎます．

MとQは除算時に利用するフラグです．プログラマは意識する必要はありません．ただし除算時に利用しているので変更しないようにしてください．

MD，RB，BLの各ビットはSH-3で追加されました．SH-2にはありません．MD(Mode)はCPU動作モードを設定/参照できるビットです．RB(Register Bank)は汎用レジスタのバンクを設定/参照できるビットです．汎用レジスタとして利用できるバンクを設定します．汎用レジスタとして利用できないバンクはコントロール・レジスタとしてアクセスできますが，演算やポインタの機能はありません．

BL(Block)は例外要求をブロックするビットです．例外要求とはリセット，アドレス・エラーなどのエラー

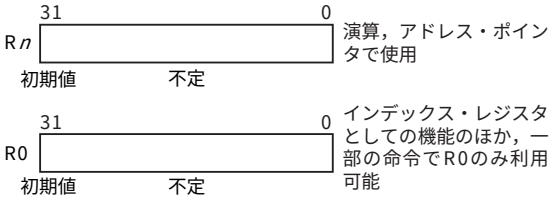


図2　汎用レジスタ

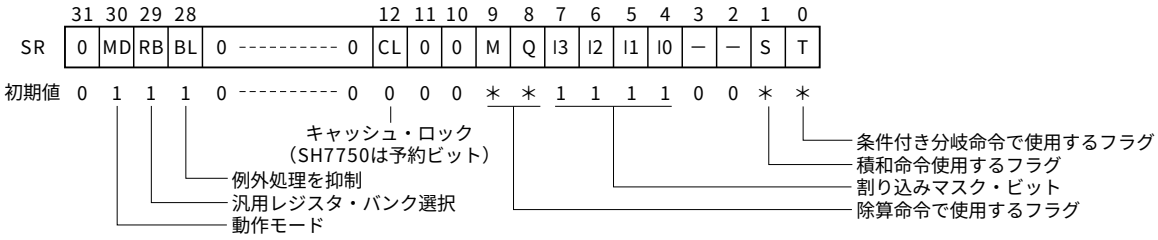
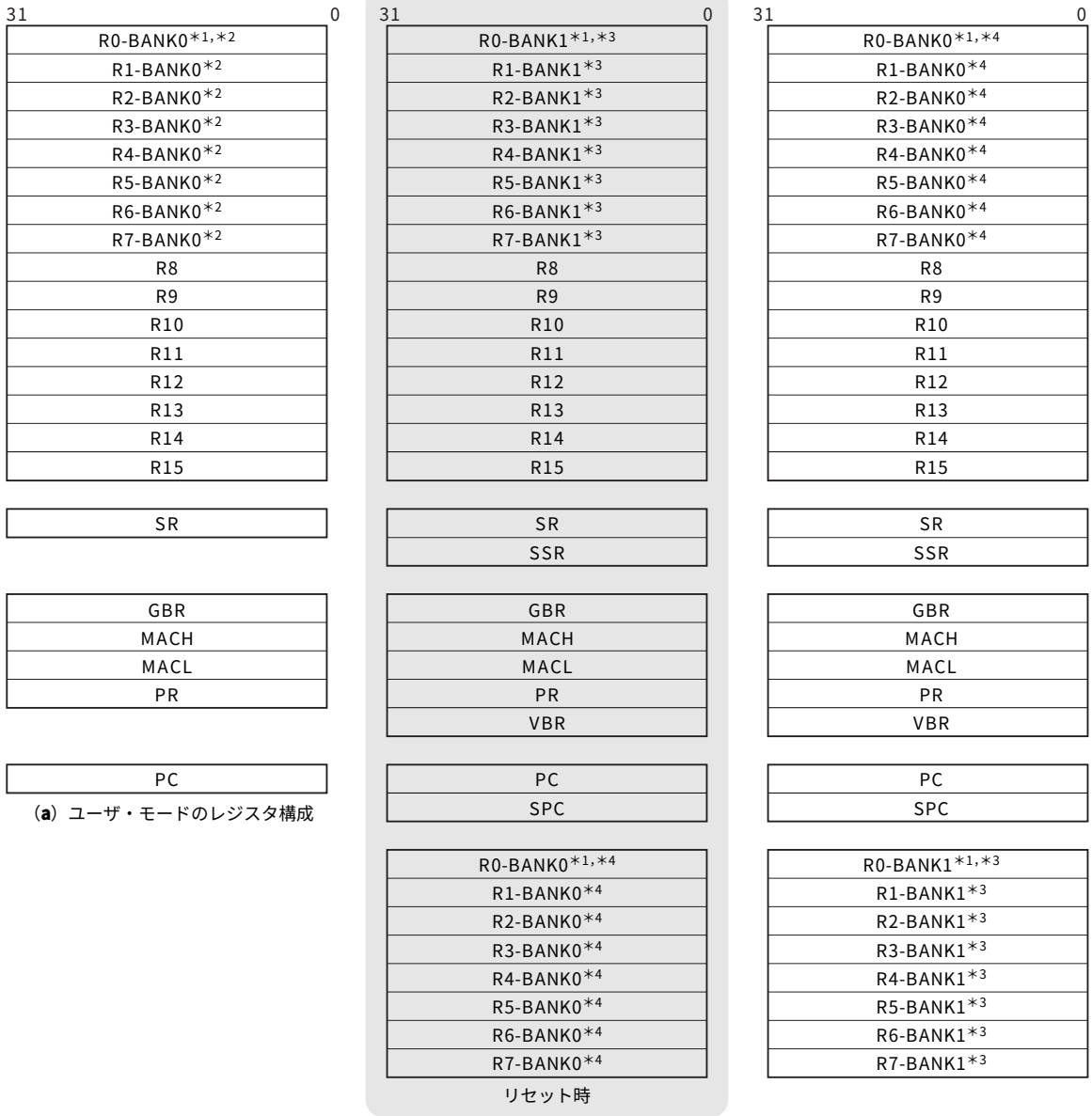
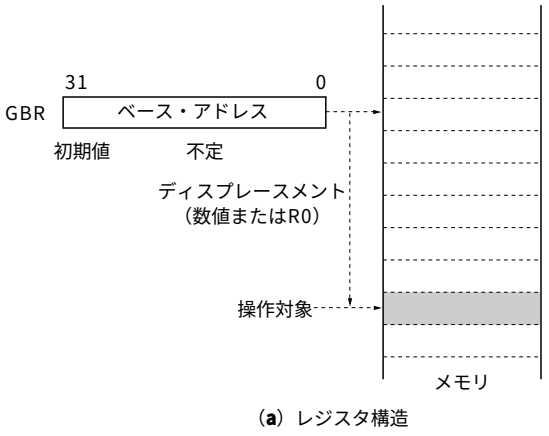


図3　SRレジスタの構成

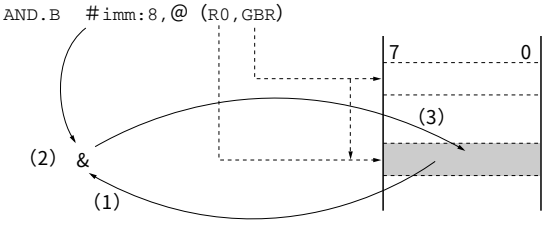


注）＊1 R0レジスタは，インデックス付きレジスタ間接アドレッシング・モードとインデックス付きGBR間接アドレッシング・モードのインデックス・レジスタとして使われる
＊2 バンク・レジスタ
＊3 バンク・レジスタ
SRレジスタのRBビットが1のとき汎用レジスタとしてアクセスされる．RBビットが0のとき，LDC/STC命令でのみアクセスされる
＊4 バンク・レジスタ
SRレジスタのRBビットが0のとき汎用レジスタとしてアクセスされる．RBビットが1のとき，LDC/STC命令でのみアクセスされる

図1　CPU内部レジスタ



（**a**）レジスタ構造



（**b**）動作例

図4　GBRレジスタの構成

検出，TRAPA命令のOS呼び出し，ハードウェアによる割り込み要求を指します．これらの例外要求のうち，リセットを除く要求を受け付けるかどうかがBLビットの効果です．BL=1の状態でハードウェア割り込みを要求しても受け付けは保留されます．そのほかの例外要求はリセット動作になります．これは例外処理の節で説明します．

▶GBRレジスタ

GBR間接アドレッシング用のベース・アドレスを記憶するレジスタです(**図4**)．ベース・アドレスを汎用レジスタで示さない分，機械語コードで利用できるビットが多くなります．

▶VBRレジスタ

例外処理プログラムの先頭番地を設定するレジスタです(**図5**)．SuperHプロセッサの例外処理は，その要因グループごとにVBR＋0x100番地またはVBR＋0x400，VBR＋0x600番地からスタートする固定番

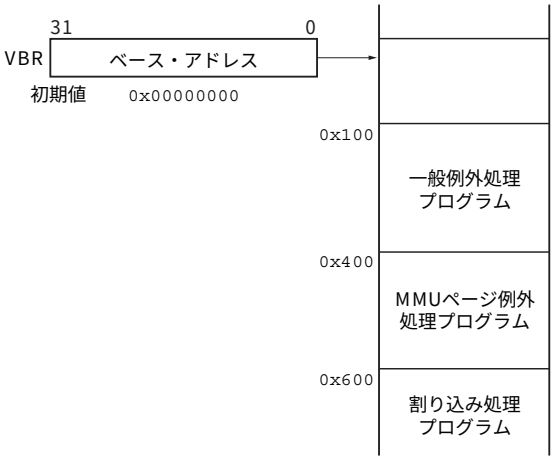


図5　VBRレジスタの構成

地スタート方式です．VBRによって処理プログラムをRAMに設定することも可能で，ROMブート時とRAM上でのOS稼働時でVBRを切り替えることができます．

▶SPC/SSRレジスタ

リセット以外の例外処理を受け付けたときのPCとSRの退避レジスタです(**図6**)．例外処理要求ではPCをSPCへ，SRをSSRへ退避します．

汎用レジスタ8本はバンク構造になっています．リセット後はバンク1が表，バンク0が裏です．表の汎用レジスタ・バンクはデータ演算，ポインタ機能で利用できますが，裏にあるレジスタ・バンクはコントロール・レジスタとしてのみアクセスできます．これらはコントロール・レジスタなので，データ演算やポインタ機能としては利用できません．

● システム・レジスタ

▶PCレジスタ

プログラム・カウンタです(**図7**)．命令フェッチするアドレスを示しています．初期値は0xA0000000です．

▶PRレジスタ

サブルーチン(C言語では関数)から戻るためのアドレスを格納するレジスタです(**図8**)．関数呼び出しにスタック領域は使いません．これはパイプライン動作

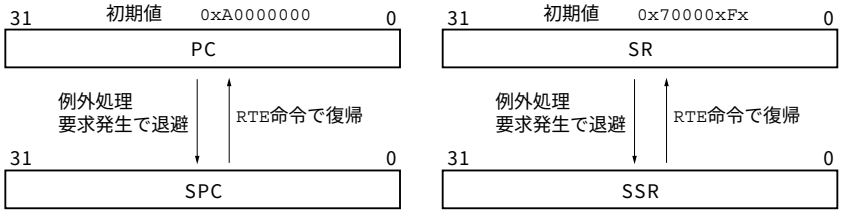


図6
SPC/SSRレジスタの構成

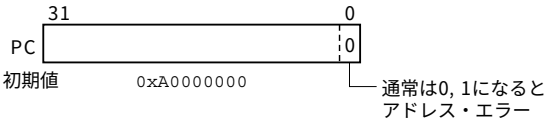


図7　PCレジスタの構成

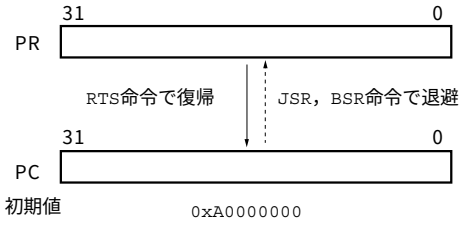


図8　PRレジスタの構成

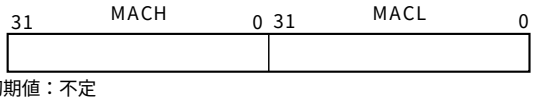


図9　MACレジスタの構成

に影響を与えないように素早く遅延分岐するための工夫です．

関数呼び出しが多重化される場合にはプログラムでスタック領域へPRを退避します．プログラムで退避することでパイプラインのストールを小さくできます．関数(サブルーチン)呼び出しではPRへPCのみを退避します．その他のCPU内部レジスタはプログラムで退避，復旧します．

これまでに紹介したようにSH-3は関数呼び出しにはPR，例外処理にはSPCとSSRの各専用レジスタに退避するため，ハードウェア的なスタック・ポインタの機能はありません．SH-3にはSuperHコントローラでは存在したハード・スタック・ポインタ(R15)は存在しません．

▶MACレジスタ

積と積和演算結果を記憶する専用レジスタです(**図9**)．

2　16ビット固定長の基本68命令

● アセンブラ命令

命令はRISC方式の基本を守り，16ビット固定長でほとんどの命令を1クロックで実行します．さらに，ディジタル・フィルタやFFT，DCTなどの行列演算を高速化することがSuperHユーザにとって求められる性能であると考え，積和演算回路と命令を実装しています．**表1**にSuperHファミリの命令を示します．

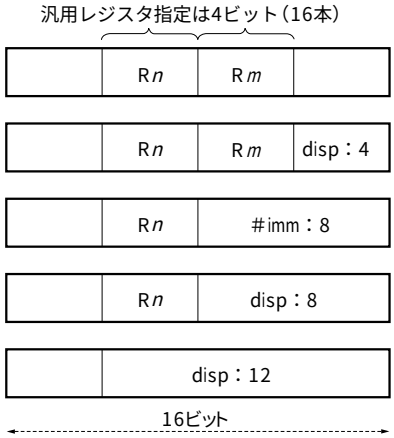


図10　機械語フォーマット

SuperHはC言語での開発を前提に仕様が作られています．そのためint型は32ビットです．汎用レジスタはつねに32ビットで演算動作をしていれば良く，C言語にある演算子に対応した16ビットや8ビットの機械語命令は必要ありません．そこで思い切って命令のオペランド・サイズを削除しました．またC言語には4ビット(ニブル)のデータはありません．パックド，アンパックドBCDもデータ型にはありません．それでもシステムを構築できない，という話は聞いたことがありません．ならば4ビットのデータ演算も考慮しなくて良いでしょう．このようにSuperHではC言語の言語仕様から命令を厳選し，必要不可欠な命令のみを実装しています．

SuperHはコンパクトな16ビット固定長の命令とC言語に合わせた命令体系で構成されているので，小さなコアでも十分に能力を発揮します．

機械語命令のフォーマットを**図10**に示します．

● SH-3の命令

SH-3の命令をいくつか紹介します．

▶データ転送

16ビット長の命令なのでイミディエイトは8ビットです．16ビットや32ビットのデータはメモリ上に格納したもののみ扱うことができます．リセットするとPCは初期値が設定され，プログラムを実行中にイミディエイトのデータが必要になります．そこでSHはPC相対でメモリに記憶したイミディエイトを汎用レジスタに読み込む命令を持っています(**図11**)．

```
MOV.L       @(disp:8, PC), Rn
```

もちろん，8ビットのイミディエイトを読み込んで汎用レジスタ内で演算(シフト演算，算術演算，論理

表1　SuperHファミリの命令

種　類	命令ニーモニック	
データ転送	MOV	汎用レジスタ間，汎用レジスタとメモリ間のデータ転送
	MOVA	R0にアドレスをPC相対で転送
	MOVT	SRレジスタのTビットを汎用レジスタに転送
	SWAP	汎用レジスタ間のデータ交換
	XTRCT	64ビット・データの中心位置から32ビットの抽出
算術演算	ADD，ADDC，ADDV	加算(キャリ付き/オーバフロー確認)
	CMP/cond	条件付きデータ比較
	DIV1，DIV0S，DIV0U	除算(初期化と1ビット除算)
	EXTS，EXTU	データの拡張(符号付き/なし)
	MAC *1	積和演算
	MULS，MULU，MUL *2，DMULS *2，DMULU *2	乗算(16×16＝32，32×32＝32，32×32＝64ビット)
	NEG，NEGC	符号変換
	SUB，SUBC，SUBV	減算(ボロー付き/アンダフロー確認)
	DT *2	ループ制御(カウンタのデクリメントと0検出)
論理演算	AND，NOT，OR，XOR	論理演算
	TAS	マルチプロセッサ・システムのセマフォ
	TST	テスト(AND演算し0検出)
シフト	ROTL，ROTR，ROTCL，ROTCR	ローテート(1ビット)
	SHAL，SHAR，SHAD *3	算術シフト(1ビット/ダイナミック)
	SHLL，SHLL*n*，SHLR，SHLR*n*，SHLD *3	論理シフト(1ビット/*n*ビット/ダイナミック) (*n*は2または8または16)
分岐	BT，BF，BT/S，BF/S	条件付き分岐(遅延スロットなし/あり)
	BRA，BSR，BRAF *2，BSRF *2	PC相対分岐(数値/汎用レジスタ)
	JMP，JSR	絶対番地分岐
	RTS	リターン
システム制御	CLRT，SETT	SRレジスタのTビットのクリア/セット
	CLRS *3，SETS *3	SRレジスタのSビットのクリア/セット
	CLRMAC	MACレジスタのクリア
	LDC，STC	コントロール・レジスタのロード/ストア
	LDS，STS	システム・レジスタのロード/ストア
	RTE	例外処理からの復帰
	NOP	ノー・オペレーション
	SLEEP	スリープまたはスタンバイ・モードへの遷移
	TRAPA	TRAP例外処理の起動
	PREF *3	キャッシュへのプリフェッチ
	MOVCA *4	キャッシュへの書き込み，ミス・ヒットしても外部メモリからのリードを行わない
	OCBP *4	ヒットしたオペランド・キャッシュ・ブロックを無効にする，コピーバック・モードで更新データはライトバック
	OCBWB *4	ヒットしたオペランド・キャッシュ・ブロックを無効にする，コピーバック・モードで更新データはライトバック
	OCBI *4	ヒットしたオペランド・キャッシュ・ブロックを無効にする
	LDTLB *3	MMUのTLBへ書き込み
FPU命令	FABS	絶対値
	FADD	加算
	FCMP/cond	比較
	FDIV	除算
	FLDI0，FLDI1	0または1のロード
	FLOAT	FPULから浮動小数点データに変換しFR*n*にロード
	FMAC	積和演算
	FLDS，FSTS	FR*n*とFPUL間のデータ転送(データ変換なし)
	FMOV	メモリ，FR*n*間のデータ転送
	FMUL	乗算

表1　SuperHファミリの命令(つづき)

種　類	命令ニーモニック	
FPU命令 (つづき)	FNEG	符号の変換
	FSQRT	ルート
	FSUB	減算
	FTRC	FR*n*をFPULに変換して転送
	FRCHG	FR*n*のレジスタ・バンクを交換(FPSCRレジスタのFRを設定)
	FSCHG＊4	演算精度を交換(FPSCRレジスタのSZを設定)
	FIPR＊4	内積演算
	FTRV＊4	ベクトル変換

注：＊1：SH-2で改良された命令
　　＊2：SH-2で追加された命令
　　＊3：SH-3で追加された命令
　　＊4：SH-4で追加された命令

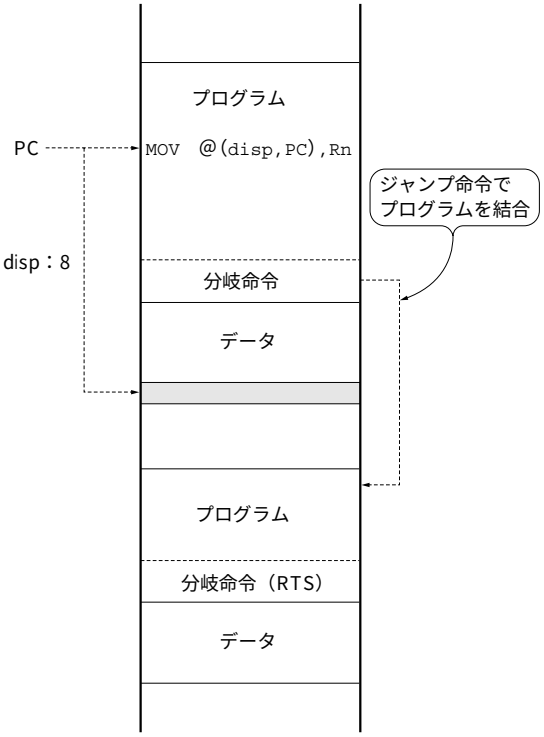


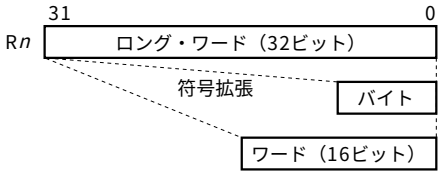
図11　PC相対データ・アクセス

演算など)で作り出すほうが速い場合もあるので，コンパイラは最適なものを選択します．

```
MOV.B     #xx, Rn
SHLL16    Rn
```

32ビットに満たないデータ転送は符号拡張されます(**図12**)．

GNUやルネサスのCコンパイラでは**表2**のように整数型を扱います．グローバル(大域)変数として，メモリに固定的に割り当てられた変数を汎用レジスタへ転送し演算しようとすると，転送時に行われた符号拡張をやり直し，ゼロ拡張しなければならない場合があります．型がunsigned char，unsigned shortの場合です．できるだけグローバル変数はsignedまたはintかlongの型としたほうが少ない命令で処理できます．



汎用レジスタの内容は，つねにロング・ワードで扱われる．
バイト・ワードのデータは，汎用レジスタにロードするときに符号拡張されロング・ワードとなる

図12　データ転送時の汎用レジスタ

▶比較

8ビットはR0との間で直接比較できます．しかし16ビットは汎用レジスタで32ビットに拡張して比較します．この場合もunsigned charとunsigned shortは処理速度的に不利です．メモリ空間が許すなら32ビットで変数を用意しましょう．

(1) 8ビット・データの比較

比較はつねに32ビットで行われます．8ビット・データは符号拡張されR0と比較されます．イミディエイト・データはR0と比較できます．R0以外のレジスタは使用できません．

```
        ;R0の内容は0x00000010と等しいか
        CMP/EQ  #H'10,R0
        ;if R0=0x10 branch
        BT      EQUAL
        ;   R0!=0x10
        ADD     R0,R1
EQUAL:  $
```

(2) 16ビット・データ

表2
コンパイラの整数型と汎用レジスタ

データ型	サイズ	値の範囲
char(signed char)	1	－128 ～127
unsigned char	1	0 ～255
short	2	－32768 ～32767
unsigned short	2	0 ～65535
int	4	－2147483648 ～214783647
unsigned int	4	0 ～4294967295
long	4	－2147483648 ～214783647
unsigned long	4	0 ～4294967295
long long	8	－9223372036854775808 ～9223372036854775807
unsigned long long	8	0 ～18446744073709551615
float	4	$7.0064923216240862e^{-46}$f ～$3.4028235677973364e^{+38}$f
double	8	$4.9406564584124655e^{-324}$ ～$1.7976931348623158e^{+308}$

(**a**) 型とサイズ

レジスタ	用　途	関数呼び出し前後のレジスタ保証
R0	リターン	保証しない
R1～R3	ワーク	
R4～R7	引き数	
R8～R14	レジスタ変数	保証する
R15	スタック・ポインタ	

(**b**) 汎用レジスタ

レジスタ	用　途	関数呼び出し前後のレジスタ保証
MACH,MACL		保証する
PR		

(**c**) 汎用レジスタ以外

16ビット・データでは比較できません．そこで汎用レジスタ内で32ビットにデータを拡張してから比較します．

●符号付きデータ

```
        MOV.W   @R0,R0
        MOV.W   @R1,R1
        ;R0はR1より大きいか？
        CMP/HI  R1,R0
```

●符号なしデータ

```
        MOV.W   @R0,R0
        EXTU.W  R0,R0   ;符号を削除
        MOV.W   @R1,R1
        EXTU.W  R1,R1   ;符号を削除
        ;R0はR1より等しいまたは大きいか？
        CMP/HS  R1,R0
```

(3) 32ビット・データ

```
        ;R2とR3を符号付きで比較(R3-R2)
        CMP/GE  R2,R3
        BF      LOW
LOW:    $
```

▶分岐

ほとんどの分岐命令は遅延分岐します．遅延分岐命令の次の番地は遅延スロットといいます．この位置に記述した命令を実行してから分岐先命令を実行します．

Cコンパイラは遅延スロットにできるだけ実行効果のある命令を配置するようにしています．どうしても配置できる命令が見つからない場合はノー・オペレーション(NOP)命令を配置します．

アセンブリ言語で最適化する場合にも遅延スロットには注意しましょう．次のプログラムでは番号をふった順番に実行されます．

```
        ;サブルーチン・コール
        BSR     sub     ①
        ;ディレイ・スロット
        MOV     #34,R0  ②
        ;サブルーチンからの戻り
        NOP             ⑫
         :
sub:
         :              ③
         :              (省略)
         :
        RTS             ⑩
        ;ディレイ・スロット
        MOV     R4,R0   ⑪
         :
```

▶サブルーチン呼び出しとPR

サブルーチン呼び出し時にはパイプラインのストールを最小限に抑えるため，リターン・アドレスをCPU内部のPRに退避します．

PRは1本しかないので，呼び出しの入れ子(ネストまたは多重呼び出し)の場合はPRを退避してからサブルーチン呼び出しをします．このとき，PRはスタック領域を利用すると良いでしょう．ただし，パイプラインがストールしないように，4の倍数のアドレス位置にスタックへ退避する命令を配置すると良いでしょう．

当然，復旧も4の倍数のアドレス位置に配置します．

```
        ;サブルーチン・コール
        BSR     SUB1
        ;ディレイ・スロット
        NOP
        ;サブルーチンからの戻り
        NOP

SUB1:   $
        ;PRのpush
        STS     PR,@-R15
        ;アドレスの取得
        MOVA    SUB2,R0
        ;サブルーチン・コール
        JSR     @R0
        ;ディレイ・スロット
        NOP
        ;サブルーチンからの戻り，PRの復旧
        LDS     @R15+,PR

        RTS
        ;ディレイ・スロット
        NOP
```

▶除算命令

除算は1クロックで動作する回路を構成するとその規模が大きくなり，高速化の妨げになります．1命令で複数のクロックを利用するか，または1クロックでは1ビットの演算しかしないようにして，複数命令を利用するかの選択となります．

SuperHは1ビット除算回路を採用しています．この方式のメリットは割り込み応答が速いことです．命令に0除算検出機能はありません．ANSIの規格が0除算を検出できるハードウェアを要求していないためです．

符号なし16ビット・ワード・サイズ(unsigned short型)の除算プログラムを示します．

```
        ;桁合わせ
        SHLL16  R0
        ;0除算検出
        TST     R0,R0
        BT      ZERO_DIV
        ;オーバフロー検出
        CMP/HS  R0,R1
        ;答えが16ビットを越える
        BT      OVER_DIV
        ;フラグの初期化
        DIV0U
        DIV1    R0,R1
        (中略)  ;DIV1命令を16個繰り返す
        DIV1    R0,R1
        ;最後の商をTビットから移す
        ROTCL   R1
        ;符号を整える
        EXTU.W  R1,R1
```

▶積和命令

積和演算命令はMACです．FIR型ディジタル・フィルタのように多くのデータを扱うことが多いと考えられるため，そしてデータが準備できてからの積和演算命令の実行時間が3クロックかかることから，パイプラインを7段構成としました．命令はメモリから直接データを読み込みながら積和演算します．また後続するほかの命令の実行に影響を与えないように，専用レジスタのMACレジスタを使います．そのためMAC命令を連続発行するだけでFIR型ディジタル・フィルタが実現できます．

FIR型ディジタル・フィルタではデータのリング・バッファ構造が必要になりますが，それらはプログラムで実現します．

なお，積和演算は符号付き固定小数点を扱うため，桁あふれしないように，MACレジスタは64ビットで結果を保存します．それでも加算でオーバフローが発生し，符号が正しく出ないことも考えられます．通常のCPU演算では演算後のオーバフローを確認しながら処理します．しかし，DSPでは演算性能を確保するため，誤差が出てもオーバフローを防ぐように演算することが求められるケースが多いのです．

FIR型ディジタル・フィルタの演算がまさにこれです．演算結果は1回のサンプリングのみに利用します．フィードバック処理がないため，次のサンプリング時の演算では誤差の含まれた前回の結果は利用せず，新たに計算を始めるのです．このような演算を飽和演算といいます．SuperHのMAC命令も飽和演算ができるようになっています．飽和演算はSRレジスタのSビ

ットを1にセットします．

```
;0->MAC
CLRMAC
;Top address of DATA1 -> R4
MOV.L       DATA1, R4
;DATA2 -> R5
MOV.L       DATA2,R5
;Multiply and Accumulate
MAC.W       @R4+,@R5+
MAC.W       @R4+,@R5+
MAC.W       @R4+,@R5+
MAC.W       @R4+,@R5+
MAC.W       @R4+,@R5+
STS MACL,R0;
```

▶ 共有メモリ・アクセス

共有メモリに対するセマフォなどのアクセスにはバスをロックする必要があります．複数の命令を用いるとバスをロックできません．また割り込み要求も禁止しなければなりません．こういった用途に利用できる命令は，

```
TAS @Rn
AND.B       #xx,@(R0,GBR)
OR.B        #xx,@(R0,GBR)
XOR.B       #xx,@(R0,GBR)
```

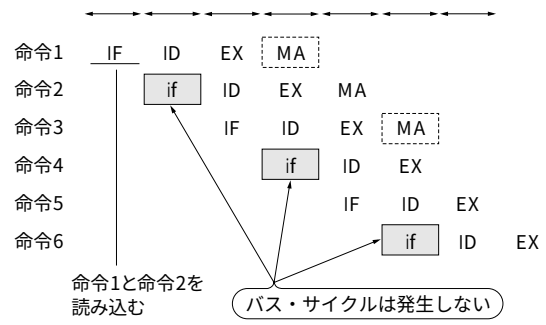
があります．

TAS命令はセマフォに利用する命令です．共有メモリの使用権を確認し，利用可能なら利用する宣言を対象となった1バイトのビット7で表示します．

GBRを利用した論理演算命令はメモリに対してリード/モディファイ/ライトを行います．この間バスをロックし，割り込み要求も受け付けません．TAS命令がビット7だけで，しかも1セット処理しかできないのに対し，これらの命令は複数のビットに対しての処理

(a)命令の配置

図14
メモリ配置とパイプライン



(b)パイプライン動作

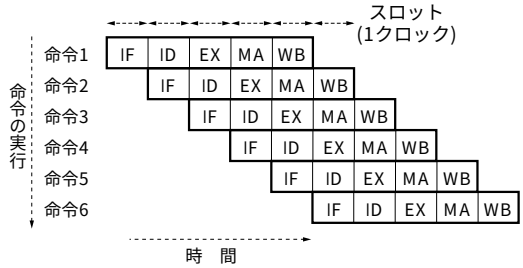


図13　基本パイプライン動作

が行えます．

● SH-3のパイプラインについて

SH-3は5段のパイプラインで構成されています(**図13**)．それぞれのステージでは次の処理を行っています．

IF：命令フェッチ
ID：命令デコード
EX：実行(メモリ・アクセス時はアドレス計算のみ)
MA：データ・アクセス
WB：内部レジスタへの書き込み

一般に，パイプラインのステージ数は少なければ分岐によるペナルティが少なく，またレイテンシも少ないためハードウェアによる調整の必要が少なくなります．逆に，ステージ数が多ければ一つ一つのステージの回路が単純になり，動作周波数を上げやすいと言われています．

SH-3の5段パイプラインは，クロック周波数や組み込まれるアプリケーションからすると，妥当なパイプライン・ステージ数といえます．

● パイプライン・ハザード

パイプライン動作を乱さないように実行すれば，1命令/クロックで実行できます(**図14**)．しかし，次に示す要因でパイプライン動作は一時停止します．

(1) 4の倍数にないアドレスにある命令のメモリ・アクセス

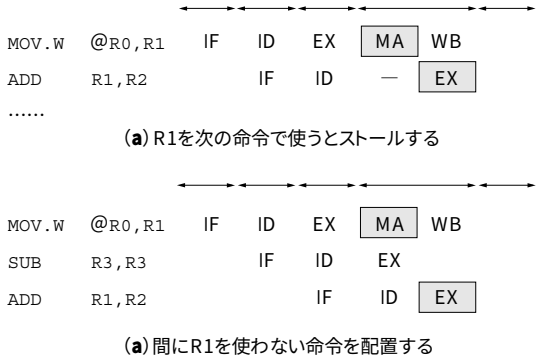


図15　命令配置とパイプライン

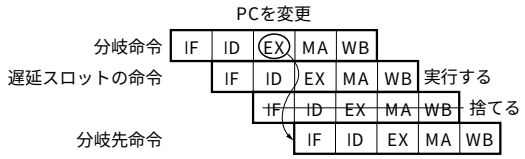


図16　遅延分岐とパイプライン

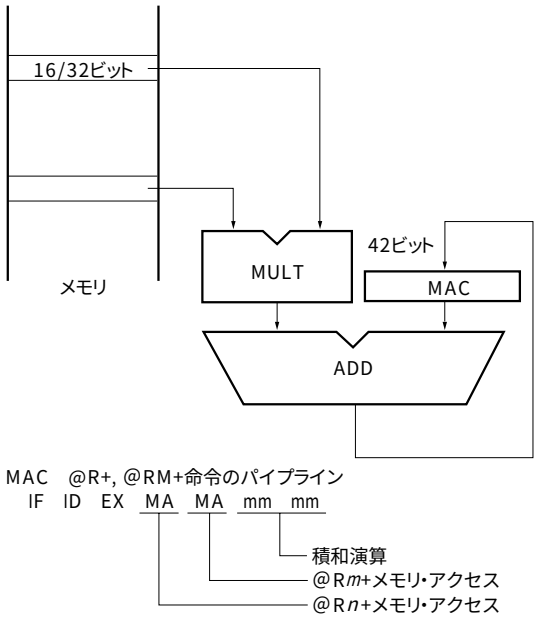


図17　MAC命令の動作

(2) 分岐命令の実行
(3) メモリからのデータ読み込みによる実行遅延

(1)は命令フェッチ(IF)とデータ・アクセス(MA)ステージの競合によるインターロック機構の動作で，できるだけ4の倍数以外のアドレスに配置した命令が，MAステージを持たないレジスタ間演算などにするよう配置することで回避できます．

(3)はメモリから読み出したデータを，すぐに演算しないように命令を配置することで回避します(**図15**)．

● 遅延分岐

分岐命令によるペナルティ，これはパイプラインで命令を実行したり，命令を先読み(プリフェッチ)しているとどうしても発生する問題です．実行しようとした命令が分岐なら，すでにプリフェッチしている命令を捨てて，分岐先命令のフェッチを開始しなければなりません．この時間をできるだけ短くするために，分岐予測や遅延分岐，また条件付き実行などさまざまな手法がとられています．

たとえば条件付き実行は，条件に合わなかったら分岐しないでノー・オペレーションに命令を変更してしまうものです．この方法は分岐先が近いときには効果が高いのですが，分岐先が遠いと逆効果となります．

SH-3はパイプラインを5ステージとしているので，分岐命令を実行し，プログラム・カウンタを変更しているEXステージでは次々命令までフェッチしています．そこで，この中の1命令を実行してしまうことで分岐ペナルティを最小限に抑えようというのが，遅延分岐の方式です(**図16**)．分岐が遠くても近くても，また分岐が多いときにも効果が高い処理方式です．

なお，以上のパイプライン動作は内蔵キャッシュ・メモリを利用した場合の動作です．外部メモリをアクセスすると1ステージの時間が数クロックとなります．実際のクロック数はCPUとバス・クロックの比，さらにメモリ種別やウェイト・サイクル数により変わります．

● 積和演算

ディジタル・フィルタ，FFTをはじめとするデータ圧縮や伸張に用いられる離散コサイン変換で行われる行列演算，さらにはカー・ナビゲーションなどで用いられる座標変換のベクトル演算も積和演算の塊です．携帯電話の音質改善やディジタル・カメラの画像圧縮/伸張，ゲーム機の画像処理など数えあげればきりがないほど積和演算は用いられます．

そこでSH-1は開発当初から積和演算回路と命令を実装しました．DSP並みの回路を持つことで，CPU＋DSPであったシステムをSH-1だけで置き換えることも可能になりました．SH-3はSH-1の上位互換なので，積和演算命令も持っています．

SH-3はSH-2と同じ，16ビットまたは32ビット固定小数点データを2～3クロックで積和演算できる回路を実装しています(**図17**)．