

日立 SuperH RISC engine  
SH7091 プログラミングマニュアル  
SH7091

---

SH7091 プログラミングマニュアル

発行年月日

平成 10 年 4 月 第 1.0 版

発行

株式会社 日立製作所

電子統括営業本部

編集

株式会社日立マイコンシステム

技術情報センタ

©株式会社 日立製作所 1998

---

---

## ご注意

---

1. 本書に記載の製品及び技術のうち「外国為替及び外国貿易法」に基づき安全保障貿易管理関連貨物・技術に該当するものを輸出する場合、または国外に持ち出す場合は日本国政府の許可が必要です。
2. 本書に記載された情報の使用に際して、弊社もしくは第三者の特許権、著作権、商標権、その他の知的所有権等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。また本書に記載された情報を使用した事により第三者の知的所有権等の権利に関わる問題が生じた場合、弊社はその責を負いませんので予めご了承ください。
3. 製品及び製品仕様は予告無く変更する場合がありますので、最終的な設計、ご購入、ご使用に際しましては、事前に最新の製品規格または仕様書をお求めになりご確認ください。
4. 弊社は品質・信頼性の向上に努めておりますが、宇宙、航空、原子力、燃焼制御、運輸、交通、各種安全装置、ライフサポート関連の医療機器等のように、特別な品質・信頼性が要求され、その故障や誤動作が直接人命を脅かしたり、人体に危害を及ぼす恐れのある用途にご使用をお考えのお客様は、事前に弊社営業担当迄ご相談をお願い致します。
5. 設計に際しては、特に最大定格、動作電源電圧範囲、放熱特性、実装条件及びその他諸条件につきましては、弊社保証範囲内でご使用いただきますようお願い致します。  
保証値を越えてご使用された場合の故障及び事故につきましては、弊社はその責を負いません。  
また保証値内のご使用であっても半導体製品について通常予測される故障発生率、故障モードをご考慮の上、弊社製品の動作が原因でご使用機器が人身事故、火災事故、その他の拡大損害を生じないようにフェールセーフ等のシステム上の対策を講じて頂きますようお願い致します。
6. 本製品は耐放射線設計をしておりません。
7. 本書の一部または全部を弊社の文書による承認なしに転載または複製することを堅くお断り致します。
8. 本書をはじめ弊社半導体についてのお問い合わせ、ご相談は弊社営業担当迄お願い致します。

---

# 目次

---

## 第 1 章 概要

1.1	SH7091 の特長 .....	1-1
1.2	ブロック図 .....	1-5

## 第 2 章 プログラミングモデル

2.1	データフォーマット .....	2-1
2.2	レジスタの構成 .....	2-2
2.2.1	特権モードとバンク .....	2-2
2.2.2	汎用レジスタ .....	2-5
2.2.3	浮動小数点レジスタ .....	2-6
2.2.4	コントロールレジスタ .....	2-8
2.2.5	システムレジスタ .....	2-9
2.3	メモリ割り付けレジスタ .....	2-11
2.4	レジスタのデータ形式 .....	2-11
2.5	メモリ上でのデータ形式 .....	2-12
2.6	処理状態 .....	2-13
2.7	処理モード .....	2-14

## 第 3 章 メモリマネジメントユニット (MMU)

3.1	概要 .....	3-1
3.1.1	特長 .....	3-1
3.1.2	MMU の役割 .....	3-1
3.1.3	レジスタの構成 .....	3-3
3.1.4	注意事項 .....	3-3
3.2	レジスタの説明 .....	3-4
3.3	メモリ空間 .....	3-7
3.3.1	物理メモリ空間 .....	3-7
3.3.2	外部メモリ空間 .....	3-10
3.3.3	仮想メモリ空間 .....	3-10
3.3.4	内蔵 RAM 空間 .....	3-12
3.3.5	アドレス変換 .....	3-12
3.3.6	単一仮想記憶モードと多重仮想記憶モード .....	3-12
3.3.7	アドレス空間識別子 (ASID) .....	3-12
3.4	TLB の機能 .....	3-13
3.4.1	共用 TLB (UTLB) の構成 .....	3-13

3.4.2	命令 TLB (ITLB) の構成 .....	3-16
3.4.3	アドレス変換方式 .....	3-17
3.5	MMU の機能 .....	3-19
3.5.1	MMU のハードウェア管理 .....	3-19
3.5.2	MMU のソフトウェア管理 .....	3-19
3.5.3	MMU の命令 (LDTLB) .....	3-19
3.5.4	ハードウェア ITLB ミスハンドリング .....	3-20
3.5.5	シノニム問題の回避 .....	3-20
3.6	MMU 例外 .....	3-21
3.6.1	命令 TLB 多重ヒット例外 .....	3-21
3.6.2	命令 TLB ミス例外 .....	3-22
3.6.3	命令 TLB 保護違反例外 .....	3-22
3.6.4	データ TLB 多重ヒット例外 .....	3-23
3.6.5	データ TLB ミス例外 .....	3-23
3.6.6	データ TLB 保護違反例外 .....	3-24
3.6.7	初期ページ書き込み例外 .....	3-25
3.7	メモリ割り付け TLB の構成 .....	3-26
3.7.1	ITLB アドレスアレイ .....	3-26
3.7.2	ITLB データアレイ 1 .....	3-27
3.7.3	ITLB データアレイ 2 .....	3-27
3.7.4	UTLB アドレスアレイ .....	3-28
3.7.5	UTLB データアレイ 1 .....	3-29
3.7.6	UTLB データアレイ 2 .....	3-30

## 第 4 章 キャッシュ

4.1	概要 .....	4-1
4.1.1	特長 .....	4-1
4.1.2	レジスタの構成 .....	4-1
4.2	レジスタの説明 .....	4-2
4.3	オペランドキャッシュ (OC) .....	4-4
4.3.1	構成 .....	4-4
4.3.2	リード動作 .....	4-5
4.3.3	ライト動作 .....	4-6
4.3.4	ライトバックバッファ .....	4-7
4.3.5	ライトスルーバッファ .....	4-7
4.3.6	RAM モード .....	4-7
4.3.7	OC インデックスモード .....	4-8
4.3.8	キャッシュと外部メモリとのコヒーレンシ .....	4-8
4.3.9	プリフェッチ動作 .....	4-9
4.4	命令キャッシュ (IC) .....	4-10
4.4.1	構成 .....	4-10
4.4.2	リード動作 .....	4-11
4.4.3	IC インデックスモード .....	4-11

4.5	メモリ割り付けキャッシュの構成 .....	4-12
4.5.1	IC アドレスアレイ .....	4-12
4.5.2	IC データアレイ .....	4-13
4.5.3	OC アドレスアレイ .....	4-14
4.5.4	OC データアレイ .....	4-15
4.6	ストアキュー .....	4-16
4.6.1	SQ の構成 .....	4-16
4.6.2	SQ への書き込み .....	4-16
4.6.3	外部メモリへの転送 .....	4-17
4.6.4	SQ へのプロテクション .....	4-17

## 第5章 例外処理

5.1	概要 .....	5-1
5.1.1	特長 .....	5-1
5.1.2	レジスタ構成 .....	5-1
5.2	レジスタの説明 .....	5-2
5.3	例外処理の機能 .....	5-3
5.3.1	例外処理の流れ .....	5-3
5.3.2	例外処理ベクタアドレス .....	5-3
5.4	例外の種類と優先順位 .....	5-4
5.5	例外フロー .....	5-6
5.5.1	例外フロー .....	5-6
5.5.2	例外要因の受け付け .....	5-7
5.5.3	例外要求と BL ビット .....	5-8
5.5.4	例外処理からの復帰 .....	5-8
5.6	各例外の説明 .....	5-9
5.6.1	リセット .....	5-9
5.6.2	一般例外 .....	5-13
5.6.3	割り込み .....	5-22
5.6.4	複数回の例外が発生する場合の優先順位 .....	5-24
5.7	注意事項 .....	5-25

## 第6章 浮動小数点ユニット

6.1	概要 .....	6-1
6.2	データフォーマット .....	6-2
6.2.1	浮動小数点フォーマット .....	6-2
6.2.2	非数 (NaN) .....	6-3
6.2.3	非正規化数 .....	6-4
6.3	レジスタ .....	6-5
6.3.1	浮動小数点レジスタ .....	6-5
6.3.2	浮動小数点ステータス / コントロールレジスタ (FPSCR) .....	6-7
6.3.3	浮動小数点通信レジスタ (FPUL) .....	6-8

6.4	丸め .....	6-8
6.5	浮動小数点例外 .....	6-9
6.6	グラフィックサポート機能 .....	6-10
6.6.1	ジオメトリック演算命令 .....	6-10
6.6.2	ペア単精度データ転送 .....	6-11

## 第7章 命令セット

7.1	実行環境 .....	7-1
7.2	アドレッシングモード .....	7-3
7.3	命令セット .....	7-6

## 第8章 バイブライン動作

8.1	パイプライン .....	8-1
8.2	並列実行性 .....	8-7
8.3	実行サイクルとパイプラインストール .....	8-10

## 第9章 低消費電力モード

9.1	概要 .....	9-1
9.1.1	低消費電力モードの種類 .....	9-1
9.1.2	レジスタ構成 .....	9-2
9.2	レジスタの説明 .....	9-2
9.2.1	スタンバイコントロールレジスタ (STBCR) .....	9-2
9.2.2	周辺モジュール端子ハイインピーダンス制御 .....	9-4
9.2.3	周辺モジュール端子プルアップ制御 .....	9-5
9.2.4	スタンバイコントロールレジスタ2 (STBCR2) .....	9-5
9.3	スリープモード .....	9-6
9.3.1	スリープモードへの遷移 .....	9-6
9.3.2	スリープモードの解除 .....	9-6
9.4	ディープスリープモード .....	9-6
9.4.1	ディープスリープモードへの遷移 .....	9-6
9.4.2	ディープスリープモードの解除 .....	9-6
9.5	スタンバイモード .....	9-7
9.5.1	スタンバイモードへの遷移 .....	9-7
9.5.2	スタンバイモードの解除 .....	9-8
9.5.3	クロックポーズ機能 .....	9-8
9.6	モジュールスタンバイ機能 .....	9-9
9.6.1	モジュールスタンバイ機能への遷移 .....	9-9
9.6.2	モジュールスタンバイ機能の解除 .....	9-9

## 第 10 章 各命令の説明

10.1	ADD ADD binary 算術演算命令 .....	10-13
10.2	ADDC ADD with Carry 算術演算命令 .....	10-14
10.3	ADDV ADD with (Vflag) overflow check 算術演算命令 .....	10-15
10.4	AND AND logical 論理演算命令 .....	10-16
10.5	BF Branch if False 分岐命令 .....	10-18
10.6	BF/S Branch if False with delay Slot 分岐命令 .....	10-19
10.7	BRA BRANch 分岐命令 .....	10-21
10.8	BRAF BRANch Far 分岐命令 .....	10-22
10.9	BSR Branch to SubRoutine 分岐命令 .....	10-23
10.10	BSRF Branch to SubRoutine Far 分岐命令 .....	10-25
10.11	BT Branch if True 分岐命令 .....	10-26
10.12	BT/S Branch if True with delay Slot 分岐命令 .....	10-27
10.13	CLRMAC CLear MAC register システム制御命令 .....	10-29
10.14	CLRS CLear Sbit システム制御命令 .....	10-30
10.15	CLRT CLear Tbit システム制御命令 .....	10-31
10.16	CMP/cond CoMPare conditionally 算術演算命令 .....	10-32
10.17	DIV0S DIVide(step0) as Signed 算術演算命令 .....	10-35
10.18	DIV0U DIVide (step0) as Unsigned 算術演算命令 .....	10-36
10.19	DIV1 DIVide 1 step 算術演算命令 .....	10-37
10.20	DMULS.L Double-length MULTiply as Signed 算術演算命令 .....	10-41
10.21	DMULU.L Double-length MULTiply as Unsigned 算術演算命令 .....	10-43
10.22	DT Decrement and Test 算術演算命令 .....	10-45
10.23	EXTS EXTend as Signed 算術演算命令 .....	10-46
10.24	EXTU EXTend as Unsigned 算術演算命令 .....	10-47
10.25	FABS Floating Point Absolute Value 浮動小数点命令 .....	10-48
10.26	FADD Floating Point Add 浮動小数点命令 .....	10-49
10.27	FCMP Floating Point Compare 浮動小数点命令 .....	10-51
10.28	FCNVDS Floating Point Convert Double to Single precision 浮動小数点命令 .....	10-54
10.29	FCNVSD Floating Point Convert Single to Double Precision 浮動小数点命令 .....	10-56
10.30	FDIV Floating Point Divide 浮動小数点命令 .....	10-58
10.31	FIPR Floating Point Inner Product 浮動小数点命令 .....	10-61
10.32	FLDI0 Floating Point Load 0.0 浮動小数点命令 .....	10-63
10.33	FLDI1 Floating Point Load 1.0 浮動小数点命令 .....	10-64
10.34	FLDS Floating Point Load to System Register 浮動小数点命令 .....	10-65
10.35	FLOAT Floating Point Convert from Integer 浮動小数点命令 .....	10-66
10.36	FMAC Floating Point Multiply and Accumulate 浮動小数点命令 .....	10-67
10.37	FMOV Floating Point Move 浮動小数点命令 .....	10-71
10.38	FMOV Floating Point Move Extension 浮動小数点命令 .....	10-74
10.39	FMUL Floating Point Multiply 浮動小数点命令 .....	10-76
10.40	FNEG Floating Point Negate Value 浮動小数点命令 .....	10-78
10.41	FRCHG FR-Bit Change 浮動小数点命令 .....	10-79
10.42	FSCHG SZ-Bit Change 浮動小数点命令 .....	10-80
10.43	FSQRT Floating Point Square Root 浮動小数点命令 .....	10-81
10.44	FSTS Floating Point Store System Register 浮動小数点命令 .....	10-84
10.45	FSUB Floating Point Subtract 浮動小数点命令 .....	10-85
10.46	FTRC Floating Point Truncate and Convert to Integer 浮動小数点命令 .....	10-87

10.47	FTRV Floating Point Transform Vector 浮動小数点命令 .....	10-90
10.48	JMP JuMP 分岐命令 .....	10-92
10.49	JSR Jump to SubRoutine 分岐命令 .....	10-93
10.50	LDC LoaD to Control register システム制御命令 .....	10-94
10.51	LDS LoaD to FPU System register システム制御命令 .....	10-95
10.52	LDTLB LoaD PTEH/PTEL/PTEA to TLB システム制御命令 .....	10-97
10.53	MAC.L MUlty and ACcumulate Long 算術演算命令 .....	10-98
10.54	MAC.W MUlty and ACcumulate Word 算術演算命令 .....	10-101
10.55	MOV MOVe data データ転送命令 .....	10-103
10.56	MOV MOVe Constant Value データ転送命令 .....	10-107
10.57	MOV MOVe Global data データ転送命令 .....	10-109
10.58	MOV MOVe structure data データ転送命令 .....	10-111
10.59	MOVA MOVe effective Address データ転送命令 .....	10-114
10.60	MOVCA.L Move with Cache Block Allocation データ転送命令 .....	10-115
10.61	MOVT MOVe Tbit データ転送命令 .....	10-116
10.62	MUL.L MUlty Long 算術演算命令 .....	10-117
10.63	MULS.W MUlty as Signed Word 算術演算命令 .....	10-118
10.64	MULU.W MUlty as Unsigned Word 算術演算命令 .....	10-119
10.65	NEG NEGate 算術演算命令 .....	10-120
10.66	NEGC NEGate with Carry 算術演算命令 .....	10-121
10.67	NOP No OPeration システム制御命令 .....	10-122
10.68	NOT NOT-logical complement 論理演算命令 .....	10-123
10.69	OCBI Operand Cache Block Invalidate データ転送命令 .....	10-124
10.70	OCBP Operand Cache Block Purge データ転送命令 .....	10-125
10.71	OCBWB Operand Cache Block Write Back データ転送命令 .....	10-126
10.72	OR OR logical 論理演算命令 .....	10-127
10.73	PREF PREFetch data to cache データ転送命令 .....	10-128
10.74	ROTCL ROTate with Carry Left シフト命令 .....	10-129
10.75	ROTCR ROTate with Carry Right シフト命令 .....	10-130
10.76	ROTL ROTate Left シフト命令 .....	10-131
10.77	ROTR ROTate Right シフト命令 .....	10-132
10.78	RTE ReTurn from Exception システム制御命令 (特権命令) .....	10-133
10.79	RTS ReTurn from SubRoutine 分岐命令 .....	10-134
10.80	SETS SET Sbit システム制御命令 .....	10-135
10.81	SETT SET Tbit システム制御命令 .....	10-136
10.82	SHAD SHift Arithmetic Dynamically シフト命令 .....	10-137
10.83	SHAL SHift Arithmetic Left シフト命令 .....	10-139
10.84	SHAR SHift Arithmetic Right シフト命令 .....	10-140
10.85	SHLD SHift Logical Dynamically シフト命令 .....	10-141
10.86	SHLL SHift Logical Left シフト命令 .....	10-143
10.87	SHLLn n bits SHift Logical Left シフト命令 .....	10-144
10.88	SHLR SHift Logical Right シフト命令 .....	10-146
10.89	SHLRn n bits SHift Logical Right シフト命令 .....	10-147
10.90	SLEEP SLEEP システム制御命令 .....	10-149
10.91	STC STore Control register システム制御命令 .....	10-150
10.92	STS STore from FPU System register システム制御命令 .....	10-151
10.93	SUB SUBtract binary 算術演算命令 .....	10-153



10.94	SUBC SUBtract with Carry 算術演算命令 .....	10-154
10.95	SUBV SUBtract with (Vflag)underflow check 算術演算命令 .....	10-155
10.96	SWAP SWAP register halves データ転送命令 .....	10-156
10.97	TAS Test And Set 論理演算命令 .....	10-157
10.98	TRAPA TRAP Always システム制御命令 .....	10-158
10.99	TST TeST logical 論理演算命令 .....	10-159
10.100	XOR eXclusive OR logical 論理演算命令 .....	10-161
10.101	XTRCT eXTRaCT データ転送命令 .....	10-162

## 付録

A.	アドレス一覧 .....	A-1
B.	命令のプリフェッチとその副作用について .....	B-1

---

# 1. 概要

---

## 1.1 SH7091 の特長

SH7091 は SH-1、SH-2、SH-3、SH-3E マイクロコンピュータとのオブジェクトコードレベルでの上位互換性を特長とする 32 ビット RISC (縮小命令セットコンピュータ) マイコンです。SH7091 は 8k バイトの命令キャッシュ、コピーバックまたはライトスルーモードの選択が可能な 16k バイトオペランドキャッシュ、4 エントリのフルアソシアティブ命令 TLB (変換ルックアサイドバッファ)、64 エントリのフルアソシアティブ共用 TLB 付き MMU (メモリ管理ユニット) を内蔵しています。

SH7091 は、外部回路なしに DRAM、シンクロナス DRAM に直結できるバスステートコントローラ (BSC) を内蔵しています。16 ビット固定長の命令セットにより、32 ビット命令に比較してプログラムコードのサイズをほぼ 50% 縮小することができます。

SH7091 の特長を表 1.1 に示します。

表 1.1 SH7091 の特長 (1)

項目	特長
LSI	<ul style="list-style-type: none"><li>・ 周波数：200MHz</li><li>・ 性能：<ul style="list-style-type: none"><li>- 360 MIPS (200 MHz)</li><li>- 1.4 GFLOPS (200 MHz)</li></ul></li><li>・ スーパスカラ：2 つの命令の並行実行</li><li>・ 電圧：1.8 V (内部)、3.3 V (IO)</li><li>・ パッケージ：256 ピン BGA</li><li>・ 外部バス：<ul style="list-style-type: none"><li>- 独立 26 ビットアドレス + 64 ビットデータ</li><li>- (内部バス周波数に対して) 1/2、1/3、1/4、1/6、1/8 外部バス周波数</li></ul></li></ul>
CPU	<ul style="list-style-type: none"><li>・ 日立オリジナルアーキテクチャ</li><li>・ 32 ビット内部データバス</li><li>・ 汎用レジスタファイル：<ul style="list-style-type: none"><li>- 16 本の 32 ビット汎用レジスタ (および 8 本の 32 ビットシャドウレジスタ)</li><li>- 7 本の 32 ビット制御レジスタ</li><li>- 4 本の 32 ビットシステムレジスタ</li></ul></li><li>・ RISC タイプ命令セット (SH シリーズと上位互換性)：<ul style="list-style-type: none"><li>- 命令長：コードの効率改善のための 16 ビット固定長</li><li>- ロードストアアーキテクチャ</li><li>- 遅延分岐命令</li><li>- 条件付き実行</li><li>- C 言語に基づく命令セット</li></ul></li><li>・ FPU を含む 2 命令同時実行型スーパスカラ</li><li>・ 命令実行時間：最大 2 命令 / サイクル</li><li>・ 仮想アドレス空間：4G バイト (448M バイト外部メモリ空間)</li><li>・ 空間識別子 ASID：8 ビット、256 仮想アドレス空間</li><li>・ 乗算器内蔵</li><li>・ 5 段パイプライン</li></ul>

## 1. 概要

表 1.1 SH7091 の特長 (2)

項目	特長
FPU	<ul style="list-style-type: none"> <li>・浮動小数点コプロセッサ内蔵</li> <li>・単精度 (32 ビット) および倍精度 (64 ビット) をサポート</li> <li>・IEEE754 に準拠したデータタイプおよび例外をサポート</li> <li>・丸めモード：最近傍への丸め、またはゼロへの丸め</li> <li>・非正規化数の扱い：0 への切捨て、または IEEE754 に準拠のための割り込み発生</li> <li>・浮動小数点レジスタ：32 ビット x16 ワード x2 バンク (単精度 x16 ワードまたは倍精度 x8 ワード) x2 バンク</li> <li>・32 ビット CPU-FPU インタフェースレジスタ (FPUL)</li> <li>・FMAC (乗算およびアキュムレート) 命令をサポート</li> <li>・FDIV (除算) / FSQRT (平方根) 命令をサポート</li> <li>・FLD0 / FLD1 (ロード定数 0/1) 命令をサポート</li> <li>・命令実行時間 <ul style="list-style-type: none"> <li>- レイテンシ (FMAC/FADD/FSUB/FMUL)：3 サイクル (単精度)、8 サイクル (倍精度)</li> <li>- ピッチ (FMAC/FADD/FSUB/FMUL)：1 サイクル (単精度)、6 サイクル (倍精度)</li> </ul> </li> </ul> <p>【注】：FMAC は単精度に対してのみサポートしています。</p> <ul style="list-style-type: none"> <li>・3D グラフィック命令 (単精度のみ)： <ul style="list-style-type: none"> <li>- 4 次元ベクトル変換および行列演算 (FTRV)、4 サイクル (ピッチ)、7 サイクル (レイテンシ)</li> <li>- 4 次元ベクトル (FIPR) の内積、1 サイクル (ピッチ)、4 サイクル (レイテンシ)</li> </ul> </li> <li>・5 段パイプライン</li> </ul>
クロックパルス発生回路 (CPG)	<ul style="list-style-type: none"> <li>・メインクロック選択可能：EXTAL の 1/2、1、3、6 倍</li> <li>・クロックモード： <ul style="list-style-type: none"> <li>- CPU 周波数：(メインクロックに対して) 1、1/2、1/3、1/4、1/6、1/8：最大 200MHz</li> <li>- バス周波数：(メインクロックに対して) 1/2、1/3、1/4、1/6、1/8：最大 100MHz</li> <li>- 周辺周波数：(メインクロックに対して) 1/2、1/3、1/4、1/6、1/8：最大 50MHz</li> </ul> </li> <li>・低消費電力モード <ul style="list-style-type: none"> <li>- スリープモード</li> <li>- スタンバイモード</li> <li>- モジュールスタンバイ機能</li> </ul> </li> <li>・1 チャネルのウォッチドッグタイマ</li> </ul>
メモリ管理ユニット (MMU)	<ul style="list-style-type: none"> <li>・4G バイトのアドレス空間、256 のアドレス空間識別子 (ASID 8 ビット)</li> <li>・単一仮想記憶モードと多重仮想記憶モード</li> <li>・複数のページサイズをサポート：1k、4k、64k、1M バイト</li> <li>・命令に対する 4 エントリのフルアソシアティブ TLB</li> <li>・命令およびオペランドに対する 64 エントリのフルアソシアティブ TLB</li> <li>・ソフトウェアによる入換方法およびランダムカウンタ方式入換アルゴリズムをサポート</li> <li>・TLB の内容はアドレスマッピングにより直接アクセス可能</li> </ul>

表 1.1 SH7091 の特長 (3)

項目	特長
キャッシュメモリ	<ul style="list-style-type: none"> <li>・命令キャッシュ(IC) <ul style="list-style-type: none"> <li>- 8k バイト、ダイレクトマッピング</li> <li>- 256 エントリ、32 バイトブロック長</li> <li>- 通常モード (8k バイトキャッシュ)</li> <li>- インデックスモード</li> </ul> </li> <li>・オペランドキャッシュ (OC) <ul style="list-style-type: none"> <li>- 16k バイト、ダイレクトマッピング</li> <li>- 512 エントリ、32 バイトブロック長</li> <li>- 通常モード (16k バイトキャッシュ)</li> <li>- インデックスモード</li> <li>- RAM モード (8k バイトキャッシュ + 8k バイト RAM)</li> <li>- 選択可能な書き込み方式 (コピーバック / ライトスルー)</li> </ul> </li> <li>・1 段コピーバックバッファ、1 段ライトスルーバッファ</li> <li>・キャッシュメモリの内容はアドレスマッピングにより直接アクセス可能 (内蔵メモリとして使用可能)。</li> <li>・ストアキュー (32 バイト×2 エントリ)</li> </ul>
割り込みコントローラ (INTC)	<ul style="list-style-type: none"> <li>・5 本の独立した外部割り込み: NMI、IRL3~IRL0</li> <li>・15 レベルの符号化した外部割り込み: IRL3~IRL0</li> <li>・内蔵周辺割り込み: モジュールごとに優先レベルを設定</li> </ul>
ユーザブレイクコントローラ (UBC)	<ul style="list-style-type: none"> <li>・ユーザブレイク割り込みによるデバッグをサポート</li> <li>・2 本のブレイクチャネル</li> <li>・アドレス、データ値、アクセスのタイプ、データサイズはすべてブレイク条件として設定可能。</li> <li>・シーケンシャルブレイク機能をサポート</li> </ul>
バスステートコントローラ (BSC)	<ul style="list-style-type: none"> <li>・外部メモリアccessをサポート <ul style="list-style-type: none"> <li>- 64/32/16/8 ビットの外部データバス</li> </ul> </li> <li>・それぞれ最大 64M バイトの 7 つのエリアに分割した外部メモリ空間、各エリアには次の機能を設定可能。 <ul style="list-style-type: none"> <li>- バスサイズ (8、16、32、または 64 ビット)</li> <li>- ウェイトサイクル数 (ハードウェアウェイト機能もサポート)</li> <li>- 空間のタイプを設定することにより、DRAM、シンクロナス DRAM、バースト ROM に対する直結が可能。</li> <li>- 高速ページモードと DRAM 用 EDO をサポート</li> <li>- PCMCIA インタフェースをサポート</li> <li>- 該当エリアに対するチップセレクト信号 (<math>\overline{CS0}</math>~<math>\overline{CS6}</math>) を出力</li> </ul> </li> <li>・DRAM / シンクロナス DRAM リフレッシュ機能 <ul style="list-style-type: none"> <li>- プログラマブルなりフレッシュ間隔</li> <li>- CAS ビフォー RAS リフレッシュモードおよびセルフリフレッシュモードをサポート</li> </ul> </li> <li>・DRAM / シンクロナス DRAM バーストアクセス機能</li> <li>・ビッグエンディアンまたはリトルエンディアンを設定可能</li> </ul>

## 1. 概要

表 1.1 SH7091 の特長 (4)

項目	特長
ダイレクト メモリアクセス コントローラ (DMAC)	<ul style="list-style-type: none"><li>・ 4 チャンネル物理アドレス DMA コントローラ</li><li>・ 転送データサイズ： 8、16、32、64 ビット、または 32 バイト</li><li>・ アドレスモード<ul style="list-style-type: none"><li>- 1 バスサイクルシングルアドレスモード</li><li>- 2 バスサイクルデュアルアドレスモード</li></ul></li><li>・ 転送要求： 外部、内蔵モジュール、またはオートリクエスト</li><li>・ バスモード： サイクルスチール、またはバーストモード</li><li>・ オンデマンドデータ転送をサポート</li></ul>
タイマ (TMU)	<ul style="list-style-type: none"><li>・ 3 チャンネルのオートリロード方式 32 ビットタイマ</li><li>・ インプットキャプチャ機能</li><li>・ 7 種類のカウンタ入力クロックを選択可能</li></ul>
リアルタイムク ロック (RTC)	<ul style="list-style-type: none"><li>・ 内蔵クロック、カレンダー機能</li><li>・ 最大 1/256 秒の分解能 (サイクル割り込み) を持つ内蔵 32kHz 水晶発振回路</li></ul>
シリアルコミュ ニケーションイ ンタフェース (SCI、SCIF)	<ul style="list-style-type: none"><li>・ 2 本の全二重通信チャンネル (SCI、SCIF)</li><li>・ チャンネル 1 (SCI)<ul style="list-style-type: none"><li>- 調歩同期式モードまたはクロック同期式モードの選択可能</li><li>- スマートカードインタフェースをサポート</li></ul></li><li>・ チャンネル 2 (SCIF)<ul style="list-style-type: none"><li>- 調歩同期式モードをサポート</li><li>- 送信部、受信部それぞれに 16 バイトの FIFO 付き</li></ul></li></ul>
パッケージ	<ul style="list-style-type: none"><li>・ 256 ピン BGA</li></ul>

## 1.2 ブロック図

図 1.1 に SH7091 の機能ブロック図を示します。

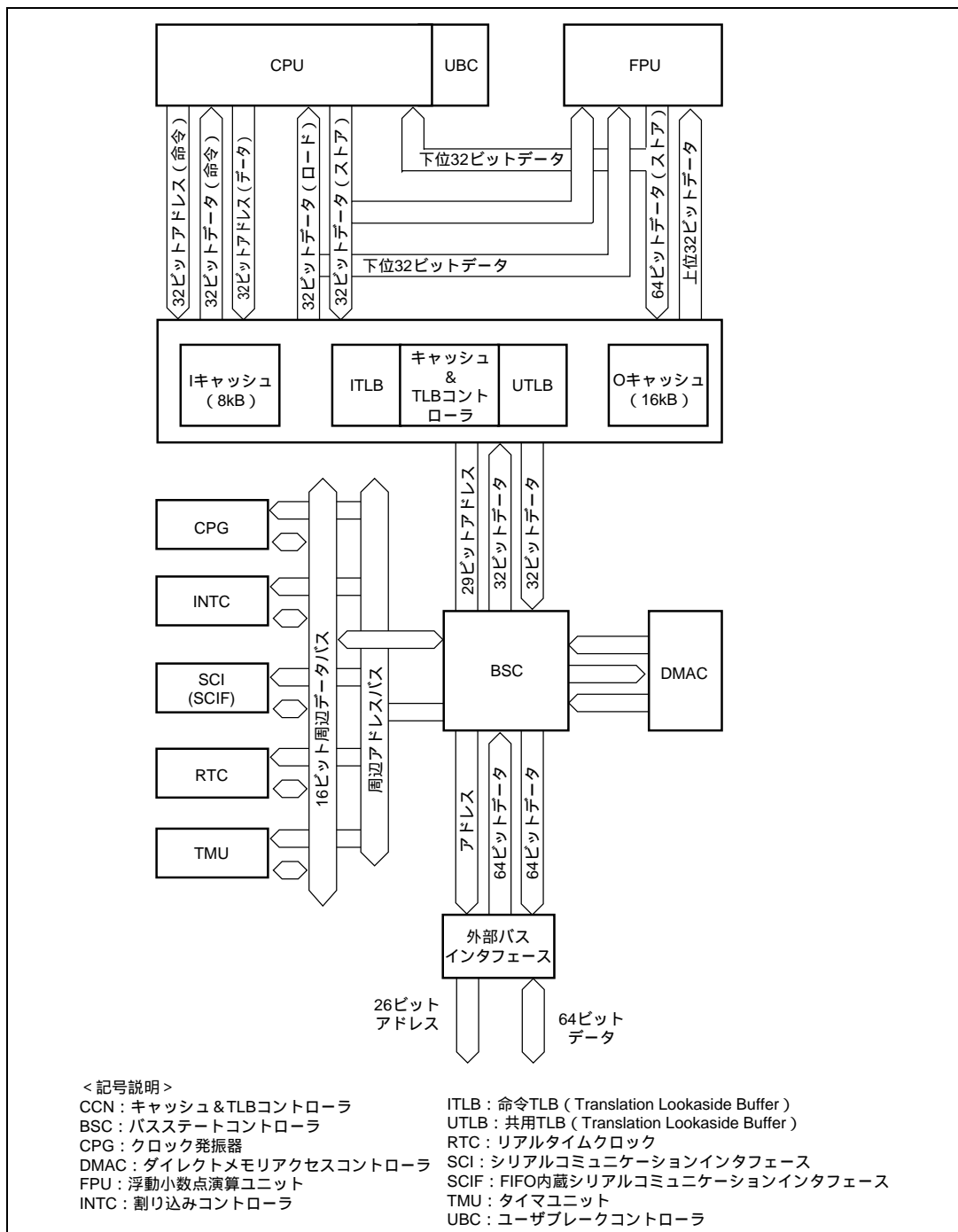


図 1.1 SH7091 機能ブロック図

---

## 2. プログラミングモデル

---

### 2.1 データフォーマット

SH7091 でサポートしているデータフォーマットを図 2.1 に示します。

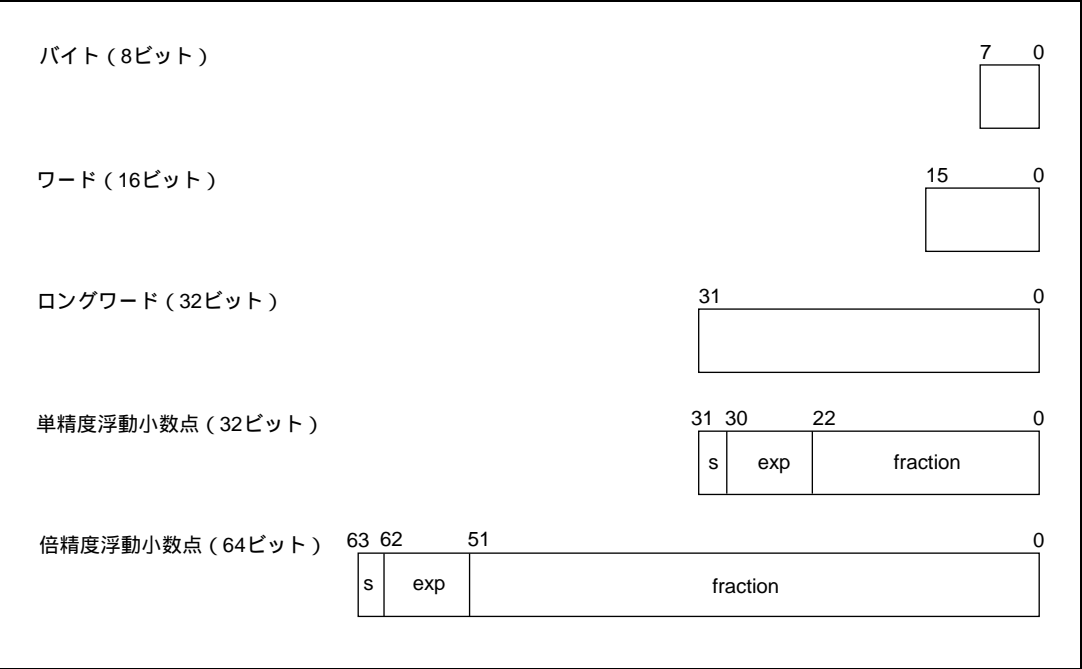


図 2.1 データフォーマット

## 2.2 レジスタの構成

### 2.2.1 特権モードとバンク

#### (1) 処理モード

処理モードにはユーザモードと特権モードの2つがあります。通常はユーザモードで動作し、例外が発生または割り込みを受け付けると特権モードになります。レジスタには、汎用レジスタ、システムレジスタ、コントロールレジスタ、および浮動小数点レジスタがあり、アクセスできるレジスタはそれぞれの処理モードで異なります。

#### (2) 汎用レジスタ

汎用レジスタには R0 から R15 までの 16 本のレジスタがあります。汎用レジスタ R0 から R7 は、バンクレジスタで、処理モードで切り換えることができます。

特権モードのとき、ステータスレジスタ (SR) のレジスタバンクビット (RB) により、汎用レジスタとしてアクセスできるレジスタとできないレジスタが決められます。汎用レジスタとしてアクセスできないレジスタは、コントロールレジスタのロード命令 (LDC) とストア命令 (STC) でアクセスします。

RB ビットが 1 のとき、つまりバンク 1 が選ばれているときは、バンク 1 の汎用レジスタ R0\_BANK1 から R7\_BANK1 とバンクに関係ない R8 から R15 との合計 16 本のレジスタが汎用レジスタ R0 から R15 としてアクセスすることができ、バンク 0 の汎用レジスタ R0\_BANK0 から R7\_BANK0 の 8 本のレジスタは LDC/STC 命令でアクセスできます。

RB ビットが 0 のとき、つまりバンク 0 が選ばれているときは、バンク 0 の汎用レジスタ R0\_BANK0 から R7\_BANK0 とバンクに関係ない R8 から R15 との合計 16 本のレジスタが汎用レジスタ R0 から R15 としてアクセスすることができ、バンク 1 の汎用レジスタ R0\_BANK1 から R7\_BANK1 の 8 本のレジスタは LDC/STC 命令でアクセスできます。

ユーザモードのときは、バンク 0 の汎用レジスタ R0\_BANK0 から R7\_BANK0 とバンクに関係ない R8 から R15 との合計 16 本のレジスタが汎用レジスタ R0 から R15 としてアクセスすることができ、バンク 1 の汎用レジスタ R0\_BANK1 から R7\_BANK1 の 8 本のレジスタはアクセスできません。

#### (3) コントロールレジスタ

コントロールレジスタには、処理モードで共通のグローバルベースレジスタ (GBR) とステータスレジスタ (SR) があり、特権モードでのみアクセスできる退避ステータスレジスタ (SSR)、退避プログラムカウンタ (SPC)、ベクタベースレジスタ (VBR)、退避ジェネラルレジスタ 15 (SGR)、デバッグベースレジスタ (DBR) があります。ステータスレジスタには、特権モードでのみアクセスできるビット (例えば RB ビット) があります。

#### (4) システムレジスタ

システムレジスタには、積和レジスタ (MACH/MACL)、プロシージャレジスタ (PR)、プログラムカウンタ (PC)、浮動小数点ステータス / コントロールレジスタ (FPSCR)、浮動小数点通信レジスタ (FPUL) があり、処理モードに関係しません。

#### (5) 浮動小数点レジスタ

浮動小数点レジスタには、FR0 ~ FR15、XF0 ~ XF15 の 32 本のレジスタがあります。FR0 ~ FR15、XF0 ~ XF15 を各々 FPR0\_BANK0 ~ FPR15\_BANK0、FPR0\_BANK1 ~ FPR15\_BANK1 のいずれのバンクに割り付けるか選択できます。

また、FR0 ~ FR15 は、DR0/2/4/6/8/10/12/14 (倍精度浮動小数点レジスタ、またはレジスタペア) の 8 本、FV0/4/8/12 (レジスタベクタ) の 4 本として使用でき、XF0 ~ XF15 は、XD0/2/4/6/8/10/12/14 (レジスタペア) の 8 本、XMTRX (レジスタ行列) の 1 本として使用できます。



リセット後のレジスタの値を表 2.1 に示します。

表 2.1 レジスタの初期値

区分	レジスタ	初期値*
汎用レジスタ	R0_BANK0 ~ R7_BANK0、 R0_BANK1 ~ R7_BANK1、 R8 ~ R15	不定
コントロールレジスタ	SR	MD ビットは 1、RB ビットは 1、BL ビットは 1、FD ビットは 0、I3 ~ I0 は 1111 (H'F)、予約ビットは 0、その他は不定
	GBR、SSR、SPC、SGR、DBR	不定
	VBR	H'00000000
システムレジスタ	MACH、MACL、PR、FPUL	不定
	PC	H'A0000000
	FPSCR	H'00040001
浮動小数点レジスタ	FR0 ~ FR15、XF0 ~ XF15	不定

【注】 \* パワーオンリセット、マニュアルリセットで初期化されます。

処理モード別の CPU レジスタ構成を図 2.2 に示します。

ユーザモードと特権モードは、ステータスレジスタの処理動作モードビット (MD) で切り換えます。

## 2. プログラミングモデル

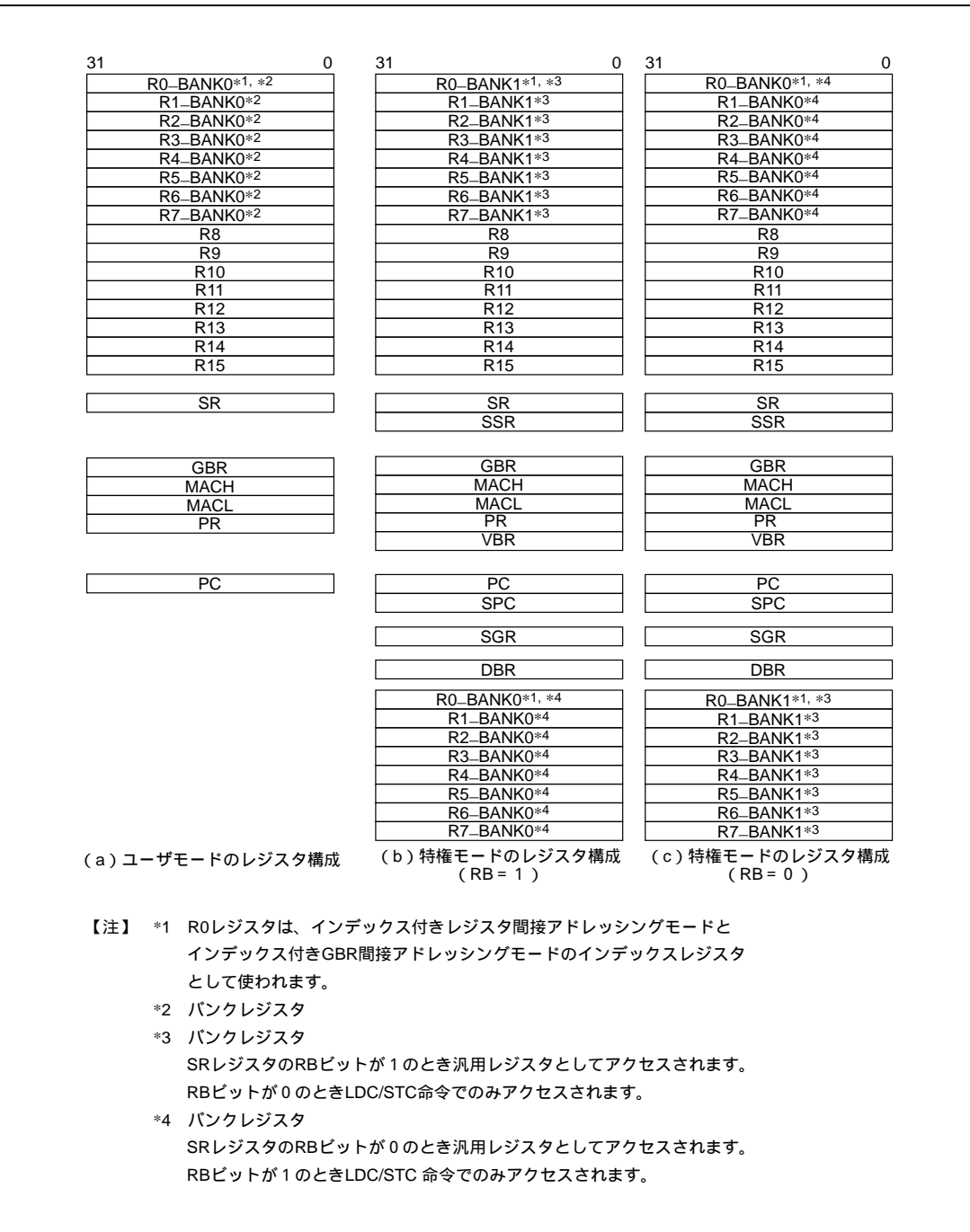


図 2.2 処理モード別の CPU レジスタ構成

## 2.2.2 汎用レジスタ

図 2.3 にプロセッサモードと汎用レジスタの関係を示します。SH7091 には 24 本の 32 ビット汎用レジスタ (R0\_BANK0~7\_BANK0、R0\_BANK1~7\_BANK1、R8~R15) があります。ただし、これらのうち 16 本のレジスタのみ 1 つのプロセッサモードで汎用レジスタ R0~R15 としてアクセスできます。SH7091 には特権モードとユーザモードの 2 つのプロセッサモードがあります。R0~R7 はその 2 つのモードにより次のように割り当てられます。

- R0\_BANK0~R7\_BANK0  
ユーザモード (SR.MD=0) では、常に R0~R7 に割り当てられます。  
特権モード (SR.MD=1) では、(SR.RB=0) の場合に限り R0~R7 に割り当てられます。
- R0\_BANK1~R7\_BANK1  
ユーザモードでは、アクセスできません。  
特権モードでは、(SR.RB=1) の場合に限り、R0~R7 に割り当てられます。

SR.MD=0 または (SR.MD=1, SR.RB=0)		(SR.MD=1, SR.RB=1)
R0	R0_BANK0	R0_BANK0
R1	R1_BANK0	R1_BANK0
R2	R2_BANK0	R2_BANK0
R3	R3_BANK0	R3_BANK0
R4	R4_BANK0	R4_BANK0
R5	R5_BANK0	R5_BANK0
R6	R6_BANK0	R6_BANK0
R7	R7_BANK0	R7_BANK0
R0_BANK1	R0_BANK1	R0
R1_BANK1	R1_BANK1	R1
R2_BANK1	R2_BANK1	R2
R3_BANK1	R3_BANK1	R3
R4_BANK1	R4_BANK1	R4
R5_BANK1	R5_BANK1	R5
R6_BANK1	R6_BANK1	R6
R7_BANK1	R7_BANK1	R7
R8	R8	R8
R9	R9	R9
R10	R10	R10
R11	R11	R11
R12	R12	R12
R13	R13	R13
R14	R14	R14
R15	R15	R15

図 2.3 汎用レジスタ

### 【プログラミング上の注意】

ユーザの R0~R7 は R0\_BANK0~R7\_BANK0 に、例外・割り込み後の R0~R7 は R0\_BANK1~R7\_BANK1 に割り当てられるので、割り込みハンドラはユーザの R0~R7 (R0\_BANK0~R7\_BANK0) を退避または復帰する必要はありません。

リセット後の R0\_BANK0~R7\_BANK0、R0\_BANK1~R7\_BANK1、R8~R15 の値は不定です。

### 2.2.3 浮動小数点レジスタ

図 2.4 に浮動小数点レジスタを示します。32 本の 32 ビット浮動小数点レジスタがあります。これらは、2 つのバンクで構成され、FPR0\_BANK0 ~ FPR15\_BANK0、FPR0\_BANK1 ~ FPR15\_BANK1 があります。また、この 32 本レジスタは FR0 ~ FR15, DR0/2/4/6/8/10/12/14、FV0/4/8/12, XF0 ~ XF15, XD0/2/4/6/8/10/12/14, XMTRX として参照されます。FPRn\_BANKi と参照名の対応は FPSCR の FR ビットによって決まります。図 2.4 を参照してください。

(1) 浮動小数点レジスタ FPRn\_BANKi (32 レジスタ)

FPR0\_BANK0, FPR1\_BANK0, FPR2\_BANK0, FPR3\_BANK0,  
FPR4\_BANK0, FPR5\_BANK0, FPR6\_BANK0, FPR7\_BANK0,  
FPR8\_BANK0, FPR9\_BANK0, FPR10\_BANK0, FPR11\_BANK0,  
FPR12\_BANK0, FPR13\_BANK0, FPR14\_BANK0, FPR15\_BANK0  
FPR0\_BANK1, FPR1\_BANK1, FPR2\_BANK1, FPR3\_BANK1,  
FPR4\_BANK1, FPR5\_BANK1, FPR6\_BANK1, FPR7\_BANK1,  
FPR8\_BANK1, FPR9\_BANK1, FPR10\_BANK1, FPR11\_BANK1,  
FPR12\_BANK1, FPR13\_BANK1, FPR14\_BANK1, FPR15\_BANK1

(2) 単精度浮動小数点レジスタ FRi (16 レジスタ)

FPSCR.FR = 0 のとき、FR0 ~ FR15 は FPR0\_BANK0 ~ FPR15\_BANK0 に割り当てられます。  
FPSCR.FR = 1 のとき、FR0 ~ FR15 は FPR0\_BANK1 ~ FPR15\_BANK1 に割り当てられます。

(3) 倍精度浮動小数点レジスタ、または単精度浮動小数点レジスタのペア DRi (8 レジスタ)

DR レジスタは、2 つの FR レジスタから構成されます。  
DR0 = {FR0, FR1}, DR2 = {FR2, FR3},  
DR4 = {FR4, FR5}, DR6 = {FR6, FR7},  
DR8 = {FR8, FR9}, DR10 = {FR10, FR11},  
DR12 = {FR12, FR13}, DR14 = {FR14, FR15}

(4) 単精度浮動小数点ベクトルレジスタ FVi (4 レジスタ)

FV レジスタは 4 つの FR レジスタから構成されます。  
FV0 = {FR0, FR1, FR2, FR3},  
FV4 = {FR4, FR5, FR6, FR7},  
FV8 = {FR8, FR9, FR10, FR11},  
FV12 = {FR12, FR13, FR14, FR15}

(5) 単精度浮動小数点拡張レジスタ XFi (16 レジスタ)

FPSCR.FR = 0 のとき、XF0 ~ XF15 は FPR0\_BANK1 ~ FPR15\_BANK1 に割り当てられます。  
FPSCR.FR = 1 のとき、XF0 ~ XF15 は FPR0\_BANK0 ~ FPR15\_BANK0 に割り当てられます。

(6) 単精度浮動小数点拡張レジスタのペア XD<sub>i</sub> (8 レジスタ)

XD レジスタは 2 つの XF レジスタから構成されます。  
XD0 = {XF0, XF1}, XD2 = {XF2, XF3},  
XD4 = {XF4, XF5}, XD6 = {XF6, XF7},  
XD8 = {XF8, XF9}, XD10 = {XF10, XF11},  
XD12 = {XF12, XF13}, XD14 = {XF14, XF15}

## (7) 単精度浮動小数点拡張レジスタ行列 XMTRX

XMTRX は 16 本の XF レジスタから構成されます。

$$\text{XMTRX} = \begin{bmatrix} \text{XF0} & \text{XF4} & \text{XF8} & \text{XF12} \\ \text{XF1} & \text{XF5} & \text{XF9} & \text{XF13} \\ \text{XF2} & \text{XF6} & \text{XF10} & \text{XF14} \\ \text{XF3} & \text{XF7} & \text{XF11} & \text{XF15} \end{bmatrix}$$

FPSCR.FR=0				FPSCR.FR=1			
FV0	DR0	FR0	FPR0_BANK0	XF0	XD0	XMTRX	
		FR1	FPR1_BANK0	XF1			
	DR2	FR2	FPR2_BANK0	XF2	XD2		
		FR3	FPR3_BANK0	XF3			
FV4	DR4	FR4	FPR4_BANK0	XF4	XD4		
		FR5	FPR5_BANK0	XF5			
	DR6	FR6	FPR6_BANK0	XF6	XD6		
		FR7	FPR7_BANK0	XF7			
FV8	DR8	FR8	FPR8_BANK0	XF8	XD8		
		FR9	FPR9_BANK0	XF9			
	DR10	FR10	FPR10_BANK0	XF10	XD10		
		FR11	FPR11_BANK0	XF11			
FV12	DR12	FR12	FPR12_BANK0	XF12	XD12		
		FR13	FPR13_BANK0	XF13			
	DR14	FR14	FPR14_BANK0	XF14	XD14		
		FR15	FPR15_BANK0	XF15			
XMTRX	XD0	XF0	FPR0_BANK1	FR0	DR0	FV0	
		XF1	FPR1_BANK1	FR1			
	XD2	XF2	FPR2_BANK1	FR2	DR2		
		XF3	FPR3_BANK1	FR3			
	XD4	XF4	FPR4_BANK1	FR4	DR4	FV4	
		XF5	FPR5_BANK1	FR5			
	XD6	XF6	FPR6_BANK1	FR6	DR6		
		XF7	FPR7_BANK1	FR7			
	XD8	XF8	FPR8_BANK1	FR8	DR8	FV8	
		XF9	FPR9_BANK1	FR9			
	XD10	XF10	FPR10_BANK1	FR10	DR10		
		XF11	FPR11_BANK1	FR11			
	XD12	XF12	FPR12_BANK1	FR12	DR12	FV12	
		XF13	FPR13_BANK1	FR13			
	XD14	XF14	FPR14_BANK1	FR14	DR14		
		XF15	FPR15_BANK1	FR15			

図 2.4 浮動小数点レジスタ

## 【プログラミング上の注意】

リセット後の FPR0\_BANK0 ~ FPR15\_BANK0、FPR0\_BANK1 ~ FPR15\_BANK1 の値は不定です。

### 2.2.4 コントロールレジスタ

- (1) ステータスレジスタ SR  
(32 ビット、特権保護、初期値 = 0111 0000 0000 0000 0000 00XX 1111 00XX)

31	30	29	28	27	16	15	14	10	9	8	7	4	3	2	1	0
—	MD	RB	BL	—	FD	—	—	M	Q	IMASK	—	—	—	S	T	—

【注】 —: 予約ビット。読み出すと常に0が読み出されます。書き込む値も常に0にしてください。  
X: 不定

- MD: プロセッサモード
  - MD=0: ユーザモード（命令の中には実行できない命令があり、リソースの中にはアクセスできないリソースがあります。）
  - MD=1: 特権モード
- RB: 特権モードでの汎用レジスタバンク指定子（リセット、例外または割り込みにより 1 にセットされます。）
  - RB=0: R0\_BANK0 ~ R7\_BANK0は、汎用レジスタR0 ~ R7としてアクセスされます（R0\_BANK1 ~ R7\_BANK1はLDC/STC R0\_BANK ~ R7\_BANK命令を使用することによってアクセスできます）。
  - RB=1: R0\_BANK1 ~ R7\_BANK1は、汎用レジスタR0 ~ R7としてアクセスされます（R0\_BANK0 ~ R7\_BANK0はLDC/STC R0\_BANK ~ R7\_BANK命令を使用することによってアクセスできます）。
- BL: 例外 / 割り込みブロックビット（リセット、例外または割り込みにより 1 にセットされます。）
  - BL=1: 割り込み要求はマスクされます。（BL=1）のときユーザブレーク以外の一般例外が発生すると、プロセッサは、リセット状態に遷移します。
- FD: FPU ディセーブルビット（リセットにより 0 にクリアされます。）
  - FD=1: FPU命令は一般FPU抑止例外を発生させ、FPU命令が遅延スロットにある場合、スロット FPU抑止例外が発生します。（FPU命令: H'F\*\*\*命令、FPUL/FPSCRに対するLDS(.L)/STS(.L)命令）
- M、Q: DIV0S、DIV0U、DIV1 命令が使用
- IMASK: 割り込みマスクレベル  
IMASKより低いレベルの外部割り込みはマスクされます。
- S: MAC 命令の飽和動作を指定します。
- T: 真 / 偽条件、またはキャリ / ボロービット

- (2) 回避ステータスレジスタ SSR（32 ビット、特権保護、初期値=不定）  
SR の内容は例外または割り込みの発生時、SSR に回避されます。

- (3) 退避プログラムカウンタ SPC (32 ビット、特権保護、初期値=不定)

割り込みの発生した命令のアドレスは SPC に退避されます。

- (4) グローバルベースレジスタ GBR (32 ビット、初期値=不定)

GBR は GBR 参照 MOV 命令のベースアドレスとして参照されます。

- (5) ベクタベースレジスタ VBR (32 ビット、特権保護、初期値=H'0000 0000)

VBR は例外および割り込み発生時、分岐先のベースアドレスとして参照されます。詳細については「第 5 章 例外処理」を参照してください。

- (6) 退避ジェネラルレジスタ 15 SGR (32 ビット、特権保護、初期値=不定)

R15 の内容は例外または割り込みの発生時 SGR に退避されます。

- (7) デバッグベースレジスタ DBR (32 ビット、特権保護、初期値=不定)

ユーザブレイクデバッグ機能を有効にする場合 (BRCR.UBDE=1)、DBR は VBR の代わりにユーザブレイクハンドラへの分岐先アドレスとして参照されます。

## 2.2.5 システムレジスタ

- (1) 積和上位レジスタ MACH (32 ビット、初期値=不定)、  
積和下位レジスタ MACL (32 ビット、初期値=不定)

MACH / MACL は、MAC 命令の加算値として用いられます。また MAC 命令、MUL の演算結果を格納するためにも用いられます。

- (2) プロシジャレジスタ PR (32 ビット、初期値=不定)

BSR、BSRF、JSR 命令を用いたサブルーチンコールの戻りアドレスは PR に格納されます。PR は、サブルーチンからの復帰命令 (RTS) によって参照されます。

- (3) プログラムカウンタ PC (32 ビット、初期値=H'A000 0000)

PC は命令フェッチアドレスを示します。

- (4) 浮動小数点ステータス / コントロールレジスタ FPSCR  
(32 ビット、初期値=H'0004 0001)

31	22	21	20	19	18	17	12	11	7	6	2	1	0
—				FR	SZ	PR	DN	Cause		Enable		Flag	RM

【注】 —: 予約ビット。読み出すと常に0が読み出されます。書き込む値も常に0にしてください。

- FR : 浮動小数点レジスタバンク

- FR=0 :

FPR0\_BANK0 ~ FPR15\_BANK0はFR0 ~ FR15に、FPR0\_BANK1 ~ FPR15\_BANK1はXF0 ~ XF15に割り当てられます。

- FR=1 :

FPR0\_BANK0 ~ FPR15\_BANK0はXF0 ~ XF15に、FPR0\_BANK1 ~ FPR15\_BANK1 はFR0 ~ FR15に割り当てられます。

## 2. プログラミングモデル

- SZ：転送サイズモード
  - SZ=0：
   
FMOV命令のデータサイズは32ビットです。
  - SZ=1：
   
FMOV命令のデータサイズは32ビットペア（64ビット）です。
- PR：精度モード
  - PR=0：
   
浮動小数点命令を単精度で実行します。
  - PR=1：
   
浮動小数点命令を倍精度で実行します（倍精度がサポートされていない命令の結果は未定義です。）

SZ と PR は同時に 1 にセットしないでください。この設定は予約されています。

[SZ, PR]=11：予約（FPU 命令演算は未定義です。）

- DN：非正規化モード
  - DN=0：非正規化数を非正規化数として扱います。
  - DN=1：非正規化数を 0 として扱います。

		FPU エラー (E)	無効演算 (V)	0 除算 (Z)	オーバ フロー(O)	アンダ フロー (U)	不正確 (I)
Cause	FPU 例外要因 フィールド	ビット 17	ビット 16	ビット 15	ビット 14	ビット 13	ビット 12
Enable	FPU 例外イネーブル フィールド	なし	ビット 11	ビット 10	ビット 9	ビット 8	ビット 7
Flag	FPU 例外フラグフィ ールド	なし	ビット 6	ビット 5	ビット 4	ビット 3	ビット 2

FPU 演算命令を実行すると、要因フィールドは最初に 0 に設定されます。次に FPU 例外が要求されると、要因フィールドとフラグフィールドの該当ビットが 1 にセットされます。

フラグフィールドは、フラグフィールドが最後にクリアされたそれ以降に発生した例外のステータスを保持します。

- RM：丸めモード
  - RM=00：近傍への丸め
  - RM=01：0 方向への丸め
  - RM=10：予約
  - RM=11：予約
- ビット 22～31：予約

(5) 浮動小数点通信レジスタ FPUL（32 ビット、初期値=不定）

FPU レジスタと CPU レジスタ間のデータ転送は、FPUL を介して行われます。

### 【プログラミング上の注意】

SZ=1 かつビッグエンディアン方式の場合、FMOV は倍精度浮動小数点ロードまたはストアとして使用できます。リトルエンディアン方式の場合、倍精度浮動小数点をロードまたはストアするためには、SZ=0 でデータサイズ 32 ビットを 2 度実行する必要があります。



## 2.3 メモリ割り付けレジスタ

付録 A にメモリに割り付けた制御レジスタを示します。制御レジスタは次のメモリ領域にダブルマッピングされています。すべてのレジスタには 2 つのアドレスがあります。

H'1F00 0000 ~ H'1FFF FFFF  
H'FF00 0000 ~ H'FFFF FFFF

以上 2 つの領域は次のように使用します。

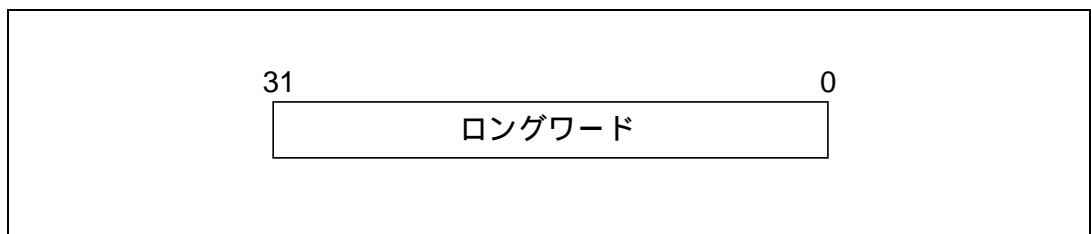
- H'1F00 0000 ~ H'1FFF FFFF  
この領域は TLB を用いたアドレス変換モード下でアクセスしなければなりません。外部メモリは、SH-4 アーキテクチャでは 29 ビットアドレス空間として定義されているので、TLB の物理ページ番号は 32 ビットアドレス空間をカバーしません。アドレス変換時、メモリ割り付けレジスタにアクセスして、この領域のページ番号を TLB の該当フィールドに設定することができます。この領域のページ番号を TLB に設定した実際のページ番号として使用してください。アドレス変換を行わない場合、この領域に対するアクセスの動作は不定です。
- H'FF00 0000 ~ H'FFFF FFFF  
この領域はアドレス変換なしにアクセスしなければなりません。  
2 つの領域のレジスタが割り付けられていないアドレスにはアクセスしないでください。レジスタが割り付けられていないアドレスに対するアクセスの動作は不定になります。また、メモリ割り付けレジスタは一定のデータサイズでアクセスしなければなりません。不正なサイズでアクセスした場合も動作は不定になります。

### 【プログラミング上の注意】

ユーザモードで領域 H'FF00 0000 ~ H'FFFF FFFF にアクセスすると、アドレスエラーが発生します。ユーザモードではメモリ割り付けレジスタはアドレス変換によるアクセスで参照することができます。

## 2.4 レジスタのデータ形式

レジスタオペランドのデータサイズは常にロングワード (32 ビット) です。メモリ上のデータをレジスタへロードするとき、メモリオペランドのデータサイズがバイト (8 ビット)、もしくはワード (16 ビット) の場合は、ロングワードに符号拡張し、レジスタに格納します。



## 2.5 メモリ上でのデータ形式

バイト、ワード、ロングワードのデータ形式があります。メモリは8ビットのバイト、16ビットのワード、32ビットのロングワードいずれの形でもアクセスすることができます。32ビットに満たないメモリオペランドは符号拡張されてレジスタに格納されます。

ワードオペランドはワード境界（2バイト刻みの偶数番地：2n 番地）から、ロングワードオペランドはロングワード境界（4バイト刻みの偶数番地：4n 番地）からアクセスしてください。これを守らない場合は、アドレスエラーになります。バイトオペランドはどの番地からでもアクセスできます。

データフォーマットは、ビックエンディアンかリトルエンディアンのどちらかのバイト順を選択できます。エンディアンはパワーオンリセット時に外部ピン（MD5 端子）で設定してください。MD5 端子がローレベルの場合ビックエンディアンに、MD5 端子がハイレベルの場合リトルエンディアンに設定されます。エンディアンは動的には変更できません。ただしビット位置は常に最上位（most-significant）から最下位（least-significant）へ左から右へ減少するように番号が付けられています。すなわち 32 ビットのロングワードでは、一番左のビット、ビット 31 が最上位ビットで、一番右のビット、ビット 0 が最下位ビットです。

メモリ上のデータ形式を図 2.5 に示します。リトルエンディアンモードのときは、バイト長（8 ビット）で書き込んだデータはバイト長で読み出してください。ワード長（16 ビット）で書き込んだデータはワード長で読み出してください。

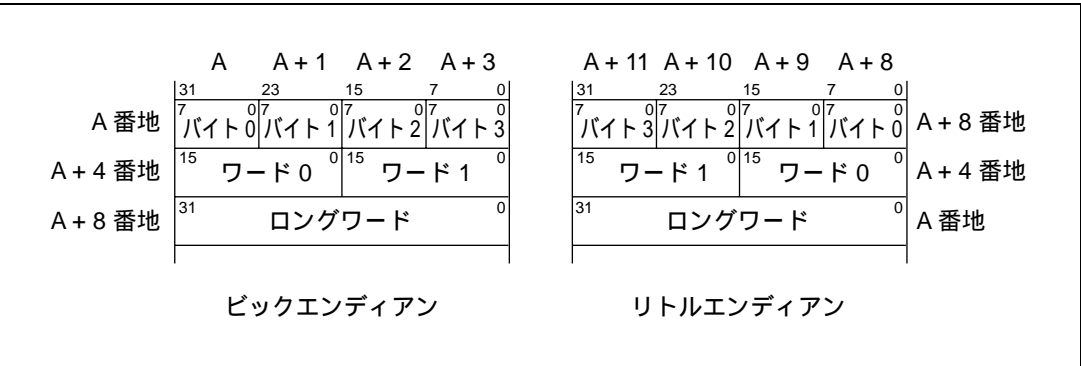


図 2.5 メモリ上のデータ形式

【注】 SH7091 では、64 ビット長データフォーマットのエンディアン変換をサポートしていません。そのため、リトルエンディアンモード下で倍精度浮動小数点フォーマット（64 ビット長）のアクセスをした場合、上位 32 ビットと下位 32 ビットが逆になります。

## 2.6 処理状態

処理状態にはリセット状態、例外処理状態、バス権解放状態、プログラム実行状態、低消費電力状態の5種類があります。

### (1) リセット状態

CPU がリセットされている状態です。 $\overline{\text{RESET}}$  端子がローレベルになるとリセット状態になります。 $\overline{\text{MRESET}}$  端子がハイレベルのときパワーオンリセット状態になり、 $\overline{\text{MRESET}}$  端子がローレベルのとき、マニュアルリセット状態になります。リセットについては、「第5章 例外処理」を参照してください。

パワーオンリセット状態では、CPU の内部状態と内蔵周辺モジュールのレジスタが初期化されます。マニュアルリセット状態では、バスステートコントローラ (BSC) を除く内蔵周辺モジュールのレジスタと CPU の内部状態とが初期化されます。マニュアルリセット状態では、BSC は初期化されませんのでリフレッシュ動作は継続しています。詳細は、ハードウェアマニュアルの各章のレジスタ構成を参照してください。

### (2) 例外処理状態

リセット、一般例外、割り込みの例外処理要因によって、CPU が処理状態の流れを変えるときの一時的な状態です。

リセットの場合は、H'A000 0000 に分岐してユーザが作成した例外処理プログラムの実行を開始します。

一般例外、割り込みの場合は、プログラムカウンタ (PC) を退避プログラムカウンタ (SPC) に、ステータスレジスタ (SR) を退避ステータスレジスタ (SSR)、R15 を退避ジェネラルレジスタ 15 (SGR) に退避します。ベクタベースアドレスの内容とベクタオフセットの和で求められたユーザ作成の例外サービスルーチンの開始アドレスに分岐して、プログラムの実行を開始します。リセット、一般例外、割り込みについては、「第5章 例外処理」を参照してください。

### (3) プログラム実行状態

CPU が順次プログラムを実行している状態です。

### (4) 低消費電力状態

CPU の動作が停止し消費電力が低い状態です。スリープ命令で低消費電力状態になります。スリープモードとスタンバイモードの2つのモードがあります。低消費電力状態の詳細は「第9章 低消費電力モード」を参照してください。

### (5) バス権解放状態

CPU がバス権を要求したデバイスにバスを解放している状態です。

状態間の遷移を図 2.6 に示します。

## 2. プログラミングモデル

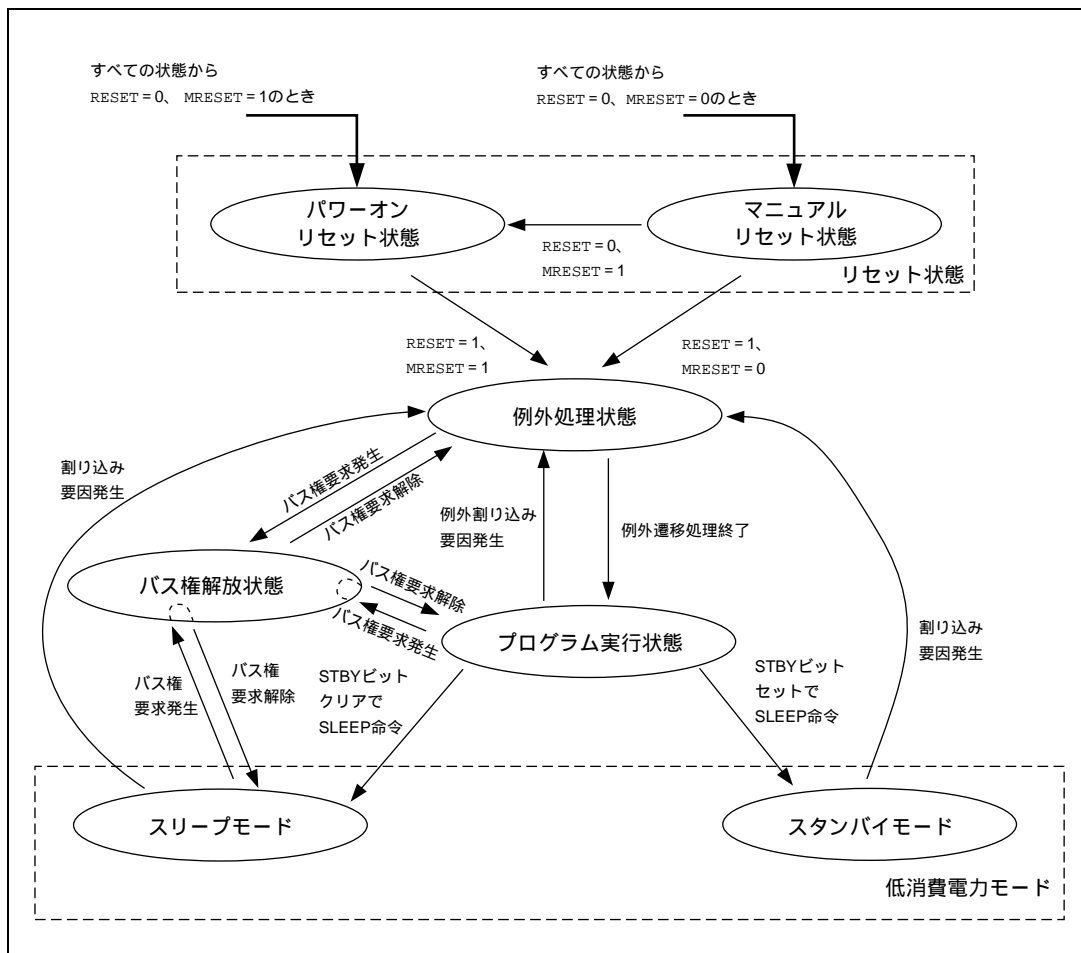


図 2.6 処理状態の状態遷移図

## 2.7 処理モード

処理モードには特権モードとユーザモードの2種類があります。ステータスレジスタ (SR) の処理モードビット (MD) で処理モードが決まります。MD ビットが0のときユーザモードになり、1のとき特権モードになります。リセット状態、例外処理状態になると、MD ビットが1になります。例外処理が終了したときは、MD ビットを0にクリアしてユーザモードに切り換えます。特権モードでのみアクセスできるレジスタとビットがあります。

---

## 3. メモリマネジメントユニット (MMU)

---

### 3.1 概要

#### 3.1.1 特長

SH7091 は 8 ビットのアドレス空間識別子と 32 ビットの仮想アドレス空間から 29 ビットの外部メモリ空間を扱うことができます。仮想アドレスから物理アドレスへのアドレス変換は SH7091 に内蔵されたメモリマネジメントユニット (MMU:Memory Management Unit) を用いて行います。MMU は変換ルックアサイドバッファ (TLB:Translation Lookaside Buffer) にユーザ作成のアドレス変換テーブルの情報をキャッシングすることにより、高速にアドレス変換を行います。SH7091 は命令 TLB (ITLB) を 4 エントリ、共用 TLB (UTLB) を 64 エントリ内蔵しており ITLB には UTLB のコピーがハードウェアにより格納されます。アドレス変換方式はページング方式で、4 種類 (1k/4k/64k/1M バイト) のページサイズをサポートしています。また特権モード、ユーザモードのそれぞれにおいて、仮想アドレス空間へのアクセス権を設定し、記憶保護を行うことができます。

#### 3.1.2 MMU の役割

MMU とは物理メモリを有効に利用するために考え出された機能です。図 3.1 に示すように、プロセスのサイズが物理メモリより少ない場合、プロセスの全てを物理メモリへマッピングすることが可能です。しかしプロセスのサイズが増大し、物理メモリに収まらない場合、プロセスを分割して実行に必要な部分を随時物理メモリへマッピングする必要性が生じます (1)。この物理メモリへのマッピングをプロセス自身が考えながら実行している、プロセスにかかる負担が増大します。この負担を軽減するために物理メモリへのマッピングを一括して行おうとして生まれた考え方が仮想記憶方式です (2)。仮想記憶方式では物理メモリに比べて十分に大きな仮想メモリを用意します。プロセスはこの仮想メモリにマッピングされます。このためプロセスは仮想メモリ上での動作だけを考えていれば良くなります。仮想メモリから物理メモリへのマッピングには、MMU が用いられます。MMU は通常 OS が管理しており、プロセスが必要とする仮想メモリを円滑に物理メモリへマッピングできるように物理メモリの入れ換えを行います。物理メモリの入れ換えは 2 次記憶などとの間で行われます。

こうして生まれた仮想記憶方式は複数のプロセスが同時に走行するタイムシェアリングシステム (TSS) の上で威力を発揮します (3)。TSS 上で走行する複数のプロセスが、おのあの物理メモリへのマッピングを意識しながら動作していたのでは効率が上がりません。この効率を上げ、各プロセスの負担を減らすために仮想記憶方式は使われます (4)。この仮想記憶方式ではプロセスごとに仮想メモリが割り当てられます。MMU は複数の仮想メモリを効率よく物理メモリへマッピングする働きをします。さらにあるプロセスが別のプロセスの物理メモリに誤ってアクセスしないように、MMU には記憶保護の機能も備わっています。

MMU を用いて仮想メモリから物理メモリへアドレス変換を行うとき、その変換情報が MMU に登録されていなかったり、別のプロセスの仮想メモリへ誤ってアクセスすることがあります。そのとき MMU は例外を発生させ、物理メモリのマッピングを変更し、新たなアドレス変換情報を登録します。

### 3. メモリマネジメントユニット (MMU)

MMU の機能はソフトウェアのみでも実現可能ですが、プロセスが物理メモリへアクセスするたびにソフトウェアで変換を行っていたのでは効率が悪くなります。そのためハードウェア上にアドレス変換のためのバッファ (TLB) を用意し、頻繁に使用されるアドレス変換情報は TLB に置いておきます。TLB はアドレス変換情報のためのキャッシュといえます。しかしキャッシュと違いアドレス変換に失敗したとき、つまり例外が発生したときの、アドレス変換情報の入れ換えは通常ソフトウェアで行います。このためソフトウェアで柔軟にメモリ管理を行うことが可能となります。

MMU が仮想メモリから物理メモリへのマッピングをする方式として、固定長のアドレス変換を用いる方式 (ページング方式) と可変長のアドレス変換を用いる方式 (セグメント方式) があります。ページング方式では固定サイズのページと呼ばれるアドレス空間 (通常 1k ~ 64k バイト) が変換の単位となります。

以下 SH7091 では仮想メモリ上のアドレス空間のことを仮想アドレス空間、物理メモリ上のアドレス空間のことを物理アドレス空間と呼ぶことにします。

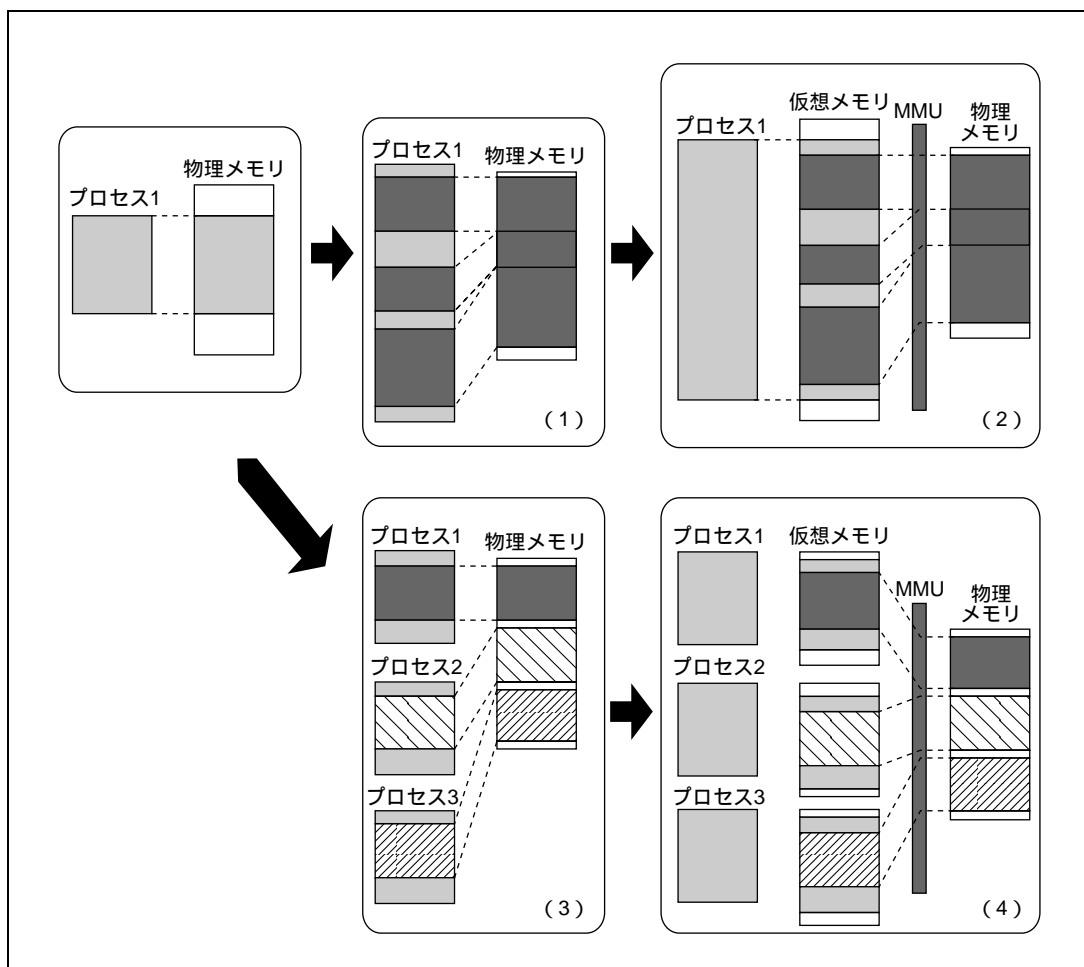


図 3.1 MMU の役割

### 3.1.3 レジスタの構成

MMU レジスタの構成を表 3.1 に示します。

表 3.1 レジスタ構成

名称	略称	R/W	初期値 <sup>*1</sup>	P4 アドレス <sup>*2</sup>	エリア 7 アドレス <sup>*2</sup>	アクセス サイズ
ページテーブルエントリ 上位レジスタ	PTEH	R/W	不定	H'FF00 0000	H'1F00 0000	32
ページテーブルエントリ 下位レジスタ	PTL	R/W	不定	H'FF00 0004	H'1F00 0004	32
ページテーブルエントリ アシスタンスレジスタ	PTEA	R/W	不定	H'FF00 0034	H'1F00 0034	32
変換テーブルベースレジスタ	TTB	R/W	不定	H'FF00 0008	H'1F00 0008	32
TLB 例外アドレスレジスタ	TEA	R/W	不定	H'FF00 000C	H'1F00 000C	32
MMU 制御レジスタ	MMUCR	R/W	H'0000 0000	H'FF00 0010	H'1F00 0010	32

【注】 \*1 初期値とはパワーオンリセット、マニュアルリセット後の値を示します。

\*2 P4 アドレスは仮想 / 物理アドレス空間の P4 領域を用いた場合のものです。TLB を用いて物理アドレス空間のエリア 7 からアクセスする場合、アドレスの上位 3 ビットが無視されます。

### 3.1.4 注意事項

本マニュアル中で予約領域とは、アクセスした場合に動作を保証しない領域を示します。

## 3.2 レジスタの説明

MMU に関連するレジスタは 6 つあります。

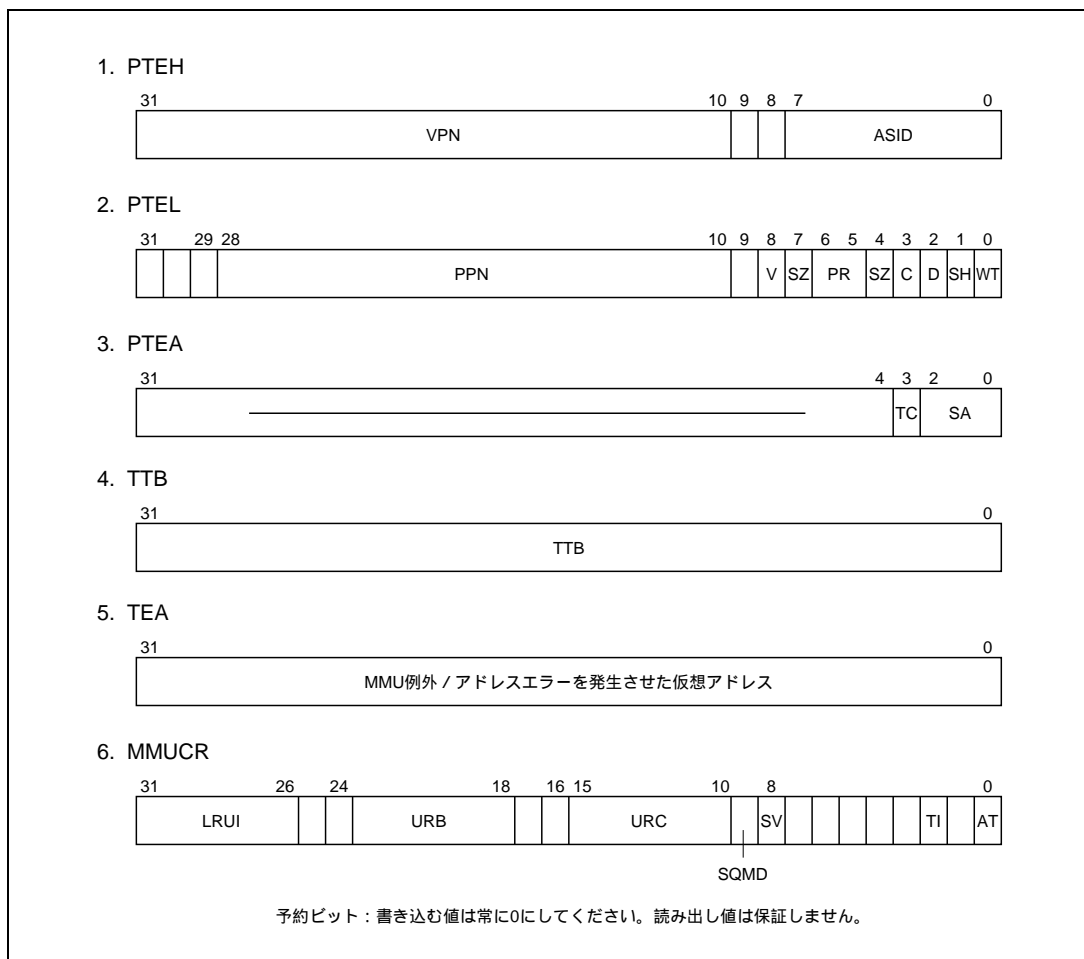


図 3.2 MMU 関連レジスタ

#### (1) ページエントリ上位レジスタ (PTEH)

PTEH は、P4 領域の H'FF00 0000 からとエリア 7 の H'1F00 0000 からロングワードサイズでアクセスすることが可能です。PTEH は仮想ページ番号 (VPN) とアドレス空間識別子 (ASID) から構成されています。VPN は MMU 例外またはアドレスエラー例外が発生した際に、ハードウェアにより例外を発生させた仮想アドレスの VPN が設定されます。VPN はページサイズにより異なりますが、例外発生時にハードウェアにより設定される VPN は例外を発生させた仮想アドレスの上位 22 ビットとなります。VPN の設定はソフトウェアにより行うことも可能です。ASID には現在実行中のプロセスの番号をソフトウェアにより設定します。ASID がハードウェアにより更新されることはありません。この VPN と ASID が LDTLB 命令により UTLB に登録されます。



## (2) ページエントリ下位レジスタ (PTEL)

PTEL へは、P4 領域の H'FF00 0004 からとエリア 7 の H'1F00 0004 からロングワードサイズでアクセスすることが可能です。PTEL は LDTLB 命令により UTLB へ登録する物理ページ番号とページ管理情報を格納するために使用されます。本レジスタはソフトウェアの指示がない限り内容が変更されることはありません。

## (3) ページテーブルエントリアシスタンスレジスタ (PTEA)

PTEA へは、P4 領域の H'FF00 0034 からとエリア 7 の H'1F00 0034 からロングワードサイズでアクセスすることが可能です。PTEA は LDTLB 命令により UTLB への PCMCIA のアクセスのためのアシスタントビットを格納するために使用されます。本レジスタはソフトウェアの指示がない限り内容が変更されることはありません。

## (4) 変換テーブルベースレジスタ (TTB)

TTB へは P4 領域の H'FF00 0008 からとエリア 7 の H'1F00 0008 からロングワードサイズでアクセスすることが可能です。このレジスタは、例えば現在使用しているページテーブルのベースアドレスの格納用に使用します。TTB はソフトウェアの指示がない限り内容が変更されることはありません。本レジスタはソフトウェアで自由に使用可能です。

## (5) TLB 例外アドレスレジスタ (TEA)

TEA へは P4 領域の H'FF00 000C からとエリア 7 の H'1F00 000C からロングワードサイズでアクセスすることが可能です。MMU 例外またはアドレスエラー例外発生後に、このレジスタへは例外を発生させた仮想アドレスがハードウェアにより設定されます。このレジスタはソフトウェアにより変更することは可能です。

## (6) MMU 制御レジスタ (MMUCR)

MMUCR には以下のビットがあります。

- LRUI:Least Recently Used ITLB
- URB:UTLB Replace Boundary
- URC:UTLB Replace Counter
- SQMD:Store Queue Mode Bit
- SV:Single Virtual Mode Bit
- TI:TLB Invalidate
- AT:Address Translation Bit

MMUCR へは P4 領域の H'FF00 0010 からとエリア 7 の H'1F00 0010 からロングワードサイズでアクセスすることが可能です。MMUCR の各ビットは以下に示すように、MMU の設定を行います。このため MMUCR の書き換えは P1、P2 領域のプログラムで行うようにしてください。MMUCR はソフトウェアにより変更可能です。ただし LRUI ビットと URC ビットはハードウェアにより更新されることもあります。

- LRUI:  
ITLBミス発生時に入れ換えるITLBのエントリを決めるため、LRU方式(Least Recently Used)を用いています。LRUIビットを用いて、ITLBの追い出すエントリを確定することができます。LRUIは以下のアルゴリズムで更新が行われます。この表で "—" は更新を行わないことを意味します。

### 3. メモリマネジメントユニット (MMU)

	LRUI					
	[5]	[4]	[3]	[2]	[1]	[0]
ITLB のエントリ 0 を用いたとき	0	0	0	—	—	—
ITLB のエントリ 1 を用いたとき	1	—	—	0	0	—
ITLB のエントリ 2 を用いたとき	—	1	—	1	—	0
ITLB のエントリ 3 を用いたとき	—	—	1	—	1	1
上記以外	—	—	—	—	—	—

またLRUIが以下の状態のとき、対応するITLBのエントリがITLBミスにより更新されます。この表で “\*” はdon't careを意味します。

	LRUI					
	[5]	[4]	[3]	[2]	[1]	[0]
ITLB のエントリ 0 が更新される	1	1	1	*	*	*
ITLB のエントリ 1 が更新される	0	*	*	1	1	*
ITLB のエントリ 2 が更新される	*	0	*	0	*	1
ITLB のエントリ 3 が更新される	*	*	0	*	0	0
上記以外	設定禁止					

上記の表で設定禁止の値にはソフトウェアの責任で設定しないようにしてください。パワーオン、マニュアルリセット後、LRUIは0に初期化されますので、ハードウェアの更新によりLRUIが上記の表の設定禁止の値になることはありません。

- URB：入れ換えを行う UTLB エントリの境界を示すビット  
URB>0のときに有効となります。
- URC：LDTLB 命令により入れ換えを行う UTLB エントリを示すためのランダムカウンタ  
UTLBへのアクセスが発生する度にインクリメントされます。ただしURB>0の場合、URC = URBの条件が成立するとURCは0にクリアされます。またソフトウェアによりURC>URBとなる値がURCに書き込まれた場合、最初はURC = H'3FになるまでURBを超えてインクリメントが行われますので注意してください。URCはLDTLB命令によってカウントアップされません。
- SQMD:ストアキューモードビット  
ストアキューへのアクセス権を指定します。
  - 0: ユーザ / 特権アクセスが可能
  - 1: 特権アクセスが可能 (ユーザアクセスの場合はアドレスエラー例外)

- SV:単一仮想記憶モード / 多重仮想記憶モードの切り換えビット
  - 0: 多重仮想記憶モード
  - 1: 単一仮想記憶モードこのビットを変更するときは、必ずTIビットにも1を書き込んでください。
- TI:  
このビットに1を書き込むと、UTLB/ITLBの有効ビットを全て無効化(0にクリア)します。読み出しは常に0です。
- AT:  
MMUのイネーブル(有効)とディスエーブル(無効)を指定します。
  - 0: MMU ディスエーブル
  - 1: MMU イネーブルATビットが0の状態ではMMU例外は発生しません。このためMMUを使用しないソフトウェアではATビットを0の状態で使用してください。

### 3.3 メモリ空間

#### 3.3.1 物理メモリ空間

SH7091は32ビットの物理メモリ空間をサポートし、4Gバイトのアドレス空間をアクセスできます。MMUCR.ATビットを0にし、MMUをディスエーブル状態にしたときのアドレス空間がこの物理メモリ空間です。物理メモリ空間は図3.3に示す通り、いくつかの領域に分かれています。物理メモリ空間は固定的に29ビットの外部メモリ空間へマッピングされ、その対応は物理メモリ空間のアドレスの上位3ビットを無視することで行えます。特権モードではP0領域からP4領域の4Gバイトの空間をアクセスすることが可能です。ユーザモードではU0領域の2Gバイトの空間をアクセスすることが可能です。ユーザモードでP1~P4領域(ストアキュー領域を除く)をアクセスした場合、アドレスエラーとなります。

### 3. メモリマネジメントユニット (MMU)

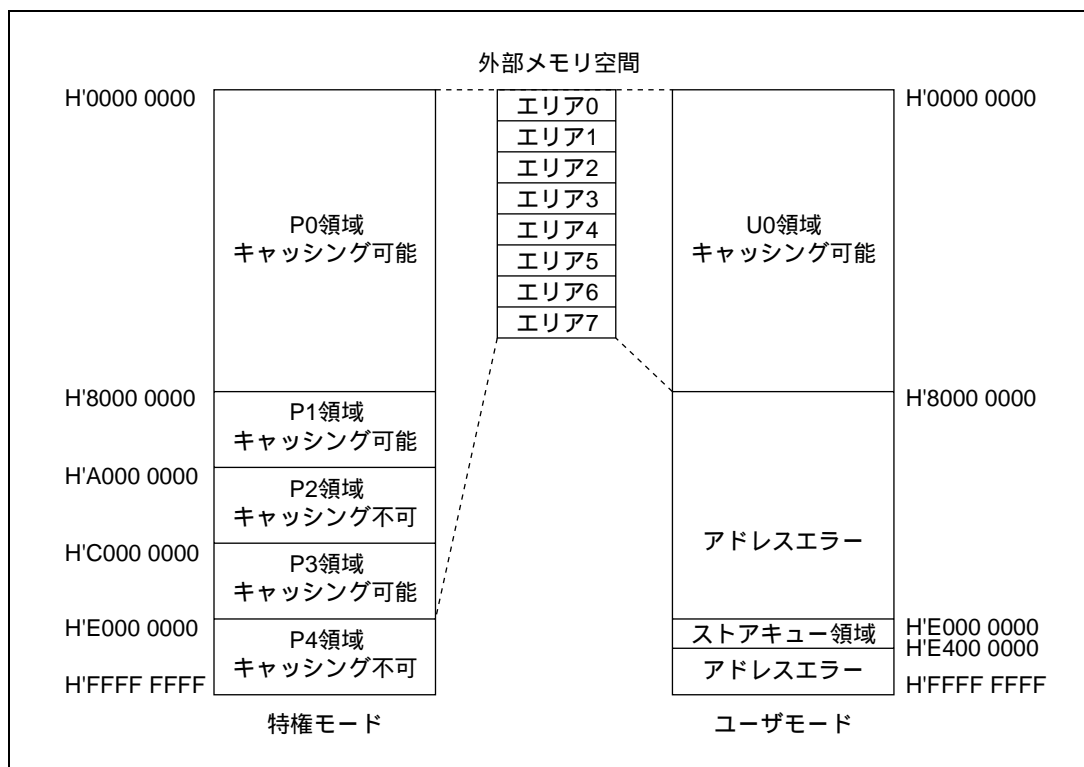


図 3.3 物理メモリ空間 (MMUCR.AT = 0)

#### (1) P0、P1、P3、U0 領域

P0、P1、P3、U0 領域はキャッシュを用いたアクセスが可能な領域です。キャッシュを用いるか、用いないかはキャッシュコントロールレジスタ (CCR) に従います。キャッシュを用いた場合、ライトアクセスにおけるコピーバック方式とライトスルー方式の切り換えは、P1 領域を除いて CCR.WT ビットの指定に従います。P1 領域の切り換えは、CCR.CB ビットの指定に従います。これらの領域のアドレスの上位 3 ビットを 0 にしたものが対応する外部メモリ空間のアドレスとなります。ただし外部メモリ空間のエリア 7 は予約領域ですので、これらの領域にも予約領域が現われることになります。

#### (2) P2 領域

P2 領域はキャッシュを用いたアクセスが行えない領域です。P2 領域ではアドレスの上位 3 ビットを 0 にしたものが対応する外部メモリ空間のアドレスとなります。ただし外部メモリ空間のエリア 7 は予約領域ですので、この領域にも予約領域が現われることになります。

#### (3) P4 領域

P4 領域は SH7091 の内蔵 I/O にマッピングされる領域です。この領域はキャッシュを用いたアクセスができません。P4 領域の詳細を図 3.4 に示します。

H'E000 0000	ストアキュー
H'E400 0000	予約領域
H'F000 0000	命令キャッシュ アドレスアレイ
H'F100 0000	命令キャッシュ データアレイ
H'F200 0000	命令TLB アドレスアレイ
H'F300 0000	命令TLB データアレイ1、2
H'F400 0000	オペランドキャッシュ アドレスアレイ
H'F500 0000	オペランドキャッシュ データアレイ
H'F600 0000	共用TLB アドレスアレイ
H'F700 0000	共用TLB データアレイ1、2
H'F800 0000	予約領域
H'FF00 0000	制御レジスタ領域

図 3.4 P4 領域

H'E000 0000 ~ H'E3FF FFFF までは、ストアキュー (SQ) にアクセスするためのアドレスです。MMU が無効な場合 (MMUCR.AT=0)、SQ のアクセス権は MMUCR.SQMD ビットで指定します。詳細は、「4.6 ストアキュー」を参照してください。

H'F000 0000 ~ H'F0FF FFFF までは、命令キャッシュのアドレスアレイを直接アクセスするための領域です。詳細は、「4.5.1 IC アドレスアレイ」を参照してください。

H'F100 0000 ~ H'F1FF FFFF までは、命令キャッシュのデータアレイを直接アクセスするための領域です。詳細は、「4.5.2 IC データアレイ」を参照してください。

H'F200 0000 ~ H'F2FF FFFF までは、命令 TLB のアドレスアレイを直接アクセスするための領域です。詳細は、「3.7.1 ITLB アドレスアレイ」を参照してください。

H'F300 0000 ~ H'F3FF FFFF までは、命令 TLB のデータアレイ 1、2 を直接アクセスするための領域です。詳細は、「3.7.2 ITLB データアレイ 1」、「3.7.3 ITLB データアレイ 2」を参照してください。

H'F400 0000 ~ H'F4FF FFFF までは、オペランドキャッシュのアドレスアレイを直接アクセスするための領域です。詳細は、「4.5.3 OC アドレスアレイ」を参照してください。

H'F500 0000 ~ H'F5FF FFFF までは、オペランドキャッシュのデータアレイを直接アクセスするための領域です。詳細は、「4.5.4 OC データアレイ」を参照してください。

### 3. メモリマネジメントユニット (MMU)

H'F600 0000 ~ H'F6FF FFFF までは、共用 TLB のアドレスアレイを直接アクセスするための領域です。詳細は、「3.7.4 UTLB アドレスアレイ」を参照してください。

H'F700 0000 ~ H'F7FF FFFF までは、共用 TLB のデータアレイ 1、2 を直接アクセスするための領域です。詳細は、「3.7.5 UTLB データアレイ 1」、「3.7.6 UTLB データアレイ 2」を参照してください。

H'FF00 0000 ~ H'FFFF FFFF までは、内蔵周辺モジュール制御レジスタの領域です。

#### 3.3.2 外部メモリ空間

SH7091 は 29 ビットの外部メモリ空間をサポートします。外部メモリ空間は図 3.5 に示す通り 8 つの領域に分かれています。エリア 0 ~ エリア 6 は SRAM、シンクロナス DRAM、DRAM、PCMCIA などのメモリにつながる領域です。エリア 7 は予約領域です。詳細はハードウェアマニュアルの「13 章 パスステートコントローラ」を参照してください。

H'0000 0000	エリア0
H'0400 0000	エリア1
H'0800 0000	エリア2
H'0C00 0000	エリア3
H'1000 0000	エリア4
H'1400 0000	エリア5
H'1800 0000	エリア6
H'1C00 0000 H'1FFF FFFF	エリア7 (予約領域)

図 3.5 外部メモリ空間

#### 3.3.3 仮想メモリ空間

MMUCR.AT ビットを 1 にすることにより、SH7091 では物理メモリ空間の P0 領域と P3 領域と U0 領域を任意の外部メモリ空間へ 1k/4k/64k/1M バイトページ単位にマッピングすることができま  
す。また 8 ビットのアドレス空間識別子を用いることにより P0、U0、P3、ストアキュー領域を 256  
個まで増やすことが可能です。これを仮想メモリ空間と呼びます。仮想メモリ空間から 29 ビットの  
外部メモリ空間へのマッピングには TLB を用います。仮想メモリ空間を用いて外部メモリ空間のエ  
リア 7 をアクセスする場合のみエリア 7 の H'1F00 0000 ~ H'1F00 0000 までの領域が予約領域ではな  
くなり、物理メモリ空間の P4 領域の制御レジスタ領域と等価になります。仮想メモリ空間を図 3.6  
に示します。

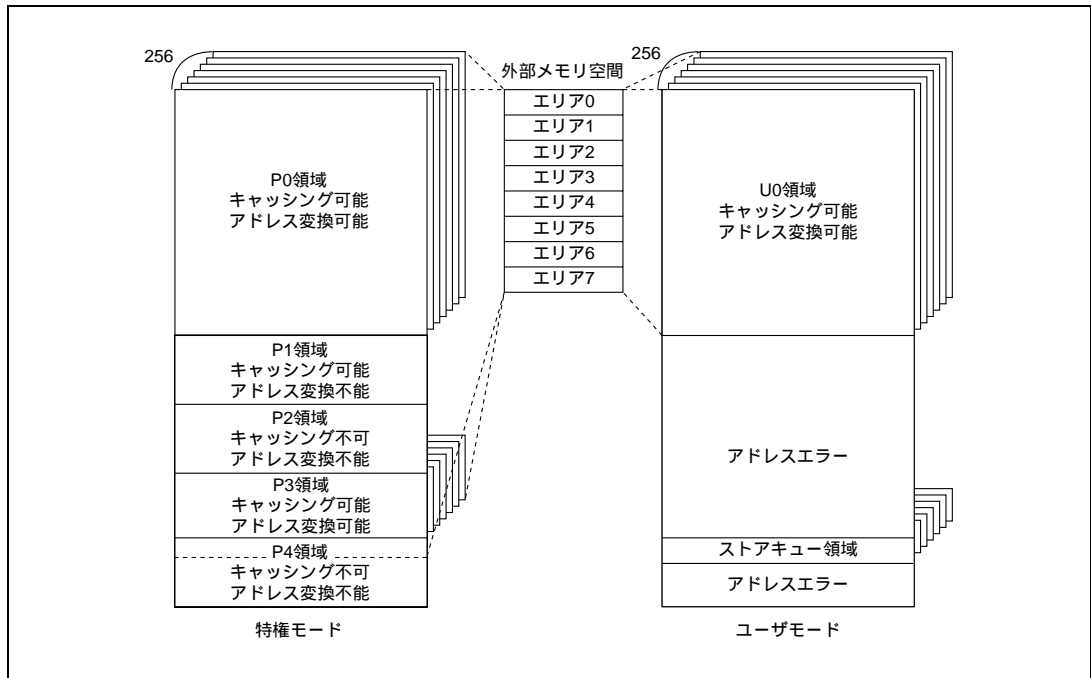


図 3.6 仮想メモリ空間 (MMUCR.AT=1)

## (1) P0、P3、U0 領域

P0(H'7C00 0000 から H'7FFF FFFF を除く)、P3、U0 領域はキャッシュを用いたアクセスと TLB を用いたアドレス変換が可能な領域です。これらの領域は TLB を用いて 1k/4k/64k/1M バイトページ単位に任意の外部メモリ空間へマッピングできます。CCR がキャッシュイネーブル状態にあり、かつ TLB のキャッシング可能ビット (C ビット) が 1 のとき、キャッシュを用いたアクセスが行えます。また、キャッシュへのライトアクセスにおけるコピーバック方式とライトスルー方式の切り換えは、TLB のライトスルービット (WT ビット) に従い、ページ単位に指定します。

P0、P3、U0 領域が TLB により外部メモリ空間へマッピングされるときのみ、外部メモリ空間のエリア 7 の H'1F00 0000 ~ H'1FFF FFFF が制御レジスタ領域に割り当てられます。これによりユーザーモードでも U0 領域から内蔵周辺モジュール制御レジスタをアクセスすることが可能となります。この場合、該当するページの C ビットには 0 を指定しなければなりません。

## (2) P1、P2、P4 領域

P1、P2、P4 領域 (ストアキュー領域を除く) に対して TLB を用いたアドレス変換は実行できません。これらの領域に対するアクセスは物理メモリ空間に対するアクセスと同じです。ストアキュー領域は MMU によって任意の外部メモリ空間にマッピングすることができます。ただし、例外処理の場合の動作は通常の P0、U0、P3 空間の場合とは異なります。詳細については「4.6 ストアキュー」を参照してください。

#### 3.3.4 内蔵 RAM 空間

SH7091 では、オペランドキャッシュ (16kB) の半分 (8kB) を内蔵 RAM として使用することが可能です。これは CCR の設定を変更することで行えます。

オペランドキャッシュを内蔵 RAM として使用する場合 (CCR.ORA = 1)、P0 領域の (H'7C00 0000 ~ H'7FFF FFFF) が内蔵 RAM 領域となります。この領域へはデータアクセス (バイト/ワード/ロングワード/クワッドワード) が可能です。ただしこの領域は、RAM モード時以外には使用できません。

#### 3.3.5 アドレス変換

MMU を使用するとき、仮想アドレス空間はページという単位に分割され、そのページ単位で物理アドレスに変換されます。外部メモリ上のアドレス変換テーブルには、仮想アドレスに対応する物理アドレスや、記憶保護コードなどの付加情報が格納され、TLB にはアドレス変換の高速化のために、外部メモリ上のアドレス変換テーブルの内容がキャッシングされます。SH7091 では命令のアクセスには ITLB を、データのアクセスには UTLB を基本的に用います。P4 領域以外へのアクセスが発生するとそのアクセスされた仮想アドレスが物理アドレスへ変換されます。その仮想アドレスが P1、P2 領域に属する場合、TLB をアクセスせずに物理アドレスが一意に決定されます。その仮想アドレスが P0、U0、P3 領域に属する場合には、仮想アドレスで TLB が検索され、その仮想アドレスが TLB に登録されている場合には、TLB ヒットとなり、TLB から対応する物理アドレスが読み出されます。またアクセスされた仮想アドレスが TLB に登録されていない場合には、TLB ミス例外が発生し、処理が TLB ミス例外ルーチンへ移ります。TLB ミス例外ルーチンでは、外部メモリ上のアドレス変換テーブルを検索し、対応する物理アドレス、ページ管理情報を TLB に登録します。そして例外処理ルーチンから復帰後、TLB ミス例外を発生させた命令を再実行します。

#### 3.3.6 単一仮想記憶モードと多重仮想記憶モード

仮想記憶方式には単一仮想記憶方式と多重仮想記憶方式があり、MMUCR.SV により選択が可能です。単一仮想記憶方式では、複数のプロセスが仮想アドレス空間を排他的に使用しながら同時に走行し、ある仮想アドレスに対応する物理アドレスは一意に定まります。多重仮想記憶方式では、複数のプロセスが仮想アドレス空間を共有して使用しながら走行するため、ある仮想アドレスはプロセスにより異なった物理アドレスに変換され得ます。単一仮想記憶方式と多重仮想記憶方式との動作上の違いは TLB のアドレス比較の方式 (「3.4.3 アドレス変換方式」参照) のみです。

#### 3.3.7 アドレス空間識別子 (ASID)

多重仮想記憶モードの場合、8 ビットのアドレス空間識別子 (ASID) は仮想アドレス空間を共有しながら同時に走行する複数のプロセスを区別するために用いられます。ASID は 8 ビットで、ソフトウェアが MMU 内の PTEH に現在走行中のプロセスの ASID をセットすることで設定可能です。また ASID によりプロセス切り換えの際に TLB をパージしないで済みます。

単一仮想記憶モードの場合、ASID は仮想アドレス空間を排他的に使用しながら同時に走行する複数のプロセスの記憶保護のために用いられます。



## 3.4 TLB の機能

### 3.4.1 共用 TLB (UTLB) の構成

UTLB は次の 2 つの目的のために使用されます。

- (1) データアクセスのとき、仮想アドレスを物理アドレスへ変換する。
- (2) 命令 TLB ミスのとき、ITLB へ登録するアドレス変換情報のテーブル。

このため共用 TLB と呼ばれます。UTLB には外部メモリ上に置かれるアドレス変換テーブルの情報がキャッシングされます。アドレス変換テーブルには仮想ページ番号とアドレス空間識別子、それに対応する物理ページ番号とページ管理情報が格納されています。図 3.7 に UTLB の構成を示します。UTLB はフルアソシアティブ方式の 64 エントリで構成されています。図 3.8 にページサイズとアドレスの関係を示します。

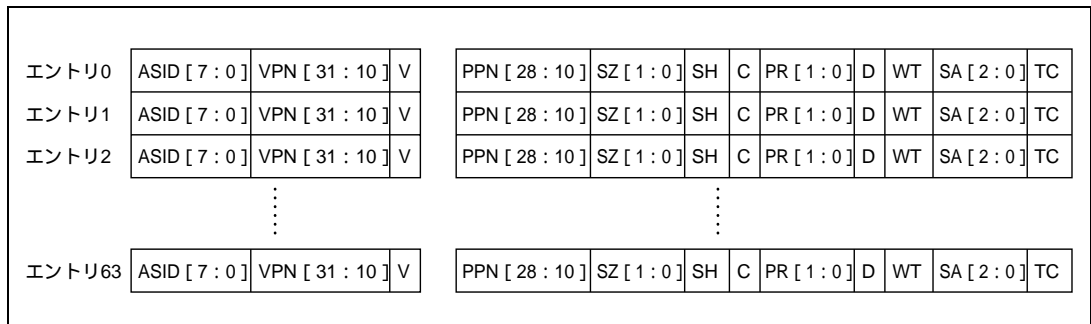


図 3.7 UTLB の構成

### 3. メモリマネジメントユニット (MMU)

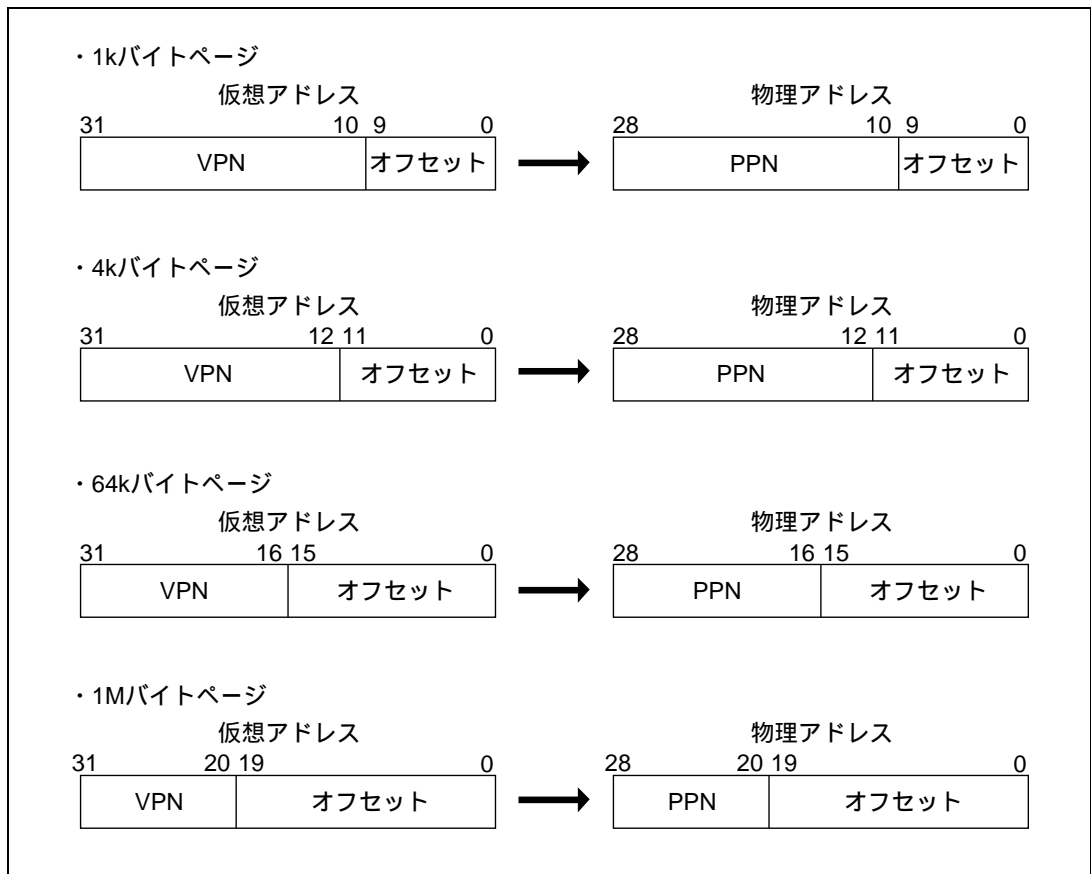


図 3.8 ページサイズとアドレスの関係

- VPN : 仮想ページ番号
  - 1k バイトページ のとき、仮想アドレス の上位 22 ビット
  - 4k バイトページ のとき、仮想アドレス の上位 20 ビット
  - 64k バイトページ のとき、仮想アドレス の上位 16 ビット
  - 1M バイトページ のとき、仮想アドレス の上位 12 ビット
- ASID : アドレス空間識別子  
仮想ページをアクセスできるプロセスを示します。  
単一仮想記憶モードかつユーザモードか、多重仮想記憶モードのときで、SHビットが0ならアドレス比較の際にPTEH中のASIDと比較されます。
- SH : 共有状態ビット
  - 0 のとき複数のプロセスでページを共有しません。
  - 1 のとき複数のプロセスでページを共有します。

- SZ : ページサイズビット  
ページサイズを指定します。
  - 00:1k バイトページ
  - 01:4k バイトページ
  - 10:64k バイトページ
  - 11:1M バイトページ
  
- V : 有効ビット  
エントリが有効かどうかを示します。
  - 0 のとき無効
  - 1 のとき有効パワーオンリセット時に0にクリアされます。  
マニュアルリセット時には変化しません。
  
- PPN : 物理ページ番号  
物理アドレスの上位22ビット
  - 1k バイトページのときは PPN [ 28 : 10 ] が有効です。
  - 4k バイトページのときは PPN [ 28 : 12 ] が有効です。
  - 64k バイトページのときは PPN [ 28 : 16 ] が有効です。
  - 1M バイトページのときは PPN [ 28 : 20 ] が有効です。またPPNの設定においてはシノニム問題に注意してください(「3.5.5 シノニム問題の回避」参照)。
  
- PR : 保護キーデータ  
ページのアクセス権をコードで表した2ビットデータ
  - 00 : 特権モードで読み出しのみ可能。
  - 01 : 特権モードで読み出し / 書き込み可能。
  - 10 : 特権 / ユーザモードで読み出しのみ可能。
  - 11 : 特権 / ユーザモードで読み出し / 書き込み可能。
  
- C : キャッシング可能ビット  
ページがキャッシング可能かどうかを示します。
  - 0 のときキャッシング不可能。
  - 1 のときキャッシング可能。制御レジスタ空間のマッピングを行う場合、このビットは0にしてください。
  
- D : ダーティビット  
ページに書き込みが行われたかどうかを示します。
  - 0 のとき書き込みが行われていない。
  - 1 のとき書き込みが行われている。
  
- WT : ライトスルービット  
キャッシュへの書き込みモードを指定します。
  - 0 : コピーバックモード
  - 1 : ライトスルーモード

### 3. メモリマネジメントユニット (MMU)

- SA : 空間属性ビット

エリア5または6 に接続するPCMCIAにページをマッピングする場合にのみ有効です。

- 000: 不定
- 001: 可変サイズの I/O 空間 (基本サイズは  $\overline{\text{IOIS16}}$  信号に従います)
- 010: 8 ビット I/O 空間
- 011: 16 ビット I/O 空間
- 100: 8 ビット共用メモリ空間
- 101: 16 ビット共用メモリ空間
- 110: 8 ビット属性メモリ空間
- 111: 16 ビット属性メモリ空間

- TC : タイミングコントロールビット

エリア5、6のバスコントロールユニットに用いられるウェイトコントロールレジスタを選択するために使用します。

- 0 : WCR2 (A5W2 ~ A5W0) と PCR (A5PCW1 ~ A5PCW0, A5TED2 ~ A5TED0, A5TEH2 ~ A5TEH0) を使用
- 1 : WCR2 (A6W2 ~ A6W0) と PCR (A6PCW1 ~ A6PCW0, A6TED2 ~ A6TED0, A6TEH2 ~ A6TEH0) を使用

#### 3.4.2 命令 TLB (ITLB) の構成

ITLB は命令アクセスのとき、仮想アドレスを物理アドレスへ変換するために用いられます。ITLB には UTLB 上に置かれるアドレス変換テーブルの情報がキャッシングされます。図 3.9 に ITLB の構成を示します。ITLB はフルアソシアティブの 4 エントリで構成されています。

エントリ0	ASID [ 7 : 0 ]	VPN [ 31 : 10 ]	V	PPN [ 28 : 10 ]	SZ [ 1 : 0 ]	SH	C	PR	SA [ 2 : 0 ]	TC
エントリ1	ASID [ 7 : 0 ]	VPN [ 31 : 10 ]	V	PPN [ 28 : 10 ]	SZ [ 1 : 0 ]	SH	C	PR	SA [ 2 : 0 ]	TC
エントリ2	ASID [ 7 : 0 ]	VPN [ 31 : 10 ]	V	PPN [ 28 : 10 ]	SZ [ 1 : 0 ]	SH	C	PR	SA [ 2 : 0 ]	TC
エントリ3	ASID [ 7 : 0 ]	VPN [ 31 : 10 ]	V	PPN [ 28 : 10 ]	SZ [ 1 : 0 ]	SH	C	PR	SA [ 2 : 0 ]	TC

【注】 1. D、WTビットをサポートしません。  
2. PRビットが1ビットになり、UTLBのPRビットの上位1ビットに対応します。

図 3.9 ITLB の構成

## 3.4.3 アドレス変換方式

図 3.10、図 3.11 に、UTLB、ITLB を用いたメモリアクセスのフローを示します。

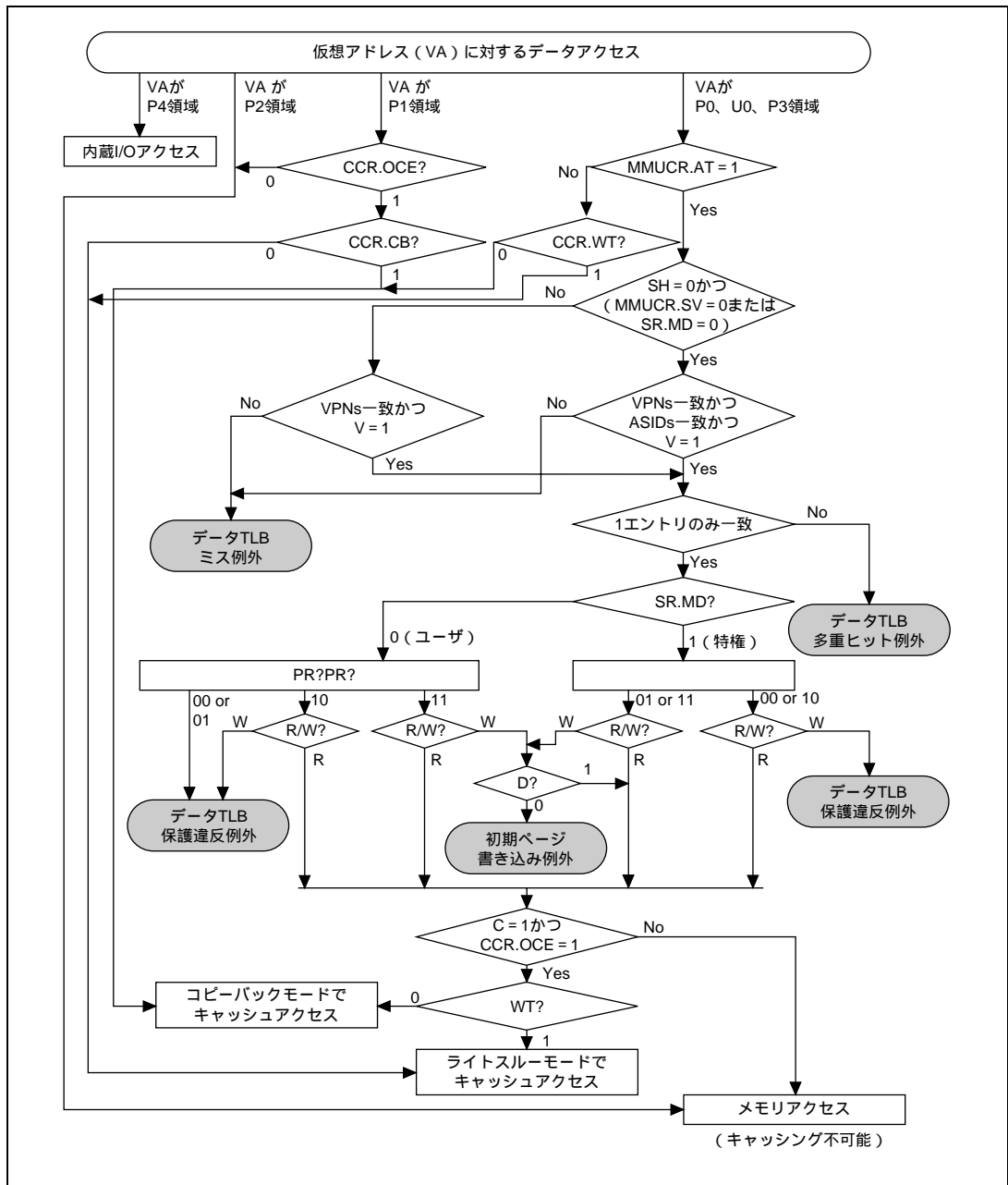
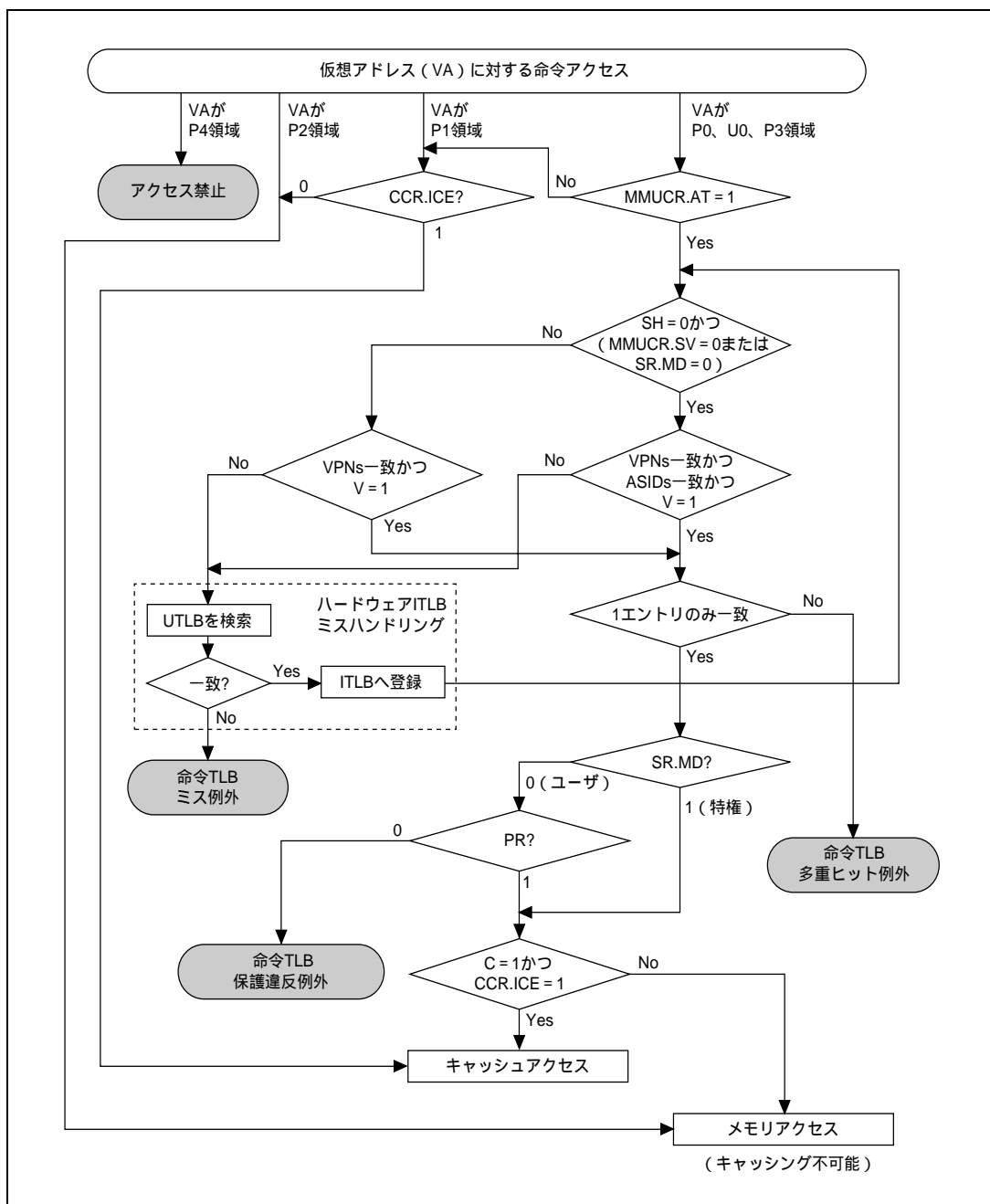


図 3.10 UTLB を用いたメモリアクセスフロー

### 3. メモリマネジメントユニット (MMU)



## 3.5 MMU の機能

### 3.5.1 MMU のハードウェア管理

SH7091 がサポートする MMU の機能として次のものがあります。

- (1) ソフトウェアがアクセスする仮想アドレスをデコードし、MMUCR の設定に従い UTLB/ITLB を制御してアドレス変換を行います。
- (2) アドレス変換の際に読み出されたページ管理情報をもとに、キャッシュへのアクセス状態を判定します (C、WT、SA、TC ビット)。
- (3) データアクセス、命令アクセスにおいて正常にアドレス変換が行われなかった場合、MMU 例外の発生により、ソフトウェアに通知します。
- (4) 命令アクセスで ITLB にアドレス変換情報が登録されていないとき、UTLB を検索し、UTLB に必要なアドレス変換情報が登録されていた場合、MMUCR.LRUI に従い ITLB にそのアドレス変換情報をコピーします。

### 3.5.2 MMU のソフトウェア管理

MMU に対するソフトウェアの処理として次のものがあります。

- (1) MMU 関連レジスタの設定。一部ハードウェアにより自動的に更新されるものもあります。
- (2) TLB エントリの登録、削除、読み出し。UTLB エントリの登録には LDTLB 命令を用いる方法と、メモリ割り付け UTLB に直接書き込む方法があります。ITLB エントリの登録はメモリ割り付け ITLB に直接書き込む方法しかありません。UTLB/ITLB エントリの削除と読み出しは、メモリ割り付け UTLB/ITLB をアクセスすることで可能です。
- (3) MMU 例外処理。MMU 例外が発生したときにハードウェア側から設定された情報を元に処理を行います。

### 3.5.3 MMU の命令 (LDTLB)

UTLB エントリを登録する命令として TLB ロード命令 (LDTLB) があります。LDTLB 命令が発行されると SH7091 は PTEH と PTEL と PTEA の内容を MMUCR.URC が指し示す UTLB エントリにコピーします。LDTLB 命令により ITLB エントリの更新は行われませんので、UTLB エントリから追い出されたアドレス変換情報が ITLB エントリに残る可能性があります。LDTLB 命令はアドレス変換情報を変更する命令のため、必ず P1、P2 領域のプログラムで発行するようにしてください。図 3.12 に LDTLB 命令の動作を示します。

### 3. メモリマネジメントユニット (MMU)

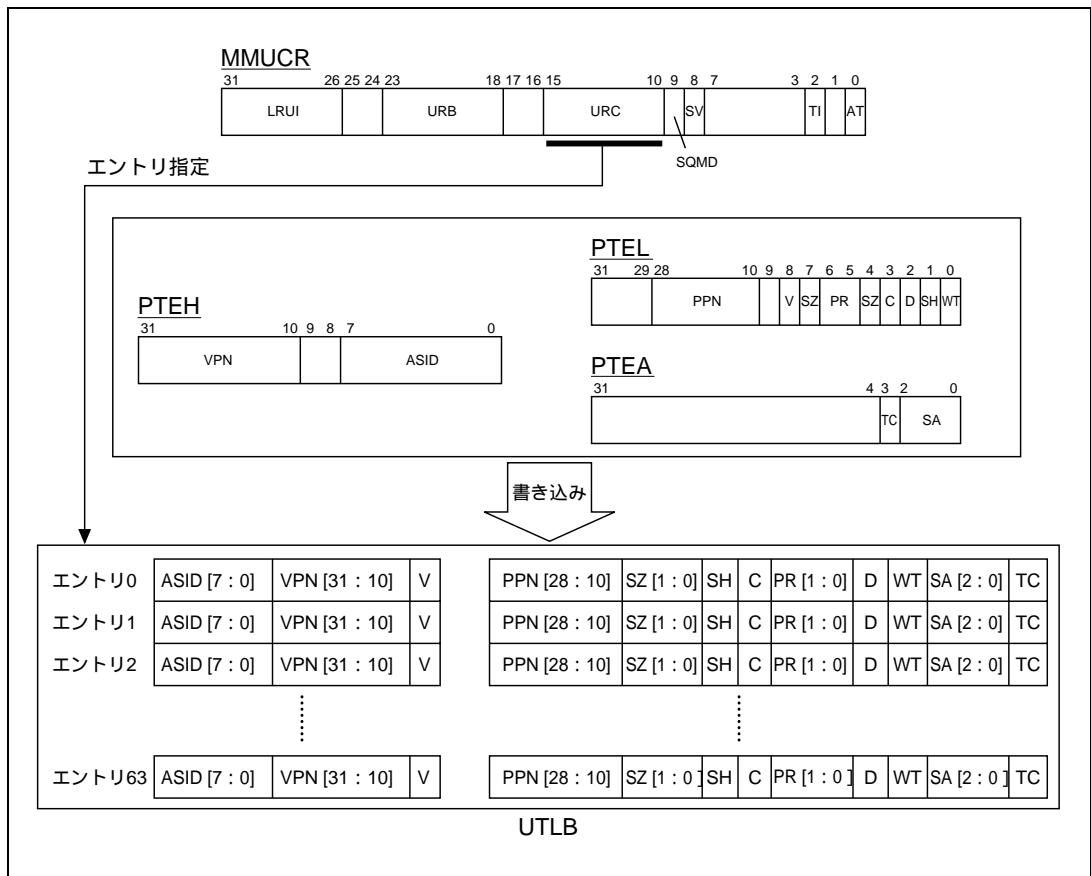


図 3.12 LDTLB 命令の動作

#### 3.5.4 ハードウェア ITLB ミスハンドリング

SH7091 は命令アクセスの際、ITLB を検索して必要なアドレス変換情報を見つけられなかった (ITLB ミス) 場合、ハードウェアにより UTLB を検索し必要なアドレス変換情報があれば ITLB への登録を行います。これをハードウェア ITLB ミスハンドリングと呼びます。UTLB を検索しても必要なアドレス変換情報が見つからない場合、命令 TLB ミス例外を発生し、処理をソフトウェアへ移します。

#### 3.5.5 シノニム問題の回避

TLB エントリに 1k, 4k バイトページを登録するときにシノニム問題が発生する可能性があります。シノニム問題とは、複数の仮想アドレスが 1 つの物理アドレスにマッピングされる場合に、キャッシュの複数のエントリに同一の物理アドレスのデータが登録されてしまい、データの一致性を保証できなくなるという問題です。この問題は命令 TLB や命令キャッシュではデータの読み出ししか行わないため発生しません。SH7091 ではオペランドキャッシュの高速動作のために仮想アドレスの [13 : 5] を用いて、エントリの指定を行います。しかし 1k バイトページでは仮想アドレスの [13 : 10] が、4k バイトページでは仮想アドレスの [13 : 12] がアドレス変換の対象になります。このため変換後の物理アドレスの [13 : 10] と仮想アドレスの [13 : 10] が異なる可能性があります。



このため UTLB エントリへのアドレス変換情報の登録には以下の制限が生じます。

- (1) 複数の 1k バイトページの UTLB エントリが同一の物理アドレスに変換されるアドレス変換情報を UTLB に登録するとき、VPN [ 13 : 10 ] は必ず等しくなるようにしてください。
- (2) 複数の 4k バイトページの UTLB エントリが同一の物理アドレスに変換されるアドレス変換情報を UTLB に登録するとき、VPN [ 13 : 12 ] は必ず等しくなるようにしてください。
- (3) 1k バイトページの UTLB エントリの物理アドレスを、異なるページサイズの UTLB エントリで使用しないでください。
- (4) 4k バイトページの UTLB エントリの物理アドレスを、異なるページサイズの UTLB エントリで使用しないでください。

上記の制限はキャッシュを用いたアクセスを行う場合に限定されます。キャッシュインデックスモードを用いた場合、VPN [ 25 ] が VPN [ 13 ] の代わりにエントリアドレスとして使用されるため、上記制限事項は、VPN [ 25 ] に対して有効となります。

【注】 将来の SH シリーズ拡張に備えて、複数のアドレス変換情報が同一の物理メモリを使用する場合、VPN [ 20 : 10 ] を等しくなるようにしてください。また異なるページサイズのアドレス変換情報で同一の物理アドレスを使用しないでください。

## 3.6 MMU 例外

MMU 例外には、命令 TLB 多重ヒット例外、命令 TLB ミス例外、命令 TLB 保護違反例外、データ TLB 多重ヒット例外、データ TLB ミス例外、データ TLB 保護違反例外、初期ページ書き込み例外の 7 つの例外があります。各例外の発生条件については図 3.10 と図 3.11 を参照してください。

### 3.6.1 命令 TLB 多重ヒット例外

命令 TLB 多重ヒット例外は、命令アクセスした仮想アドレスに一致する ITLB エントリが複数存在した場合に発生します。ハードウェア ITLB ミスハンドリングにより UTLB を検索する際に UTLB で多重ヒットが発生した場合は、データ TLB 多重ヒット例外となります。

命令 TLB 多重ヒット例外が発生すると、リセットになり、この場合キャッシュのコヒーレンスは保証しません。

#### ● ハードウェア処理

命令 TLB 多重ヒット例外のとき、ハードウェアは次の処理を行います。

- (1) 例外の発生した仮想アドレスを TEA に設定します。
- (2) 例外コード H'140 を EXPEVT に設定します。
- (3) リセット処理ルーチン(H'A000 0000)に分岐します。

#### ● ソフトウェア処理(リセットルーチン)

リセット処理ルーチンで多重ヒットが発生させた ITLB エントリを確認します。この例外はプログラムのデバッグ時に用いるためのもので、通常はこの例外が発生させないでください。

#### 3.6.2 命令 TLB ミス例外

命令 TLB ミス例外は、ハードウェア ITLB ミスハンドリングにより UTLB エントリに命令アクセスした仮想アドレスに対応するアドレス変換情報が見つからなかったときに発生します。命令 TLB ミス例外のハードウェアで行われる処理と、ソフトウェアで行う処理は次のとおりです。これはデータ TLB ミス例外時の処理と同じです。

- ハードウェア処理  
命令 TLB ミス例外のとき、ハードウェアは次の処理を行います。
  - (1) 例外が発生した仮想アドレスの VPN を PTEH に設定します。
  - (2) 例外の発生した仮想アドレスを TEA に設定します。
  - (3) 例外コード H'040 を、EXPEVT に設定します。
  - (4) 例外が発生した命令のアドレスを指す PC の値を SPC に設定します。もし例外が遅延スロットで発生した場合は、遅延分岐命令のアドレスを指す PC の値を SPC に設定します。
  - (5) 例外が発生したときの SR の内容を SSR に設定します。
  - (6) SR の MD ビットを 1 に設定し、特権モードに切り換えます。
  - (7) SR の BL ビットを 1 に設定し、これ以降の例外要求をマスクします。
  - (8) SR の RB ビットを 1 に設定します。
  - (9) VBR の内容にオフセット H'0000 0400 を加えたアドレスに分岐し、命令 TLB ミス例外処理ルーチンを開始します。
- ソフトウェア処理(命令 TLB ミス例外処理ルーチン)  
外部メモリのページテーブルを検索し、必要なページテーブルエントリを割り当てるのはソフトウェアの責任です。必要なページテーブルエントリを探して割り当てるために、ソフトウェアでは次のように処理してください。
  - (1) 外部メモリのアドレス変換テーブルに記録されているページテーブルエントリの PPN、PR、SZ、C、D、SH、V、WT の各ビットの値を、PTEL に書き込みます。必要なら SA、TC の値を PTEA に書き込みます。
  - (2) エントリ置き換えで置き換えられるエントリをソフトウェアで指定する場合、その値を MMUCR レジスタの URC に書き込みます。このとき URC が URB を超えるような場合、LDTLB 命令発行後に適切な値に変更してください。
  - (3) LDTLB 命令を実行させ、PTEH、PTEL、PTEA の内容を TLB に書き込みます。
  - (4) 最後に、例外処理からの復帰命令 (RTE) を実行させ、例外処理ルーチンを終わらせ、制御を通常の流れに戻してください。ただし、LDTLB 命令の次の命令以降に RTE 命令を発行してください。

#### 3.6.3 命令 TLB 保護違反例外

命令 TLB 保護違反例外は、命令アクセスした仮想アドレスに一致するアドレス変換情報が ITLB エントリに存在するにもかかわらず、実際のアクセスタイプが PR ビットで指定されるアクセス権で許されていない場合に発生します。命令 TLB 保護違反例外のハードウェアで行われる処理と、ソフトウェアで行う処理は次の通りです。

- ハードウェア処理  
命令 TLB 保護違反例外のとき、ハードウェアは次の処理を行います。
  - (1) 例外が発生した仮想アドレスの VPN を PTEH に設定します。
  - (2) 例外の発生した仮想アドレスを TEA に設定します。
  - (3) 例外コード H'0A0 を EXPEVT に設定します。

- (4) 例外が発生した命令のアドレスを指す PC の値を SPC に設定します。もし例外が遅延スロットで発生した場合は、遅延分岐命令のアドレスを指す PC の値を SPC に設定します。
- (5) 例外が発生したときの SR の内容を SSR に設定します。
- (6) SR の MD ビットを 1 に設定し、特権モードに切り換えます。
- (7) SR の BL ビットを 1 に設定し、これ以降の例外要求をマスクします。
- (8) SR の RB ビットを 1 に設定します。
- (9) VBR の内容にオフセット H'0000 0100 を加えたアドレスに分岐し、命令 TLB 保護違反例外処理ルーチンを開始します。

- ソフトウェア処理(命令 TLB 保護違反例外処理ルーチン)

命令 TLB 保護違反を解決し、例外処理からの復帰命令(RTE)を実行させ、例外処理ルーチンを終わらせ、制御を通常の流れに戻してください。ただし LDTLB 命令の次の命令以降に RTE 命令を発行してください。

### 3.6.4 データ TLB 多重ヒット例外

データ TLB 多重ヒット例外は、データアクセスした仮想アドレスに一致する UTLB エントリが複数存在した場合に発生します。ハードウェア ITLB ミスハンドリングにより UTLB を検索する際に UTLB で多重ヒットが発生した場合にも、データ TLB 多重ヒット例外となります。

データ TLB 多重ヒット例外が発生すると、リセットになり、この場合キャッシュのコヒーレンシは保証しません。また例外発生以前の UTLB 内の PPN の内容は壊れることがあります。

- ハードウェア処理

データ TLB 多重ヒット例外のとき、ハードウェアは次の処理を行います。

- (1) 例外の発生した仮想アドレスを TEA に設定します。
- (2) 例外コード H'140 を EXPEVT に設定します。
- (3) リセット処理ルーチン(H'A000 0000)に分岐します。

- ソフトウェア処理(リセットルーチン)

リセット処理ルーチンで多重ヒットを発生させた UTLB エントリを確認します。この例外はプログラムのデバッグ時に用いるためのもので、通常はこの例外を発生させないでください。

### 3.6.5 データ TLB ミス例外

データ TLB ミス例外は、データアクセスした仮想アドレスに対応するアドレス変換情報が UTLB 内に見つからなかったときに発生します。データ TLB ミス例外のハードウェアで行われる処理と、ソフトウェアで行う処理は次のとおりです。

- ハードウェア処理

データ TLB ミス例外のとき、ハードウェアは次の処理を行います。

- (1) 例外が発生した仮想アドレスの VPN を PTEH に設定します。
- (2) 例外の発生した仮想アドレスを TEA に設定します。
- (3) 読み出しのとき例外コード H'040 を、書き込みのとき例外コード H'060 を、EXPEVT に設定します (OCBP、OCWBW: 読み出し; OCBI、MOVCA.L: 書き込み)。
- (4) 例外が発生した命令のアドレスを指す PC の値を SPC に設定します。もし例外が遅延スロットで発生した場合は、遅延分岐命令のアドレスを指す PC の値を SPC に設定します。
- (5) 例外が発生したときの SR の内容を SSR に設定します。

### 3. メモリマネジメントユニット (MMU)

---

- (6) SR の MD ビットを 1 に設定し、特権モードに切り換えます。
  - (7) SR の BL ビットを 1 に設定し、これ以降の例外要求をマスクします。
  - (8) SR の RB ビットを 1 に設定します。
  - (9) VBR の内容にオフセット H'0000 0400 を加えたアドレスに分岐し、データ TLB ミス例外処理ルーチンを開始します。
- ソフトウェア処理(データ TLB ミス例外処理ルーチン)  
外部メモリのページテーブルを検索し、必要なページテーブルエントリを割り当てるのはソフトウェアの責任です。必要なページテーブルエントリを探して割り当てるために、ソフトウェアでは次のように処理してください。
    - (1) 外部メモリのアドレス変換テーブルに記録されているページテーブルエントリの PPN、PR、SZ、C、D、SH、V、WT の各ビットの値を、PTEL に書き込みます。また、必要なら SA と TC の値を PTEA に書き込んでください。
    - (2) エントリ置き換えで置き換えられるエントリをソフトウェアで指定する場合、その値を MMUCR レジスタの URC に書き込みます。このとき URC が URB を超えるような場合、LDTLB 命令発行後に適切な値に変更してください。
    - (3) LDTLB 命令を実行させ、PTEH、PTEL、PTEA の内容を UTLB に書き込みます。
    - (4) 最後に、例外処理からの復帰命令 (RTE) を実行させ、例外処理ルーチンを終わらせ、制御を通常の流れに戻してください。ただし、LDTLB 命令の次の命令以降に RTE 命令を発行してください。

#### 3.6.6 データ TLB 保護違反例外

データ TLB 保護違反例外は、データアクセスした仮想アドレスに一致するアドレス変換情報が UTLB エントリに存在するにもかかわらず、実際のアクセスタイプが PR ビットで指定されるアクセス権で許されていない場合に発生します。データ TLB 保護違反例外のハードウェアで行われる処理と、ソフトウェアで行う処理は次のとおりです。

- ハードウェア処理  
データ TLB 保護違反例外のとき、ハードウェアは次の処理を行います。
  - (1) 例外が発生した仮想アドレスの VPN を PTEH に設定します。
  - (2) 例外の発生した仮想アドレスを TEA に設定します。
  - (3) 読み出しのとき例外コード H'0A0 を、書き込みのとき例外コード H'0C0 を、EXPEVT に設定します (OCBP、OCBWB: 読み出し; OCBI、MOVCA.L: 書き込み)。
  - (4) 例外が発生した命令のアドレスを指す PC の値を SPC に設定します。もし例外が遅延スロットで発生した場合は、遅延分岐命令のアドレスを指す PC の値を SPC に設定します。
  - (5) 例外が発生したときの SR の内容を SSR に設定します。
  - (6) SR の MD ビットを 1 に設定し、特権モードに切り換えます。
  - (7) SR の BL ビットを 1 に設定し、これ以降の例外要求をマスクします。
  - (8) SR の RB ビットを 1 に設定します。
  - (9) VBR の内容にオフセット H'0000 0100 を加えたアドレスに分岐し、データ TLB 保護違反例外処理ルーチンを開始します。
- ソフトウェア処理(データ TLB 保護違反例外処理ルーチン)  
データ TLB 保護違反を解決し、例外処理からの復帰命令 (RTE) を実行させ、例外処理ルーチンを終わらせ、制御を通常の流れに戻してください。ただし LDTLB 命令の次の命令以降に RTE 命令を発行してください。

### 3.6.7 初期ページ書き込み例外

初期ページ書き込み例外は、データアクセス（書き込み）した仮想アドレスに一致するアドレス変換情報が UTLB エントリに存在し、アクセス権も許されているにもかかわらず、D ビットが 0 であった場合に発生します。初期ページ書き込み例外のハードウェアで行われる処理と、ソフトウェアで行う処理は次のとおりです。

- ハードウェア処理
 

初期ページ書き込み例外のとき、ハードウェアは次の処理を行います。

  - (1) 例外が発生した仮想アドレスの VPN を PTEH に設定します。
  - (2) 例外の発生した仮想アドレスを TEA に設定します。
  - (3) 例外コード H'080 を EXPEVT に設定します。
  - (4) 例外が発生した命令のアドレスを指す PC の値を SPC に設定します。もし例外が遅延スロットで発生した場合は、遅延分岐命令のアドレスを指す PC の値を SPC に設定します。
  - (5) 例外が発生したときの SR の内容を SSR に設定します。
  - (6) SR の MD ビットを 1 に設定し、特権モードに切り換えます。
  - (7) SR の BL ビットを 1 に設定し、これ以降の例外要求をマスクします。
  - (8) SR の RB ビットを 1 に設定します。
  - (9) VBR の内容にオフセット H'0000 0100 を加えたアドレスに分岐し、初期ページ書き込み例外処理ルーチンを開始します。
- ソフトウェア処理(初期ページ書き込み例外処理ルーチン)
 

ソフトウェアの責任で、次のように処理してください。

  - (1) 外部メモリから必要なページテーブルエントリを探し出します。
  - (2) 外部メモリのページテーブルエントリの D ビットに 1 を書き込んでください。
  - (3) 外部メモリに記憶されているページテーブルエントリの PPN、PR、SZ、C、D、WT、SH、V のビットの値を PTEL に書き込みます。また必要なら SA と TC の値を PTEA に書き込んでください。
  - (4) エントリ置き換えで置き換えられるエントリをソフトウェアで指定する場合、その値を MMUCR レジスタの URC に書き込みます。このとき URC が URB を超えるような場合、LDTLB 命令発行後に適切な値に変更してください。
  - (5) LDTLB 命令を実行させ、PTEH、PTEL、PTEA の内容を UTLB に書き込みます。
  - (6) 最後に、例外処理からの復帰命令 (RTE) を実行させ、例外処理ルーチンを終わらせ、制御を通常の流れに戻してください。ただし、LDTLB 命令の次の命令以降に RTE 命令を発行してください。

### 3.7 メモリ割り付け TLB の構成

ITLB/UTLB をソフトウェアで管理するために、特権モードのとき、P2 領域のプログラムから MOV 命令によって ITLB/UTLB の内容の読み出し、書き込みが可能です。別の領域のプログラムからアクセスする場合、動作の保証はありません。P2 領域以外への分岐は、この MOV 命令の 8 命令以降に行うようにしてください。ITLB/UTLB は物理メモリ空間の P4 領域に割り付けられています。ITLB では VPN、V、ASID をアドレスアレイとして、PPN、V、SZ、PR、C、SH をデータアレイ 1 として、また SA、TC をデータアレイ 2 としてアクセス可能です。

UTLB では VPN、D、V、ASID をアドレスアレイとして、PPN、V、SZ、PR、C、D、WT、SH をデータアレイ 1 として、また SA、TC をデータアレイ 2 としてアクセス可能です。V と D はアドレスアレイ側からとデータアレイ側からの両方からアクセスできるようになっています。アクセスサイズはロングワードサイズのみ可能です。この領域に対して命令フェッチは行えません。予約ビットに対しては、書き込み値として 0 を指定してください。読み出し値は保証しません。

#### 3.7.1 ITLB アドレスアレイ

ITLB のアドレスアレイは P4 領域の H'F200 0000 ~ H'F2FF FFFF に割り付けられています。アドレスアレイのアクセスには、32 ビットのアドレス部の指定（読み出し / 書き込み時）と 32 ビットのデータ部の指定（書き込み時）が必要です。アドレス部はアクセスするエントリを選択するための情報を指定し、データ部にはアドレスアレイに書き込む VPN、V、ASID を指定します。

アドレス部は、[ 31 : 24 ] が ITLB アドレスアレイを示す H'F2 になっており、[ 9 : 8 ] でエントリを選択するようになっています。アドレス部 [ 1 : 0 ] はロングワードアクセスのため 0 を指定してください。

データ部は、[ 31 : 10 ] が VPN を、[ 8 ] が V を、[ 7 : 0 ] が ASID を示します。

ITLB アドレスアレイに対しては以下の 2 種類の操作が可能です。

(1) ITLB アドレスアレイ リード

アドレス部に設定されたエントリに対応する ITLB エントリから、データ部へ VPN、V、ASID を読み出します。

(2) ITLB アドレスアレイ ライト

アドレス部に設定されたエントリに対応する ITLB エントリに対して、データ部で指定された VPN、V、ASID を書き込みます。

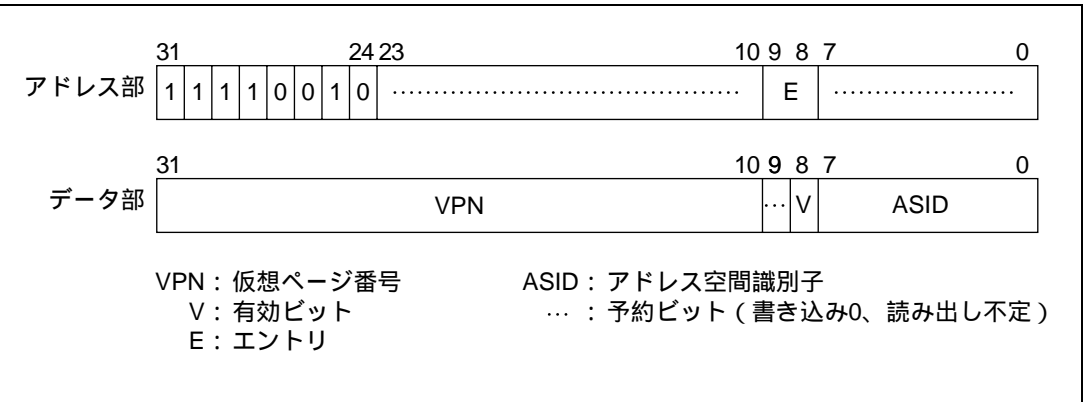


図 3.13 メモリ割り付け ITLB アドレスアレイ

### 3.7.2 ITLB データアレイ 1

ITLB のデータアレイ 1 は P4 領域の H'F300 0000 ~ H'F37F FFFF に割り付けられています。データアレイのアクセスには、32 ビットのアドレス部の指定（読み出し / 書き込み時）と 32 ビットのデータ部の指定（書き込み時）が必要です。アドレス部はアクセスするエントリを選択するための情報を指定し、データ部にはデータアレイ 1 に書き込む PPN、V、SZ、PR、C、SH を指定します。

アドレス部は、[31:23] が ITLB データアレイ 1 を示す H'F30 になっており、[9:8] でエントリを選択するようになっています。

データ部は、[28:10] が PPN を、[8] が V を、[7]、[4] が SZ を、[6] が PR を、[3] が C を、[1] が SH を示します。

ITLB データアレイ 1 に対しては以下の 2 種類の操作が可能です。

#### (1) ITLB データアレイ 1 リード

アドレス部に設定されたエントリに対応する ITLB エントリから、データ部へ PPN、V、SZ、PR、C、SH を読み出します。

#### (2) ITLB データアレイ 1 ライト

アドレス部に設定されたエントリに対応する ITLB エントリに対して、データ部で指定された PPN、V、SZ、PR、C、SH を書き込みます。

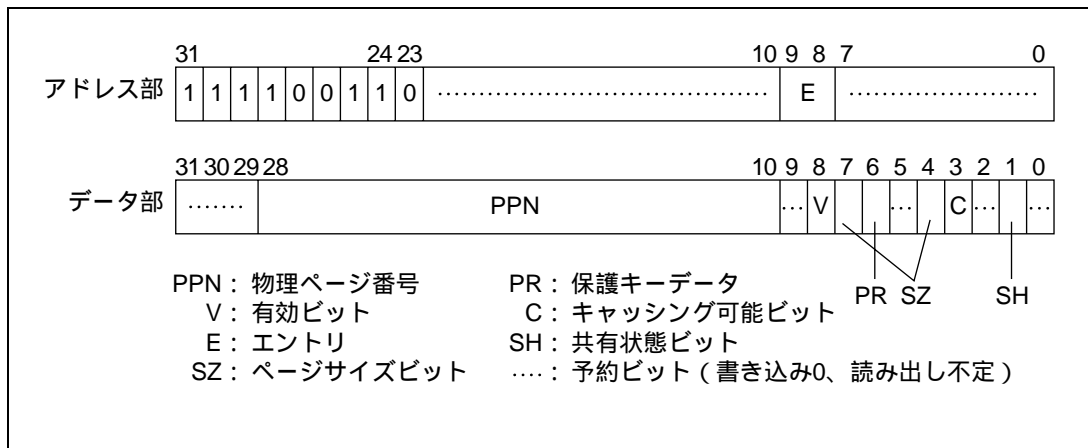


図 3.14 メモリ割り付け ITLB データアレイ 1

### 3.7.3 ITLB データアレイ 2

ITLB のデータアレイ 2 は P4 領域の H'F380 0000 ~ H'F3FF FFFF に割り付けられています。データアレイのアクセスには、32 ビットのアドレス部の指定（読み出し / 書き込み時）と 32 ビットのデータ部の指定（書き込み時）が必要です。アドレス部はアクセスするエントリを選択するための情報を指定し、データ部にはデータアレイ 2 に書き込む SA、TC を指定します。

アドレス部は、[31:23] が ITLB データアレイ 2 を示す H'F38 になっており、[9:8] でエントリを選択するようになっています。

データ部は、[2:0] が SA を、[3] が TC を示します。

ITLB データアレイ 2 に対しては以下の 2 種類の操作が可能です。

### 3. メモリマネジメントユニット (MMU)

#### (1) ITLB データアレイ 2 リード

データ部に設定されたエントリに対応するITLBエントリから、データ部へSAとTCを読み出します。

#### (2) ITLB データアレイ 2 ライト

アドレス部に設定されたエントリに対応するITLBエントリに対して、データ部で指定されたSAとTCを書き込みます。



図 3.15 メモリ割り付け ITLB データアレイ 2

### 3.7.4 UTLB アドレスアレイ

UTLB のアドレスアレイは P4 領域の H'F600 0000 ~ H'F6FF FFFF に割り付けられています。アドレスアレイのアクセスには、32 ビットのアドレス部の指定 (読み出し / 書き込み時) と 32 ビットのデータ部の指定 (書き込み時) が必要です。アドレス部はアクセスするエントリを選択するための情報を指定し、データ部にはアドレスアレイに書き込む VPN、D、V、ASID を指定します。

アドレス部は、[ 31 : 24 ] が UTLB アドレスアレイを示す H'F6 になっており、[ 13 : 8 ] でエントリを選択するようになっています。アドレス部 [ 7 ] の連想ビット (A ビット) は、UTLB アドレスアレイへの書き込みのときのアドレス比較の有無を指定します。

データ部は、[ 31 : 10 ] が VPN を、[ 9 ] が D を、[ 8 ] が V を、[ 7 : 0 ] が ASID を示します。UTLB アドレスアレイに対しては以下の 3 種類の操作が可能です。

#### (1) UTLB アドレスアレイ リード

アドレス部に設定されたエントリに対応するUTLBエントリから、データ部へVPN、D、V、ASIDを読み出します。リードの場合、アドレス部に指定される連想ビットは1でも0でも連想動作は行いません。

#### (2) UTLB アドレスアレイ ライト (連想なし)

アドレス部に設定されたエントリに対応するUTLBエントリに対して、データ部で指定されたVPN、D、V、ASIDを書き込みます。アドレス部のAビットは0にしてください。



## (3) UTLB アドレスアレイ ライト (連想あり)

アドレス部のAビットが1でライトのとき、データ部で指定されたVPNとPTEH.ASIDを用い、UTLBの全エントリとの間で比較が行われます。比較は通常のアドレス比較の規則に従いますが、TLBミス例外が発生した場合はノーオペレーションとなります。比較によりデータ部で指定したVPNに対応するUTLBエントリが存在した場合、そのエントリに対してデータ部で指定したDとVを書き込みます。一致するエントリが複数存在する場合は、データTLB多重ヒット例外となります。この連想動作はITLBに対しても同時に行われ、ITLB内に一致するエントリが存在した場合はそのエントリに対してVを書き込みます。UTLBでの比較でノーオペレーションとなってもITLBで一致していればITLB側にのみ書き込みは行います。またUTLBとITLBの両方で一致した場合、UTLBの情報がITLBへも書き込まれます。

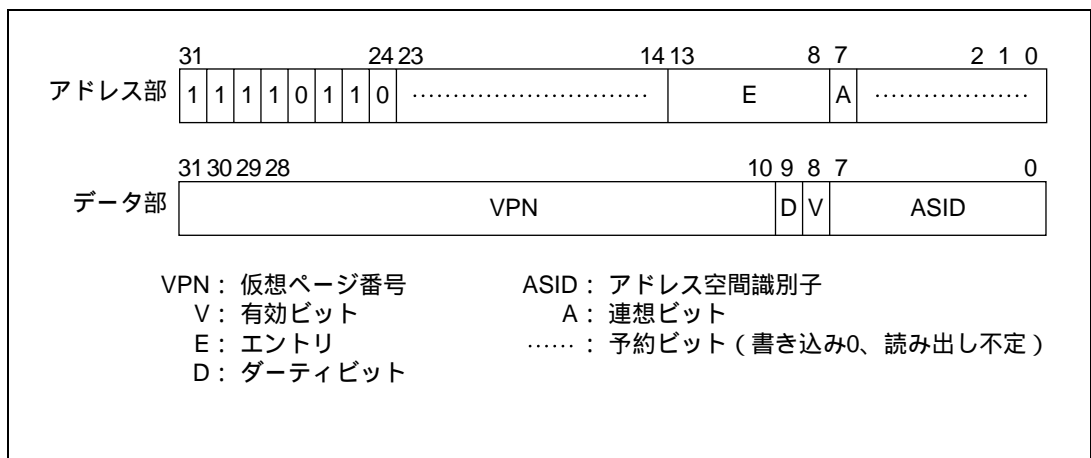


図 3.16 メモリ割り付け UTLB アドレスアレイ

## 3.7.5 UTLB データアレイ 1

UTLB のデータアレイ 1 は P4 領域の H'F700 0000 ~ H'F77F FFFF に割り付けられています。アドレスアレイのアクセスには、32 ビットのアドレス部の指定 (読み出し / 書き込み時) と 32 ビットのデータ部の指定 (書き込み時) が必要です。アドレス部はアクセスするエントリを選択するための情報を指定し、データ部にはデータアレイに書き込む PPN、V、SZ、PR、C、D、SH、WT を指定します。

アドレス部は、[ 31 : 23 ] が UTLB データアレイ 1 を示す H'F70 になっており、[ 13 : 8 ] でエントリを選択するようになっています。

データ部は、[ 28 : 10 ] が PPN を、[ 8 ] が V を、[ 7 ]、[ 4 ] が SZ を、[ 6 : 5 ] が PR を、[ 3 ] が C を、[ 2 ] が D を、[ 1 ] が SH を、[ 0 ] が WT を示します。

UTLB データアレイ 1 に対しては以下の 2 種類の操作が可能です。

## (1) UTLB データアレイ 1 リード

アドレス部に設定されたエントリに対応する UTLB エントリから、データ部へ PPN、V、SZ、PR、C、D、SH、WT を読み出します。

## (2) UTLB データアレイ 1 ライト

アドレス部に設定されたエントリに対応する UTLB エントリに対して、データ部で指定された PPN、V、SZ、PR、C、D、SH、WT を書き込みます。

### 3. メモリマネジメントユニット (MMU)

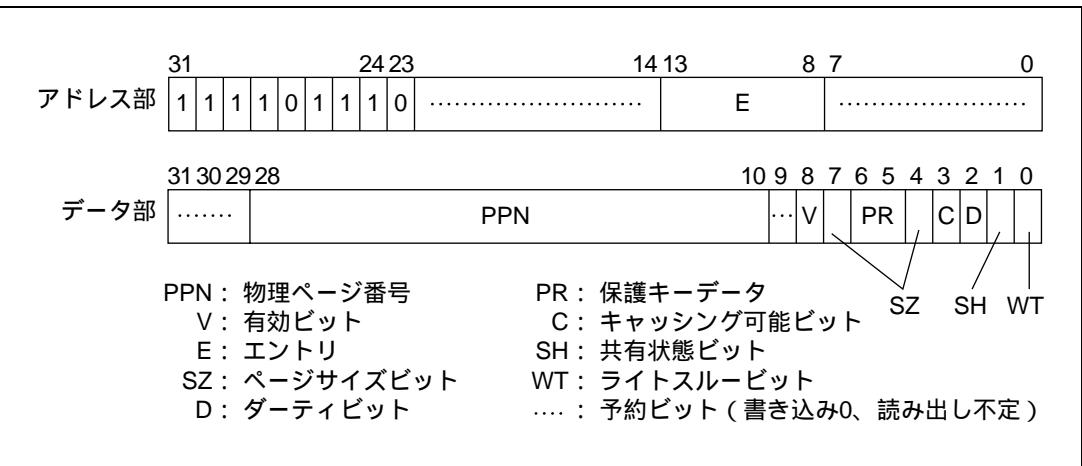


図 3.17 メモリ割り付け UTLB データアレイ 1

#### 3.7.6 UTLB データアレイ 2

UTLB のデータアレイ 2 は P4 領域の H'F780 0000 ~ H'F7FF FFFF に割り付けられています。データアレイのアクセスには、32 ビットのアドレス部の指定 (読み出し / 書き込み時) と 32 ビットのデータ部の指定 (書き込み時) が必要です。アドレス部はアクセスするエントリを選択するための情報を指定し、データ部にはデータアレイ 2 に書き込む SA、TC を指定します。

アドレス部は、[ 31 : 23 ] が UTLB データアレイ 2 を示す H'F78 になっており、[ 13 : 8 ] でエントリを選択するようになっています。

データ部は、[ 3 ] が TC を、[ 2 : 0 ] が SA を示します。

UTLB データアレイ 2 に対しては以下の 2 種類の操作が可能です。

(1) UTLB データアレイ 2 リード

アドレス部に設定されたエントリに対応する UTLB エントリから、データ部へ SA と TC を読み出します。

(2) UTLB データアレイ 2 ライト

アドレス部に設定されたエントリに対応する UTLB エントリに対して、データ部で指定された SA と TC を書き込みます。

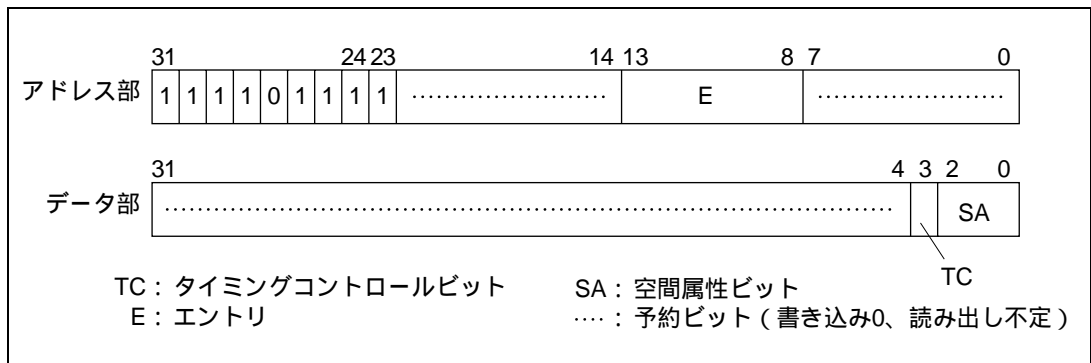


図 3.18 メモリ割り込み UTLB データアレイ 2

## 4. キャッシュ

### 4.1 概要

#### 4.1.1 特長

SH7091 は命令用に 8k バイトの命令キャッシュ (IC) を、データ用に 16k バイトのオペランドキャッシュ (OC) を内蔵しています。またオペランドキャッシュの半分のメモリ (8k バイト) を内蔵 RAM としても利用できます。キャッシュの特長を表 4.1 に示します。

表 4.1 キャッシュの特長

項目	命令キャッシュ	オペランドキャッシュ
容量	8k バイトキャッシュ	16k バイトキャッシュもしくは 8k バイトキャッシュ+8k バイト RAM
方式	ダイレクトマップ	ダイレクトマップ
ラインサイズ	32 バイト	32 バイト
エントリ数	256 エントリ	512 エントリ
ライト方式		コピーバック/ライトスルー選択可能

項目	ストアキュー
容量	2 × 32 バイト
アドレス	H'E000 0000 ~ H'E3FF FFFF
ライト	ストア命令 (1 サイクルライト)
ライトバック	プリフェッチ命令
アクセス権	MMU off : MMUCR.SQMD による MMU on : 個々のページ PR による

#### 4.1.2 レジスタの構成

キャッシュ制御レジスタの構成を表 4.2 に示します。

表 4.2 レジスタの構成

名称	略称	R/W	初期値 <sup>*1</sup>	P4 アドレス <sup>*2</sup>	エリア アドレス <sup>*2</sup>	アクセス サイズ
キャッシュ制御レジスタ	CCR	R/W	H'0000 0000	H'FF00 001C	H'1F00 001C	32
キューアドレス制御レジスタ 0	QACR0	R/W	不定	H'FF00 0038	H'1F00 0038	32
キューアドレス制御レジスタ 1	QACR1	R/W	不定	H'FF00 003C	H'1F00 003C	32

【注】 \*1 初期値とはパワーオンリセット、マニユアルリセット後の値のことを指します。

\*2 P4 アドレスは仮想 / 物理アドレス空間の P4 領域を用いた場合のものです。TLB を用いて物理アドレス空間のエリア 7 からアクセスする場合、アドレスの上位 3 ビットが無視されます。

## 4.2 レジスタの説明

キャッシュに関連するレジスタとして、キャッシュ制御レジスタ（CCR）があります。

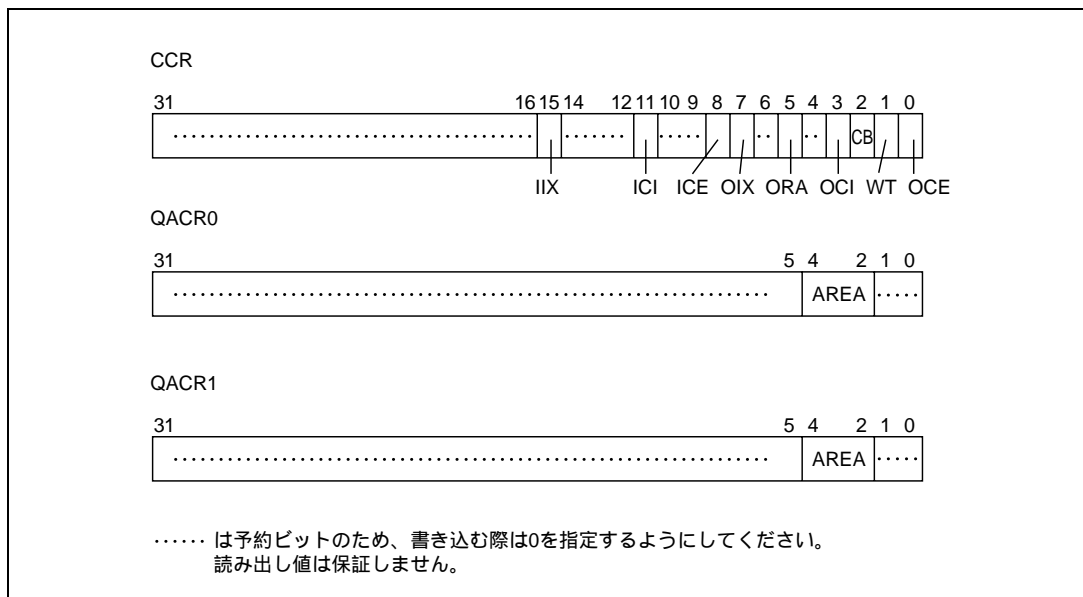


図 4.1 キャッシュ制御レジスタ（CCR）

### (1) キャッシュ制御レジスタ（CCR）

CCR には以下のビットがあります。

- IIX:IC index enable
- ICI:IC Invalidation
- ICE:IC Enable
- OIX:OC index enable
- ORA:OC RAM enable
- OCI:OC Invalidation
- CB:Copy-Back enable
- WT:Write-Through enable
- OCE:OC Enable

CCR へのロングワードアクセスは、P4 領域の H'FF00 001C とエリア 7 の H'1F00 001C から実行されます。CCR ビットは下記に示すようなキャッシュの設定に使われます。したがって、CCR の修正は非キャッシュの P2 領域のプログラムのみで行わなければなりません。

- IIX：IC インデックス有効ビット
  - 0：アドレス [ 12：5 ] が IC のエントリ選択に使われる
  - 1：アドレス [ 25 ]、[ 11：5 ] が IC のエントリ選択に使われる
- ICI：IC 無効化ビット
 

このビットに 1 を書き込むと IC の全エントリの V ビットを 0 にします。読み出すと常に 0 が読めます。

- ICE : IC 有効ビット  
ICを使用するかどうかを示します。ただし、アドレス変換が行われる場合はページ管理情報のCビットも1でなければICを使用できません。
  - 0 : IC を使用しない
  - 1 : IC を使用する
- OIX : OC インデックス有効ビット
  - 0 : アドレス [ 13 : 5 ] が OC のエントリ選択に使われる
  - 1 : アドレス [ 25 ]、[ 12 : 5 ] が OC のエントリ選択に使われる
- ORA : OC RAM ビット  
OCが有効 ( OCE = 1 ) のとき、OCのエントリ128 ~ 255と384 ~ 511の8kバイトをRAMとして使用するかどうかを指定します。OCが有効でない ( OCE = 0 ) ときは、ORAビットは0に設定してください。
  - 0 : 16k バイトをキャッシュとして使用
  - 1 : 8k バイトをキャッシュ、8k バイトを RAM として使用
- OCI : OC 無効化ビット  
このビットに1を書き込むとOCの全エントリのV、Uビットを0にします。読み出すと常に0が読めます。
- CB:コピーバックビット  
P1領域のキャッシュへの書き込みモードを示します。
  - 0 : ライトスルーモード
  - 1 : コピーバックモード
- WT : ライトスルービット  
P0、U0、P3領域のキャッシュへの書き込みモードを示します。  
ただし、アドレス変換が行われる場合はページ管理情報のWTビットの値を優先します。
  - 0 : コピーバックモード
  - 1 : ライトスルーモード
- OCE : OC 有効ビット  
OCを使用するかどうかを示します。ただしアドレス変換が行われる場合はページ管理情報のCビットも1でなければOCを使用できません。
  - 0 : OC を使用しない
  - 1 : OC を使用する

## (2) キューアドレス制御レジスタ 0 ( QACR0 )

QACR0 へのロングワードアクセスはP4領域のH'FF00 0038 とエリア7のH'1F00 0038 から実行されます。QACR0 はMMU がオフのとき、ストアキュー 0 ( SQ0 ) がマップされているエリアを設定します。

(3) キューアドレス制御レジスタ 1 (QACR1)

QACR1 へのロングワードアクセスは P4 領域の H'FF00 003C とエリア 7 の H'1F00 003C から実行されます。QACR1 は MMU がオフのとき、ストアキュー 1 (SQ1) がマップされているエリアを設定します。

### 4.3 オペランドキャッシュ (OC)

### 4.3.1 構成

図 4.2 にオペランドキャッシュの構成を示します。

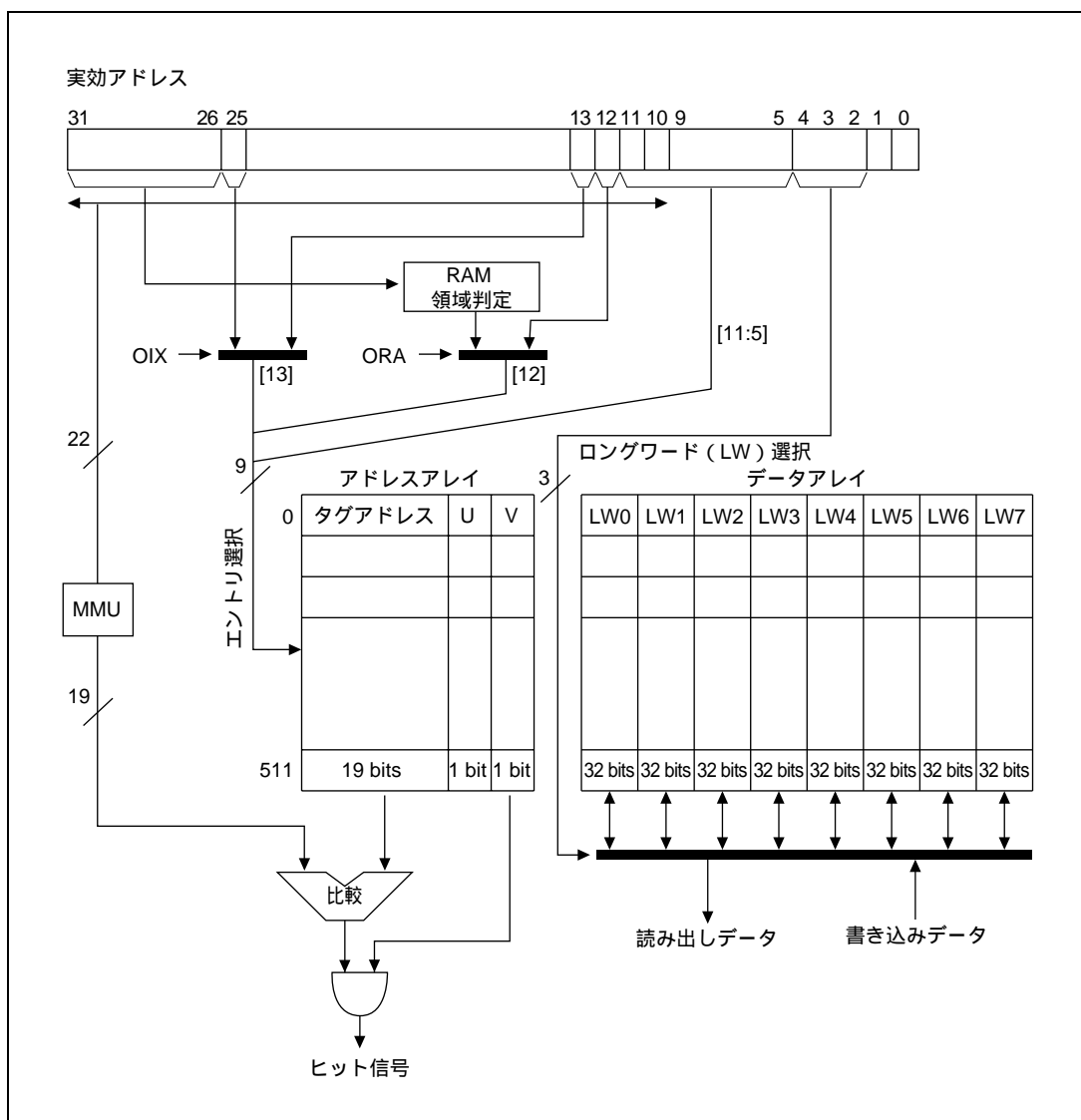


図 4.2 オペランドキャッシュの構造

オペランドキャッシュは 512 本のキャッシュラインから構成され、それぞれのラインは 19 ビットのタグ、V ビット、U ビットおよび 32 バイトのデータから成ります。

(1) タグ

キャッシュされるデータラインの外部メモリアドレス 29 ビットの上位 19 ビットを格納します。タグはパワーオンリセット、マニュアルリセットで初期化されません。

(2) V ビット (有効ビット)

キャッシュラインに有効なデータが格納されているかを示します。このビットが 1 のとき、そのキャッシュラインのデータは有効となります。V ビットはパワーオンリセットで 0 に初期化されますが、マニュアルリセットでは値を保持します。

(3) U ビット (ダーティビット)

コピーバックモードでキャッシュを使用中に、キャッシュラインヘータを書き込んだとき、U ビットが 1 になります。つまり U ビットはキャッシュライン中のデータと外部メモリ中のデータとの不一致を示します。メモリ割り付けキャッシュ (「4.5 メモリ割り付けキャッシュの構成」参照) をアクセスすることにより U ビットを書き換えない限り、ライトスルーモードでキャッシュを使用中は U ビットが 1 になることはありません。U ビットはパワーオンリセットで 0 に初期化されますが、マニュアルリセットでは値を保持します。

(4) データ部

データ部には 1 キャッシュラインあたり 32 バイト (256 ビット) のデータが格納されます。データレイはパワーオンリセット、マニュアルリセットで初期化されません。

### 4.3.2 リード動作

OC が有効 (CCR.OCE = 1) かつキャッシング可能な領域から実効アドレスによってデータを読み出す場合、キャッシュは以下のように動作します。

- (1) 実効アドレスのビット [13 : 5] でインデックスされるキャッシュラインからタグと V ビットと U ビットを読み出します。
- (2) 実効アドレスを MMU により変換したアドレスのビット [28 : 10] とタグを比較し、
  - ・ タグが一致かつ V ビットが 1 の場合 (3A)
  - ・ タグが一致かつ V ビットが 0 の場合 (3B)
  - ・ タグが不一致かつ V ビットが 0 の場合 (3B)
  - ・ タグが不一致かつ V ビットが 1 かつ U ビットが 0 の場合 (3B)
  - ・ タグが不一致かつ V ビットが 1 かつ U ビットが 1 の場合 (3C)

(3A) キャッシュヒット

実効アドレスのビット [13 : 5] でインデックスされるキャッシュラインのデータ部から、実効アドレスのビット [4 : 0] でインデックスされるデータをアクセスサイズ (クワッドワード / ロングワード / ワード / バイト) に応じて読み出します。

(3B) キャッシュミス (書き戻しなし)

実効アドレスに対応する外部メモリ空間から、キャッシュラインヘータを読み込みます。データの読み込みは実効アドレスに対応するロングワードデータから順にラップアラウンド方式で行い、該当するデータがキャッシュへ到着した時点で、CPU へ読み出しデータを返します。残りのキャッシュ 1 ライン分のデータが読み込まれている間、CPU は次の処理



を実行することができます。キャッシュは1ライン分のデータの読み込みが完了した時点で、実効アドレスに対応するタグを登録し、Vビットに1を書き込みます。

(3C) キャッシュミス（書き戻しあり）

実効アドレスのビット[13:5]でインデックスされるキャッシュラインのタグとデータ部をライトバックバッファへ退避します。そして実効アドレスに対応する外部メモリ空間から、キャッシュラインへデータを読み込みます。データの読み込みは実効アドレスに対応するロングワードデータから順にラップアラウンド方式で行い、該当するデータがキャッシュへ到着した時点で、CPUへ読み出しデータを返します。残りのキャッシュ1ライン分のデータが読み込まれている間、CPUは次の処理を実行することができます。キャッシュは1ライン分のデータの読み込みが完了した時点で、実効アドレスに対応するタグを登録し、Vビットに1をUビットに0を書き込みます。その後ライトバックバッファのデータを外部メモリへ書き戻します。

### 4.3.3 ライト動作

OCが有効（CCR.OCE=1）かつキャッシング可能な領域に対し実効アドレスによってデータが書き込まれる場合、キャッシュは以下のように動作します。

(1) 実効アドレスのビット[13:5]でインデックスされるキャッシュラインからタグとVビットとUビットを読み出します。

(2) 実効アドレスをMMUにより変換したアドレスのビット[28:10]とタグを比較し、

	コピーバック	ライトスルー
・タグが一致かつVビットが1の場合	(3A)	(3B)
・タグが一致かつVビットが0の場合	(3C)	(3D)
・タグが不一致かつVビットが0の場合	(3C)	(3D)
・タグが不一致かつVビットが1かつUビットが0の場合	(3C)	(3D)
・タグが不一致かつVビットが1かつUビットが1の場合	(3E)	(3D)

(3A) キャッシュヒット（コピーバック）

実効アドレスのビット[13:5]でインデックスされるキャッシュラインのデータ部の実効アドレスのビット[4:0]でインデックスされるデータに対し、アクセスサイズ（クワッドワード/ロングワード/ワード/バイト）によりデータの書き込みを行います。そしてUビットに1を設定します。

(3B) キャッシュヒット（ライトスルー）

実効アドレスのビット[13:5]でインデックスされるキャッシュラインのデータ部の実効アドレスのビット[4:0]でインデックスされるデータに対し、アクセスサイズ（クワッドワード/ロングワード/ワード/バイト）によりデータの書き込みを行います。書き込みは指定されたアクセスサイズを用いた外部メモリと対応して実行します。

(3C) キャッシュミス（コピーバック、ライトバックなし）

実効アドレスのビット[13:5]でインデックスされるキャッシュラインのデータ部の実効アドレスのビット[4:0]でインデックスされるデータに対し、アクセスサイズ（クワッドワード/ロングワード/ワード/バイト）によりデータの書き込みを行います。そして実効アドレスに対応する外部メモリ空間から、キャッシュラインへデータを読み込みます。データの読み込みは実効アドレスに対応するロングワードデータから順にラップアラウンド方式で行い、書き込んだデータを除いたキャッシュ1ライン分のデータが読み込まれます。この間、CPUは次の処理を実行することができます。キャッシュは1ライン分のデータの読み込みが完了した時点で、実効アドレスに対応するタグを登録し、VビットとUビットに1を書き込みます。

## (3D) キャッシュミス（ライトスルー）

実効アドレスに対応した外部メモリへ、設定されたアクセスサイズのライトを行います。  
この場合、キャッシュへのライトは行われません。

## (3E) キャッシュミス（コピーバック、ライトバックあり）

実効アドレスのビット [13:5] でインデックスされるキャッシュラインのタグとデータ部をライトバックバッファへ退避した後、実効アドレスのビット [13:5] でインデックスされるキャッシュラインのデータ部の実効アドレスのビット [4:0] でインデックスされるデータに対し、アクセスサイズ（クワッドワード/ロングワード/ワード/バイト）によりデータの書き込みを行います。そして実効アドレスに対応する外部メモリ空間から、キャッシュラインへデータを読み込みます。データの読み込みは実効アドレスに対応するロングワードデータから順にラップアラウンド方式で行い、書き込んだデータを除いたキャッシュ 1 ライン分のデータが読み込まれます。この間、CPU は次の処理を実行することができます。キャッシュは 1 ライン分のデータの読み込みが完了した時点で、実効アドレスに対応するタグを登録し、VビットとUビットに 1 を書き込みます。その後ライトバックバッファのデータを外部メモリへ書き戻します。

## 4.3.4 ライトバックバッファ

キャッシュミスによりダーティなキャッシュのエントリを外部メモリに追い出す必要が生じた場合、キャッシュへのデータの読み込みを優先させ性能を向上させるために、追い出すキャッシュラインのデータを格納するためのライトバックバッファを SH7091 は内蔵しています。ライトバックバッファはキャッシュ 1 ライン分のデータと追い出す先の物理アドレスで構成されます。

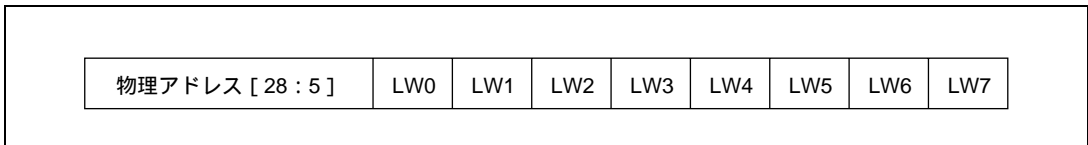


図 4.3 ライトバックバッファの構成

## 4.3.5 ライトスルーバッファ

ライトスルーモード時のデータの書き込みや、キャッシング不可能な領域に対する書き込み動作において、書き込みデータを保持するための 64 ビットのバッファを SH7091 は内蔵しています。これにより CPU はライトスルーバッファへの書き込みが完了すると、外部メモリへの書き込みの完了を待たずに次の動作へ移ります。

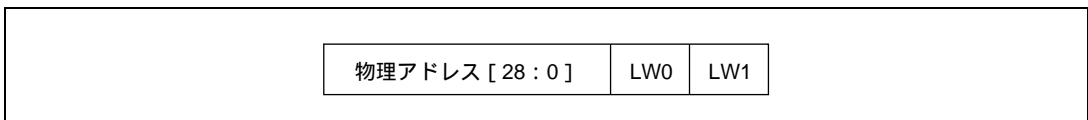


図 4.4 ライトスルーバッファの構成

## 4.3.6 RAM モード

CCR.ORA を 1 にセットすると、オペランドキャッシュの 8k バイトを RAM として使用することができます。RAM となるエントリは、オペランドキャッシュのエントリ 128 ~ 255 と 384 ~ 511 までです。それ以外のエントリはキャッシュとして利用できます。RAM へはアドレスの H'7C00 0000 ~ H'7FFF FFFF を用いてアクセスができます。オペランドキャッシュの RAM 領域へはバイト/ワード/ロングワード/クワッドワードサイズのデータの読み出し/書き込みが可能です。この領域に対して命令フェッチは行えません。

## 4. キャッシュ

RAMの使用例を以下に示します。ここではOC エントリ 128～255 の4kB をRAM 領域1 とし、OC エントリ 384～511 までの4kB をRAM 領域2 とします。

- OC インデックスモードがオフの場合 (CCR.OIX = 0)
  - H'7C00 0000 ~ H'7C00 0FFF (4kB) : RAM領域1に対応
  - H'7C00 1000 ~ H'7C00 1FFF (4kB) : RAM領域1に対応
  - H'7C00 2000 ~ H'7C00 2FFF (4kB) : RAM領域2に対応
  - H'7C00 3000 ~ H'7C00 3FFF (4kB) : RAM領域2に対応
  - H'7C00 4000 ~ H'7C00 4FFF (4kB) : RAM領域1に対応
  - : : :

以下H'7FFF FFFFまでのRAM領域1、2が8kB置きに繰り返し現れます。

このため連続した8kBのRAM領域を確保する場合、たとえば、H'7C00 1000 ~ H'7C00 2FFF の領域を用います。

- OC インデックスモードがオンの場合 (CCR.OIX = 1)
  - H'7C00 0000 ~ H'7C00 0FFF (4kB) : RAM領域1に対応
  - H'7C00 1000 ~ H'7C00 1FFF (4kB) : RAM領域1に対応
  - H'7C00 2000 ~ H'7C00 2FFF (4kB) : RAM領域1に対応
  - : : :
  - H'7DFF F000 ~ H'7DFF FFFF (4kB) : RAM領域1に対応
  - H'7E00 0000 ~ H'7E00 0FFF (4kB) : RAM領域2に対応
  - H'7E00 1000 ~ H'7E00 1FFF (4kB) : RAM領域2に対応
  - : : :
  - H'7FFF F000 ~ H'7FFF FFFF (4kB) : RAM領域2に対応

RAM領域1、2の区別はアドレス [ 25 ] で行われるため、連続した8kBのRAM領域の確保はH'7DFF F000 ~ H'7E00 0FFFの領域で行ってください。

### 4.3.7 OC インデックスモード

CCR.OIX を 1 にセットすると、実効アドレスの [ 25 ] を用いて OC のインデックスを実行することができます。これを OC インデックスモードと呼びます。通常モードでは CCR.OIX が 0 の状態で、実効アドレスの [ 13 : 5 ] を用いて OC のインデックスを実行します。したがって、連続した 16 k バイト以上のデータを処理する場合、このデータは OC のすべてを利用します。この結果、キャッシュミスが頻発するようになります。インデックスモードを使用すると実効アドレスの [ 25 ] により OC を 2 つの 8k バイト領域として処理することができ、キャッシュの効率的な利用が可能です。

### 4.3.8 キャッシュと外部メモリとのコヒーレンシ

キャッシュと外部メモリとのコヒーレンシはソフトウェアで保証してください。SH7091 ではキャッシュを操作する命令として新たに次の 4 命令をサポートしています。各命令の詳細は「10 章 各命令の説明」を参照してください。

- |             |                  |                      |
|-------------|------------------|----------------------|
| • インバリデイト命令 | : OCBI @Rn       | : キャッシュの無効化 (書き戻しなし) |
| • パージ命令     | : OCBP @Rn       | : キャッシュの無効化 (書き戻しあり) |
| • ライトバック命令  | : OCBWB @Rn      | : キャッシュの書き戻し         |
| • アロケート命令   | : MOVCA.L R0,@Rn | : キャッシュの確保           |

### 4.3.9 プリフェッチ動作

キャッシュミスにより発生するキャッシュフィルのペナルティを削減するために、SH7091 ではプリフェッチ命令をサポートしています。リード動作、ライト動作によりキャッシュミスの発生することがわかっていた場合、プリフェッチ命令によりあらかじめキャッシュヘデータをフィルしておく、リード動作、ライト動作においてキャッシュミスを発生させないようにできます。これによりソフトウェアの性能が向上します。すでにキャッシュに格納されているデータに対して、プリフェッチ命令を実行した場合やプリフェッチしようとしたアドレスで MMU 例外を起こした場合、ノーオペレーションとなり例外を発生させません。プリフェッチ命令の詳細は「10.73 PREF」を参照してください。

- プリフェッチ命令 : PREF @Rn

## 4.4 命令キャッシュ (IC)

### 4.4.1 構成

図 4.5 に命令キャッシュの構成を示します。

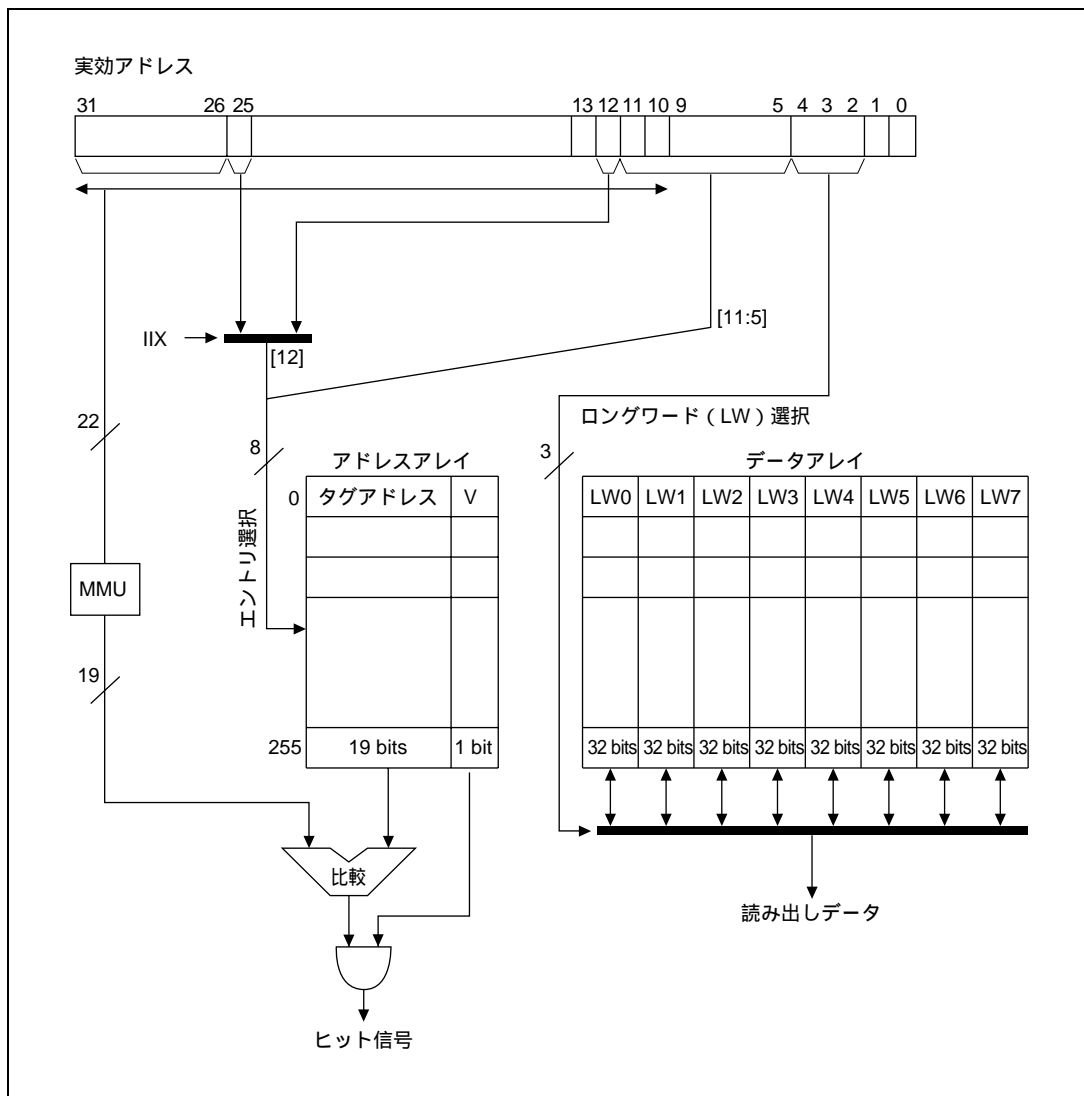


図 4.5 命令キャッシュの構成

命令キャッシュは 256 本のキャッシュラインから構成され、それぞれのラインは 19 ビットのタグ、V ビット、および 32 バイトのデータ（16 命令）から成ります。

#### (1) タグ

キャッシュされるデータラインの外部メモリアドレス 29 ビットの上位 19 ビットを格納します。タグはパワーオンリセット、マニュアルリセットで初期化されません。

- (2) V ビット (有効ビット)  
キャッシュラインに有効なデータが格納されているかを示します。このビットが1のとき、そのキャッシュラインのデータは有効となります。Vビットはパワーオンリセットで0に初期化されますが、マニュアルリセットでは値を保持します。
- (3) データアレイ  
データ部には1 キャッシュラインあたり32バイト (256ビット) のデータが格納されます。データアレイはパワーオンリセット、マニュアルリセットで初期化されません。

#### 4.4.2 リード動作

IC が有効 (CCR.ICE = 1) かつキャッシング可能な領域から実効アドレスによって命令フェッチを行う場合、命令キャッシュは以下のように動作します。

- (1) 実効アドレスのビット [12 : 5] でインデックスされるキャッシュラインからタグと V ビットを読み出します。
- (2) 実効アドレスを MMU により変換したアドレスのビット [28 : 10] とタグを比較し、
  - ・ タグが一致かつ V ビットが1の場合 (3A)
  - ・ タグが一致かつ V ビットが0の場合 (3B)
  - ・ タグが不一致かつ V ビットが0の場合 (3B)
  - ・ タグが不一致かつ V ビットが1の場合 (3B)
- (3A) キャッシュヒット  
実効アドレスのビット [12 : 5] でインデックスされるキャッシュラインのデータ部から、実効アドレスのビット [4 : 2] でインデックスされるデータを命令として読み出します。
- (3B) キャッシュミス  
実効アドレスに対応する外部メモリ空間から、キャッシュラインヘデータを読み込みます。データの読み込みは実効アドレスに対応するロングワードデータから順にラップアラウンド方式で行い、該当するデータがキャッシュへ到着した時点で、CPUへ読み出しデータを命令として返します。残りのキャッシュ1ライン分のデータが読み込まれている間、CPUは次の処理を実行することができます。キャッシュは1ライン分のデータの読み込みが完了した時点で、実効アドレスに対応するタグを登録し、Vビットに1を書き込みます。

#### 4.4.3 IC インデックスモード

CCR.IIX を1にセットすると、実効アドレスの [25] を用いて IC のインデックスを実行することができます。これを IC インデックスモードと呼びます。通常モードでは CCR.IIX が0の状態、実効アドレスの [12 : 5] を用いて IC のインデックスを実行します。したがって、連続した 8k バイト以上のプログラムを処理する場合、このプログラムは IC のすべてを利用します。この結果、キャッシュミスが頻発するようになります。インデックスモードを使用すると有効アドレスの [25] により IC を2つの 4k バイト領域として処理することができ、キャッシュの効率的な利用が可能です。

### 4.5 メモリ割り付けキャッシュの構成

IC、OCをソフトウェアで管理するために、特権モードのとき、P2領域のプログラムからMOV命令によってIC、OCの内容の読み出し／書き込みが可能です。他の領域のプログラムからのアクセスは保証しません。この場合、P0、U0、P1、P3領域への分岐命令はこのMOV命令の8命令以降に実行するようにしてください。IC、OCは物理メモリ空間のP4領域に割り付けられています。ICのアドレスアレイ／データアレイ、OCのアドレスアレイ／データアレイともにデータアクセスのみ可能でアクセスサイズはロングワード固定です。この領域に対して命令フェッチは行えません。予約ビットには0を設定するようにしてください。予約ビットの読み出し値は不定です。

#### 4.5.1 IC アドレスアレイ

ICのアドレスアレイはP4領域のH'F000 0000～H'FOFF FFFFに割り付けられています。アドレスアレイのアクセスには32ビットのアドレス部の指定（読み出し／書き込み時）と32ビットのデータ部の指定が必要です。アドレス部ではアクセスするエントリを指定し、データ部には書き込みタグとVビットを指定します。

アドレス部は[31:24]がICアドレスアレイを示すH'F0になっており、[12:5]でエントリを指定するようになっています。CCR.IIXはこのエントリ指定に影響を与えません。アドレス部[3]の連想ビット（Aビット）はICアドレスアレイへの書き込みのときに連想を行うかどうかを指定します。アクセスはロングワードサイズ固定なのでアドレス部[1:0]は0を指定してください。

データ部は[31:10]がタグを、[0]がVビットを示します。ICアドレスアレイのタグは19ビットのためデータ部[31:29]は連想を行わない書き込みのときには使用されません。データ部[31:29]は連想を行う書き込みのときのみ仮想アドレスの指定のため用います。

ICアドレスアレイに対しては次の3種類の操作が可能です。

##### (1) IC アドレスアレイ リード

アドレス部に設定されたエントリに対応するICエントリから、データ部へタグとVビットを読み出します。リードの場合アドレス部に指定される連想ビットは1でも0でも連想動作は行いません。

##### (2) IC アドレスアレイ ライト（連想なし）

アドレス部に設定されたエントリに対応するICエントリに対して、データ部で指定されたタグとVビットを書き込みます。アドレス部のAビットは0にしてください。

##### (3) IC アドレスアレイ ライト（連想あり）

アドレス部のAビットが1でライトのとき、アドレス部で指定されたエントリに格納されているタグとデータ部で指定されたタグとの間で一致判定が行われます。このときMMUがイネーブルなら、データ部[31:10]で指定した仮想アドレスをITLBを用い物理アドレスに変換してから一致判定を行います。アドレスが一致しVビットが1であったなら、データ部で指定したVビットをICのエントリに書き込みます。本動作はICの特定のエントリの無効化に用いられます。アドレス変換の際に命令TLBミス例外が発生したり、一致判定で不一致になった場合、ノーオペレーションとなり書き込みは行われません。アドレス変換の際に命令TLB多重ヒット例外が発生した場合は、命令TLB多重ヒット例外処理ルーチンへ処理が移ります。

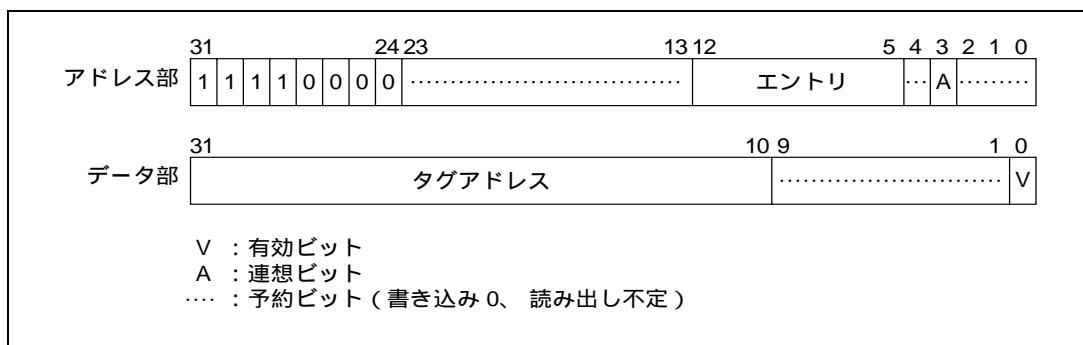


図 4.6 メモリ割り付け IC アドレスアレイ

### 4.5.2 IC データアレイ

IC のデータアレイは P4 領域の H'F100 0000 ~ H'F1FF FFFF に割り付けられています。データアレイのアクセスには 32 ビットのアドレス部の指定 (読み出し / 書き込み時) と 32 ビットのデータ部の指定が必要です。アドレス部ではアクセスするエントリを指定し、データ部には書き込むロングワードデータを指定します。

アドレス部は [31 : 24] が IC データアレイを示す H'F1 になっており、[12 : 5] でエントリを指定するようになっています。CCR.IIX はこのエントリ指定に影響を与えません。アドレス部 [4 : 2] はエントリ内のロングワードデータの指定に用います。アクセスはロングワードサイズ固定なのでアドレス部 [1 : 0] は 0 を指定してください。

データ部はロングワードデータの指定に用います。

IC データアレイに対しては次の 2 種類の操作が可能です。

#### (1) IC データアレイ リード

アドレス部に設定されたエントリに対応する IC エントリのうち、アドレス部のロングワード指定ビットで指定されたデータから、データ部へロングワードデータを読み出します。

#### (2) IC データアレイ ライト

アドレス部に設定されたエントリに対応する IC エントリのうち、アドレス部のロングワード指定ビットで指定されたデータに対して、データ部で指定されたロングワードデータを書き込みます。

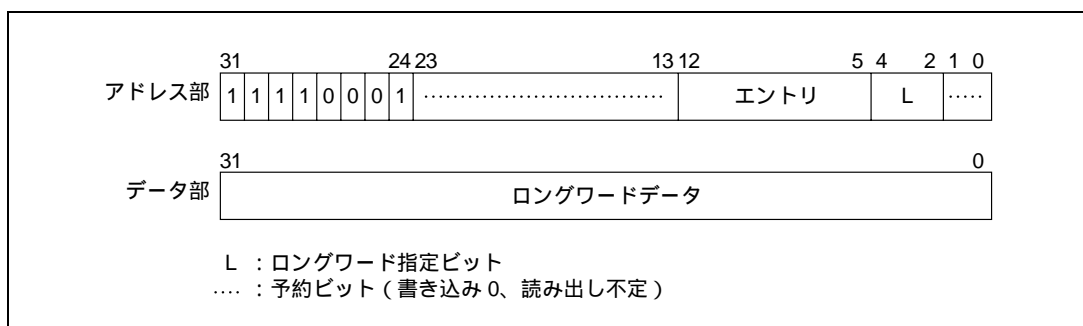


図 4.7 メモリ割り付け IC データアレイ



### 4.5.3 OC アドレスアレイ

OC のアドレスアレイは P4 領域の H'F400 0000 ~ H'F4FF FFFF に割り付けられています。アドレスアレイのアクセスには 32 ビットのアドレス部の指定（読み出し / 書き込み時）と 32 ビットのデータ部の指定が必要です。アドレス部ではアクセスするエントリを指定し、データ部には書き込みタグと U ビットと V ビットを指定します。

アドレス部は [31 : 24] が OC アドレスアレイを示す H'F4 になっており、[13 : 5] でエントリを指定するようになっていきます。CCR.OIX および CCR.ORA はこのエントリ指定に影響を与えません。アドレス部 [3] の連想ビット（A ビット）は OC アドレスアレイへの書き込みのときに連想を行うかどうかを指定します。アクセスはロングワードサイズ固定ですのでアドレス部 [1 : 0] は 0 を指定してください。

データ部は [31 : 10] がタグを、[1] が U ビットを、[0] が V ビットを示します。OC アドレスアレイのタグは 19 ビットのため、データ部 [31 : 29] は連想を行わない書き込みのときには使用されません。データ部 [31 : 29] は連想を行う書き込みのときのみ仮想アドレスの指定のため用います。

OC アドレスアレイに対しては次の 3 種類の操作が可能です。

#### (1) OC アドレスアレイ リード

アドレス部に設定されたエントリに対応する OC エントリから、データ部へタグと U ビットと V ビットを読み出します。リードの場合、アドレス部に指定される連想ビットは 1 でも 0 でも連想動作は行いません。

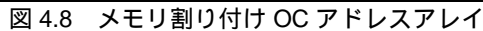
#### (2) OC アドレスアレイ ライト（連想なし）

アドレス部に設定されたエントリに対応する OC エントリに対して、データ部で指定されたタグと U ビットと V ビットを書き込みます。アドレス部の A ビットは 0 にしてください。

書き込みを U ビットが 1、V ビットが 1 のキャッシュラインに対して行った場合、そのキャッシュラインの書き戻しを行った後、データ部で指定されたタグと U ビットと V ビットを書き込みます。

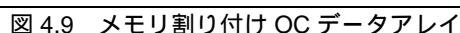
#### (3) OC アドレスアレイ ライト（連想あり）

アドレス部の A ビットが 1 でライトのとき、アドレス部で指定されたエントリに格納されているタグとデータ部で指定されたタグとの間で一致判定が行われます。このとき MMU がイネーブルなら、データ部 [31 : 10] で指定した仮想アドレスを UTLB を用い物理アドレスに変換してから一致判定を行います。アドレスが一致し V ビットが 1 であったなら、データ部で指定した U ビットと V ビットを OC のエントリに書き込みます。本動作は OC の特定のエントリの無効化に用いられます。このとき OC のエントリの U ビットが 1 で、V ビットに 0 もしくは U ビットに 0 を書き込んだ場合、書き戻しが発生します。アドレス変換の際にデータ TLB ミス例外が発生したり、一致判定で不一致になった場合、ノーオペレーションとなり書き込みは行われません。アドレス変換の際にデータ TLB 多重ヒット例外が発生した場合はデータ TLB 多重ヒット例外処理ルーチンへ処理が移ります。



OC データレイに対しては次の 2 種類の操作が可能です。

アドレス部に設定されたエントリに対応するOCエントリのうち、アドレス部のロングワード指定ビットで指定されたデータに対して、データ部で指定されたロングワードデータを書き込みます。この書き込みによりアドレスアレイ側のUビットは1になりません。



## 4.6 ストアキュー

外部メモリへの高速な書き込みを行うために 32 バイト×2 のストアキュー (SQ) をサポートします。

### 4.6.1 SQ の構成

SQ は図 4.10 に示す通り、32 バイトの SQ0 と 32 バイトの SQ1 から成り立っています。SQ0、1 はそれぞれ独立に設定することが可能です。

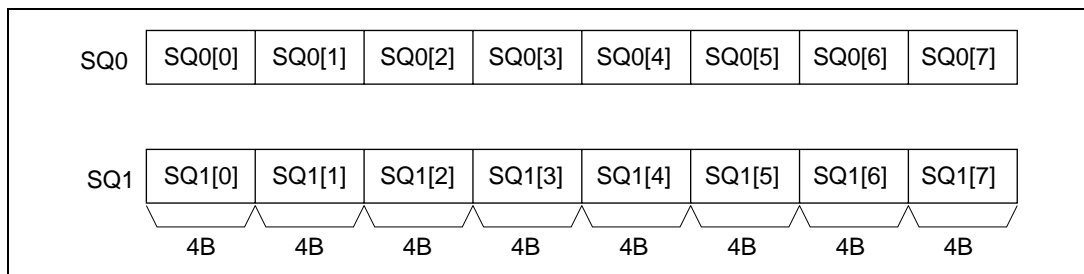


図 4.10 ストアキューの構成

### 4.6.2 SQ への書き込み

SQ への書き込みは P4 領域の H'E000 0000 ~ H'E3FF FFFC に対するストア命令 (MOV) で行うことができます。アクセスサイズはロングワード、もしくはクワッドワードが可能です。このアドレスは以下の意味を持ちます。

[31:26]	: 111000	: ストアキュー指定
[25:6]	: Don't care	: 外部メモリへの転送・アクセス権で使用
[5]	: 0/1	: 0:SQ0 指定 1:SQ1 指定
[4:2]	: LW 指定	: SQ0、SQ1 内のロングワード位置を指定
[1:0]	: 00	: 0 固定

### 4.6.3 外部メモリへの転送

SQ から外部メモリへの転送は、プリフェッチ命令 (PREF) により行えます。PREF 命令を P4 領域の H'E000 0000 ~ H'E3FF FFFC に対して発行することにより、SQ から外部メモリへのバースト転送が開始します。バースト転送は 32 バイト固定で、開始アドレスは必ず 32 バイト境界となります。一方の SQ を外部メモリへ転送中に、もう一方の SQ への書き込みはペナルティサイクルなしに行えますが、外部メモリへ転送中の SQ への書き込みは外部メモリへの転送が完了するまで待たされます。

SQ の転送先の外部メモリアドレス[28:0]は MMU オン / オフにより次のように指定します。

#### (1) MMU オン

UTLBのVPNにSQ領域 (H'E000 0000 ~ H'E3FF FFFF) を、PPNに転送先の外部メモリアドレスを設定します。ASID、V、SZ、SH、PR、Dビットは通常のアドレス変換と同様の意味を持ちますが、C、WTビットはこのページに関しては意味を持ちません。なおPCMCIA領域に対するバースト転送も禁止のため、SA、TCビットも意味を持ちません。

SQ領域へのプリフェッチ命令が発行されると、アドレス変換を行い、SZビットの指定に従い外部メモリアドレス[28:10]を生成します。外部メモリアドレスの[9:5]についてはMMUオフと同様にアドレス変換前のアドレスから生成します。外部メモリアドレスの[4:0]は 0 固定です。SQから外部メモリへの転送はこのアドレスに対して行われます。

#### (2) MMU オフ

プリフェッチを行うアドレスにSQ領域 (H'E000 0000 ~ H'E3FF FFFF) を指定します。このアドレス[31:0]は次の意味を持ちます。

[31:26]	: 111000	: ストアキュー指定
[25:6]	: アドレス	: 外部メモリアドレス[25:6]
[5]	: 0/1	: 0:SQ0指定 1:SQ1指定 かつ 外部メモリアドレス[5]
[4:2]	: Don't care	: プリフェッチのときは意味を持たない。
[1:0]	: 00	: 0 固定

上記のアドレスから生成できない外部メモリアドレス[28:26]は、QACR0、1レジスタから生成します。

QACR0[4:2]	: SQ0に対する外部メモリアドレス[28:26]
QACR1[4:2]	: SQ1に対する外部メモリアドレス[28:26]

外部メモリアドレスの[4:0]は、バースト転送の開始が32バイト境界のため常に 0 固定となります。

### 4.6.4 SQ へのプロテクション

SQ への書き込み、および外部メモリへの転送に対してプロテクションを設定することができます。SQ への書き込みがプロテクションに違反した場合、例外は発生しますが、SQ の内容は壊されます。SQ から外部メモリへの転送 (プリフェッチ命令) がプロテクションに違反した場合、外部メモリへの転送は抑止され、例外が発生します。

(1) MMU オンの場合

UTLBに登録されたアドレス変換情報とMMUCR.SQMDに従います。SQへの書き込みはライトタイプ、SQから外部メモリへの転送(PREF命令)はリードタイプとして例外判定が行われ、TLBミス例外、保護違反例外、初期ページ書き込み例外が発生します。ただし、MMUCR.SQMDによりSQへのアクセスを特権モードのみ許可している場合、ユーザモードでアドレス変換に成功してもアドレスエラーとなります。

(2) MMU オフの場合

MMUCR.SQMDに従います。

0：特権 / ユーザアクセス可能

1：特権アクセス可能

MMUCR.SQMDが1のときに、ユーザモードでSQ領域をアクセスするとアドレスエラーが発生します。

---

## 5. 例外処理

---

### 5.1 概要

#### 5.1.1 特長

例外処理とは、リセット、一般例外、割り込みが検出されたときに、通常とは異なるプログラムで必要な処理を行うことをいいます。例えば、実行中の命令の異常終了が発生した場合、適切な処置をすることで、元のプログラムに復帰したり、異常を報告して終了するといった制御が必要になります。このような機能をサポートするために、異常終了に対して、例外処理要求を発生させ、ユーザが作成した例外処理ルーチンに制御の流れが渡ることなどを総称して例外処理と呼びます。

SH7091 の例外処理は、リセット、一般例外、割り込みの 3 つに分類されます。

#### 5.1.2 レジスタ構成

例外処理に関するレジスタ構成を表 5.1 に示します。

表 5.1 レジスタ構成（アドレス）

名称	略称	R/W	初期値	P4 アドレス*2	エリア 7 アドレス*2	アクセス サイズ
TRAPA 例外レジスタ	TRA	R/W	不定	H'FF00 0020	H'1F00 0020	32
例外事象レジスタ	EXPEVT	R/W	H'0000 0000/ H'0000 0020*1	H'FF00 0024	H'1F00 0024	32
割り込み事象レジスタ	INTEVT	R/W	不定	H'FF00 0028	H'1F00 0028	32

【注】 \*1 パワーオンリセット時に H'0000 0000、マニュアルリセット時に H'0000 0020 がセットされます。

\*2 P4 アドレスは仮想 / 物理アドレス空間の P4 領域を用いた場合のものです。TLB を用いて物理アドレス空間のエリア 7 からアクセスする場合、アドレスの上位 3 ビットが無視されます。

## 5.2 レジスタの説明

例外処理に関するレジスタは、3 本あります。これらはメモリ上に割り付けられており、P4 アドレスまたはエリア 7 アドレスを指定することでアクセスできます。

- (1) 例外事象レジスタ (EXPEVT) は、P4 アドレス H'FF00 0024 番地に配置されていて、例外コード 12 ビットから構成されています。EXPEVT に設定される例外コードは、リセットと一般例外事象による例外コードです。例外コードは例外発生時にハードウェアにより自動的に設定されます。EXPEVT はソフトウェアからも変更が可能です。
- (2) 割り込み事象レジスタ (INTEVT) は、P4 アドレス H'FF00 0028 番地に配置されていて、例外コード 12 ビットから構成されています。INTEVT に設定される例外コードは、割り込み要求による例外コードです。例外コードは例外発生時にハードウェアにより自動的に設定されます。INTEVT はソフトウェアからも変更が可能です。
- (3) TRAPA 例外レジスタ (TRA) は、P4 アドレス H'FF00 0020 番地に配置されていて、TRAPA 命令の 8 ビットイミディエイトデータ (imm) から構成されています。TRA は TRAPA 命令実行時にハードウェアにより自動的に設定されます。TRA はソフトウェアからも変更が可能です。

EXPEVT、INTEVT、TRA のビット構成を図 5.1 に示します。

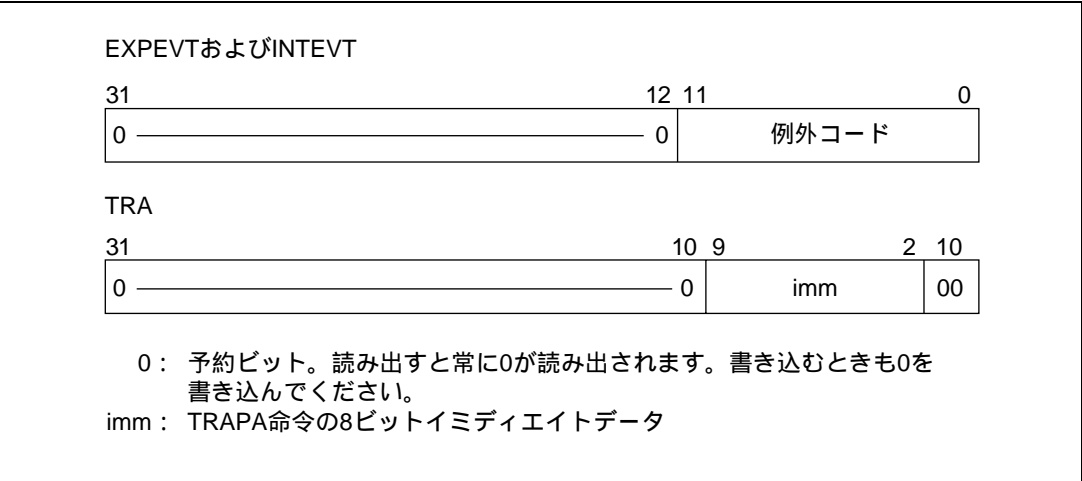


図 5.1 レジスタのビット構成

## 5.3 例外処理の機能

### 5.3.1 例外処理の流れ

例外処理では、プログラムカウンタ (PC)、ステータスレジスタ (SR) の内容がそれぞれ退避プログラムカウンタ (SPC)、退避ステータスレジスタ (SSR) に退避され、ベクタアドレスに従って対応する例外処理ルーチンの実行を開始します。例外処理ルーチンとは、ユーザによって、個々の例外の内容に応じて作成されたプログラムです。例外処理ルーチンを終了させ、元のプログラムに戻るためには、例外処理からの復帰命令 (RTE) を実行します。本命令によって、PC と SR の内容が復帰し、例外などが発生した時点での通常処理ルーチンに戻ることができます。

基本的な例外処理の流れは次のようになります。SR のビットの意味の詳細は、「2 章 プログラミングモデル」を参照してください。

- (1) PC と SR の内容がそれぞれ SPC と SSR に退避されます。
- (2) SR のブロックビット (BL) が 1 に設定されます。
- (3) SR のモードビット (MD) が 1 に設定されます。
- (4) SR のレジスタバンクビット (RB) が 1 に設定されます。
- (5) リセット時、SR の FPU ディスエーブルビット (FD) が 0 に設定されます。
- (6) 例外コードは、例外要因の例外事象レジスタ (EXPEVT)、または、割り込み事象レジスタ (INTEVT) のビット 11 ~ 0 に書き込まれます。
- (7)決められた例外処理のベクタアドレスに分岐して、例外処理ルーチンを開始します。

### 5.3.2 例外処理ベクタアドレス

リセットベクタアドレスは H'A000 0000 に固定されています。例外、割り込みのベクタアドレスはベクタベースアドレスに各事象のオフセットの値を加えたアドレスです。ベクタベースアドレスはベクタベースレジスタ (VBR) にソフトウェアで設定します。例えば、TLB ミス例外のオフセットは H'0000 0400 ですから、VBR に H'9C08 0000 を設定しておくと、例外処理ベクタアドレスは H'9C08 0400 になります。例外処理ベクタアドレスでさらに例外が発生すると、2 重例外となり、回復が困難になりますので、ベクタアドレスは固定物理アドレス (P1、P2) を指定してください。



## 5. 例外処理

### 5.4 例外の種類と優先順位

表 5.2 に、例外の種類、優先順位、ベクタアドレス、および例外 / 割り込みコードを示します。

表 5.2 例外一覧 (1)

例外区分	実行形態	例外	優先 レベル	優先 順位	ベクタベース	オフセット	例外コード
リセット	中断型	パワーオンリセット	1	1	H'A000 0000	-	H'000
		マニュアルリセット	1	2	H'A000 0000	-	H'020
		Hitachi-UDI リセット	1	1	H'A000 0000	-	H'000
		命令 TLB 多重ヒット例外	1	3	H'A000 0000	-	H'140
		データ TLB 多重ヒット例外	1	4	H'A000 0000	-	H'140
一般例外	再実行型	命令実行前ユーザブレイク *1	2	0	(VBR/DBR)	H'100/ -	H'1E0
		命令アドレスエラー	2	1	(VBR)	H'100	H'0E0
		命令 TLB ミス例外	2	2	(VBR)	H'400	H'040
		命令 TLB 保護違反例外	2	3	(VBR)	H'100	H'0A0
		一般不当命令例外	2	4	(VBR)	H'100	H'180
		スロット不当命令例外	2	4	(VBR)	H'100	H'1A0
		一般 FPU 抑止例外	2	4	(VBR)	H'100	H'800
		スロット FPU 抑止例外	2	4	(VBR)	H'100	H'820
		データアドレスエラー (読み出し)	2	5	(VBR)	H'100	H'0E0
		データアドレスエラー (書き込み)	2	5	(VBR)	H'100	H'100
		データ TLB ミス例外 (読み出し)	2	6	(VBR)	H'400	H'040
		データ TLB ミス例外 (書き込み)	2	6	(VBR)	H'400	H'060
		データ TLB 保護違反例外 (読み出し)	2	7	(VBR)	H'100	H'0A0
		データ TLB 保護違反例外 (書き込み)	2	7	(VBR)	H'100	H'0C0
		FPU 例外	2	8	(VBR)	H'100	H'120
		初期ページ書き込み例外	2	9	(VBR)	H'100	H'080
	完了型	無条件トラップ (TRAPA)	2	4	(VBR)	H'100	H'160
		命令実行後ユーザブレイク *1	2	10	(VBR/DBR)	H'100/ -	H'1E0
割り込み	完了型	ノンマスカブル割り込み	3	-	(VBR)	H'600	H'1C0
		外部割り込み IRL3 ~ 0	4	*2	(VBR)	H'600	H'200
							H'220
							H'240
							H'260
							H'280
							H'2A0
							H'2C0
							H'2E0
							H'300
							H'320
							H'340
							H'360
							H'380
							H'3A0
							H'3C0

表 5.2 例外一覧 (2)

例外区分	実行形態	例外		優先 レベル	優先 順位	ベクタベース	オフセット	例外コード	
割り込み	完了型	周辺モジュール 割込み (モジュール/要因)	TMU0	TUNI0	4	*2	( VBR )	H'600	H'400
			TMU1	TUNI1					H'420
			TMU2	TUNI2					H'440
				TICPI2					H'460
			RTC	ATI					H'480
				PRI					H'4A0
				CUI					H'4C0
			SCI	ERI					H'4E0
			SCI	RXI					H'500
				TXI					H'520
				TEI					H'540
			WDT	ITI					H'560
			REF	RCMI					H'580
				ROVI					H'5A0
			Hitachi- UDI	Hitachi- UDI					H'600
			DMAC	DMTE0					H'640
				DMTE1					H'660
				DMTE2					H'680
				DMTE3					H'6A0
				DMAE					H'6C0
			SCIF	ERI					H'700
				RXI					H'720
				BRI					H'740
				TXI					H'760

優先度：           まず優先レベルで順位付けし、同一レベル内を優先順位で順位付けします（より小さい数値が優先度が高くなります）。

例外遷移先：       リセットでは H'A000 0000、その他では (VBR+オフセット) へ制御が移ります。

例外コード：       リセット、一般例外では EXPEVT、割込みでは INTEVT に格納されます。

IRL：               割込み要求レベル (IRL3~0 端子)

モジュール/要因：   ハードウェアマニュアルの各周辺モジュールの章を参照してください。

【注】 \*1   BR CR.UBDE=1 のとき PC=DBR。その他は PC=VBR+H'100

\*2   外部割込みおよび周辺モジュール割込みの優先順位はソフトウェアによって設定可能です。

## 5.5 例外フロー

### 5.5.1 例外フロー

図 5.2 に、命令実行と例外処理の基本動作を概念的に示します。ここでは説明の都合上、命令を 1 命令ずつ逐次的に実行することを基本として説明しています。図 5.2 には、例外種別（リセット、一般例外、割り込み）間の優先順位が表されています。なお図 5.2 では、例外成立時のレジスタ設定を SSR、SPC、EXPEVT/INTEVT、SR、および PC に限っていますが、例外によってはこの他にもハードウェアによって自動的に設定されるレジスタがあります。詳細は、「5.6 各例外の説明」を参照してください。また、遅延分岐命令と遅延スロット命令を実行中の例外処理や、2 回データアクセスが発生する命令については「5.6.4 複数回の例外が発生する場合の優先順位」を参照してください。

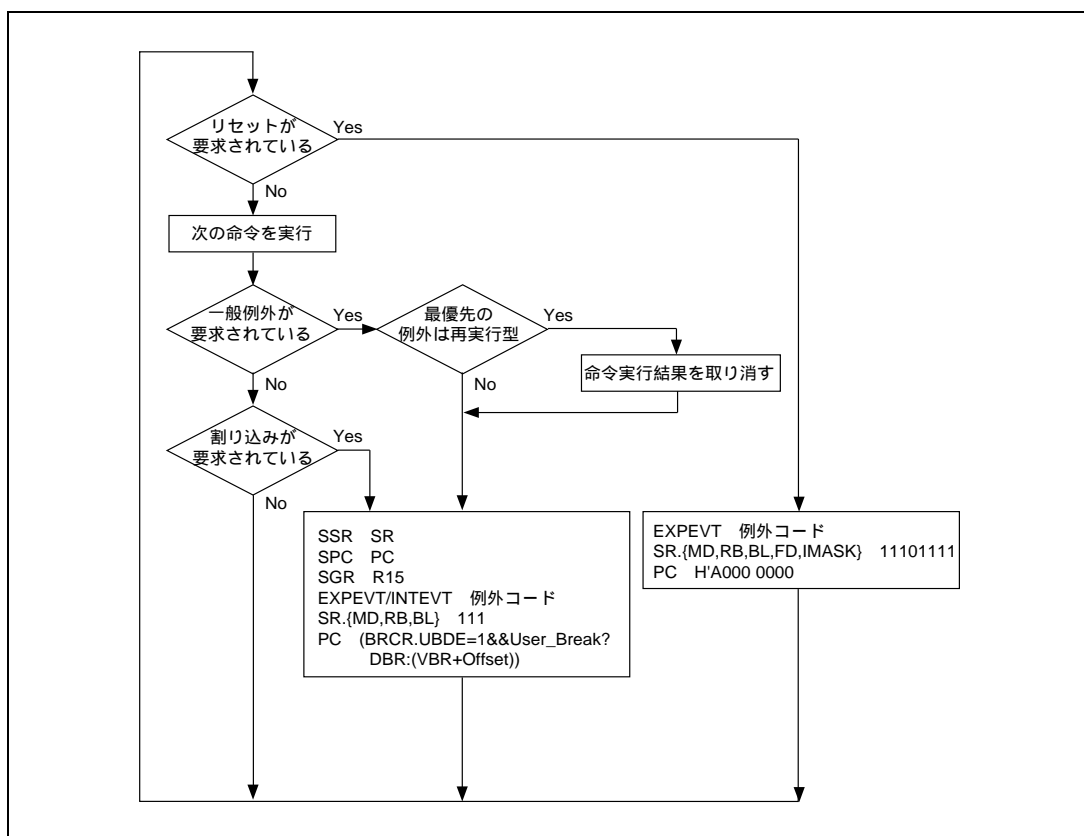


図 5.2 命令実行と例外処理

### 5.5.2 例外要因の受け付け

2 つ以上の例外が同時に発生したときに受け付ける例外を決定するため、すべての例外には優先順位が決められています。一般例外の中の一般不当命令例外、スロット不当命令例外、一般 FPU 抑止例外、スロット FPU 抑止例外、無条件トラップ例外の 5 つは、それぞれの命令解析の過程で検出され、命令パイプラインの中では同時に発生しない例外です。このため優先順位は同じ値になっています。一般例外は命令実行に従った順序で検出されます。しかし、例外処理は命令の流れの順序（プログラム順）に従って処理されます。つまり、先の命令の例外が、後続の命令の例外よりも優先されて受け付けられます。一般例外の受け付け順序の例を図 5.3 に示します。

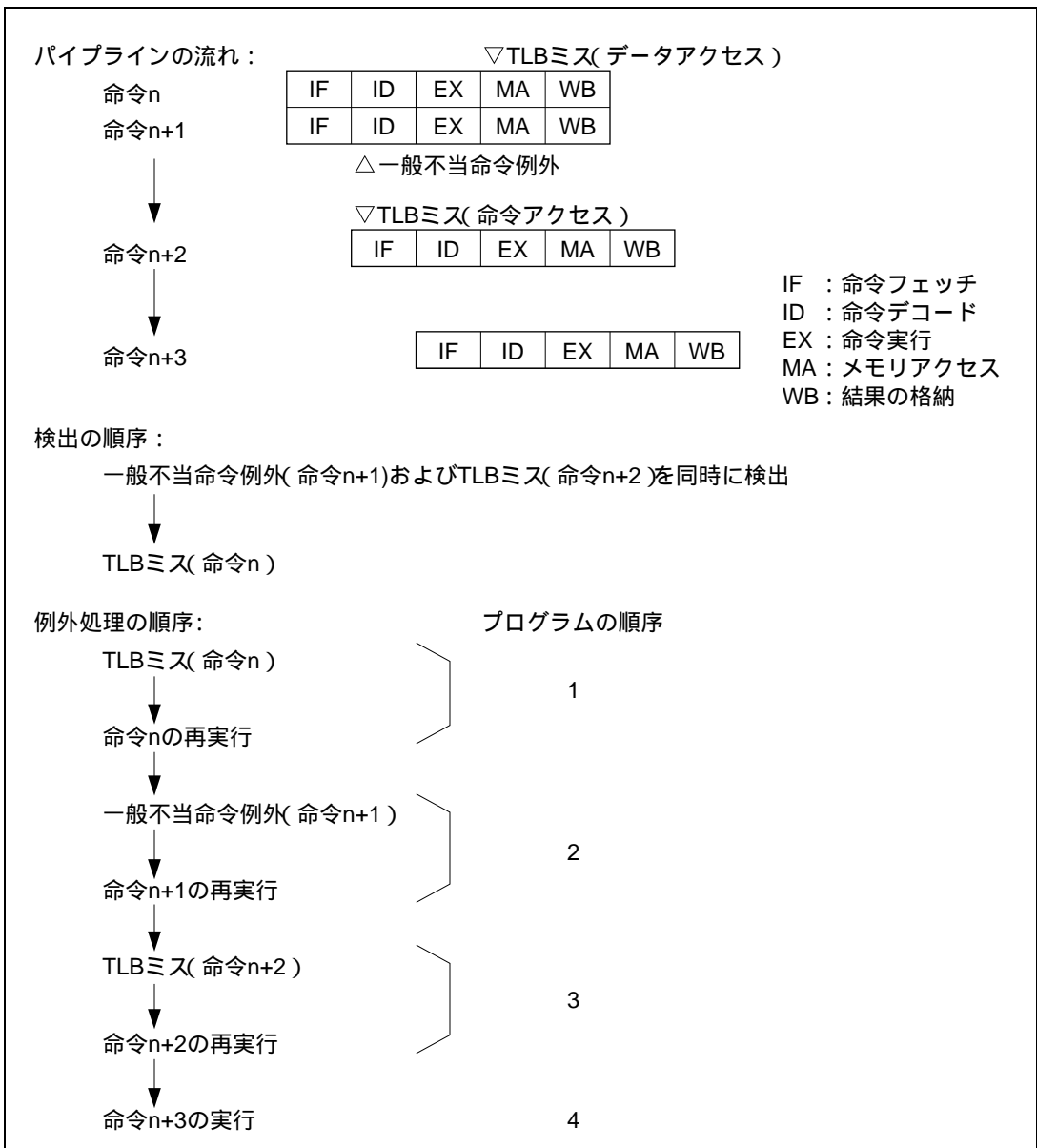


図 5.3 一般例外の受け付け順序の例

### 5.5.3 例外要求と BL ビット

SR の BL ビットが 0 のとき、例外、割り込みを受け付けます。

SR の BL ビットが 1 のときに、ユーザブレイクを除く例外が発生した場合には、CPU の内部レジスタは、リセット後の状態になり、他のモジュールのレジスタは、例外発生前の内容を保持した状態でリセットと同アドレス（H'A000 0000）に分岐します。ユーザブレイクが発生した場合の動作についてはハードウェアマニュアルの「20 章 ユーザブレイクコントローラ」を参照してください。また、通常の割り込みが発生した場合には、割り込み要求は保留され、ソフトウェアで BL ビットが 0 にクリアされてから受け付けられます。ノンマスカブル割り込み（NMI）が発生した場合は、保留するか、受け付けるかをソフトウェアによって設定可能です。

このように、通常は例外状態を多重に受け付け可能にするためには、SPC と SSR を退避させ、その後 SR の BL ビットを 0 クリアします。

### 5.5.4 例外処理からの復帰

例外処理からの復帰は、RTE 命令を使用します。RTE 命令により、SPC が PC に、SSR が SR に回復され、SPC のアドレスに分岐して、例外処理ルーチンから復帰します。もし、メモリに SPC、SSR を退避していた場合には、SR の BL ビットを 1 にセットしてから、SPC と SSR を回復し、RTE 命令を発行してください。

## 5.6 各例外の説明

個別の例外処理動作について、発生要因、発生時の遷移先アドレス、遷移時のプロセッサの動作を説明します。

### 5.6.1 リセット

#### (1) パワーオンリセット

- 要因
  - $\overline{\text{SCK2}}$  端子ハイレベルおよび  $\overline{\text{RESET}}$  端子ローレベル
  - WTCSR の WT/IT ビットが 1 かつ WTCSR の RTS ビットが 0 の状態で、ウォッチドッグタイマがオーバフローした場合。詳細はハードウェアマニュアルの「10 章 クロック発振回路」を参照してください。
- 遷移先アドレス： H'A000 0000
- 遷移時動作：
 

例外コード H'000 を EXPEVT にセットします。VBR、SR の初期化を行い、PC = H'A000 0000 に分岐します。

初期化により、VBR レジスタは H'0000 0000 にセットされます。SR は、MD、RB、BL ビットが 1 にセットされ、FD ビットが 0 にクリアされ、割り込みマスクビット (I3 ~ I0) が B'1111 にセットされます。

CPU および内蔵周辺モジュールの初期化を行います。詳細は、ハードウェアマニュアルの各章のレジスタの説明を参照してください。また、CPU の一部の機能については、 $\overline{\text{TRST}}$  端子ローレベルおよび  $\overline{\text{RESET}}$  端子ローレベルにする必要があります。そのため、電源投入時には必ずパワーオンリセットと、 $\overline{\text{TRST}}$  端子をローレベルに設定してください。

```
Power_on_reset()
{
    EXPEVT = H'00000000;
    VBR = H'00000000;
    SR.MD = 1;
    SR.RB = 1;
    SR.BL = 1;
    SR.(I0-I3) = B'1111;
    SR.FD=0;
    Initialize_CPU();
    Initialize_Module(PowerOn);
    PC = H'A0000000;
}
```

## (2) マニュアルリセット

- 要因
  - $\overline{\text{SCK2}}$  端子ローレベルおよび  $\overline{\text{RESET}}$  端子ローレベル
  - SR の BL ビットが 1 のときにユーザブ레이크を除く一般例外が発生した場合
  - WTCSR の RSTS ビットが 1 のとき、ウォッチドッグタイマがオーバフローした場合。詳細はハードウェアマニュアルの「10 章 クロック発振回路」を参照してください。
- 遷移先アドレス： H'A000 0000
- 遷移時動作：
 

例外コード H'020 を EXPEVT にセットします。VBR、SR の初期化を行い、PC = H'A000 0000 に分岐します。

初期化により、VBR レジスタは H'0000 0000 にセットされます。SR は、MD、RB、BL ビットが 1 にセットされ、FD ビットが 0 にクリアされ、割り込みマスクビット (I3 ~ I0) が B'1111 にセットされます。

CPU および内蔵周辺モジュールの初期化を行います。詳細は、ハードウェアマニュアルの各章のレジスタの説明を参照してください。

```
Manual_reset()
{
    EXPEVT = H'00000020;
    VBR = H'00000000;
    SR.MD = 1;
    SR.RB = 1;
    SR.BL = 1;
    SR.(I0-I3) = B'1111;
    SR.FD = 0;
    Initialize_CPU();
    Initialize_Module(Manual);
    PC = H'A0000000;
}
```

表 5.3 リセットの種類

種類	リセット状態への遷移条件		内部状態	
	$\overline{\text{SCK2}}$	$\overline{\text{RESET}}$	CPU	内蔵周辺モジュール
パワーオンリセット	ハイレベル	ローレベル	初期化	ハードウェアマニュアルの各章のレジスタ構成を参照
マニュアルリセット	ローレベル	ローレベル	初期化	成を参照

## (3) Hitachi-UDI リセット

- 要因：SDIR.TI3 ~ 0 が B'0110 (ネゲート)、または B'0111 (アサート)
- 遷移先アドレス：H'A000 0000
- 遷移時動作：
 

例外コードH'000をEXPEVTにセットします。VBR、SRの初期化を行い、PC = H'A000 0000 に分岐します。

初期化により、VBRレジスタはH'0000 0000にセットされます。SRは、MD、RB、BLビットが1にセットされ、FDビットが0にクリアされ、割り込みマスクビット (I3 ~ I0) がB'1111 にセットされます。

CPUおよび内蔵周辺モジュールの初期化を行います。詳細は、ハードウェアマニュアルの各章のレジスタの説明を参照してください。

```
Hitachi-UDI_reset()
{
    EXPEVT = H'00000000;
    VBR = H'00000000;
    SR.MD = 1;
    SR.RB = 1;
    SR.BL = 1;
    SR.(I0~I3) = B'1111;
    SR.FD = 0;
    Initialize_CPU();
    Initialize_Module(PowerOn);
    PC = H'A0000000;
}
```

## (4) 命令 TLB 多重ヒット例外

- 要因：ITLB のアドレスが多重に一致
- 遷移先アドレス：H'A000 0000
- 遷移時動作：
 

本例外を発生させた仮想アドレス (32ビット) をTEAに、対応する仮想ページ番号 (22ビット) をPTEH [ 31 : 10 ] にセットします。PTEHのASIDは本例外発生時のASIDを示します。

例外コードH'140をEXPEVTにセットします。VBR、SRの初期化を行い、PC = H'A000 0000 に分岐します。

初期化により、VBRレジスタはH'0000 0000にセットされます。SRは、MD、RB、BLビットが1にセットされ、FDビットが0にクリアされ、割り込みマスクビット (I3 ~ I0) がB'1111 にセットされます。

CPUおよび内蔵周辺モジュールの初期化をマニュアルリセットの場合と同様に行います。詳細は、ハードウェアマニュアルの各章のレジスタの説明を参照してください。



```
TLB_multi_hit()
{
    TEA = EXCEPTION_ADDRESS;
    PTEH.VPN = PAGE_NUMBER;
    EXPEVT = H'00000140;
    VBR = H'00000000;
    SR.MD = 1;
    SR.RB = 1;
    SR.BL = 1;
    SR.(I0-I3) = B'1111;
    SR.FD = 0;
    Initialize_CPU();
    Initialize_Module(Manual);
    PC = H'A0000000;
}
```

(5) データ TLB 多重ヒット例外

- 要因：UTLB のアドレスが多重に一致
- 遷移先アドレス： H'A000 0000
- 遷移時動作：

本例外を発生させた仮想アドレス（32ビット）をTEAに、対応する仮想ページ番号（22ビット）をPTEH [ 31 : 10 ] にセットします。PTEHのASIDは本例外発生時のASIDを示します。例外コードH'140をEXPEVTにセットします。VBR、SRの初期化を行い、PC = H'A000 0000 に分岐します。

初期化により、VBRレジスタはH'0000 0000にセットされます。SRは、MD、RB、BLビットが1にセットされ、FDビットが0にクリアされ、割り込みマスクビット（I3～I0）がB'1111にセットされます。

CPUおよび内蔵周辺モジュールの初期化をマニュアルリセットの場合と同様に行います。詳細は、ハードウェアマニュアルの各章のレジスタの説明を参照してください。

```
TLB_multi_hit()
{
    TEA = EXCEPTION_ADDRESS;
    PTEH.VPN = PAGE_NUMBER;
    EXPEVT = H'00000140;
    VBR = H'00000000;
    SR.MD = 1;
    SR.RB = 1;
    SR.BL = 1;
    SR.(I0-I3) = B'1111;
    SR.FD = 0;
    Initialize_CPU();
    Initialize_Module(PowerOn);
    PC = H'A0000000;
}
```

## 5.6.2 一般例外

### (1) データ TLB ミス例外

- 要因：UTLB のアドレス比較の結果、アドレスが不一致
- 遷移先アドレス： VBR + H'0000 0400
- 遷移時動作：
 

本例外を発生させた仮想アドレス（32ビット）をTEAに、対応する仮想ページ番号（22ビット）をPTEH [ 31 : 10 ] にセットします。PTEHのASIDは本例外発生時のASIDを示します。本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避します。読み出しの場合は例外コードH'040を書き込みの場合は例外コードH'060をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC = VBR + H'0400に分岐します。

TLBミス処理高速化のために、他の例外とオフセットを分けています。

```
Data_TLB_miss_exception()
{
    TEA = EXCEPTION_ADDRESS;
    PTEH.VPN = PAGE_NUMBER;
    SPC = PC;
    SSR = SR;
    EXPEVT = read_access ? H'00000040 : H'00000060;
    SR.MD = 1;
    SR.RB = 1;
    SR.BL = 1;
    PC = VBR + H'00000400;
}
```

### (2) 命令 TLB ミス例外

- 要因：ITLB のアドレス比較の結果、アドレスが不一致
- 遷移先アドレス： VBR + H'0000 0400
- 遷移時動作：
 

本例外を発生させた仮想アドレス（32ビット）をTEAに、対応する仮想ページ番号（22ビット）をPTEH [ 31 : 10 ] にセットします。PTEHのASIDは本例外発生時のASIDを示します。本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避します。例外コードH'040をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC = VBR + H'0400に分岐します。

TLBミス処理高速化のために、他の例外とオフセットを分けています。

```
ITLB_miss_exception()  
{  
    TEA = EXCEPTION_ADDRESS;  
    PTEH.VPN = PAGE_NUMBER;  
    SPC = PC;  
    SSR = SR;  
    EXPEVT = H'00000040;  
    SR.MD = 1;  
    SR.RB = 1;  
    SR.BL = 1;  
    PC = VBR + H'00000400;  
}
```

(3) 初期ページ書き込み例外

- 要因：ストアアクセスで TLB にヒットしたが、ダーティビット D=0
- 遷移先アドレス： VBR + H'0000 0100
- 遷移時動作：  
本例外を発生させた仮想アドレス (32ビット) を TEA に、対応する仮想ページ番号 (22ビット) を PTEH [ 31 : 10 ] にセットします。PTEH の ASID は本例外発生時の ASID を示します。  
本例外を発生させた命令の PC、SR をそれぞれ SPC、SSR に退避します。  
例外コード H'080 を EXPEVT にセットします。SR の BL ビット、MD ビット、RB ビットを 1 にセットし、PC = VBR + H'0100 に分岐します。

```
Initial_write_exception()  
{  
    TEA = EXCEPTION_ADDRESS;  
    PTEH.VPN = PAGE_NUMBER;  
    SPC = PC;  
    SSR = SR;  
    EXPEVT = H'00000080;  
    SR.MD = 1;  
    SR.RB = 1;  
    SR.BL = 1;  
    PC = VBR + H'00000100;  
}
```

## (4) データ TLB 保護違反例外

- 要因：アクセスが以下に示す UTLB のプロテクション情報（PR ビット）に反する。

PR	特権モード	ユーザモード
00	読み出しのみ可	アクセス不可
01	読み出し / 書き込み可	アクセス不可
10	読み出しのみ可	読み出しのみ可
11	読み出し / 書き込み可	読み出し / 書き込み可

- 遷移先アドレス：VBR + H'0000 0100
- 遷移時動作：
 

本例外を発生させた仮想アドレス（32ビット）をTEAに、対応する仮想ページ番号（22ビット）をPTEH [ 31 : 10 ] にセットします。PTEHのASIDは本例外発生時のASIDを示します。本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避します。

読み出しの場合には例外コードH'0A0を、書き込みの場合には例外コードH'0C0をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC = VBR + H'0100に分岐します。

```
Data_TLB_protection_violation_exception()
{
    TEA = EXCEPTION_ADDRESS;
    PTEH.VPN = PAGE_NUMBER;
    SPC = PC;
    SSR = SR;
    EXPEVT = read_access ? H'000000A0 : H'000000C0;
    SR.MD = 1;
    SR.RB = 1;
    SR.BL = 1;
    PC = VBR + H'00000100;
}
```

## (5) 命令 TLB 保護違反例外

- 要因：アクセスが以下に示す ITLB のプロテクション情報 (PR ビット) に反する。

PR	特権モード	ユーザモード
0	アクセス可	アクセス不可
1	アクセス可	アクセス可

- 遷移先アドレス：VBR + H'0000 0100
- 遷移時動作：
 

本例外を発生させた仮想アドレス (32ビット) を TEA に、対応する仮想ページ番号 (22ビット) を PTEH [31:10] にセットします。PTEH の ASID は本例外発生時の ASID を示します。本例外を発生させた命令の PC、SR をそれぞれ SPC、SSR に退避します。例外コード H'0A0 を EXPEVT にセットします。SR の BL ビット、MD ビット、RB ビットを 1 にセットし、PC = VBR + H'0100 に分岐します。

```
ITLB_protection_violation_exception()
{
    TEA = EXCEPTION_ADDRESS;
    PTEH.VPN = PAGE_NUMBER;
    SPC = PC;
    SSR = SR;
    EXPEVT = H'000000A0;
    SR.MD = 1;
    SR.RB = 1;
    SR.BL = 1;
    PC = VBR + H'00000100;
}
```

## (6) データアドレスエラー

- 要因：
  - ワードデータをワード境界以外 ( $2n+1$ ) からアクセス
  - ロングワードデータをロングワードデータ境界以外 ( $4n+1$ ,  $4n+2$ ,  $4n+3$ ) からアクセス
  - クワッドワードをクワッドワードデータ境界以外 ( $8n+1$ ,  $8n+2$ ,  $8n+3$ ,  $8n+4$ ,  $8n+5$ ,  $8n+6$ ,  $8n+7$ ) からアクセス
  - ユーザモードでの領域 H'8000 0000 ~ H'FFFF FFFF へのアクセス
- 遷移先アドレス：VBR + H'0000 0100
- 遷移時動作：
 

本例外を発生させた仮想アドレス (32ビット) を TEA に、対応する仮想ページ番号 (22ビット) を PTEH [31:10] にセットします。PTEH の ASID は本例外発生時の ASID を示します。本例外を発生させた命令の PC、SR をそれぞれ SPC、SSR に退避します。読み出しの場合は例外コード H'0E0 を、書き込みの場合は例外コード H'100 を EXPEVT にセットします。SR の BL ビット、MD ビット、RB ビットを 1 にセットし、PC = VBR + H'0100 に分岐します。詳細は「第3章 メモリマネジメントユニット」を参照してください。

```

Data_address_error()
{
    TEA = EXCEPTION_ADDRESS;
    PTEN.VPN = PAGE_NUMBER;
    SPC = PC;
    SSR = SR;
    EXPEVT = read_access? H'000000E0: H'00000100;
    SR.MD = 1;
    SR.RB = 1;
    SR.BL = 1;
    PC = VBR + H'00000100;
}

```

## (7) 命令アドレスエラー

## • 要因：

(a) ワード境界以外 ( $2n+1$ ) から命令フェッチ

(b) ユーザモードでの領域 H'8000 0000 ~ H'FFFF FFFF から命令フェッチ

## • 遷移先アドレス： VBR + H'0000 0100

## • 遷移時動作：

本例外を発生させた仮想アドレス (32ビット) をTEAに、対応する仮想ページ番号(22ビット)をPTEH [ 31 : 10 ] にセットします。PTEHのASIDは本例外発生時のASIDを示します。

本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避します。

例外コードH'0E0をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC = VBR + H'0100に分岐します。詳細は「第3章 メモリマネジメントユニット」を参照してください。

```

Instruction_address_error()
{
    TEA = EXCEPTION_ADDRESS;
    PTEN.VPN = PAGE_NUMBER;
    SPC = PC;
    SSR = SR;
    EXPEVT = H'000000E0;
    SR.MD = 1;
    SR.RB = 1;
    SR.BL = 1;
    PC = VBR + H'00000100;
}

```

### (8) 無条件トラップ

- 要因：TRAPA 命令の実行
- 遷移先アドレス：VBR + H'0000 0100
- 遷移時動作：  
処理完了型の例外のため、TRAPA命令の次の命令のPCをSPCに退避します。TRAPA命令実行時のSRをSSRに退避します。TRAPA命令中の8ビットのイミディエイトを4倍して、TRA [9:0] にセットします。例外コードH'160をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC = VBR + H'0100に分岐します。

```
TRAPA_exception()  
{  
    SPC = PC + 2;  
    SSR = SR;  
    TRA = imm << 2;  
    EXPEVT = H'00000160;  
    SR.MD = 1;  
    SR.RB = 1;  
    SR.BL = 1;  
    PC = VBR + H'00000100;  
}
```

### (9) 一般不当命令例外

- 要因：
  - (a) 遅延スロット以外にある未定義命令をデコード  
遅延分岐命令：JMP、JSR、BRA、BRAf、BSR、BSRf、RTS、RTE、BT/S、BF/S  
未定義命令：H'FFFD
  - (b) 遅延スロット以外にある特権命令をユーザモードでデコード  
特権命令：LDC、STC、RTE、LDTLB、SLEEP、  
ただし、LDC、STCでGBRをアクセスする命令を除く
- 遷移先アドレス：VBR + H'0000 0100
- 遷移時動作：  
本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避します。  
例外コードH'180をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC = VBR + H'0100に分岐します。なお、H'FFFD以外の未定義コードをデコードした場合には動作を保証しません。

```

General_illegal_instruction_exception()
{
    SPC = PC;
    SSR = SR;
    EXPEVT = H'00000180;
    SR.MD = 1;
    SR.RB = 1;
    SR.BL = 1;
    PC = VBR + H'00000100;
}

```

## (10) スロット不当命令例外

- 要因：
  - (a) 遅延スロットにある未定義命令をデコード  
 遅延分岐命令：JMP、JSR、BRA、BRAf、BSR、BSRf、RTS、RTE、BT/S、BF/S  
 未定義命令：H'FFFD
  - (b) 遅延スロット内の PC を書き換える命令をデコード  
 PCを書き換える命令：JMP、JSR、BRA、BRAf、BSR、BSRf、RTS、RTE、BT、BF、BT/S、  
 BF/S、TRAPA、LDC Rm、SR、LDC.L @Rm+、SR
  - (c) 遅延スロット内の特権命令をユーザモードでデコード  
 特権命令：LDC、STC、RTE、LDTLB、SLEEP、ただし、LDC、STCでGBRをアクセスする命令を除く
  - (d) 遅延スロット内の PC 相対 MOV 命令、MOVA 命令をデコード
    - 遷移先アドレス：VBR + H'0000 0100
    - 遷移時動作：
 

直前の遅延分岐命令のPCをSPCに退避します。本例外発生時のSRをSSRに退避します。  
 例外コードH'1A0をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1に  
 セットし、PC = VBR + H'0100に分岐します。なお、H'FFFD以外の未定義命令をデコードし  
 た場合には動作を保証しません。

```

Slot_illegal_instruction_exception()
{
    SPC = PC - 2;
    SSR = SR;
    EXPEVT = H'000001A0;
    SR.MD = 1;
    SR.RB = 1;
    SR.BL = 1;
    PC = VBR + H'00000100;
}

```



### (11) 一般 FPU 抑止例外

- 要因：遅延スロット以外にある FPU 命令\*1 を SR.FD=1 でデコード
- 遷移先アドレス：VBR + H'0000 0100
- 遷移時動作：  
本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避します。  
例外コードH'800をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC = VBR + H'0100に分岐します。

```
General_fpu_disable_exception()  
{  
    SPC = PC;  
    SSR = SR;  
    EXPEVT = H'00000800;  
    SR.MD = 1;  
    SR.RB = 1;  
    SR.BL = 1;  
    PC = VBR + H'00000100;  
}
```

【注】\*1 FPU 命令とは命令コードの最初の4ビットがFである命令(ただし、未定義命令 H'FFFD を除く)と、FPUL、FPSCR に対する LDS、STS、LDC.L、STC.L 命令です。

### (12) スロット FPU 抑止例外

- 要因：遅延スロットにある FPU 命令を SR.FD=1 でデコード
- 遷移先アドレス：VBR + H'0000 0100
- 遷移時動作：  
直前の遅延分岐命令のPCをSPCに退避します。本例外発生時のSRをSSRに退避します。  
例外コードH'820をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC = VBR + H'0100に分岐します。

```
Slot_fpu_disable_exception()  
{  
    SPC = PC - 2;  
    SSR = SR;  
    EXPEVT = H'00000820;  
    SR.MD = 1;  
    SR.RB = 1;  
    SR.BL = 1;  
    PC = VBR + H'00000100;  
}
```

## (13) ユーザブレークポイントトラップ

- 要因：ユーザブレークポイントコントローラに設定したブレーク条件が成立
- 遷移先アドレス： VBR + H'0000 0100、または DBR
- 遷移時動作：
 

実行後ブレークの場合、ブレークポイントを設定した命令の直後の命令のPCをSPCに退避します。実行前ブレークの場合、ブレークポイントを設定した命令のPCをSPCに退避します。

ブレーク発生時のSRをSSRに退避します。例外コードH'1E0をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC = VBR + H'0100に分岐します。ただし、PC=DBRに分岐することも可能です。

データブレークを設定した場合のPCについてなど、詳細はハードウェアマニュアルの「第20章 ユーザブレークコントローラ」を参照してください。

```
User_break_exception()
{
    SPC = (pre_execution break? PC : PC + 2);
    SSR = SR;
    EXPEVT = H'000001E0;
    SR.MD = 1;
    SR.RB = 1;
    SR.BL = 1;
    PC = (BRCCR.UBDE==1 ? DBR : VBR + H'00000100);
}
```

## (14) FPU 例外

- 要因：浮動小数点演算実行による例外
- 遷移先アドレス： VBR + H'0000 0100
- 遷移時動作：
 

本例外を発生させた命令のPC、SRをそれぞれSPC、SSRに退避します。例外コードH'120をEXPEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC=VBR+H'0100に分岐します。

```
FPU_exception()
{
    SPC = PC;
    SSR = SR;
    EXPEVT = H'00000120;
    SR.MD = 1;
    SR.RB = 1;
    SR.BL = 1;
    PC = VBR + H'00000100;
}
```

### 5.6.3 割り込み

#### (1) NMI

- 要因：NMI 端子のエッジ検出
- 遷移先アドレス：VBR + H'0000 0600
- 遷移時動作：

本割り込みを受け付けた命令の直後のPC、SRを、それぞれSPC、SSRに退避します。  
例外コードH'1C0をINTEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、PC = VBR + H'0600に分岐します。本割り込みは、SRのBLビットが0のときはSRの割り込みマスクビットによってマスクされず、最優先で受け付けられます。SRのBLビットが1のとき本割り込みがマスクされるか、受け付けるかをソフトウェアによって設定可能です。詳細はハードウェアマニュアルの「第19章 割り込みコントローラ」を参照してください。

```
NMI()  
{  
    SPC = PC;  
    SSR = SR;  
    INTEVT = H'000001C0;  
    SR.MD = 1;  
    SR.RB = 1;  
    SR.BL = 1;  
    PC = VBR + H'00000600;  
}
```

#### (2) IRL 割り込み

- 要因：  
SRの割り込みマスクビットがIRL (3-0) レベルより小さく、かつSRのBLビットが0 (命令の切れ目で受け付けます)。
- 遷移先アドレス：VBR + H'0000 0600
- 遷移時動作：  
受け付けた命令の直後のPCをSPCにセットします。受け付けた時点のSRをSSRにセットします。  
IRL (3-0) レベルに対応したコードをINTEVTにセットします。対応コードはハードウェアマニュアルの「表19.5 割り込み例外処理要因と優先順位」を参照してください。SRのBLビット、MDビット、RBビットを1にセットし、VBR + H'0600に分岐します。受け付けレベルをSRの割り込みマスクビットにセットしません。SRのBLビットが1のときは、マスクされます。詳細はハードウェアマニュアルの「第19章 割り込みコントローラ」を参照してください。

```

IRL()
{
    SPC = PC;
    SSR = SR;
    INTEVT = H'00000200 ~ H'000003C0;
    SR.MD = 1;
    SR.RB = 1;
    SR.BL = 1;
    PC = VBR + H'00000600;
}

```

### (3) 周辺モジュール割り込み

- 要因：  
SRの割り込みマスクビットが周辺モジュール( Hitachi-UDI、DMAC、TMU、RTC、SCI、SCIF、WDT、REF )割り込みレベルより小さく、かつSRのBLが0( 命令の切れ目で受け付けます。)
- 遷移先アドレス：VBR + H'0000 0600
- 遷移時動作：  
受け付けた命令の直後のPCをSPCにセットします。受け付けた時点のSRをSSRにセットします。  
各割り込み要因に対応したコードをINTEVTにセットします。SRのBLビット、MDビット、RBビットを1にセットし、VBR + H'0600に分岐します。モジュール割り込みのレベルは、割り込みコントローラ内の割り込み優先レベル設定レジスタ( IRPA ~ IRPC )にB'0000からB'1111までの値をセットしてください。詳細はハードウェアマニュアルの「第19章 割り込みコントローラ」を参照してください。

```

Module_interruption()
{
    SPC = PC;
    SSR = SR;
    INTEVT = H'00000400 ~ H'00000760;
    SR.MD = 1;
    SR.RB = 1;
    SR.BL = 1;
    PC = VBR + H'00000600;
}

```

### 5.6.4 複数回の例外が発生する場合の優先順位

メモリを2回アクセスする命令や、不可分である遅延付き分岐命令と遅延スロット命令などでは、複数回例外が発生します。この場合、通常の例外優先順位と異なるので、注意が必要です。

#### (1) メモリを2回アクセスする命令

MAC 命令やメモリ メモリ間論理演算命令、TAS 命令は1つの命令でデータ転送が2回あるため、それぞれのデータ転送時に例外の発生を検出します。そのため、以下の順位で判定します。

- (a) 1 回目のデータ転送のデータアドレスエラー
- (b) 1 回目のデータ転送の TLB ミス
- (c) 1 回目のデータ転送の TLB 保護違反
- (d) 1 回目のデータ転送の初期ページ書き込み例外
- (e) 2 回目のデータ転送のデータアドレスエラー
- (f) 2 回目のデータ転送の TLB ミス
- (g) 2 回目のデータ転送の TLB 保護違反
- (h) 2 回目のデータ転送の初期ページ書き込み例外

#### (2) 不可分である遅延付き分岐命令と遅延スロット命令

遅延付き分岐命令と遅延スロット命令は不可分であるため、1つの命令として扱われます。そのため、それぞれの命令における例外についても、優先順位が通常と異なります。遅延スロット命令が1回のデータ転送しか持たない場合の順位を示します。

- (a) 遅延付き分岐命令における優先レベル1、2をチェックします。
- (b) 遅延スロット命令における優先レベル1、2をチェックします。
- (c) 遅延付き分岐命令における優先レベル3と遅延スロット命令における優先レベル3をチェックします（この2つの間の優先順位はありません）。
- (d) 遅延付き分岐命令における優先レベル4と遅延スロット命令における優先レベル4をチェックします（この2つの間の優先順位はありません）。

遅延スロット命令が2回目のデータ転送を持つ場合、b)において、(1)の様に2回チェックを行います。

なお、受け付けた例外（最も優先度が高い例外）が遅延スロット命令の再実行型例外である場合、分岐命令のPR レジスタ書き込み動作（BSR、BSRF、JSR のPC PR 動作）は抑止されません。

## 5.7 注意事項

- (1) 例外処理からの復帰
  - (a) SR の BL ビットをソフトウェアでチェックしてください。メモリに SPC、SSR を退避していた場合には、SR の BL ビットを 1 にしてからそれらを回復してください。
  - (b) RTE 命令を発行してください。RTE 命令により、SPC が PC に、SSR が SR にセットされ、SPC のアドレスに分岐して、例外処理から復帰します。
- (2) SR.BL = 1 のときに例外または割り込みが発生した場合
  - (a) 例外  
ユーザブレイクを除く例外が発生した場合には、CPUの内部レジスタ、他のモジュールのレジスタは、マニュアルリセット後の状態になり、リセットと同アドレス (H'A000 0000) に分岐します。このときEXPEVTは、H'0000 0020となり、SPC、SSRの各レジスタは不定値となります。
  - (b) 割り込み  
通常の割り込みが発生した場合には、割り込み要求は保留され、ソフトウェアでSRのBLビットが0にクリアされてから受け付けられます。ノンマスカブル割り込み (NMI) が発生した場合は、保留するか、受け付けるかをソフトウェアによって設定可能です。  
ただし、スリープまたはスタンバイ状態では、SRのBLビットが1であっても、割り込みを受け付けます。
- (3) 例外発生時の SPC
  - (a) 再実行型の例外  
例外が発生した命令のPCがSPCにセットされ、例外処理から復帰後に再実行されます。ただし、遅延スロット命令で発生した場合、直前の遅延分岐命令の条件が成立する、しないに関係なく遅延分岐命令のPCがSPCにセットされます。
  - (b) 完了型の例外、割り込み  
例外が発生した命令の次の命令のPCがSPCにセットされます。ただし、遅延スロット付き分岐命令で発生した場合、分岐先のPCがSPCにセットされます。
- (4) RTE 命令の遅延スロットで例外を発生させないでください。発生した場合、動作は保証されません。

---

## 6. 浮動小数点ユニット

---

### 6.1 概要

FPU には次のような特長があります。

- IEEE754 規格に準拠
- 32 本の単精度浮動小数点レジスタ（16 本の倍精度レジスタとしても参照できます）
- 2 つの丸めモード：近傍および 0 方向への丸め
- 2 つの非正規化数処理モード：0 方向へのフラッシュと非正規化数の扱い
- 6 つの例外要因：  
FPU エラー、無効演算、0 による除算、オーバフロー、アンダフロー、不正確
- 包括命令：  
単精度、倍精度、グラフィックサポート、システム制御

SR の FD ビットを 1 にセットすると、浮動小数点ユニット（FPU）は使用できなくなり、FPU 命令を実行しようとする FPU 抑止例外が発生します。

## 6.2 データフォーマット

### 6.2.1 浮動小数点フォーマット

浮動小数点は次の 3 つのフィールドから構成されています。

- 符号
- 指数
- 小数部

SH7091 は図 6.1 と図 6.2 に示すフォーマットを用いて単精度、倍精度浮動小数点を扱うことができます。

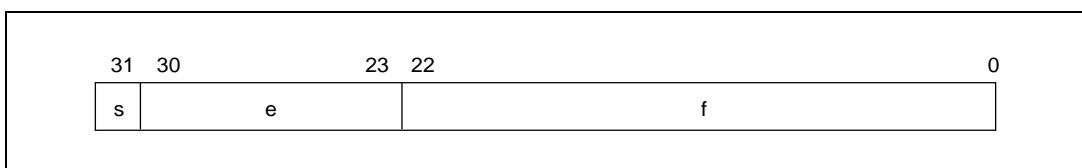


図 6.1 単精度浮動小数点フォーマット



図 6.2 倍精度浮動小数点フォーマット

指数は次のようにバイアス付きで表します。

$$e = E + \text{bias}$$

バイアスのない指数  $E$  の範囲は、 $E_{\min} - 1$  から  $E_{\max} + 1$  までです。 $E_{\min} - 1$  と  $E_{\max} + 1$  の 2 つの値は次のように区別します。 $E_{\min} - 1$  は 0 (正、負両方の符号) と非正規化数を表し、 $E_{\max} + 1$  は正または負の無限大または非数 (NaN) を表します。表 6.1 に  $E_{\min}$  と  $E_{\max}$  の値を示します。



表 6.1 浮動小数点のフォーマットとパラメータ

パラメータ	単精度	倍精度
総ビット幅	32 ビット	64 ビット
符号ビット	1 ビット	1 ビット
指数フィールド	8 ビット	11 ビット
小数フィールド	23 ビット	52 ビット
精度	24 ビット	53 ビット
バイアス	+127	+1023
$E_{\max}$	+127	+1023
$E_{\min}$	-126	-1022

浮動小数点の数値  $v$  は次のようにして決められます。

$E = E_{\max} + 1$  かつ  $f = 0$  の場合、 $v$  は符号  $s$  に関係なく非数 (NaN) です。  
 $E = E_{\max} + 1$  かつ  $f = 0$  の場合、 $v$  は  $(-1)^s$  (無限) 「正または負の無限」です。  
 $E_{\min} \leq E \leq E_{\max}$  の場合、 $v$  は  $(-1)^s 2^E (1.f)$  「正規化数」です。  
 $E = E_{\min} - 1$  かつ  $f = 0$  の場合、 $v$  は  $(-1)^s 2^{E_{\min}} (0.f)$  「非正規化数」です。  
 $E = E_{\min} - 1$  かつ  $f = 0$  の場合、 $v$  は  $(-1)^s 0$  「正または負の 0」です。

表 6.2 に 16 進数による各数の範囲を示します。

表 6.2 浮動小数点の範囲

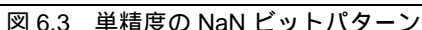
タイプ	単精度	倍精度
シグナリング非数	H'7FFFFFFF ~ H'7FC00000	H'7FFFFFFF H'FFFFFFFF ~ H'7F800000 H'00000000
クワイアット非数	H'7FBFFFFFF ~ H'7F800001	H'7F7FFFFFFF H'FFFFFFFF ~ H'7FF00000 H'00000001
正の無限大	H'7F800000	H'7FF00000 H'000000
正の正規化数	H'7F7FFFFFFF ~ H'00800000	H'7FEFFFFFFF H'FFFFFFFF ~ H'00100000 H'00000000
正の非正規化数	H'007FFFFFFF ~ H'00000001	H'000FFFFFFF H'FFFFFFFF ~ H'00000000 H'00000001
正のゼロ	H'00000000	H'00000000 H'00000000
負のゼロ	H'80000000	H'80000000 H'00000000
負の非正規化数	H'80000001 ~ H'807FFFFFFF	H'00000000 H'00000000 ~ H'800FFFFFFF H'FFFFFFFF
負の正規化数	H'80800000 ~ H'FF7FFFFFFF	H'80000000 H'00000001 ~ H'FFEFFFFFFF H'FFFFFFFF
負の無限大	H'FF800000	H'FFF00000 H'00000000
クワイアット非数	H'FF800001 ~ H'FFBFFFFFFF	H'FFF00000 H'00000001 ~ H'FFF7FFFF H'FFFFFFFF
シグナリング非数	H'FFC00000 ~ H'FFFFFFFF	H'FFF80000 H'00000000 ~ H'FFFFFFFF H'FFFFFFFF

### 6.2.2 非数 (NaN)

図 6.3 に非数 (NaN) のビットパターンを示します。次の場合の値は NaN です。

- 符号ビット： don't care
- 指数フィールド： すべて 1 のビット
- 小数フィールド： MSB を除くビットの少なくとも 1 つが 1 のとき

NaN は、小数フィールドの MSB が 1 の場合はシグナリング非数 (sNaN) であり、0 の場合はクワイアット非数 (qNaN) です。



## 6.3 レジスタ

### 6.3.1 浮動小数点レジスタ

図 6.4 に浮動小数点レジスタの構成を示します。FR0 ~ FR15、DR0/2/4/6/8/10/12/14、FV0/4/8/12、XF0 ~ XF15、XD0/2/4/6/8/10/12/14、または XMTRX を指定することによって参照される 32 本の 32 ビット浮動小数点レジスタがあります。

- (1) 浮動小数点レジスタ：FPRi\_BANKj (32 レジスタ)  
FPR0\_BANK0 ~ FPR15\_BANK0  
FPR0\_BANK1 ~ FPR15\_BANK1
- (2) 単精度浮動小数点レジスタ：FRi (16 レジスタ)  
FPSCR.FR = 0 のとき FR0 ~ FR15 は FPR0\_BANK0 ~ FPR15\_BANK0 を示します。  
FPSCR.FR = 1 のとき FR0 ~ FR15 は FPR0\_BANK1 ~ FPR15\_BANK1 を示します。
- (3) 倍精度浮動小数点レジスタ：DRi (8 レジスタ)  
DR レジスタは 2 つの FR レジスタから構成されます。  
DR0 = {FR0, FR1}、DR2 = {FR2, FR3}、DR4 = {FR4, FR5}、DR6 = {FR6, FR7}、  
DR8 = {FR8, FR9}、DR10 = {FR10, FR11}、DR12 = {FR12, FR13}、DR14 = {FR14, FR15}
- (4) 単精度浮動小数点ベクトルレジスタ、FVi (4 レジスタ)  
FV レジスタは 4 つの FR レジスタから構成されます。  
FV0 = {FR0, FR1, FR2, FR3}、FV4 = {FR4, FR5, FR6, FR7}、  
FV8 = {FR8, FR9, FR10, FR11}、FV12 = {FR12, FR13, FR14, FR15}
- (5) 単精度浮動小数点拡張レジスタ：XF<sub>i</sub> (16 レジスタ)  
FPSCR.FR = 0 のとき XF0 ~ XF15 は FPR0\_BANK1 ~ FPR15\_BANK1 を示します。  
FPSCR.FR = 1 のとき、XF0 ~ XF15 は FPR0\_BANK0 ~ FPR15\_BANK0 を示します。
- (6) 倍精度浮動小数点拡張レジスタ：XD<sub>i</sub> (8 レジスタ)  
XD レジスタは 2 つの XF レジスタから構成されます。  
XD0 = {XF0, XF1}、XD2 = {XF2, XF3}、XD4 = {XF4, XF5}、XD6 = {XF6, XF7}、  
XD8 = {XF8, XF9}、XD10 = {XF10, XF11}、XD12 = {XF12, XF13}、XD14 = {XF14, XF15}
- (7) 単精度浮動小数点拡張レジスタ行列、XMTRX  
XMTRX は 16 の XF レジスタから構成されます。  

XF0	XF4	XF8	XF12
XF1	XF5	XF9	XF13
XF2	XF6	XF10	XF14
XF3	XF7	XF11	XF15

## 6. 浮動小数点ユニット

FPSCR.FR=0				FPSCR.FR=1			
FV0	DR0	FR0	FPR0 BANK0	XF0	XD0	XMTRX	
		FR1	FPR1 BANK0	XF1			
	DR2	FR2	FPR2 BANK0	XF2	XD2		
FV4		FR3	FPR3 BANK0	XF3			
	DR4	FR4	FPR4 BANK0	XF4	XD4		
		FR5	FPR5 BANK0	XF5			
FV8	DR6	FR6	FPR6 BANK0	XF6	XD6		
		FR7	FPR7 BANK0	XF7			
	DR8	FR8	FPR8 BANK0	XF8	XD8		
FV12		FR9	FPR9 BANK0	XF9			
	DR10	FR10	FPR10 BANK0	XF10	XD10		
		FR11	FPR11 BANK0	XF11			
	DR12	FR12	FPR12 BANK0	XF12	XD12		
		FR13	FPR13 BANK0	XF13			
	DR14	FR14	FPR14 BANK0	XF14	XD14		
		FR15	FPR15 BANK0	XF15			
XMTRX	XD0	XF0	FPR0 BANK1	FR0	DR0	FV0	
		XF1	FPR1 BANK1	FR1			
	XD2	XF2	FPR2 BANK1	FR2	DR2		
		XF3	FPR3 BANK1	FR3			
	XD4	XF4	FPR4 BANK1	FR4	DR4		
		XF5	FPR5 BANK1	FR5			
	XD6	XF6	FPR6 BANK1	FR6	DR6		
		XF7	FPR7 BANK1	FR7			
	XD8	XF8	FPR8 BANK1	FR8	DR8		
		XF9	FPR9 BANK1	FR9			
	XD10	XF10	FPR10 BANK1	FR10	DR10		
		XF11	FPR11 BANK1	FR11			
	XD12	XF12	FPR12 BANK1	FR12	DR12	FV12	
		XF13	FPR13 BANK1	FR13			
	XD14	XF14	FPR14 BANK1	FR14	DR14		
		XF15	FPR15 BANK1	FR15			

図 6.4 浮動小数点レジスタ

### 6.3.2 浮動小数点ステータス / コントロールレジスタ (FPSCR)

(1) 浮動小数点ユニットステータス / コントロールレジスタ、FPSCR (32 ビット、初期値 = H'00040001)

- FR : 浮動小数点レジスタバンク
  - FR=0 :  
FPR0\_BANK0 ~ FPR15\_BANK0はFR0 ~ FR15に、 FPR0\_BANK1 ~ FPR15\_BANK1はXF0 ~ XF15に割り当てられます。
  - FR=1 :  
FPR0\_BANK0 ~ FPR15\_BANK0はXF0 ~ XF15に、 FPR0\_BANK1 ~ FPR15\_BANK1はFR0 ~ FR15に割り当てられます。
- SZ : 転送サイズモード
  - SZ=0 : FMOV 命令のデータサイズは 32 ビットです。
  - SZ=1 : FMOV 命令のデータサイズは 32 ビットペア (64 ビット) です。
- PR : 精度モード
  - PR=0 :  
浮動小数点命令を単精度演算として実行します。
  - PR=1 :  
浮動小数点命令を倍精度演算として実行します (グラフィックサポート命令は未定義です)。

SZ と PR は同時に 1 にセットしないでください。この設定は予約されています。

[SZ, PR] = 11 : 予約 (FPU 命令演算は未定義です)

- DN : 非正規化モード
  - DN=0 : 非正規化数を非正規化数として扱います。
  - DN=1 : 非正規化数を 0 として扱います。

		FPUエラー (E)	無効演算 (V)	0 除算 (Z)	オーバ フロー(O)	アンダ フロー(U)	不正確 (I)
Cause	FPU 例外要因 フィールド	ビット 17	ビット 16	ビット 15	ビット 14	ビット 13	ビット 12
Enable	FPU 例外イネー ブルフィールド	なし	ビット 11	ビット 10	ビット 9	ビット 8	ビット 7
Flag	FPU 例外フラグ フィールド	なし	ビット 6	ビット 5	ビット 4	ビット 3	ビット 2

FPU例外が要求されると、cause/flagフィールドに該当するビットは1にセットされます。FPU演算命令が実行されるたびに、causeフィールドはまず0にクリアされます。flagフィールドはソフトウェアによって0にクリアされるまで1の値を保持します。

- RM：丸めモード
  - RM=00：近傍への丸め
  - RM=01：0 方向に丸め
  - RM=10：予約
  - RM=11：予約
- ビット 22～ビット 31：予約

【注】 SH7718 の FPU と比較して、SH7091 の FPU には以下の機能が追加されています。

- (1) FR、SZ、PR ビットが追加されました。
- (2) 要因、イネーブル、フラグの各フィールド (cause、enable、flag) に、例外 O (オーバフロー)、U (アンダフロー)、I (不正確) のビットが追加されました。
- (3) 要因フィールド (cause) に、例外 E (FPU エラー) のビットが追加されました。

### 6.3.3 浮動小数点通信レジスタ (FPUL)

FPU と CPU 間の情報伝達は FPUL レジスタを介して行われます。32 ビットの FPUL レジスタはシステムレジスタで、LDS、STS 命令によって CPU からアクセスします。たとえば、汎用レジスタ R1 に格納した整数を浮動小数点に変換する処理フローは次のとおりです。

R1      (LDS 命令)      FPUL      (FLOAT 命令)      FR1

## 6.4 丸め

浮動小数点命令において、丸めは中間結果から最終演算結果を生成する際に実行されます。したがって、FMAC、FTRV、FIPR のような組み合わせ命令の結果は、FADD、FSUB、FMUL などの基本命令だけを用了結果とは異なります。FMAC は 1 度、FADD、FSUB および FMUL は 2 度というように丸めの回数が異なるためです。

丸めには 2 つの方法があり、使用する方法は FPSCR の RM フィールドで決まります。

RM=00：近傍への丸め  
RM=01：0 方向に丸め

#### (1) 近傍への丸め

値はもっとも近い表現可能な値に丸められます。もっとも近い表現可能な値が 2 つある場合、LSB が 0 の方を選択します。

ただし、丸め前の値が表現可能な最大絶対値の数の 1.5 倍以上の場合、無限になります。

#### (2) 0 方向への丸め

丸め前の値の丸めビット以下の桁は切り捨てられます。

ただし、丸め前の値が表現可能な最大絶対値数よりも大きい場合、表現可能な最大絶対値の数になります。

## 6.5 浮動小数点例外

FPU 関連の例外は次のとおりです。

### (1) 一般 FPU 抑止 / スロット FPU 抑止例外

SR.FD=1 のときに FPU 命令を実行すると発生します。

### (2) FPU 例外

例外要因は次のとおりです。

- FPU エラー (E) : FPSCR.DN=0 かつ非正規化数の入力時
- 無効演算 (V) : NaN 入力のような無効な演算の場合
- 0 による除算 (Z) : 除数 0 による除算
- オーバフロー (O) : 演算結果がオーバフローする場合
- アンダフロー (U) : 演算結果がアンダフローする場合
- 不正確例外 (I) : オーバフロー、アンダフロー、丸めが発生する場合

FPSCR の要因フィールドには上記要因 E、V、Z、O、U、I のすべてに該当するビットが含まれ、FPSCR のフラグおよびイネーブルフィールドには要因 V、Z、O、U、I に該当するビットが含まれていますが E に該当するビットは含まれていません。このように FPU エラーはディスエーブルにすることができません。

例外が発生すると、要因フィールドの該当するビットは 1 にセットされフラグフィールドに該当するビットに 1 が累積されます。例外要因が発生しない場合、要因フィールドの該当するビットは 0 にクリアされ、フラグフィールドに該当するビットは変更されません。

### (3) イネーブル / ディスエーブル例外処理

SH7091 はイネーブル例外処理とディスエーブル例外処理をサポートしています。

イネーブル例外処理は次の場合に発生します。

- FPU エラー (E) : FPSCR.DN=0 かつ非正規化数の入力時
- 無効演算 (V) : FPSCR.EN.V=1 かつ (命令=FTRV または無効演算) の場合
- 0 による除算 (Z) : FPSCR.EN.Z=1 かつ除数 0 による除算
- オーバフロー (O) : FPSCR.EN.O=1 かつ演算結果がオーバフローする可能性のある命令
- アンダフロー (U) : FPSCR.EN.U=1 かつ演算結果がアンダフローする可能性のある命令
- 不正確例外 (I) : FPSCR.EN.I=1 かつ演算結果が不正確になる可能性のある命令

各可能性については各命令の説明で示します。FPU 演算に起因するすべての例外事象は、同一の例外事象として割り付けられています。例外の意味内容は、システムレジスタ FPSCR を読み出して、保持されている情報を解釈することでソフトウェアにより決定します。FPSCR の要因フィールド中のビットが一つもセットされていない場合、イネーブルフィールドの O、U、I および V (FTRV の場合のみ) ビットのうち一つまたは複数のビットがセットされていても実際の例外は発生しないことを示しています。また、いかなるイネーブル例外処理動作によっても、デスティネーションレジスタは変更されません。

上記以外、FPU は例外処理をディスエーブルにします。すべての処理では要因 V、Z、O、U、I に対する該当ビットを 1 にセットし、各例外にはそれぞれのディスエーブル例外処理があります。

- 無効演算 (V) : 結果として qNaN を生成します。
- 0 による除算 (Z) : 正しい符号付きの無限大を生成します。

- オーバフロー(O) :
  - 丸めモード=RZ のとき、正しい符号付き最大正規化数を生成します。
  - 丸めモード=RN のとき、正しい符号付き無限大を生成します。
- アンダフロー(U) :
  - FPSCR.DN=0 のとき、正しい符号付き 0 を生成します。
  - FPSCR.DN=1 のとき、正しい符号付き非正規化数、または 0 を生成します。
- 不正確例外(I) : 不正確な結果を生成します。

## 6.6 グラフィックサポート機能

SH7091 は 2 種類のグラフィック機能をサポートしています。1 つはジオメトリック演算用の新規命令であり、もう一つは高速データ転送を可能にするペア単精度転送命令です。

### 6.6.1 ジオメトリック演算命令

ジオメトリック演算命令は近似値演算です。最小のハードウェアで高速演算を可能とするため、SH7091 は 4 つの乗算の部分的演算結果のうち相対的に小さな値を無視します。したがって、演算結果には以下に示す誤差が生じます。

$$\text{最大誤差} = \text{MAX} ( \text{各乗算結果} \times 2^{-\text{MIN} ( \text{乗数、被乗数の有効数は桁数} - 1 ) } ) + \text{MAX} ( \text{結果値} \times 2^{-23}、2^{-149} )$$

ただし、有効数字桁数は正規化数が 24、非正規化数が 23 (小数部のリーディングゼロの桁数)

(1) FIPR FV<sub>m</sub>,FV<sub>n</sub>(*m,n*: 0, 4, 8, 12)

この命令は基本的に以下の用途に使用されます。

- 内積 (*m = n*) :
  - 一般的に、この演算はポリゴン表面の表面 / 裏面を判定するために使用されます。
- 各要素の平方和 (*m=n*) :
  - 一般的に、この演算はベクトルの長さを得るために使用されます。

高速演算を可能とするため近似値演算を行うことから、FIPR 命令を実行すると、要因フィールドおよびフラグフィールドの不正確例外 (I) ビットが常に 1 にセットされます。したがって、イネーブルフィールドの対応するビットがセットされていれば、イネーブル例外処理が実行されます。



## (2) FTRV XMTRX, FVn (n: 0, 4, 8, 12)

この命令は基本的に以下の用途に使用されます。

- 行列 ( $4 \times 4$ ) ・ ベクトル (4) :  
一般的に、この演算は、視点の変更、角度の変更、または移動といったベクトル変換 (4次元) に使用されます。基本的に、角度 + 平行移動のためのアフィン変換処理は、 $4 \times 4$  行列を必要とします。したがって、SH7091は4次元演算をサポートしています。
- 行列 ( $4 \times 4$ )  $\times$  行列 ( $4 \times 4$ ) :  
この演算を行うためには、FTRV命令を4回実行する必要があります。

高速演算を可能とするため近似値演算を行うことから、FTRV 命令を実行すると、要因フィールドおよびフラグフィールドの不正確例外 (I) ビットが常に1にセットされます。したがって、イネーブルフィールドのIビットがセットされていれば、イネーブル例外処理が実行されます。また、同様の理由で、FTRV 命令の実行の際、レジスタ内のすべてのデータタイプを実行前にチェックすることができません。イネーブルフィールドのVビットがセットされていると、イネーブル例外処理が実行されます。

## (3) FRCHG

この命令はバンクレジスタを変更します。例えば、FTRV 命令を使用する場合、背後にあるバンク上に行列の要素を設定する必要があります。しかし、変換行列の要素自体を作成するには、前面にあるバンクのレジスタを使用する方が簡単です。FPSCR に対する LDS 命令を使用すると、この命令はFPUの状態を維持するために、4~5サイクルを費やします。FRCHG 命令ではFPSCR.FR ビットの変更を1サイクル期間で行うことができます。

## 6.6.2 ペア単精度データ転送

新規に設けられた強力なジオメトリック演算命令に加えて、SH7091 は高速データ転送命令をサポートしています。

FPSCR.SZ = 1 のとき、SH7091 はペア単精度データ転送命令によるデータ転送を行えます。

- FMOV DRm/XDm, DRn/XDRn ( m, n: 0, 2, 4, 6, 8, 10, 12, 14 )
- FMOV DRm/XDm, @Rn ( m: 0, 2, 4, 6, 8, 10, 12, 14, n: 0 ~ 15 )

これらの命令により、2つの単精度 ( $2 \times 32$  ビット) データを転送することができます。したがって、データバンド幅の2倍の転送性能が得られます。

- FSCHG  
この命令はFPSCRのSZビットの値を変更します。ペア単精度データ転送を行うか行わないかを高速に切り換えることができます。

---

# 7. 命令セット

---

## 7.1 実行環境

(1) PC

PC はその命令自身の命令アドレスを示します。

データサイズとデータタイプ： SH7091 の命令セットは固定長 16 ビット命令で実現されます。SH7091 はバイト（8 ビット）、ワード（16 ビット）、ロングワード（32 ビット）、クワッドワード（64 ビット）のデータサイズでメモリにアクセスします。単精度浮動小数点データ（32 ビット）は、ロングワードまたはクワッドワードサイズでメモリとのやりとりが可能です。倍精度浮動小数点データ（64 ビット）は、ロングワードサイズでメモリとのやりとりが可能です。倍精度浮動小数点演算を指定すると（FPSCR.PR=1）、クワッドワードアクセスの演算結果は不定になります。SH7091 がバイトサイズおよびワードサイズのデータをメモリからレジスタに移動するとデータは符号拡張されます。

(2) ロード/ストアアーキテクチャ

SH7091 は基本的演算をレジスタで実行するロード/ストアアーキテクチャを特長としています。メモリで直接実行する論理 AND 演算のようなビット操作演算を除き、メモリアクセスを必要とする演算はレジスタにロードした後、レジスタで実行されます。

(3) 遅延分岐

SH7091 の分岐命令および RTE は、BF、BT の 2 つの分岐命令を除き遅延分岐です。遅延分岐上で分岐の次の命令は分岐先命令の前に実行されます。遅延分岐後のこの実行スロットは「遅延スロット」と呼ばれます。たとえば、BRA 実行シーケンスは次のとおりです。

静的シーケンス	動的シーケンス	
BRA TARGET	BRA TARGET	
ADD R1, R0 next_2	ADD R1, R0 target_instr	遅延スロットの ADD は TARGET に分岐する前に実行されます

(4) 遅延スロット

命令によっては遅延スロットで実行するとスロット不当命令例外が発生します。「第 5 章 例外処理」を参照してください。分岐が成立しなかった BF/S、BT/S の次の命令も遅延スロット命令です。

## 7. 命令セット

---

### (5) T ビット

ステータスレジスタ (SR) の T ビットは、比較演算の結果を示すために使用し、条件付き分岐命令で参照します。たとえば、以下に条件付き分岐命令例を示します。

```
ADD    #1, R0          ; T ビットは ADD 演算で変更されない。  
CMP/EQ R1, R0          ; R0=R1 のとき T ビットは 1 にセットされる。  
BT     TARGET          ; T ビット=1(R0=R1) のとき TARGET に分岐する。
```

RTE の遅延スロットで、ステータスレジスタ (SR) ビットは次のように参照されます。命令アクセスは変更の前に MD ビットを使用し、データアクセスは変更後の MD ビットにアクセスします。変更後の他の S、T、M、Q、FD、BL、RB ビットを遅延スロットの命令実行のために使用します。STC、STC.L SR 命令は、変更後すべての SR ビットにアクセスします。

### (6) 定数値

8 ビットの定数値は命令コード、イミディエイト値で指定できます。また 16 ビット、32 ビットの定数値はメモリで文字通りの定数値として定義することができ、PC 相対ロード命令で参照できます。

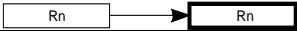
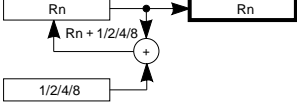
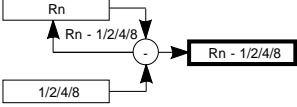
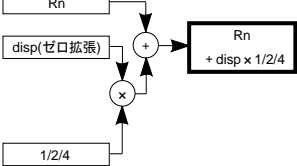
```
MOV.W   @(disp, PC), Rn  
MOV.L   @(disp, PC), Rn
```

浮動小数点に対する PC 相対ロード命令はありません。ただし、単精度浮動小数点レジスタに対して FLDI0、FLDI1 命令を使用することによって、0.0 または 1.0 にセットすることができます。

## 7.2 アドレッシングモード

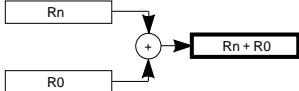
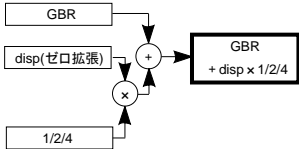
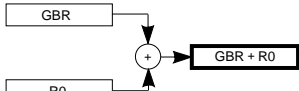
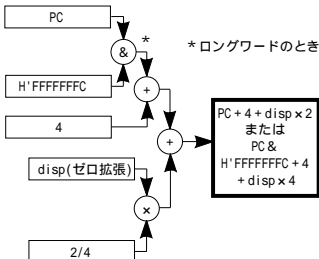
表 7.1 にアドレッシングモードと実効アドレスの計算を示します。仮想メモリ空間のある位置をアクセスすると (MMUCR.AT=1)、実効アドレスは物理メモリアドレスに変換されます。複数の仮想メモリ空間システムを選択した場合 (MMUCR.SV=0)、PTEH の最下位ビットもアクセスの ASID として参照されます。「3 章 メモリマネジメントユニット」を参照してください。

表 7.1 アドレッシングモードと実効アドレス (1)

アドレッシングモード	命令フォーマット	実効アドレスの計算方法	計算式
レジスタ直接	Rn	実効アドレスはレジスタ Rn です。 (オペランドはレジスタ Rn の内容です。)	-
レジスタ間接	@Rn	実効アドレスはレジスタ Rn の内容です。 	Rn EA (EA: 実効アドレス)
ポストインクリメントレジスタ間接	@Rn+	実効アドレスはレジスタ Rn の内容です。命令実行後 Rn に定数を加算します。定数はオペランドサイズがバイトのとき 1、ワードのとき 2、ロングワードのとき 4、クワッドワードのとき 8 です。 	Rn EA 命令実行後 バイト: Rn + 1 Rn ワード: Rn + 2 Rn ロングワード: Rn + 4 Rn クワッドワード: Rn + 8 Rn
プリデクリメントレジスタ間接	@- Rn	実効アドレスは、あらかじめ定数を減算したレジスタ Rn の内容です。定数はバイトのとき 1、ワードのとき 2、ロングワードのとき 4、クワッドワードのとき 8 です。 	バイト: Rn - 1 Rn ワード: Rn - 2 Rn ロングワード: Rn - 4 Rn クワッドワード: Rn - 8 Rn Rn EA (計算後の Rn で命令実行)
ディスプレースメント付きレジスタ間接	@ (disp:4, Rn)	実効アドレスはレジスタ Rn に 4 ビットディスプレースメント disp を加算した内容です。disp はゼロ拡張後、オペランドサイズによってバイトで 1 倍、ワードで 2 倍、ロングワードで 4 倍します。 	バイト: Rn + disp EA ワード: Rn + disp x 2 EA ロングワード: Rn + disp x 4 EA

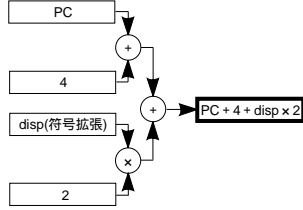
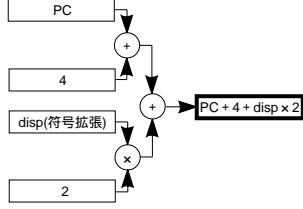
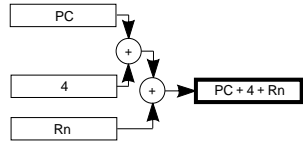
## 7. 命令セット

表 7.1 アドレッシングモードと実効アドレス (2)

アドレッシングモード	命令フォーマット	実効アドレスの計算方法	計算式
インデックス付きレジスタ間接	@ (R0, Rn)	<p>実効アドレスはレジスタ Rn に R0 を加算した内容です。</p> 	$Rn + R0$ EA
ディスプレースメント付き GBR 間接	@ (disp:8, GBR)	<p>実効アドレスはレジスタ GBR に 8 ビットディスプレースメント disp を加算した内容です。disp はゼロ拡張後、オペランドサイズによってバイトで 1 倍、ワードで 2 倍、ロングワードで 4 倍します。</p> 	バイト : $GBR + disp$ EA ワード : $GBR + disp \times 2$ EA ロングワード : $GBR + disp \times 4$ EA
インデックス付き GBR 間接	@ (R0, GBR)	<p>実効アドレスはレジスタ GBR に R0 を加算した内容です。</p> 	$GBR + R0$ EA
ディスプレースメント付き PC 相対	@ (disp:8, PC)	<p>実効アドレスは PC + 4 に 8 ビットディスプレースメント disp を加算した内容です。disp はゼロ拡張後、オペランドサイズによってワードで 2 倍、ロングワードで 4 倍します。さらにロングワードのときは PC の下位 2 ビットをマスクします。</p> 	ワード : $PC + 4 + disp \times 2$ EA ロングワード : $PC \& H'FFFFFFFC + 4 + disp \times 4$ EA

( 続く )

表 7.1 アドレッシングモードと実効アドレス (3)

アドレッシングモード	命令フォーマット	実効アドレスの計算方法	計算式
PC 相対	disp:8	<p>実効アドレスは PC + 4 に 8 ビットディスプレースメント disp を符号拡張後 2 倍し、加算した内容です。</p> 	PC + 4 + disp × 2 Branch-Target
	disp:12	<p>実効アドレスは PC + 4 に 12 ビットディスプレースメント disp を符号拡張後 2 倍し、加算した内容です。</p> 	PC + 4 + disp × 2 Branch-Target
	Rn	<p>実効アドレスは PC + 4 に Rn を加算した内容です。</p> 	PC + 4 + Rn Branch-Target
イミディエイト	#imm:8	TST, AND, OR, XOR 命令の 8 ビットイミディエイト imm はゼロ拡張します。	
	#imm:8	MOV, ADD, CMP/EQ 命令の 8 ビットイミディエイト imm は符号拡張します。	
	#imm:8	TRAPA 命令の 8 ビットイミディエイト imm はゼロ拡張後、4 倍します。	

【注】 下記のディスプレースメント (disp) を伴うアドレッシングモードにおいて、本マニュアルのアセンブラ記述は、オペランドサイズに応じたスケーリング (×1、×2、×4) を行う前の値を書いています。これは、LSI の動作を明確にするためで、実際のアセンブラの記述は、各アセンブラの表記ルールを参照してください。

@ ( disp:4, Rn ) ;ディスプレースメント付きレジスタ間接  
 @ ( disp:8, GBR ) ;ディスプレースメント付き GBR 間接  
 @ ( disp:8, PC ) ;ディスプレースメント付き PC 相対  
 disp : 8, disp :12 ;PC 相対

## 7.3 命令セット

SH 命令の次のリストに使用する表記を表 7.2 に示します。

表 7.2 命令リストの表記

項目	フォーマット	説明
命令ニーモニック	OP.Sz SRC,DEST	OP : オペレーションコード Sz : サイズ SRC : ソースオペランド DEST : ソースおよび / またはデスティネーションオペランド
演算の要約		、 転送方向 (xx) メモリオペランド M/Q/T SR のフラグビット & 各ビットの論理積   各ビットの論理和 ^ 各ビット排他的論理和 ~ 各ビットの論理否定 <<n, >>n n ビットシフト
命令コード	MSB LSB	mmmm : レジスタ番号 (Rm, FRm) nnnn : レジスタ番号 (Rn, FRn) 0000 : R0, FR0 0001 : R1, FR1 : 1111 : R15, FR15 mmm : レジスタ番号 (DRm, XDm, Rm_BANK) nnn : レジスタ番号 (DRm, XDm, Rn_BANK) 000 : DR0, XD0, R0_BANK 001 : DR2, XD2, R1_BANK : 111 : DR14, XD14, R7_BANK mm : レジスタ番号 (FVn) nn : レジスタ番号 (FVn) 00 : FV0 01 : FV4 10 : FV8 11 : FV12 iiii : イミディエイト値 dddd : ディスプレースメント
特権モード		「特権」と記載してある場合、特権モードでのみ実行可能です。
T ビット	命令実行後の T ビットの値	- : 変更なし

【注】スケーリング (x1、x2、x4、x8) は命令オペランドのサイズに応じて実行されます。

表 7.3 固定小数点転送命令

命令	動作	命令コード	特権	T ビット
MOV #imm,Rn	imm 符号拡張 Rn	1110nnnniiiiiii		
MOV.W @(disp,PC),Rn	(disp × 2+PC+4) 符号拡張 Rn	1001nnnnddddddd		
MOV.L @(disp,PC),Rn	(disp × 4+PC&H'FFFFFFC+4) Rn	1101nnnnddddddd		
MOV Rm,Rn	Rm Rn	0110nnnnmmmm0011		
MOV.B Rm,@Rn	Rm (Rn)	0010nnnnmmmm0000		
MOV.W Rm,@Rn	Rm (Rn)	0010nnnnmmmm0001		
MOV.L Rm,@Rn	Rm (Rn)	0010nnnnmmmm0010		
MOV.B @Rm,Rn	(Rm) 符号拡張 Rn	0110nnnnmmmm0000		
MOV.W @Rm,Rn	(Rm) 符号拡張 Rn	0110nnnnmmmm0001		
MOV.L @Rm,Rn	(Rm) Rn	0110nnnnmmmm0010		
MOV.B Rm,@-Rn	Rn-1 Rn, Rm (Rn)	0010nnnnmmmm0100		
MOV.W Rm,@-Rn	Rn-2 Rn, Rm (Rn)	0010nnnnmmmm0101		
MOV.L Rm,@-Rn	Rn-4 Rn, Rm (Rn)	0010nnnnmmmm0110		
MOV.B @Rm+,Rn	(Rm) 符号拡張 Rn, Rm+1 Rm	0110nnnnmmmm0100		
MOV.W @Rm+,Rn	(Rm) 符号拡張 Rn, Rm+2 Rm	0110nnnnmmmm0101		
MOV.L @Rm+,Rn	(Rm) Rn, Rm+4 Rm	0110nnnnmmmm0110		
MOV.B R0,@(disp,Rn)	R0 (disp+Rn)	10000000nnnnddd		
MOV.W R0,@(disp,Rn)	R0 (disp × 2+Rn)	10000001nnnnddd		
MOV.L Rm,@(disp,Rn)	Rm (disp × 4+Rn)	0001nnnnmmmmddd		
MOV.B @(disp,Rm),R0	(disp+Rm) 符号拡張 R0	10000100mmmmddd		
MOV.W @(disp,Rm),R0	(disp × 2+Rm) 符号拡張 R0	10000101mmmmddd		
MOV.L @(disp,Rm),Rn	(disp × 4+Rm) Rn	0101nnnnmmmmddd		
MOV.B Rm,@(R0,Rn)	Rm (R0+Rn)	0000nnnnmmmm0100		
MOV.W Rm,@(R0,Rn)	Rm (R0+Rn)	0000nnnnmmmm0101		
MOV.L Rm,@(R0,Rn)	Rm (R0+Rn)	0000nnnnmmmm0110		
MOV.B @(R0,Rm),Rn	(R0+Rm) 符号拡張 Rn	0000nnnnmmmm1100		
MOV.W @(R0,Rm),Rn	(R0+Rm) 符号拡張 Rn	0000nnnnmmmm1101		
MOV.L @(R0,Rm),Rn	(R0+Rm) Rn	0000nnnnmmmm1110		
MOV.B R0,@(disp,GBR)	R0 (disp+GBR)	11000000ddddddd		
MOV.W R0,@(disp,GBR)	R0 (disp × 2+GBR)	11000001ddddddd		
MOV.L R0,@(disp,GBR)	R0 (disp × 4+GBR)	11000010ddddddd		
MOV.B @(disp,GBR),R0	(disp+GBR) 符号拡張 R0	11000100ddddddd		
MOV.W @(disp,GBR),R0	(disp × 2+GBR) 符号拡張 R0	11000101ddddddd		
MOV.L @(disp,GBR),R0	(disp × 4+GBR) R0	11000110ddddddd		
MOVA @(disp,PC),R0	disp × 4+PC&H'FFFFFFC+4 R0	11000111ddddddd		
MOVT Rn	T Rn	0000nnnn00101001		
SWAP.B Rm,Rn	Rm 下位 2 バイトの 上下バイト交換 Rn	0110nnnnmmmm1000		
SWAP.W Rm,Rn	Rm 上下ワード交換 Rn	0110nnnnmmmm1001		
XTRCT Rm,Rn	Rm:Rn の中央 32 ビット Rn	0010nnnnmmmm1101		



## 7. 命令セット

表 7.4 算術演算命令 (1)

命令	動作	命令コード	特権	T ビット
ADD Rm,Rn	Rn+Rm Rn	0011nnnnmmmm1100		
ADD #imm,Rn	Rn+imm Rn	0111nnnniiiiiii		
ADDC Rm,Rn	Rn+Rm+T Rn, キャリ T	0011nnnnmmmm1110		キャリ
ADDV Rm,Rn	Rn+Rm Rn, オーバフロー T	0011nnnnmmmm1111		オーバ フロー
CMP/EQ #imm,R0	R0=imm のとき 1 T それ以外るとき 0 T	10001000iiiiiii		比較 結果
CMP/EQ Rm,Rn	Rn=Rm のとき 1 T それ以外るとき 0 T	0011nnnnmmmm0000		比較 結果
CMP/HS Rm,Rn	無符号で Rn Rm のとき 1 T それ以外るとき 0 T	0011nnnnmmmm0010		比較 結果
CMP/GE Rm,Rn	有符号で Rn Rm のとき 1 T それ以外るとき 0 T	0011nnnnmmmm0011		比較 結果
CMP/HI Rm,Rn	無符号で Rn>Rm のとき 1 T それ以外るとき 0 T	0011nnnnmmmm0110		比較 結果
CMP/GT Rm,Rn	有符号で Rn>Rm のとき 1 T それ以外るとき 0 T	0011nnnnmmmm0111		比較 結果
CMP/PZ Rn	Rn 0 のとき 1 T それ以外るとき 0 T	0100nnnn00010001		比較 結果
CMP/PL Rn	Rn>0 のとき 1 T それ以外るとき 0 T	0100nnnn00010101		比較 結果
CMP/STR Rm,Rn	いずれかのバイトが等しいとき 1 T それ以外るとき 0 T	0010nnnnmmmm1100		比較 結果
DIV1 Rm,Rn	1 ステップ除算 (Rn÷Rm)	0011nnnnmmmm0100		計算 結果
DIV0S Rm,Rn	Rn の MSB Q, Rm の MSB M, M^Q T	0010nnnnmmmm0111		計算 結果
DIV0U	0 M/Q/T	000000000011001		0
DMULS.L Rm,Rn	符号付きで Rn×Rm MAC, 32×32 64 ビット	0011nnnnmmmm1101		
DMULU.L Rm,Rn	符号なしで Rn×Rm MAC, 32×32 64 ビット	0011nnnnmmmm0101		
DT Rn	Rn-1 Rn, Rn が 0 のとき 1 T Rn が 0 以外るとき 0 T	0100nnnn00010000		比較 結果
EXTS.B Rm,Rn	Rm をバイトから符号拡張 Rn	0110nnnnmmmm1110		
EXTS.W Rm,Rn	Rm をワードから符号拡張 Rn	0110nnnnmmmm1111		
EXTU.B Rm,Rn	Rm をバイトからゼロ拡張 Rn	0110nnnnmmmm1100		
EXTU.W Rm,Rn	Rm をワードからゼロ拡張 Rn	0110nnnnmmmm1101		
MAC.L @Rm+,@Rn+	符号付きで (Rn)×(Rm)+MAC MAC Rn+4 Rn, Rm+4 Rm 32×32+64 64 ビット	0000nnnnmmmm1111		

表 7.4 算術演算命令 (2)

命令	動作	命令コード	特権	T ビット
MAC.W      @Rm+, @Rn+	符号付きで (Rn) × (Rm) + MAC    MAC Rn+2   Rn, Rm+2   Rm 16 × 16 + 64   64 ビット	0100nnnnnnmmmm1111		
MUL.L      Rm, Rn	Rn × Rm    MACL 32 × 32   32 ビット	0000nnnnnnmmmm0111		
MULS.W     Rm, Rn	符号付きで Rn × Rm    MACL 16 × 16   32 ビット	0010nnnnnnmmmm1111		
MULU.W     Rm, Rn	符号なしで Rn × Rm    MACL 16 × 16   32 ビット	0010nnnnnnmmmm1110		
NEG        Rm, Rn	0-Rm   Rn	0110nnnnnnmmmm1011		
NEGC       Rm, Rn	0-Rm-T   Rn, ポロー   T	0110nnnnnnmmmm1010		ポロー
SUB        Rm, Rn	Rn-Rm   Rn	0011nnnnnnmmmm1000		
SUBC       Rm, Rn	Rn-Rm-T   Rn, ポロー   T	0011nnnnnnmmmm1010		ポロー
SUBV       Rm, Rn	Rn-Rm   Rn, アンダフロー   T	0011nnnnnnmmmm1011		アンダ フロー

表 7.5 論理演算命令

命令	動作	命令コード	特権	T ビット
AND        Rm, Rn	Rn & Rm    Rn	0010nnnnnnmmmm1001		
AND        #imm, R0	R0 & imm    R0	11001001iiiiiiiiii		
AND.B      #imm, @(R0, GBR)	(R0+GBR) & imm    (R0+GBR)	11001101iiiiiiiiii		
NOT        Rm, Rn	~Rm    Rn	0110nnnnnnmmmm0111		
OR         Rm, Rn	Rn   Rm    Rn	0010nnnnnnmmmm1011		
OR         #imm, R0	R0   imm    R0	11001011iiiiiiiiii		
OR.B       #imm, @(R0, GBR)	(R0+GBR)   imm    (R0+GBR)	11001111iiiiiiiiii		
TAS.B      @Rn	(Rn) が 0 のとき 1   T それ以外とき 0   T 両方に対して 1   (Rn) の MSB	0100nnnnn00011011		テスト 結果
TST        Rm, Rn	Rn & Rm,    結果が 0 のとき 1   T それ以外とき 0   T	0010nnnnnnmmmm1000		テスト 結果
TST        #imm, R0	R0 & imm,    結果が 0 のとき 1   T それ以外とき 0   T	11001000iiiiiiiiii		テスト 結果
TST.B      #imm, @(R0, GBR)	(R0+GBR) & imm, 結果が 0 のとき 1   T それ以外とき 0   T	11001100iiiiiiiiii		テスト 結果
XOR        Rm, Rn	Rn ^ Rm    Rn	0010nnnnnnmmmm1010		
XOR        #imm, R0	R0 ^ imm    R0	11001010iiiiiiiiii		
XOR.B      #imm, @(R0, GBR)	(R0+GBR) ^ imm    (R0+GBR)	11001110iiiiiiiiii		

## 7. 命令セット

表 7.6 シフト命令

命令	動作	命令コード	特権	T ビット
ROTL Rn	T Rn MSB	0100nnnn00000100		MSB
ROTR Rn	LSB Rn T	0100nnnn00000101		LSB
ROTCL Rn	T Rn T	0100nnnn00100100		MSB
ROTCR Rn	T Rn T	0100nnnn00100101		LSB
SHAD Rm, Rn	Rm 0 のとき Rn<<Rm Rn, Rm<0 のとき Rn>>Rm [MSB Rn]	0100nnnnmmmm1100		
SHAL Rn	T Rn 0	0100nnnn00100000		MSB
SHAR Rn	MSB Rn T	0100nnnn00100001		LSB
SHLD Rm, Rn	Rm 0 のとき Rn<<Rm Rn, Rm<0 のとき Rn>>Rm [0 Rn]	0100nnnnmmmm1101		
SHLL Rn	T Rn 0	0100nnnn00000000		MSB
SHLR Rn	0 Rn T	0100nnnn00000001		LSB
SHLL2 Rn	Rn<<2 Rn	0100nnnn00001000		
SHLR2 Rn	Rn>>2 Rn	0100nnnn00001001		
SHLL8 Rn	Rn<<8 Rn	0100nnnn00011000		
SHLR8 Rn	Rn>>8 Rn	0100nnnn00011001		
SHLL16 Rn	Rn<<16 Rn	0100nnnn00101000		
SHLR16 Rn	Rn>>16 Rn	0100nnnn00101001		

表 7.7 分岐命令

命令	動作	命令コード	特権	T ビット
BF label	T=0 のとき disp×2+PC+4 PC, T=1 のとき nop	10001011dddddddd		
BF/S label	遅延分岐, T=0 のとき disp×2+PC+4 PC, T=1 のとき nop	10001111dddddddd		
BT label	T=1 のとき disp×2+PC+4 PC, T=0 のとき nop	10001001dddddddd		
BT/S label	遅延分岐, T=1 のとき disp× 2+PC+4 PC, T=0 のとき nop	10001101dddddddd		
BRA label	遅延分岐, disp×2+PC+4 PC	1010dddddddddddd		
BRAF Rn	遅延分岐, Rn+PC+4 PC	0000nnnn00100011		
BSR label	遅延分岐, PC+4 PR, disp×2+PC+4 PC	1011dddddddddddd		
BSRF Rn	遅延分岐, PC+4 PR, Rn+PC+4 PC	0000nnnn00000011		
JMP @Rn	遅延分岐, Rn PC	0100nnnn00101011		
JSR @Rn	遅延分岐, PC+4 PR, Rn PC	0100nnnn00001011		
RTS	遅延分岐, PR PC	000000000001011		

表 7.8 システム制御命令 (1)

命令	動作	命令コード	特権	T ビット
CLRMACH	0 MACH,MACL	0000000000101000		
CLRS	0 S	0000000001001000		
CLRT	0 T	0000000000001000		0
LDC Rm,SR	Rm SR	0100mmmm00001110	特権	LSB
LDC Rm,GBR	Rm GBR	0100mmmm00011110		
LDC Rm,VBR	Rm VBR	0100mmmm00101110	特権	
LDC Rm,SSR	Rm SSR	0100mmmm00111110	特権	
LDC Rm,SPC	Rm SPC	0100mmmm01001110	特権	
LDC Rm,DBR	Rm DBR	0100mmmm11111010	特権	
LDC Rm,Rn_BANK	Rm Rn_BANK(n=0~7)	0100mmmm1nnn1110	特権	
LDC.L @Rm+,SR	(Rm) SR, Rm+4 Rm	0100mmmm00000111	特権	LSB
LDC.L @Rm+,GBR	(Rm) GBR, Rm+4 Rm	0100mmmm00010111		
LDC.L @Rm+,VBR	(Rm) VBR, Rm+4 Rm	0100mmmm00100111	特権	
LDC.L @Rm+,SSR	(Rm) SSR, Rm+4 Rm	0100mmmm00110111	特権	
LDC.L @Rm+,SPC	(Rm) SPC, Rm+4 Rm	0100mmmm01000111	特権	
LDC.L @Rm+,DBR	(Rm) DBR, Rm+4 Rm	0100mmmm11110110	特権	
LDC.L @Rm+,Rn_BANK	(Rm) Rn_BANK, Rm+4 Rm	0100mmmm1nnn0111	特権	
LDS Rm,MACH	Rm MACH	0100mmmm00001010		
LDS Rm,MACL	Rm MACL	0100mmmm00011010		
LDS Rm,PR	Rm PR	0100mmmm00101010		
LDS.L @Rm+,MACH	(Rm) MACH, Rm+4 Rm	0100mmmm00000110		
LDS.L @Rm+,MACL	(Rm) MACL, Rm+4 Rm	0100mmmm00010110		
LDS.L @Rm+,PR	(Rm) PR, Rm+4 Rm	0100mmmm00100110		
LDTLB	PTEH/PTEL TLB	000000000111000	特権	
MOVCA.L R0,@Rn	(キャッシュブロックをフェッチせず)R0 (Rn)	0000nnnn11000011		
NOP	無操作	0000000000001001		
OCBI @Rn	オペランドキャッシュブロックを無効にする	0000nnnn10010011		
OCBP @Rn	オペランドキャッシュブロックをライトバックし無効にする	0000nnnn10100011		
OCBWB @Rn	オペランドキャッシュブロックをライトバックする	0000nnnn10110011		
PREF @Rn	(Rn) オペランドキャッシュ	0000nnnn10000011		
RTE	遅延分岐, SSR/SPC SR/PC	000000000101011	特権	
SETS	1 S	0000000001011000		
SETT	1 T	0000000000011000		1
SLEEP	スリープもしくはスタンバイ	0000000000011011	特権	

## 7. 命令セット

表 7.8 システム制御命令 (2)

命令	動作	命令コード	特権	T ビット
STC SR,Rn	SR Rn	0000nnnn00000010	特権	
STC GBR,Rn	GBR Rn	0000nnnn00010010		
STC VBR,Rn	VBR Rn	0000nnnn00100010	特権	
STC SSR, Rn	SSR Rn	0000nnnn00110010	特権	
STC SPC,Rn	SPC Rn	0000nnnn01000010	特権	
STC SGR,Rn	SGR Rn	0000nnnn00111010	特権	
STC DBR,Rn	DBR Rn	0000nnnn11111010	特権	
STC Rm_BANK,Rn	Rm_BANK Rn (m=0 ~ 7)	0000nnnn1mmm0010	特権	
STC.L SR,@-Rn	Rn-4 Rn, SR (Rn)	0100nnnn00000011	特権	
STC.L GBR,@-Rn	Rn-4 Rn, GBR (Rn)	0100nnnn00010011		
STC.L VBR,@-Rn	Rn-4 Rn, VBR (Rn)	0100nnnn00100011	特権	
STC.L SSR,@-Rn	Rn-4 Rn, SSR (Rn)	0100nnnn00110011	特権	
STC.L SPC,@-Rn	Rn-4 Rn, SPC (Rn)	0100nnnn01000011	特権	
STC.L SGR,@-Rn	Rn-4 Rn, SGR (Rn)	0100nnnn00110010	特権	
STC.L DBR,@-Rn	Rn-4 Rn, DBR (Rn)	0100nnnn11110010	特権	
STC.L Rm_BANK,@-Rn	Rn-4 Rn, Rm_BANK (Rn) (m=0 ~ 7)	0100nnnn1mmm0011	特権	
STS MACH,Rn	MACH Rn	0000nnnn00001010		
STS MACL,Rn	MACL Rn	0000nnnn00011010		
STS PR,Rn	PR Rn	0000nnnn00101010		
STS.L MACH,@-Rn	Rn-4 Rn, MACH (Rn)	0100nnnn00000010		
STS.L MACL,@-Rn	Rn-4 Rn, MACL (Rn)	0100nnnn00010010		
STS.L PR,@-Rn	Rn-4 Rn, PR (Rn)	0100nnnn00100010		
TRAPA #imm	PC+2 SPC, SR SSR, #imm <<2 TRA, H'160 EXPEVT, VBR+ H'0100 PC	1100001111111111		

表 7.9 浮動小数点単精度命令

命令		動作	命令コード	特権	T ビット
FLDI0	FRn	H'00000000 FRn	1111nnnn10001101		
FLDI1	FRn	H'3F800000 FRn	1111nnnn10011101		
FMOV	FRm, FRn	FRm FRn	1111nnnnnnmm1100		
FMOV.S	@Rm, FRn	(Rm) FRn	1111nnnnnnmm1000		
FMOV.S	@(R0,Rm),FRn	(R0 + Rm) FRn	1111nnnnnnmm0110		
FMOV.S	@Rm+,FRn	(Rm) FRn,Rm+4 Rm	1111nnnnnnmm1001		
FMOV.S	FRm, @Rn	FRm (Rn)	1111nnnnnnmm1010		
FMOV.S	FRm, @-Rn	Rn-4 Rn, FRm (Rn)	1111nnnnnnmm1011		
FMOV.S	FRm, @(R0,Rn)	FRm (R0+Rn)	1111nnnnnnmm0111		
FMOV	DRm, DRn	DRm DRn	1111nnnn0mmm01100		
FMOV	@Rm, DRn	(Rm) DRn	1111nnnn0mmm1000		
FMOV	@(R0,Rm),DRn	(R0 + Rm) DRn	1111nnnn0mmm0110		
FMOV	@Rm+,DRn	(Rm) DRn,Rm+8 Rm	1111nnnn0mmm1001		
FMOV	DRm, @Rn	DRm (Rn)	1111nnnnnnmm01010		
FMOV	DRm, @-Rn	Rn-8 Rn,DRm (Rn)	1111nnnnnnmm01011		
FMOV	DRm, @(R0,Rn)	DRm (R0+Rn)	1111nnnnnnmm00111		
FLDS	FRm,FPUL	FRm FPUL	1111mmmm00011101		
FSTS	FPUL, FRn	FPUL FRn	1111nnnn00001101		
FABS	FRn	FRn & H'7FFF FFFF FRn	1111nnnn01011101		
FADD	FRm, FRn	FRn + FRm FRn	1111nnnnnnmm0000		
FCMP/EQ	FRm, FRn	FRn = FRm のとき 1 T それ以外のとき 0 T	1111nnnnnnmm0100		比較 結果
FCMP/GT	FRm, FRn	FRn > FRm のとき 1 T それ以外のとき 0 T	1111nnnnnnmm0101		比較 結果
FDIV	FRm, FRn	FRn / FRm FRn	1111nnnnnnmm0011		
FLOAT	FPUL, FRn	(float)FPUL FRn	1111nnnn00101101		
FMAC	FR0, FRm, FRn	FR0 * FRm + FRn FRn	1111nnnnnnmm1110		
FMUL	FRm, FRn	FRn * FRm FRn	1111nnnnnnmm0010		
FNEG	FRn	FRn ^ H'80000000 FRn	1111nnnn01001101		
FSQRT	FRn	$\sqrt{\text{FRn}}$ FRn	1111nnnn01101101		
FSUB	FRm, FRn	FRn - FRm FRn	1111nnnnnnmm0001		
FTRC	FRm, FPUL	(long)FRm FPUL	1111mmmm00111101		

## 7. 命令セット

表 7.10 浮動小数点倍精度命令

命令	動作	命令コード	特権	T ビット
FABS DRn	DRn & H'7FFF FFFF FFFF FFFF DRn	1111nnnn001011101		
FADD DRm, DRn	DRn + DRm DRn	1111nnnn0mmm00000		
FCMP/EQ DRm, DRn	DRn = DRm のとき 1 T それ以外るとき 0 T	1111nnnn0mmm00100		比較 結果
FCMP/GT DRm, DRn	DRn > DRm のとき 1 T それ以外るとき 0 T	1111nnnn0mmm00101		比較 結果
FDIV DRm, DRn	DRn / DRm DRn	1111nnnn0mmm00011		
FCNVDS DRm, FPUL	double_to_float[DRm] FPUL	1111mmmm010111101		
FCNVSD FPUL, DRn	float_to_double[FPUL] DRn	1111nnnn010101101		
FLOAT FPUL, DRn	(float)FPUL DRn	1111nnnn000101101		
FMUL DRm, DRn	DRn * DRm DRn	1111nnnn0mmm00010		
FNEG DRn	DRn ^ H'8000 0000 0000 0000 DRn	1111nnnn001001101		
FSQRT DRn	$\sqrt{\text{DRn}}$ DRn	1111nnnn001101101		
FSUB DRm, DRn	DRn - DRm DRn	1111nnnn0mmm00001		
FTRC DRm, FPUL	(long)DRm FPUL	1111mmmm000111101		

表 7.11 浮動小数点制御命令

命令	動作	命令コード	特権	T ビット
LDS Rm, FPSCR	Rm FPSCR	0100mmmm01101010		
LDS Rm, FPUL	Rm FPUL	0100mmmm01011010		
LDS.L @Rm+, FPSCR	(Rm) FPSCR, Rm+4 Rm	0100mmmm01100110		
LDS.L @Rm+, FPUL	(Rm) FPUL, Rm+4 Rm	0100mmmm01010110		
STS FPSCR, Rn	FPSCR Rn	0000nnnn01101010		
STS FPUL, Rn	FPUL Rn	0000nnnn01011010		
STS.L FPSCR, @-Rn	Rn-4 Rn, FPSCR (Rn)	0100nnnn01100010		
STS.L FPUL, @-Rn	Rn-4 Rn, FPUL (Rn)	0100nnnn01010010		

表 7.12 浮動小数点グラフィック強化命令

命令	動作	命令コード	特権	T ビット
FMOV DRm ,XDn	DRm XDn	1111nnn1mmm01100		
FMOV XDm ,DRn	XDm DRn	1111nnn0mmm11100		
FMOV XDm ,XDn	XDm XDn	1111nnn1mmm11100		
FMOV @Rm, XDn	(Rm) XDn	1111nnn1mmm1000		
FMOV @Rm+, XDn	(Rm) XDn, Rm+8 Rm	1111nnn1mmm1001		
FMOV @(R0,Rm),DRn	(R0 + Rm) DRn	1111nnn1mmm0110		
FMOV XDm ,@Rn	XDm (Rn)	1111nnnnmmm11010		
FMOV XDm ,@-Rn	Rn-8 Rn,XDm (Rn)	1111nnnnmmm11011		
FMOV XDm ,@(R0,Rn)	XDm (R0+Rn)	1111nnnnmmm10111		
FIPR FVm ,FVn	inner_product[FVm, FVn] FR[n+3]	1111nnnm11101101		
FTRV XMTRX ,FVn	transform_vector[XMTRX, FVn] FVn	1111nn0111111101		
FRCHG	~ FPSCR.FR FPSCR.FR	1111101111111101		
FSCHG	~ FPSCR.SZ FPSCR.SZ	1111001111111101		



## 8. パイプライン動作

SH7091 は 2 命令並列型 (2-ILP, Instruction-Level-Parallelism) のスーパスカラパイプライン処理マイクロプロセッサです。命令実行はパイプライン化され、2 つの命令を並行して実行できます。実行サイクルはプロセッサの実装方法によって異なります。本章での定義は SH7091 以外の SH-4 シリーズの他のモデルには適用できない場合があります。

### 8.1 パイプライン

図 8.1 に基本パイプラインを示します。通常、パイプラインは命令フェッチ (I)、デコード・レジスタリード (D)、実行 (EX, SX, F0、F1、F2、または F3)、データアクセス (NA、または MA)、ライトバック (S、または FS) の 5、または 6 ステージから構成されます。1 つの命令は基本パイプラインの組み合わせとして実行されます。図 8.2 に命令実行パターンを示します。



図 8.1 基本パイプライン

## 8. パイプライン動作

### (1) 1ステップ演算 ; 1 発行サイクル

EXT[SU].[BW], MOV, MOV#, MOVA, MOVT, SWAP.[BW], XTRCT,  
ADD\*, CMP\*, DIV\*, DT, NEG\*, SUB\*,  
AND, AND#, NOT, OR, OR#, TST, TST#, XOR, XOR#,  
ROT\*, SHA\*, SHL\*, BF\*, BT\*, BRA,  
NOP, CLRS, CLRT, SETS, SETT,  
FPULへのLDS, FPUL/FPSCRからのSTS,  
FLDI0, FLDI1, FMOV, FLDS, FSTS,  
単精度 / 倍精度 FABS/FNEG

I	D	EX	NA	S
---	---	----	----	---

### (2) ロード / ストア ; 1発行サイクル

MOV.[BWL], FMOV\*@, FPULへのLDS.L, LDTLB, PREF,  
FPUL/FPSCRからのSTS.L

I	D	EX	MA	S
---	---	----	----	---

### (3) GBRベースロード / ストア ; 1発行サイクル

MOV.[BWL]@(d,GBR)

I	D	SX	MA	S
---	---	----	----	---

### (4) JMP, RTS, BRAF ; 2発行サイクル

I	D	EX	NA	S	
		D	EX	NA	S

### (5) TST.B ; 3発行サイクル

I	D	SX	MA	S		
		D	SX	NA	S	
			D	SX	NA	S

### (6) AND.B, OR.B, XOR.B ; 4発行サイクル

I	D	SX	MA	S				
		D	SX	NA	S			
			D	SX	NA	S		
				D	SX	MA	S	

### (7) TAS.B ; 5発行サイクル

I	D	EX	MA	S					
		D	EX	MA	S				
			D	EX	NA	S			
				D	EX	NA	S		
					D	EX	MA	S	

### (8) RTE ; 5発行サイクル

I	D	EX	NA	S					
		D	EX	NA	S				
			D	EX	NA	S			
				D	EX	NA	S		
					D	EX	NA	S	
						D	EX	NA	S

### (9) SLEEP ; 4発行サイクル

I	D	EX	NA	S				
		D	EX	NA	S			
			D	EX	NA	S		
				D	EX	NA	S	

図 8.2 命令実行パターン (1)

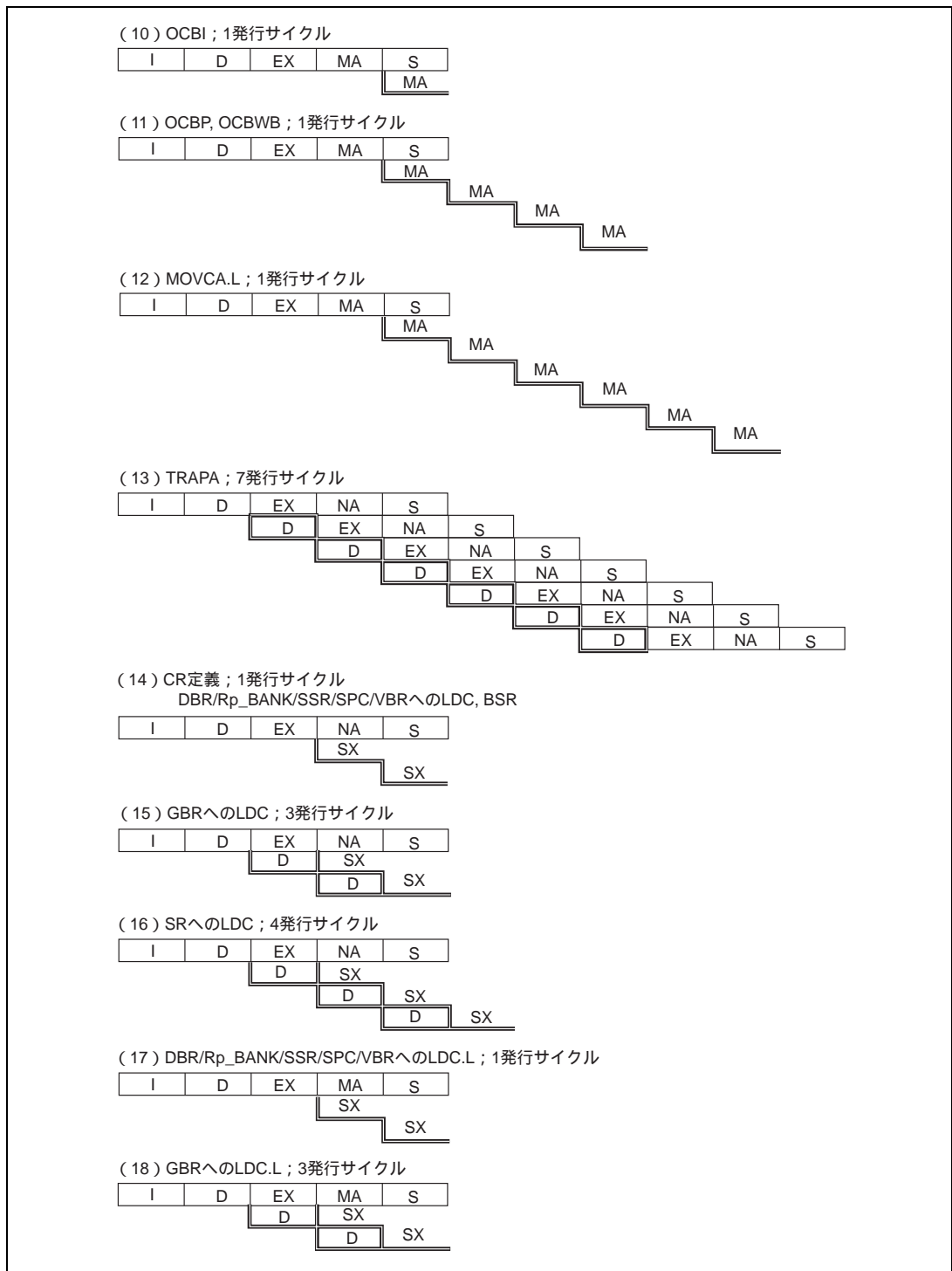
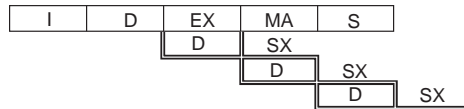


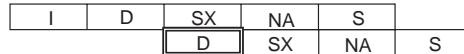
図 8.2 命令実行パターン (2)

## 8. パイプライン動作

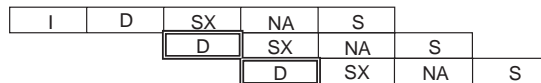
(19) SRへのLDC.L ; 4発行サイクル



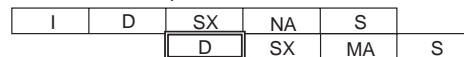
(20) DBR/GBR/Rp\_BANK/SR/SSR/SPC/VBRからのSTC ; 2発行サイクル



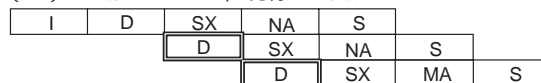
(21) SGRからのSTC ; 3発行サイクル



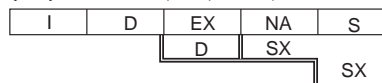
(22) DBR/GBR/Rp\_BANK/SR/SSR/SPC/VBRからのSTC.L ; 2発行サイクル



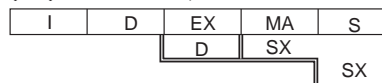
(23) SGRからのSTC.L ; 3発行サイクル



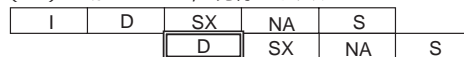
(24) PRへのLDS,JSR,BSRF ; 2発行サイクル



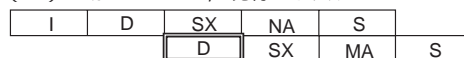
(25) PRへのLDS.L ; 2発行サイクル



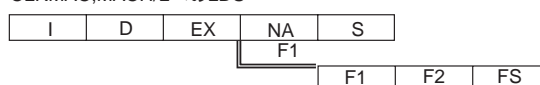
(26) PRからのSTS ; 2発行サイクル



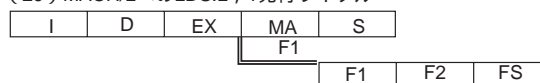
(27) PRからのSTS.L ; 2発行サイクル



(28) MACH/L 定義;1発行サイクル  
CLRMAC,MACH/LへのLDS



(29) MACH/LへのLDS.L ; 1発行サイクル



(30) MACH/LからのSTS ; 1発行サイクル

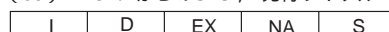


図 8.2 命令実行パターン (3)

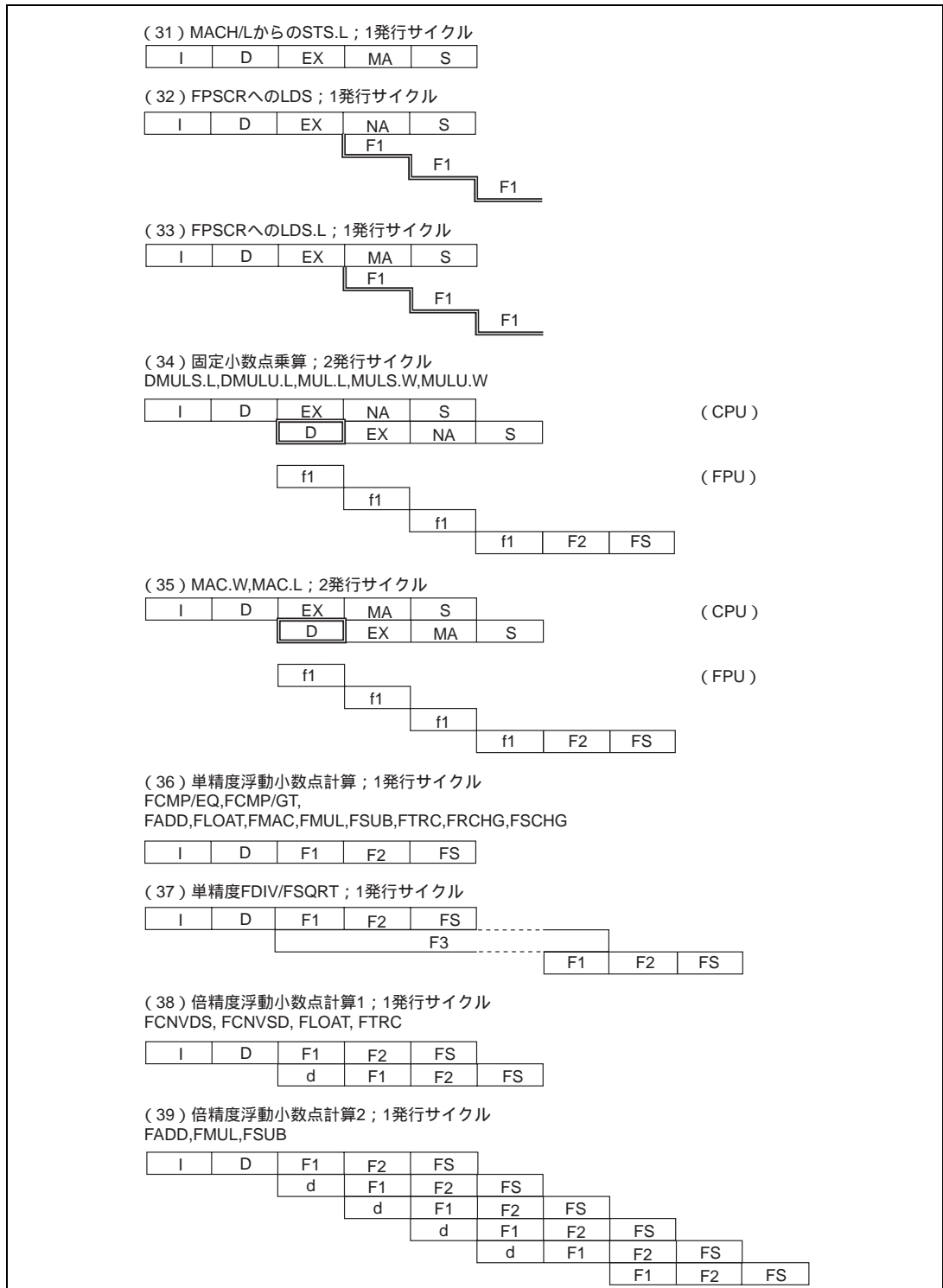


図 8.2 命令実行パターン (4)

## 8. パイプライン動作

(40) 倍精度FCMP ; 2発行サイクル  
FCMP/EQ,FCMP/GT

I	D	F1	F2	FS	
		D	F1	F2	FS

(41) 倍精度FDIV/FSQRT ; 1発行サイクル  
FDIV, FSQRT

I	D	F1	F2	FS					
		d	F1	F2					
		F3							
					F1	F2	FS		
						F1	F2	FS	
							F1	F2	FS

(42) FIPR ; 1発行サイクル

I	D	F0	F1	F2	FS
---	---	----	----	----	----

(43) FTRV ; 1発行サイクル

I	D	F0	F1	F2	FS				
		d	F0	F1	F2	FS			
			d	F0	F1	F2	FS		
				d	F0	F1	F2	FS	

- 【注】
- |    |  |
|----|--|
| ?? | 2つの命令が並列実行される場合を除き、同種のステージとのオーバーラップ不可能   |
| D  | Dステージをロック                                |
| d  | レジスタ読み出しのみ                               |
| ?? | ロックするが演算は実行しない                           |
| f1 | 別の"f1"とはオーバーラップ可能であるが、別の"F1"とはオーバーラップ不可能 |

図 8.2 命令実行パターン (5)

## 8.2 並列実行性

表 8.1 に示すように、命令は利用する内部機能ブロックにより 6 つのグループに分類されます。表 8.2 に並列実行可能な 2 つの命令の組み合わせをグループごとに示します。たとえば、EX グループに分類された ADD と BR グループの BRA は並列実行できます。

表 8.1 命令グループ (1)

### (1) MT グループ

CLRT		CMP/Hi	Rm,Rn	MOV	Rm,Rn
CMP/EQ	#imm,R0	CMP/HS	Rm,Rn	NOP	
CMP/EQ	Rm,Rn	CMP/PL	Rn	SETT	
CMP/GE	Rm,Rn	CMP/PZ	Rn	TST	#imm,R0
CMP/GT	Rm,Rn	CMP/STR	Rm,Rn	TST	Rm,Rn

### (2) EX グループ

ADD	#imm,Rn	MOVt	Rn	SHLL2	Rn
ADD	Rm,Rn	NEG	Rm,Rn	SHLL8	Rn
ADDC	Rm,Rn	NEGC	Rm,Rn	SHLR	Rn
ADDV	Rm,Rn	NOT	Rm,Rn	SHLR16	Rn
AND	#imm,R0	OR	#imm,R0	SHLR2	Rn
AND	Rm,Rn	OR	Rm,Rn	SHLR8	Rn
DIV0S	Rm,Rn	ROTCL	Rn	SUB	Rm,Rn
DIV0U		ROTCR	Rn	SUBC	Rm,Rn
DIV1	Rm,Rn	ROTL	Rn	SUBV	Rm,Rn
DT	Rn	ROTR	Rn	SWAP.B	Rm,Rn
EXTS.B	Rm,Rn	SHAD	Rm,Rn	SWAP.W	Rm,Rn
EXTS.W	Rm,Rn	SHAL	Rn	XOR	#imm,R0
EXTU.B	Rm,Rn	SHAR	Rn	XOR	Rm,Rn
EXTU.W	Rm,Rn	SHLD	Rm,Rn	XTRCT	Rm,Rn
MOV	#imm,Rn	SHLL	Rn		
MOVA	@(disp,PC),R0	SHLL16	Rn		

### (3) BR グループ

BF	disp	BRA	disp	BT	disp
BF/S	disp	BSR	disp	BT/S	disp

## 8. パイプライン動作

表 8.1 命令グループ (2)

### (4) LS グループ

FABS	DRn	FMOV.S	@Rm+,FRn	MOV.L	R0,@(disp,GBR)
FABS	FRn	FMOV.S	FRm,@(R0,Rn)	MOV.L	Rm,@(disp,Rn)
FLDI0	FRn	FMOV.S	FRm,@-Rn	MOV.L	Rm,@(R0,Rn)
FLDI1	FRn	FMOV.S	FRm,@Rn	MOV.L	Rm,@-Rn
FLDS	FRm,FPUL	FNEG	DRn	MOV.L	Rm,@Rn
FMOV	@(R0,Rm),DRn	FNEG	FRn	MOV.W	@(disp,GBR),R0
FMOV	@(R0,Rm),XDn	FSTS	FPUL,FRn	MOV.W	@(disp,PC),Rn
FMOV	@Rm,DRn	LDS	Rm,FPUL	MOV.W	@(disp,Rm),R0
FMOV	@Rm,XDn	MOV.B	@(disp,GBR),R0	MOV.W	@(R0,Rm),Rn
FMOV	@Rm+,DRn	MOV.B	@(disp,Rm),R0	MOV.W	@Rm,Rn
FMOV	@Rm+,XDn	MOV.B	@(R0,Rm),Rn	MOV.W	@Rm+,Rn
FMOV	DRm,@(R0,Rn)	MOV.B	@Rm,Rn	MOV.W	R0,@(disp,GBR)
FMOV	DRm,@-Rn	MOV.B	@Rm+,Rn	MOV.W	R0,@(disp,Rn)
FMOV	DRm,@Rn	MOV.B	R0,@(disp,GBR)	MOV.W	Rm,@(R0,Rn)
FMOV	DRm,DRn	MOV.B	R0,@(disp,Rm)	MOV.W	Rm,@-Rn
FMOV	DRm,XDn	MOV.B	Rm,@(R0,Rn)	MOV.W	Rm,@Rn
FMOV	FRm,FRn	MOV.B	Rm,@-Rn	MOVCA.L	R0,@Rn
FMOV	XDm,@(R0,Rn)	MOV.B	Rm,@Rn	OCBI	@Rn
FMOV	XDm,@-Rn	MOV.L	@(disp,GBR),R0	OCBP	@Rn
FMOV	XDm,@Rn	MOV.L	@(disp,PC),Rn	OCBWB	@Rn
FMOV	XDm,DRn	MOV.L	@(disp,Rm),Rn	PREF	@Rn
FMOV	XDm,XDn	MOV.L	@(R0,Rm),Rn	STS	FPUL,Rn
FMOV.S	@(R0,Rm),FRn	MOV.L	@Rm,Rn		
FMOV.S	@Rm,FRn	MOV.L	@Rm+,Rn		

### (5) FE グループ

FADD	DRm,DRn	FIPR	FVm,FVn	FSQRT	DRn
FADD	FRm,FRn	FLOAT	FPUL,DRn	FSQRT	FRn
FCMP/EQ	FRm,FRn	FLOAT	FPUL,FRn	FSUB	DRm,DRn
FCMP/GT	FRm,FRn	FMAC	FR0,FRm,FRn	FSUB	FRm,FRn
FCNVDS	DRm,FPUL	FMUL	DRm,DRn	FTRC	DRm,FPUL
FCNVSD	FPUL,DRn	FMUL	FRm,FRn	FTRC	FRm,FPUL
FDIV	DRm,DRn	FRCHG		FTRV	XMTRX,FVn
FDIV	FRm,FRn	FSCHG			



表 8.1 命令グループ (3)

## (6) CO グループ

AND.B	#imm,@(R0,GBR)	LDS	Rm,FPSCR	STC	SR,Rn
BRAF	Rm	LDS	Rm,MACH	STC	SSR,Rn
BSRF	Rm	LDS	Rm,MACL	STC	VBR,Rn
CLRMAC		LDS	Rm,PR	STC.L	DBR,@-Rn
CLRS		LDS.L	@Rm+,FPSCR	STC.L	GBR,@-Rn
DMULS.L	Rm,Rn	LDS.L	@Rm+,FPUL	STC.L	Rp_BANK,@-Rn
DMULU.L	Rm,Rn	LDS.L	@Rm+,MACH	STC.L	SGR,@-Rn
FCMP/EQ	DRm,DRn	LDS.L	@Rm+,MACL	STC.L	SPC,@-Rn
FCMP/GT	DRm,DRn	LDS.L	@Rm+,PR	STC.L	SR,@-Rn
JMP	@Rn	LDTLB		STC.L	SSR,@-Rn
JSR	@Rn	MAC.L	@Rm+@Rn+	STC.L	VBR,@-Rn
LDC	Rm,DBR	MAC.W	@Rm+@Rn+	STS	FPSCR,Rn
LDC	Rm,GBR	MUL.L	Rm,Rn	STS	MACH,Rn
LDC	Rm,Rp_BANK	MULS.W	Rm,Rn	STS	MACL,Rn
LDC	Rm,SPC	MULU.W	Rm,Rn	STS	PR,Rn
LDC	Rm,SR	OR.B	#imm,@(R0,GBR)	STS.L	FPSCR,@-Rn
LDC	Rm,SSR	RTE		STS.L	FPUL,@-Rn
LDC	Rm,VBR	RTS		STS.L	MACH,@-Rn
LDC.L	@Rm+,DBR	SETS		STS.L	MACL,@-Rn
LDC.L	@Rm+,GBR	SLEEP		STS.L	PR,@-Rn
LDC.L	@Rm+,Rp_BANK	STC	DBR,Rn	TAS.B	@Rn
LDC.L	@Rm+,SPC	STC	GBR,Rn	TRAPA	#imm
LDC.L	@Rm+,SR	STC	Rp_BANK,Rn	TST.B	#imm,@(R0,GBR)
LDC.L	@Rm+,SSR	STC	SGR,Rn	XOR.B	#imm,@(R0,GBR)
LDC.L	@Rm+,VBR	STC	SPC,Rn		

表 8.2 並列実行性

		第 2 命令					
		MT	EX	BR	LS	FE	CO
第 1 命令	MT						×
	EX		×				×
	BR			×			×
	LS				×		×
	FE					×	×
	CO	×	×	×	×	×	×

: 並列実行可能

× : 並列実行不可能

### 8.3 実行サイクルとパイプラインストール

本プロセッサには、I クロック、B クロック、P クロックの 3 つの基準クロックがあります。各ハードウェアユニットは次のように 3 つのクロックのいずれかで動作します。

- ・ I クロック : CPU、FPU、MMU、キャッシュ
- ・ B クロック : 外部バスコントローラ
- ・ P クロック : 周辺ユニット

3 つのクロックの周波数比は、FRQCR (周波数コントロールレジスタ) によって決まります。特別の指定がない限り、この章ではマシンサイクルは I クロックを基準にします。FRQCR の詳細についてはハードウェアマニュアルの「第 10 章 クロック発振回路」を参照してください。

命令の実行サイクルを表 8.3 に示します。ただし、ここではパイプラインストールによるペナルティサイクルは考慮していません。

- ・ 発行レート : 命令の発行と次の命令の発行の間隔
- ・ レイテンシ : 命令の発行とその結果生成 (完了) の間隔
- ・ 命令実行パターン (図 8.2 を参照)
- ・ ロックしたパイプラインステージ
- ・ 命令の発行とロック開始の間隔
- ・ ロック時間 : マシンサイクル単位のロック周期

命令の実行シーケンスは、図 8.2 に示す実行パターンの組み合わせで表現します。各命令とその次の命令の間は、その発行レートのマシンサイクル数だけ離れます。通常、実行、データアクセス、ライトバックの各ステージは他の命令の同じステージとオーバーラップさせることはできません。並列実行性の条件により 2 命令が並列実行される場合のみ、例外的にオーバーラップ可能となります。この単純な例として図 8.3 の(a)~(d)を参照してください。

レイテンシは命令の発行と完了の間隔であり、また相互依存関係を持つ 2 命令の実行間隔でもあります。同時にフェッチされた 2 命令間に依存関係が存在する場合、2 命令のうち後の命令は次のサイクル数だけストールします。

- ・ フロー依存関係 (read-after-write、書き込み後の読み出し) が存在するとき (レイテンシ) サイクル
- ・ 出力依存関係 (write-after-write、書き込み後の書き込み) が存在するとき (レイテンシ - 2) サイクル
- ・ 次のような逆フロー依存関係 (write-after-read、読み出し後の書き込み) が存在するとき 1 サイクルまたは 2 サイクル
  - (a) FTRV が先行するとき 1 サイクル
  - (b) 倍精度 FADD、FSUB、FMUL が先行するとき 2 サイクル

フロー依存関係が存在する場合、連続した命令の組み合わせによりレイテンシが例外的に増加 / 減少します (図 8.3 (e))。

- ・ 浮動小数点 (FP) 計算に FP レジスタストアが続くと、FP 計算のレイテンシは 1 サイクル減少する場合があります。

- SHAD、SHLD の直前にシフト量のロードが存在すると、ロードのレイテンシは1サイクル増加します。
- FP レジスタに対するライトバックを含み、レイテンシが2サイクル未満の命令の次に倍精度 FP 命令、FIPR または FTRV が続く場合、最初の命令のレイテンシは2サイクルに増加します。

フロー依存関係によるパイプラインのストールについては、依存性をもつ命令の組み合わせや、フェッチのタイミングによって、そのサイクル数にはバリエーションが生じます。図 8.3 (e)も参照してください。

出力依存関係を持つ命令のストールサイクルについては、「レイテンシ - 2」に代入するものとして、すべてのデスティネーションオペランドのうち、最も遅いライトバックに対する最長のレイテンシを適用しなければなりません（図 8.3 (f)を参照）。ただし、FP 演算の結果を反映する FPSCR に対する出力依存関係によるストールは決して起こりません。たとえば、FDIV の次に FP レジスタ間に依存関係のない FADD が続く場合、2 つの命令が FPSCR の要因 (cause) フィールドを更新するにも関わらず、FADD はストールしません。

逆フロー依存関係は、先行する倍精度 FADD、FMUL、FSUB または FTRV とそれに続く FMOV、FLDI0、FLDI1、FABS、または FNEG の間でのみ発生する可能性があります。図 8.3 (g)を参照してください。

実行中の命令がいずれかのリソース、すなわち基本演算を行う機能ブロックをロックする場合、偶然ロックされたリソースを使用しようとしていた後続の命令はストールされなければなりません（図 8.3 (h)）。このようなストールはロックされたリソースとは無関係な命令を1つまたはそれ以上挿入し、干渉する命令を分離することによって補償することができます。たとえば、ロード命令とロードした値を参照する ADD 命令が連続している場合、依存性のない3つの命令を間に挿入することにより、ADD に対する2サイクルのストールが除かれます。このような命令スケジューリングによってソフトウェアの性能を向上させることができます。

例外発生、または外部データへのアクセス発生により他のペナルティが現れます。

- 命令 TLB ミス。ペナルティは7 CPU クロックです。
- 外部メモリに対する命令アクセス（命令キャッシュミス等）。
- 外部メモリに対するデータアクセス（オペランドキャッシュミス等）。ペナルティは2 CPU クロック + 3 バスクロックです。
- メモリ割り付けコントロールレジスタに対するデータアクセス。ペナルティは2 CPU クロック + 3 内蔵周辺クロックです。

命令 TLB ミスおよび外部命令アクセスのペナルティサイクル中、命令は発行されませんが、発行済みの命令の実行は継続されます。データアクセスに対するペナルティは、パイプラインのフリーズ、すなわち、未完了の命令の実行は要求したデータが到着するまで中断されます。命令アクセスとデータアクセスに対するペナルティサイクル数は、ユーザのメモリサブシステムに大きく依存します。

## 8. パイプライン動作

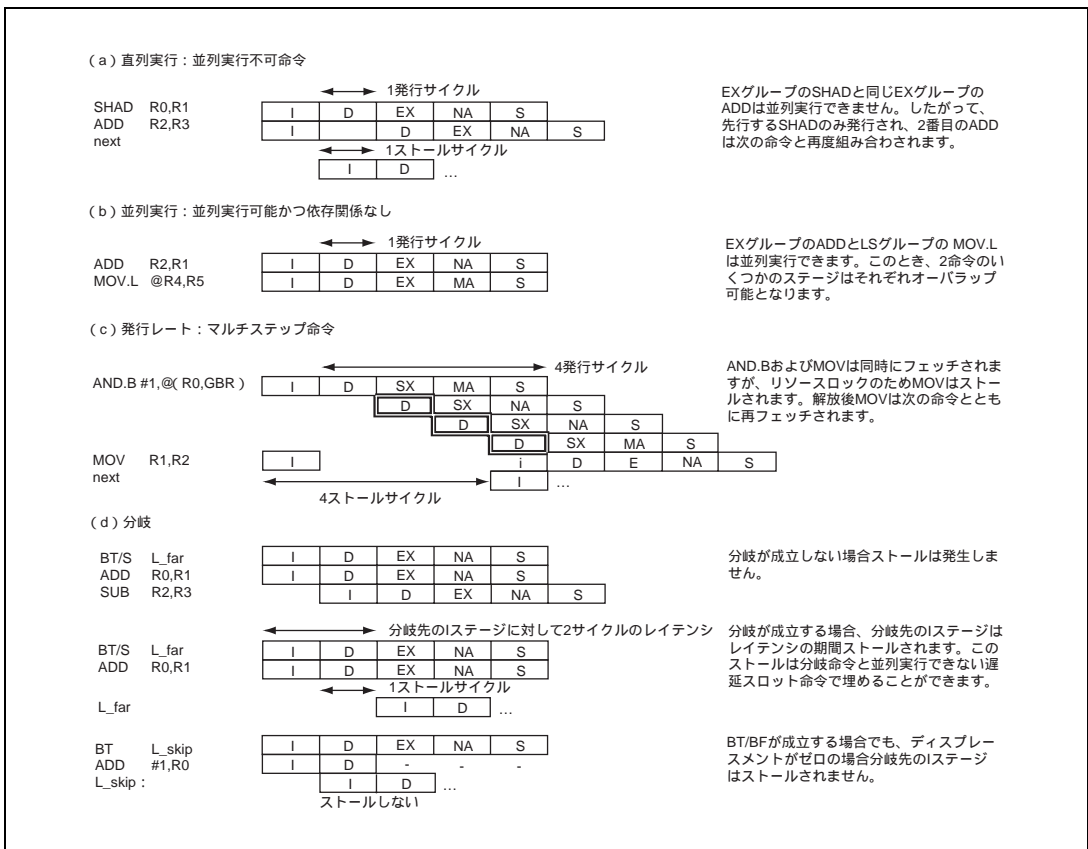


図 8.3 パイプライン実行の例 (1)

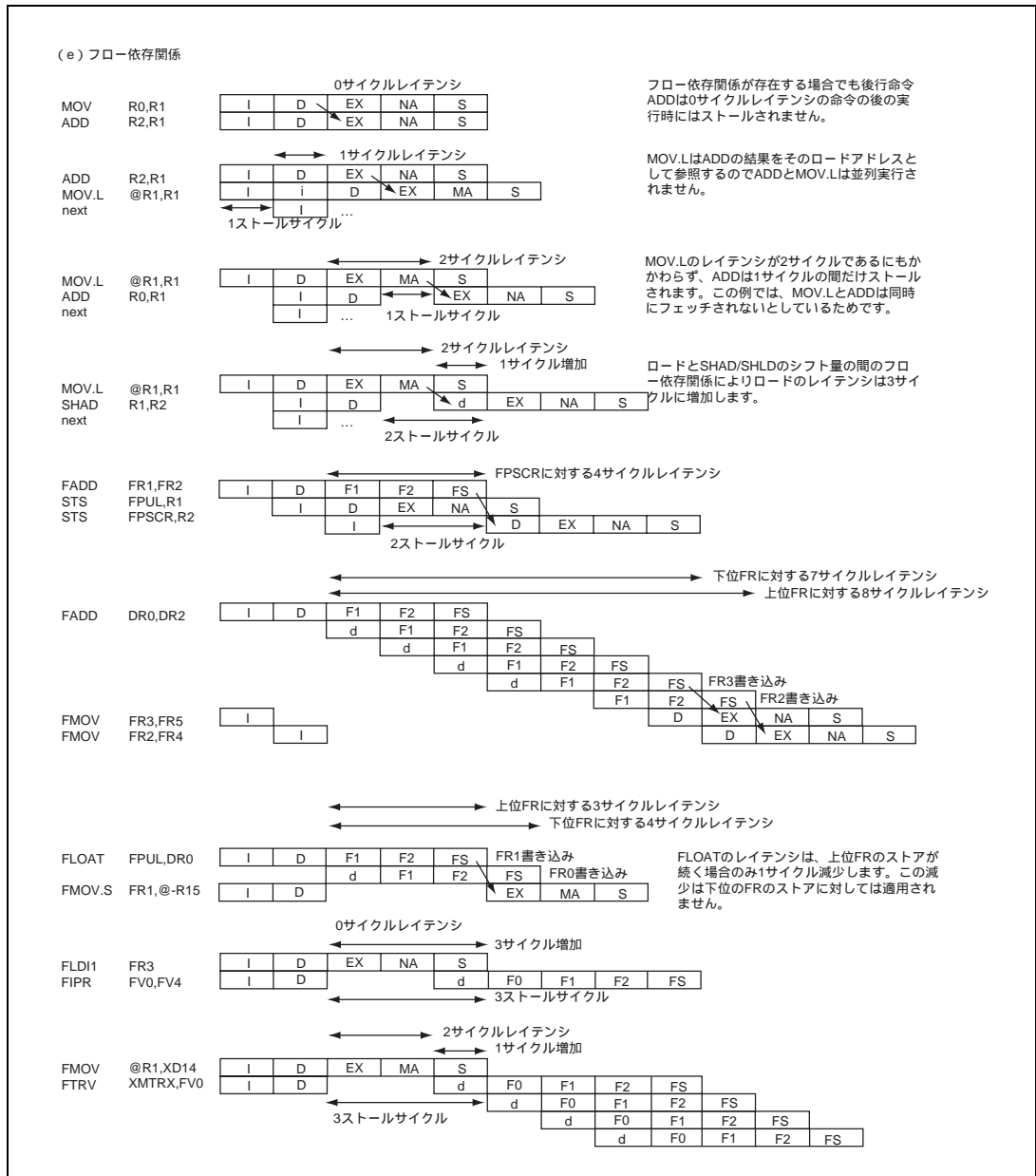


図 8.3 パイプライン実行の例 (2)



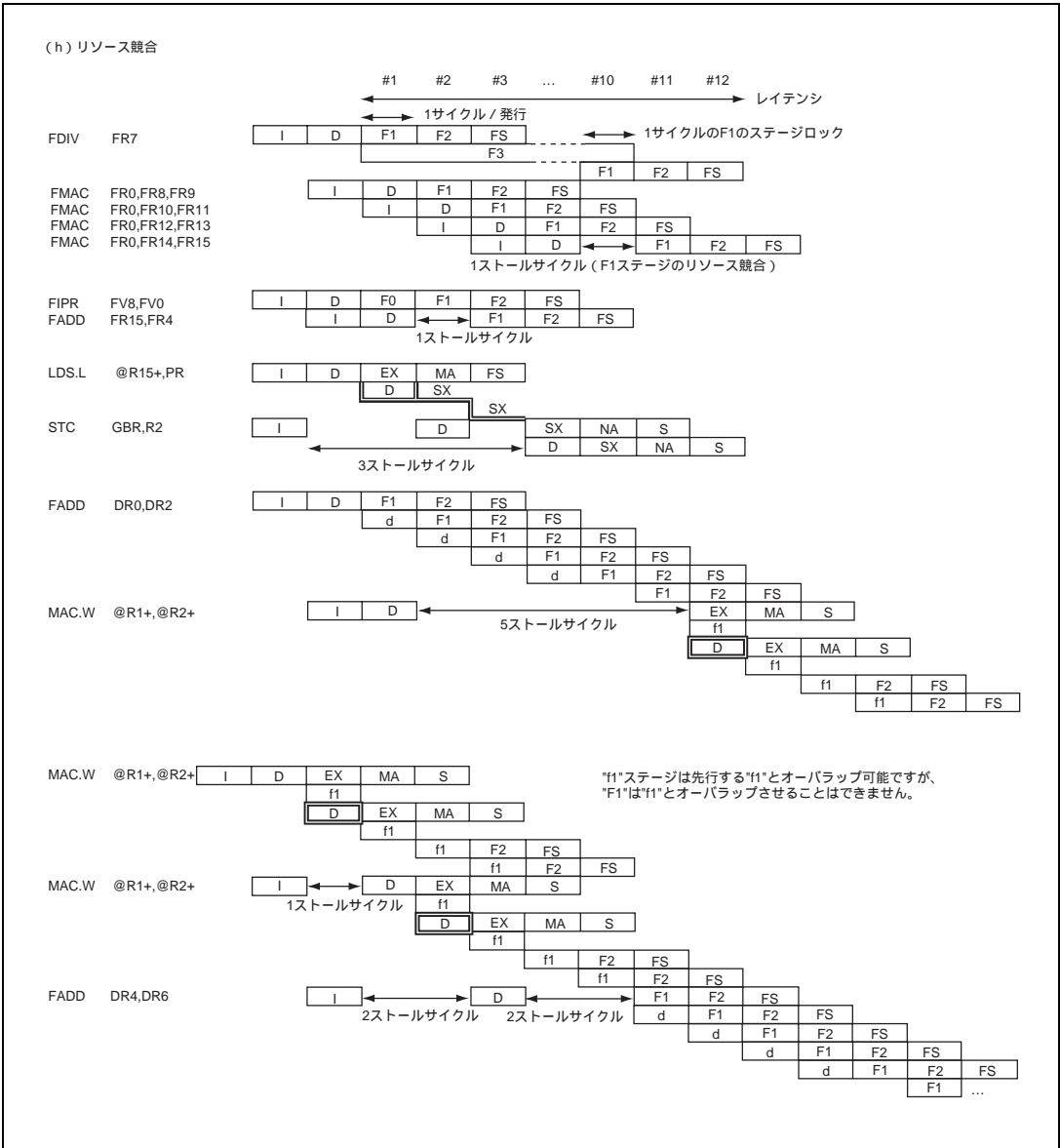


図 8.3 パイプライン実行の例 (4)

## 8. パイプライン動作

表 8.3 実行サイクル (1)

機能 分類	No.	命令		命令 グループ	発行 レート	レイテ ンシ	実行 パターン	ロック		
								ステージ	開始	サイクル
データ 転送 命令	1	EXTS.B	Rm,Rn	EX	1	1	#1	-	-	-
	2	EXTS.W	Rm,Rn	EX	1	1	#1	-	-	-
	3	EXTU.B	Rm,Rn	EX	1	1	#1	-	-	-
	4	EXTU.W	Rm,Rn	EX	1	1	#1	-	-	-
	5	MOV	Rm,Rn	MT	1	0	#1	-	-	-
	6	MOV	#Imm,Rn	EX	1	1	#1	-	-	-
	7	MOVA	@(disp,PC),R0	EX	1	1	#1	-	-	-
	8	MOV.W	@(disp,PC),Rn	LS	1	2	#2	-	-	-
	9	MOV.L	@(disp,PC),Rn	LS	1	2	#2	-	-	-
	10	MOV.B	@Rm,Rn	LS	1	2	#2	-	-	-
	11	MOV.W	@Rm,Rn	LS	1	2	#2	-	-	-
	12	MOV.L	@Rm,Rn	LS	1	2	#2	-	-	-
	13	MOV.B	@Rm+,Rn	LS	1	1/2	#2	-	-	-
	14	MOV.W	@Rm+,Rn	LS	1	1/2	#2	-	-	-
	15	MOV.L	@Rm+,Rn	LS	1	1/2	#2	-	-	-
	16	MOV.B	@(disp,Rm),R0	LS	1	2	#2	-	-	-
	17	MOV.W	@(disp,Rm),R0	LS	1	2	#2	-	-	-
	18	MOV.L	@(disp,Rm),Rn	LS	1	2	#2	-	-	-
	19	MOV.B	@(R0,Rm),Rn	LS	1	2	#2	-	-	-
	20	MOV.W	@(R0,Rm),Rn	LS	1	2	#2	-	-	-
	21	MOV.L	@(R0,Rm),Rn	LS	1	2	#2	-	-	-
	22	MOV.B	@(disp,GBR),R0	LS	1	2	#3	-	-	-
	23	MOV.W	@(disp,GBR),R0	LS	1	2	#3	-	-	-
	24	MOV.L	@(disp,GBR),R0	LS	1	2	#3	-	-	-
	25	MOV.B	Rm,@Rn	LS	1	1	#2	-	-	-
	26	MOV.W	Rm,@Rn	LS	1	1	#2	-	-	-
	27	MOV.L	Rm,@Rn	LS	1	1	#2	-	-	-
	28	MOV.B	Rm,@-Rn	LS	1	1/1	#2	-	-	-
	29	MOV.W	Rm,@-Rn	LS	1	1/1	#2	-	-	-
	30	MOV.L	Rm,@-Rn	LS	1	1/1	#2	-	-	-
	31	MOV.B	R0,@(disp,Rn)	LS	1	1	#2	-	-	-
	32	MOV.W	R0,@(disp,Rn)	LS	1	1	#2	-	-	-
	33	MOV.L	Rm,@(disp,Rn)	LS	1	1	#2	-	-	-
	34	MOV.B	Rm,@(R0,Rn)	LS	1	1	#2	-	-	-
	35	MOV.W	Rm,@(R0,Rn)	LS	1	1	#2	-	-	-
	36	MOV.L	Rm,@(R0,Rn)	LS	1	1	#2	-	-	-
	37	MOV.B	R0,@(disp,GBR)	LS	1	1	#3	-	-	-
	38	MOV.W	R0,@(disp,GBR)	LS	1	1	#3	-	-	-
	39	MOV.L	R0,@(disp,GBR)	LS	1	1	#3	-	-	-
	40	MOVCA.L	R0,@Rn	LS	1	3~7	#12	MA	4	3~7
	41	MOVT	Rn	EX	1	1	#1	-	-	-
	42	OCBI	@Rn	LS	1	1~2	#10	MA	4	1~2



表 8.3 実行サイクル (2)

機能 分類	No.	命令		命令 グループ	発行 レート	レイテ ンシ	実行 パターン	ロック		
								ステージ	開始	サイクル
データ 転送 命令	43	OCBP	@Rn	LS	1	1~5	#11	MA	4	1~5
	44	OCBWB	@Rn	LS	1	1~5	#11	MA	4	1~5
	45	PREF	@Rn	LS	1	1	#2	-	-	-
	46	SWAP.B	Rm,Rn	EX	1	1	#1	-	-	-
	47	SWAP.W	Rm,Rn	EX	1	1	#1	-	-	-
	48	XTRCT	Rm,Rn	EX	1	1	#1	-	-	-
固定 小数点 算術 命令	49	ADD	Rm,Rn	EX	1	1	#1	-	-	-
	50	ADD	#imm,Rn	EX	1	1	#1	-	-	-
	51	ADDC	Rm,Rn	EX	1	1	#1	-	-	-
	52	ADDV	Rm,Rn	EX	1	1	#1	-	-	-
	53	CMP/EQ	#imm,R0	MT	1	1	#1	-	-	-
	54	CMP/EQ	Rm,Rn	MT	1	1	#1	-	-	-
	55	CMP/GE	Rm,Rn	MT	1	1	#1	-	-	-
	56	CMP/GT	Rm,Rn	MT	1	1	#1	-	-	-
	57	CMP/HI	Rm,Rn	MT	1	1	#1	-	-	-
	58	CMP/HS	Rm,Rn	MT	1	1	#1	-	-	-
	59	CMP/PL	Rn	MT	1	1	#1	-	-	-
	60	CMP/PZ	Rn	MT	1	1	#1	-	-	-
	61	CMP/STR	Rm,Rn	MT	1	1	#1	-	-	-
	62	DIV0S	Rm,Rn	EX	1	1	#1	-	-	-
	63	DIV0U		EX	1	1	#1	-	-	-
	64	DIV1	Rm,Rn	EX	1	1	#1	-	-	-
	65	DMULS.L	Rm,Rn	CO	2	4/4	#34	F1	4	2
	66	DMULU.L	Rm,Rn	CO	2	4/4	#34	F1	4	2
	67	DT	Rn	EX	1	1	#1	-	-	-
	68	MAC.L	@Rm+,@Rn+	CO	2	2/2/4/4	#35	F1	4	2
	69	MAC.W	@Rm+,@Rn+	CO	2	2/2/4/4	#35	F1	4	2
	70	MUL.L	Rm,Rn	CO	2	4/4	#34	F1	4	2
	71	MULS.W	Rm,Rn	CO	2	4/4	#34	F1	4	2
	72	MULU.W	Rm,Rn	CO	2	4/4	#34	F1	4	2
	73	NEG	Rm,Rn	EX	1	1	#1	-	-	-
	74	NEGC	Rm,Rn	EX	1	1	#1	-	-	-
	75	SUB	Rm,Rn	EX	1	1	#1	-	-	-
	76	SUBC	Rm,Rn	EX	1	1	#1	-	-	-
	77	SUBV	Rm,Rn	EX	1	1	#1	-	-	-
論理 命令	78	AND	Rm,Rn	EX	1	1	#1	-	-	-
	79	AND	#imm,R0	EX	1	1	#1	-	-	-
	80	AND.B	#imm,@(R0,GBR)	CO	4	4	#6	-	-	-
	81	NOT	Rm,Rn	EX	1	1	#1	-	-	-
	82	OR	Rm,Rn	EX	1	1	#1	-	-	-
	83	OR	#imm,R0	EX	1	1	#1	-	-	-
	84	OR.B	#imm,@(R0,GBR)	CO	4	4	#6	-	-	-
	85	TAS.B	@Rn	CO	5	5	#7	-	-	-

## 8. パイプライン動作

表 8.3 実行サイクル (3)

機能 分類	No.	命令		命令 グループ	発行 レート	レイテ ンシ	実行 パターン	ロック		
								ステージ	開始	サイクル
論理 命令	86	TST	Rm,Rn	MT	1	1	#1	-	-	-
	87	TST	#imm,R0	MT	1	1	#1	-	-	-
	88	TST.B	#imm,@(R0,GBR)	CO	3	3	#5	-	-	-
	89	XOR	Rm,Rn	EX	1	1	#1	-	-	-
	90	XOR	#imm,R0	EX	1	1	#1	-	-	-
	91	XOR.B	#imm,@(R0,GBR)	CO	4	4	#6	-	-	-
シフト 命令	92	ROTL	Rn	EX	1	1	#1	-	-	-
	93	ROTR	Rn	EX	1	1	#1	-	-	-
	94	ROTCL	Rn	EX	1	1	#1	-	-	-
	95	ROTCR	Rn	EX	1	1	#1	-	-	-
	96	SHAD	Rm,Rn	EX	1	1	#1	-	-	-
	97	SHAL	Rn	EX	1	1	#1	-	-	-
	98	SHAR	Rn	EX	1	1	#1	-	-	-
	99	SHLD	Rm,Rn	EX	1	1	#1	-	-	-
	100	SHLL	Rn	EX	1	1	#1	-	-	-
	101	SHLL2	Rn	EX	1	1	#1	-	-	-
	102	SHLL8	Rn	EX	1	1	#1	-	-	-
	103	SHLL16	Rn	EX	1	1	#1	-	-	-
	104	SHLR	Rn	EX	1	1	#1	-	-	-
	105	SHLR2	Rn	EX	1	1	#1	-	-	-
	106	SHLR8	Rn	EX	1	1	#1	-	-	-
	107	SHLR16	Rn	EX	1	1	#1	-	-	-
分岐 命令	108	BF	disp	BR	1	2(or1)	#1	-	-	-
	109	BF/S	disp	BR	1	2(or1)	#1	-	-	-
	110	BT	disp	BR	1	2(or1)	#1	-	-	-
	111	BT/S	disp	BR	1	2(or1)	#1	-	-	-
	112	BRA	disp	BR	1	2	#1	-	-	-
	113	BRAF	Rm	CO	2	3	#4	-	-	-
	114	BSR	disp	BR	1	2	#14	SX	3	2
	115	BSRF	Rm	CO	2	3	#24	SX	3	2
	116	JMP	@Rn	CO	2	3	#4	-	-	-
	117	JSR	@Rn	CO	2	3	#24	SX	3	2
	118	RTS		CO	2	3	#4	-	-	-
システ ム制御 命令	119	NOP		MT	1	0	#1	-	-	-
	120	CLRMAC		CO	1	3	#28	F1	3	2
	121	CLRS		CO	1	1	#1	-	-	-
	122	CLRT		MT	1	1	#1	-	-	-
	123	SETS		CO	1	1	#1	-	-	-
	124	SETT		MT	1	1	#1	-	-	-
	125	TRAPA	#imm	CO	7	7	#13	-	-	-
	126	RTE		CO	5	5	#8	-	-	-
	127	SLEEP		CO	4	4	#9	-	-	-
	128	LDTLB		CO	1	1	#2	-	-	-

表 8.3 実行サイクル (4)

機能 分類	No.	命令		命令 グループ	発行 レート	レイテ ンシ	実行 パターン	ロック		
								ステージ	開始	サイクル
システ ム制御 命令	129	LDC	Rm,DBR	CO	1	3	#14	SX	3	2
	130	LDC	Rm,GBR	CO	3	3	#15	SX	3	2
	131	LDC	Rm,Rp_BANK	CO	1	3	#14	SX	3	2
	132	LDC	Rm,SR	CO	4	4	#16	SX	3	2
	133	LDC	Rm,SSR	CO	1	3	#14	SX	3	2
	134	LDC	Rm,SPC	CO	1	3	#14	SX	3	2
	135	LDC	Rm,VBR	CO	1	3	#14	SX	3	2
	136	LDC.L	@Rm+,DBR	CO	1	1/3	#17	SX	3	2
	137	LDC.L	@Rm+,GBR	CO	3	3/3	#18	SX	3	2
	138	LDC.L	@Rm+,Rp_BANK	CO	1	1/3	#17	SX	3	2
	139	LDC.L	@Rm+,SR	CO	4	4/4	#19	SX	3	2
	140	LDC.L	@Rm+,SSR	CO	1	1/3	#17	SX	3	2
	141	LDC.L	@Rm+,SPC	CO	1	1/3	#17	SX	3	2
	142	LDC.L	@Rm+,VBR	CO	1	1/3	#17	SX	3	2
	143	LDS	Rm,MACH	CO	1	3	#28	F1	3	2
	144	LDS	Rm,MACL	CO	1	3	#28	F1	3	2
	145	LDS	Rm,PR	CO	2	3	#24	SX	3	2
	146	LDS.L	@Rm+,MACH	CO	1	1/3	#29	F1	3	2
	147	LDS.L	@Rm+,MACL	CO	1	1/3	#29	F1	3	2
	148	LDS.L	@Rm+,PR	CO	2	2/3	#25	SX	3	2
	149	STC	DBR,Rn	CO	2	2	#20	-	-	-
	150	STC	SGR,Rn	CO	3	3	#21	-	-	-
	151	STC	GBR,Rn	CO	2	2	#20	-	-	-
	152	STC	Rp_BANK,Rn	CO	2	2	#20	-	-	-
	153	STC	SR,Rn	CO	2	2	#20	-	-	-
	154	STC	SSR,Rn	CO	2	2	#20	-	-	-
	155	STC	SPC,Rn	CO	2	2	#20	-	-	-
	156	STC	VBR,Rn	CO	2	2	#20	-	-	-
	157	STC.L	DBR,@-Rn	CO	2	2/2	#22	-	-	-
	158	STC.L	SGR,@-Rn	CO	3	3/3	#23	-	-	-
	159	STC.L	GBR,@-Rn	CO	2	2/2	#22	-	-	-
	160	STC.L	Rp_BANK,@-Rn	CO	2	2/2	#22	-	-	-
	161	STC.L	SR,@-Rn	CO	2	2/2	#22	-	-	-
	162	STC.L	SSR,@-Rn	CO	2	2/2	#22	-	-	-
	163	STC.L	SPC,@-Rn	CO	2	2/2	#22	-	-	-
	164	STC.L	VBR,@-Rn	CO	2	2/2	#22	-	-	-
	165	STS	MACH,Rn	CO	1	3	#30	-	-	-
	166	STS	MACL,Rn	CO	1	3	#30	-	-	-
	167	STS	PR,Rn	CO	2	2	#26	-	-	-
	168	STS.L	MACH,@-Rn	CO	1	1/1	#31	-	-	-
	169	STS.L	MACL,@-Rn	CO	1	1/1	#31	-	-	-
	170	STS.L	PR,@-Rn	CO	2	2/2	#27	-	-	-

## 8. パイプライン動作

表 8.3 実行サイクル (5)

機能 分類	No.	命令		命令 グループ	発行 レート	レイテ ンシ	実行 パターン	ロック		
								ステージ	開始	サイクル
単精度 浮動 小数点 命令	171	FLDI0	FRn	LS	1	0	#1	-	-	-
	172	FLDI1	FRn	LS	1	0	#1	-	-	-
	173	FMOV	FRm,FRn	LS	1	0	#1	-	-	-
	174	FMOV.S	@Rm,FRn	LS	1	2	#2	-	-	-
	175	FMOV.S	@Rm+,FRn	LS	1	1/2	#2	-	-	-
	176	FMOV.S	@(R0,Rm),FRn	LS	1	2	#2	-	-	-
	177	FMOV.S	FRm,@Rn	LS	1	1	#2	-	-	-
	178	FMOV.S	FRm,@-Rn	LS	1	1/1	#2	-	-	-
	179	FMOV.S	FRm,@(R0,Rn)	LS	1	1	#2	-	-	-
	180	FLDS	FRm,FPUL	LS	1	0	#1	-	-	-
	181	FSTS	FPUL,FRn	LS	1	0	#1	-	-	-
	182	FABS	FRn	LS	1	0	#1	-	-	-
	183	FADD	FRm,FRn	FE	1	3/4	#36	-	-	-
	184	FCMP/EQ	FRm,FRn	FE	1	2/4	#36	-	-	-
	185	FCMP/GT	FRm,FRn	FE	1	2/4	#36	-	-	-
	186	FDIV	FRm,FRn	FE	1	12/13	#37	F3	2	10
								F1	11	1
	187	FLOAT	FPUL,FRn	FE	1	3/4	#36	F1	2	2
	188	FMAC	FR0,FRm,FRn	FE	1	3/4	#36	-	-	-
	189	FMUL	FRm,FRn	FE	1	3/4	#36	-	-	-
	190	FNEG	FRn	LS	1	0	#1	-	-	-
	191	FSQRT	FRn	FE	1	11/12	#37	F3	2	9
								F1	10	1
	192	FSUB	FRm,FRn	FE	1	3/4	#36	-	-	-
	193	FTRC	FRm,FPUL	FE	1	3/4	#36	-	-	-
194	FMOV	DRm,DRn	LS	1	0	#1	-	-	-	
195	FMOV	@Rm,DRn	LS	1	2	#2	-	-	-	
196	FMOV	@Rm+,DRn	LS	1	1/2	#2	-	-	-	
197	FMOV	@(R0,Rm),DRn	LS	1	2	#2	-	-	-	
198	FMOV	DRm,@Rn	LS	1	1	#2	-	-	-	
199	FMOV	DRm,@-Rn	LS	1	1/1	#2	-	-	-	
200	FMOV	DRm,@(R0,Rn)	LS	1	1	#2	-	-	-	
倍精度 浮動 小数点 命令	201	FABS	DRn	LS	1	0	#1	-	-	-
	202	FADD	DRm,DRn	FE	1	(7,8)/9	#39	F1	2	6
	203	FCMP/EQ	DRm,DRn	CO	2	3/5	#40	F1	2	2
	204	FCMP/GT	DRm,DRn	CO	2	3/5	#40	F1	2	2
	205	FCNVDS	DRm,FPUL	FE	1	4/5	#38	F1	2	2
	206	FCNVSD	FPUL,DRn	FE	1	(3,4)/5	#38	F1	2	2
	207	FDIV	DRm,DRn	FE	1	(24,25)/26	#41	F3	2	21
							F1	20	3	
	208	FLOAT	FPUL,DRn	FE	1	(3,4)/5	#38	F1	2	2
	209	FMUL	DRm,DRn	FE	1	(7,8)/9	#39	F1	2	6

表 8.3 実行サイクル (6)

機能 分類	No.	命令		命令 グループ	発行 レート	レイテ ンシ	実行 パターン	ロック		
								ステージ	開始	サイクル
倍精度 浮動 小数点 命令	210	FNEG	DRn	LS	1	0	#1	-	-	-
	211	FSQRT	DRn	FE	1	(23,24)/25	#41	F3	2	20
								F1	19	3
	212	FSUB	DRm,DRn	FE	1	(7,8)/9	#39	F1	2	6
FPU システ ム制御 命令	213	FTRC	DRm,FPUL	FE	1	4/5	#38	F1	2	2
	214	LDS	Rm,FPUL	LS	1	1	#1	-	-	-
	215	LDS	Rm,FPSCR	CO	1	4	#32	F1	3	3
	216	LDS.L	@Rm+,FPUL	CO	1	1/2	#2	-	-	-
	217	LDS.L	@Rm+,FPSCR	CO	1	1/4	#33	F1	3	3
	218	STS	FPUL,Rn	LS	1	3	#1	-	-	-
	219	STS	FPSCR,Rn	CO	1	3	#1	-	-	-
	220	STS.L	FPUL,@-Rn	CO	1	1/1	#2	-	-	-
	221	STS.L	FPSCR,@-Rn	CO	1	1/1	#2	-	-	-
グラフ イクス 強化 命令	222	FMOV	DRm,XDn	LS	1	0	#1	-	-	-
	223	FMOV	XDm,DRn	LS	1	0	#1	-	-	-
	224	FMOV	XDm,XDn	LS	1	0	#1	-	-	-
	225	FMOV	@Rm,XDn	LS	1	2	#2	-	-	-
	226	FMOV	@Rm+,XDn	LS	1	1/2	#2	-	-	-
	227	FMOV	@(R0,Rm),XDn	LS	1	2	#2	-	-	-
	228	FMOV	XDm,@Rn	LS	1	1	#2	-	-	-
	229	FMOV	XDm,@-Rn	LS	1	1/1	#2	-	-	-
	230	FMOV	XDm,@(R0,Rn)	LS	1	1	#2	-	-	-
	231	FIPR	FVm,FVn	FE	1	4/5	#42	F1	3	1
	232	FRCHG		FE	1	1/4	#36	-	-	-
	233	FSCHG		FE	1	1/4	#36	-	-	-
	234	FTRV	XMTRX,FVn	FE	1	(5,5,6,7)/8	#43	F0	2	4
								F1	3	4

- 【注】
- 命令グループについては表 8.1 を参照してください。
  - レイテンシ"L1/L2..." : MACH/MACL/FPSCR を含む各レジスタへの書き込みに対応するレイテンシ。  
「例」 MOV.B @Rm+,Rn "1/2" : Rm に対するレイテンシは 1 サイクルで Rn に対するレイテンシは 2 サイクル
  - 分岐のレイテンシ : 分岐先命令がフェッチされるまでの間隔
  - 条件分岐のレイテンシ"2 (または 1) " : 0 以外のディスプレースメントに対するレイテンシは 2 で、0 ディスプレースメントに対するレイテンシは 1 です。
  - 倍精度浮動小数点命令のレイテンシ"(L1,L2)/L3" : L1 は FR[ n+1 ]、L2 は FR[ n ]、L3 は FPSCR に対するレイテンシです。
  - FTRV のレイテンシ"(L1,L2,L3,L4)/L5" : L1 は FR[ n ]、L2 は FR[ n+1 ]、L3 は FR[ n+2 ]、L4 は FR[ n+3 ]、L5 は FPSCR に対するレイテンシです。
  - MAC.L、MAC.W 命令のレイテンシ"L1/L2/L3/L4" : L1 は Rm、L2 は Rn、L3 は MACH、および L4 は MACL に対するレイテンシです。

## 8. パイプライン動作

---

8. MUL.L、MULS.W、MULU.W、DMULS.L、DMULU.L 命令のレイテンシ"L1/L2" : L1 は MACH、L2 は MACL に対するレイテンシです。
9. 実行パターン : 命令実行のパターン番号 (図 8.2 参照)
10. ロック / ステージ : 命令がロックするステージ
11. ロック / 開始 : ロッキングの開始サイクル ; 1 は命令の最初の D ステージ
12. ロック / サイクル : ロックしたサイクル数

例外 :

1. 浮動小数点計算命令に浮動小数点ストアが続く場合、浮動小数点計算のレイテンシは 1 サイクル減少します。
2. 先行命令が次の SHAD/SHLD のシフト量をロードする場合、ロードのレイテンシは 1 サイクル増加します。
3. 3 サイクル未満のレイテンシを持つ LS グループ命令に倍精度浮動小数点命令、FIPR または FTRV が続く場合、最初の命令のレイテンシは 3 サイクルに増加します。  
「例」"FMOV FR4,FR0"および"FIPR FV0,FV4"の場合、FIPR は 2 サイクルストールされます。
4. MAC\*/MUL\*に STS MAC\*, @-Rn が続く場合、MAC\*/MUL\*のレイテンシは 5 サイクルです。
5. MAC.W/MAC.L が連続実行された場合、レイテンシは 2 サイクルに減少します。
6. MAC\*への LDS に MAC\*からの STS が続く場合、MAC\*への LDS のレイテンシは 4 サイクルです。
7. MAC\*への LDS に MAC.W/MAC.L が続く場合、MAC\*への LDS のレイテンシは 1 サイクルです。
8. FSCHG または FRCHG 命令に、浮動小数点レジスタを読み出し / 書き込みする LS グループ命令が続く場合、前記 LS グループの命令は並行実行できません。
9. 単精度 FTRC 命令に STS FPUL, Rn、STS.L FPUL, @-Rn 命令が続く場合、レイテンシはそれぞれ 1、2 サイクルです。
10. 倍精度 FTRC 命令に STS FPUL, Rn、STS.L FPUL, @-Rn 命令が続く場合、レイテンシはそれぞれ 2、3 サイクルです。

---

## 9. 低消費電力モード

---

### 9.1 概要

低消費電力モードでは、内蔵周辺モジュールの一部と CPU が機能を停止します。これによって、消費電力を低減させることができます。

#### 9.1.1 低消費電力モードの種類

低消費電力モードには、次のようなモード、機能があります。

- (1) スリープモード
- (2) ディープスリープモード
- (3) スタンバイモード
- (4) モジュールスタンバイ機能 (TMU、RTC、SCI/SCIF、DMAC の内蔵モジュール)

プログラム実行状態から各モードへ遷移する条件、各モードでの CPU や周辺モジュールなどの状態、各モードの解除方法を、表 9.1 に示します。

表 9.1 低消費電力モードの状態

低消費電力モード	遷移状態	状態						解除方法
		CPG	CPU	内蔵メモリ	内蔵周辺モジュール	端子	外部メモリ	
スリープ	STBCR の STBY ビットが 0 の状態で SLEEP 命令を実行	動作	停止 (レジスタは保持)	保持	動作	保持	リフレッシュ	(1) 割り込み (2) リセット
ディープスリープ	STBCR の STBY ビットが 0、STBCR2 の DSLP ビットが 1 の状態で SLEEP 命令を実行	動作	停止 (レジスタは保持)	保持	動作 (DMA は停止)	保持	セルフリフレッシュ	(1) 割り込み (2) リセット
スタンバイ	STBCR の STBY ビットが 1 の状態で SLEEP 命令を実行	停止	停止 (レジスタは保持)	保持	停止*	保持	セルフリフレッシュ	(1) 割り込み (2) リセット
モジュールスタンバイ	STBCR の MSTP ビットを 1 とする	動作	動作	保持	指定モジュールが停止*	保持	リフレッシュ	(1) MSTP ビットを 0 とする (2) リセット

【注】 \* RTC は、RCR2 の START ビットが 1 のとき、動作します (ハードウェアマニュアルの「第 11 章 リアルタイムクロック」参照)。

## 9. 低消費電力モード

### 9.1.2 レジスタ構成

低消費電力モード関連のレジスタ構成を表 9.2 に示します。

表 9.2 レジスタ構成

名称	略称	R/W	初期値	P4 アドレス	エリア 7 アドレス	アクセス サイズ
スタンバイコントロール レジスタ	STBCR	R/W	H'00	H'FFC00004	H'1FC00004	8
スタンバイコントロール レジスタ 2	STBCR2	R/W	H'00	H'FFC00010	H'1FC00010	8

## 9.2 レジスタの説明

### 9.2.1 スタンバイコントロールレジスタ (STBCR)

スタンバイコントロールレジスタ (STBCR) は、低消費電力モードの状態を指定します。STBCR レジスタは、読出し / 書込み可能な 8 ビットのレジスタです。RESET 端子およびウォッチドッグタイマのオーバフローによるパワーオンリセットで H'00 に初期化されます。

ビット:	7	6	5	4	3	2	1	0
	STBY	PHZ	PPU	MSTP4	MSTP3	MSTP2	MSTP1	MSTP0
初期値:	0	0	0	0	0	0	0	0
R/W:	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ビット7: スタンバイ (STBY)

スタンバイモードへの遷移を指定します。

ビット7	説明
STBY	
0	SLEEP 命令の実行で、スリープモードへ遷移 (初期値)
1	SLEEP 命令の実行で、スタンバイモードへ遷移



## ビット6：周辺モジュール端子ハイインピーダンス制御（PHZ）

スタンバイモード時の、周辺モジュール関連端子の状態を制御します。PHZ を 1 にセットすると、スタンバイモード時に、周辺モジュール関連端子がハイインピーダンス状態になります。

対象端子は「9.2.2 周辺モジュール端子ハイインピーダンス制御」を参照してください。

ビット6	説明
PHZ	
0	周辺モジュール関連端子を通常状態（初期値）
1	周辺モジュール関連端子をハイインピーダンス状態

## ビット5：周辺モジュール端子プルアップ制御（PPU）

周辺モジュール関連端子の状態を制御します。PPU ビットを 0 にクリアすると、周辺モジュール関連端子が入力またはハイインピーダンス状態のときに、プルアップ抵抗がオンになります。

対象端子は「9.2.3 周辺モジュール端子プルアップ制御」を参照してください。

ビット5	説明
PPU	
0	周辺モジュール関連端子のプルアップ抵抗オン（初期値）
1	周辺モジュール関連端子のプルアップ抵抗オフ

## ビット4：モジュールストップ4（MSTP4）

DMAC へのクロック供給の停止を指定します。

MSTP4 ビットを 1 にセットすると DMAC へのクロック供給を停止します。

DMA 転送時は、転送を停止した後、MSTP4 ビットを 1 に設定してください。

MSTP4 ビットを 0 に設定した後、DMA 転送を行う場合は、DMAC の設定を再度行ってください。

ビット4	説明
MSTP4	
0	DMAC は動作（初期値）
1	DMAC へのクロックの供給を停止

## ビット3：モジュールストップ3（MSTP3）

内蔵周辺モジュールのうち、シリアルコミュニケーションインタフェースチャネル 2（SCIF）へのクロック供給の停止を指定します。

MSTP3 ビットを 1 にセットすると SCIF へのクロック供給を停止します。

ビット3	説明
MSTP3	
0	SCIF は動作（初期値）
1	SCIF へのクロックの供給を停止

## 9. 低消費電力モード

### ビット2：モジュールストップ2 (MSTP2)

内蔵周辺モジュールのうち、タイマユニット (TMU) へのクロック供給の停止を指定します。  
MSTP2 ビットを 1 にセットすると TMU へのクロック供給を停止します。

ビット 2	説明
MSTP2	
0	TMU は動作 (初期値)
1	TMU へのクロックの供給を停止

### ビット1：モジュールストップ1 (MSTP1)

内蔵周辺モジュールのうち、リアルタイムクロック (RTC) へのクロック供給の停止を指定します。

MSTP1 ビットを 1 にセットすると RTC へのクロック供給を停止します。クロック供給が停止されると RTC の各レジスタのアクセスはできなくなりますが、カウンタは動作を続けます。

ビット 1	説明
MSTP1	
0	RTC は動作 (初期値)
1	RTC へのクロックの供給を停止

### ビット0：モジュールストップ0 (MSTP0)

内蔵周辺モジュールのうち、シリアルコミュニケーションインタフェースチャネル 1 (SCI) へのクロック供給の停止を指定します。

MSTP0 ビットを 1 にセットすると SCI へのクロック供給を停止します。

ビット 0	説明
MSTP0	
0	SCI は動作 (初期値)
1	SCI へのクロックの供給を停止

## 9.2.2 周辺モジュール端子ハイインピーダンス制御

スタンバイコントロールレジスタ (STBCR) のビット 6 に 1 をセットすると、スタンバイモード時に、周辺モジュール関連端子をハイインピーダンス状態にします。

### (1) 対象端子

SCI 関連端子	MD0/SCK	MD1/TXD2
	MD7/TXD	MD8/RTS2
	CTS2	
DMA 関連端子	DACK0	DRAK0
	DACK1	DRAK1

### (2) その他

上記端子がポートの出力端子として使用されている時は、ハイインピーダンス制御は行いません。

### 9.2.3 周辺モジュール端子プルアップ制御

スタンバイコントロールレジスタ (STBCR) のビット 5 に 0 をセットすると、周辺モジュール関連端子が入力またはハイインピーダンス状態のときにプルアップされます。

(1) 対象端子

SCI 関連端子	MD0/SCK	MD1/TXD2	MD2/RXD2
	MD7/TXD	MD8/RTS2	SCK2/MRESET
	RXD	CTS2	
DMA 関連端子	DREQ0	DACK0	DRAK0
	DREQ1	DACK1	DRAK1
TMU 関連	TCLK		

### 9.2.4 スタンバイコントロールレジスタ 2 (STBCR2)

スタンバイコントロールレジスタ 2 (STBCR2) は、スリープモードとディープスリープモードの遷移条件を指定します。STBCR2 レジスタは、読出し / 書込み可能な 8 ビットのレジスタです。  
RESET 端子およびウォッチドッグタイマのオーバーフローによるパワーオンリセットで H'00 に初期化されます。

ビット:	7	6	5	4	3	2	1	0
	DSLP	-	-	-	-	-	-	-
初期値:	0	0	0	0	0	0	0	0
R/W:	R/W	R	R	R	R	R	R	R

ビット7: ディープスリープ (DSLP)

ディープスリープモードへの遷移を指定します。

ビット7	説明
DSLP	
0	SLEEP 命令の実行で、STBCR レジスタの STBY ビットの設定に従いスリープモードまたはスタンバイモードへ遷移する (初期値)
1	SLEEP 命令の実行で、ディープスリープモードへ遷移*

【注】 \* STBCR レジスタの STBY ビットが 0 の場合

ビット6~0: 予約ビット

書き込む値は常に 0 にしてください。1 を書き込んだ場合、動作は保証できません。  
読み出すときは常に 0 です。

### 9.3 スリープモード

#### 9.3.1 スリープモードへの遷移

STBCR レジスタの STBY ビットが 0 の状態で、SLEEP 命令を実行すると、プログラム実行状態からスリープモードに遷移します。CPU は SLEEP 命令実行後に停止しますが、CPU のレジスタ内容は保持されます。内蔵周辺モジュールは動作を続けます。CKIO 端子にはクロックが出力され続けます。

スリープモードでは、STATUS1 端子にハイレベルが、STATUS0 端子にローレベルが出力されません。

#### 9.3.2 スリープモードの解除

スリープモードは、割り込み（NMI、IRL、内蔵周辺）、リセットにより解除されます。

スリープモード中は、SR レジスタの BL ビットが 1 でも、割り込みを受け付けません。必要ならば、SLEEP 命令実行前に SPC、SSR をスタックに退避してください。

##### (1) 割り込みによる解除

NMI、IRL、内蔵周辺の各割り込みが発生すると、スリープモードが解除され、割り込み例外処理が実行されます。INTEVT レジスタには、割り込み要因に対応したコードがセットされます。

##### (2) リセットによる解除

RESET 端子によるパワーオンリセット、マニュアルリセット、およびウォッチドッグタイマオーバーフロー時に発生するパワーオンリセット、マニュアルリセットにより、スリープモードは解除されます。

### 9.4 ディープスリープモード

#### 9.4.1 ディープスリープモードへの遷移

STBCR レジスタの STBY ビットが 0、STBCR2 レジスタの DSLP ビットが 1 の状態で、SLEEP 命令を実行すると、プログラム実行状態からディープスリープモードに遷移します。CPU は SLEEP 命令実行後に停止しますが、CPU のレジスタ内容は保持されます。DMAC を除く内蔵周辺モジュールは動作を続けます。CKIO 端子にはクロックが出力され続けます。

ディープスリープモードでは、STATUS1 端子にハイレベルが、STATUS0 端子にローレベルが出力されます。

#### 9.4.2 ディープスリープモードの解除

ディープスリープモードは、スリープモードと同様に、割り込み（NMI、IRL、内蔵周辺モジュール）、リセットにより解除されます。

## 9.5 スタンバイモード

### 9.5.1 スタンバイモードへの遷移

STBCR レジスタの STBY ビットが 1 の状態で SLEEP 命令を実行すると、プログラム実行状態からスタンバイモードに遷移します。スタンバイモードでは、CPU だけでなくクロックや内蔵周辺モジュールも停止します。CKIO 端子からのクロック出力も停止します。

CPU、キャッシュのレジスタ内容は保持されます。内蔵周辺モジュールのレジスタに関しては初期化されるものがあります。スタンバイモード時の周辺モジュールのレジスタの状態を表 9.4 に示します。

表 9.4 スタンバイモード時のレジスタの状態

モジュール	初期化されるレジスタ	内容が保持されるレジスタ
割り込みコントローラ		全レジスタ
ユーザブレイクコントローラ		全レジスタ
バスステートコントローラ		全レジスタ
内蔵発振回路		全レジスタ
タイマユニット	TSTR レジスタ*	左記以外のレジスタ
リアルタイムクロック		全レジスタ
ダイレクトメモリアクセスコントローラ		全レジスタ
シリアルコミュニケーションインタフェース	「付録 A アドレス一覧」参照	「付録 A アドレス一覧」参照

【注】 \* ハードウェアマニュアルの「第 12 章 タイマユニット」を参照してください。

【注】 スタンバイモードへ遷移させる場合は、DMA 転送を終了させてください。

転送中にスタンバイモードへ遷移させると転送結果は保証されません。

スタンバイモードへ遷移する手順を以下に示します。

- (1) WDT のタイマコントロールレジスタ (WTCSR) の TME ビットを 0 にし、WDT を停止させます。  
WDT のタイマカウンタ (WTCNT) にカウントアップ時の初期値を、WTCSR レジスタの CKS2 ~ CKS 0 ビットに、カウントアップに使用するクロックを設定します。
- (2) STBCR レジスタの STBY ビットに 1 を設定した後、SLEEP 命令を実行させます。
- (3) スタンバイモードに入り、LSI 内部のクロックが停止すると、STATUS1 端子からローレベル、STATUS0 端子からハイレベルが出力されます。

### 9.5.2 スタンバイモードの解除

スタンバイモードは、割り込み（NMI、IRL、内蔵周辺）、 $\overline{\text{RESET}}$  端子によるリセットにより解除されます。

#### (1) 割り込みによる解除

内蔵 WDT によるホットスタートができます。NMI、IRL<sup>\*1</sup>、内蔵周辺（インターバルタイマを除く）<sup>\*2</sup>の各割り込みが検出されると、WDT がカウントを開始します。カウントオーバー後、LSI 全体にクロックが供給され、スタンバイモードが解除されて、STATUS1、STATUS0 端子がどちらもローレベルになります。この後割り込み例外処理が実行され、割り込み要因に対応したコードが INTEVT に設定されます。またスタンバイモード中は、SR レジスタの BL ビットが 1 のときでも割り込みを受け付けますので、必要ならば SLEEP 命令実行前に SPC、SSR をスタックに退避してください。

割り込み検出直後から、スタンバイモードが解除されるまでの間には、CKIO 端子のクロック出力の位相が不安定になることがあります。

【注】\*1 RTC クロック(32.768kHz)が発振しているとき(ハードウェアマニュアルの「19.2.2 IRL 割り込み」参照)、IRL3～IRL0 でスタンバイモードを解除できます(条件は、IRL3～IRL0 レベルが SR レジスタの I3～I0 のマスクレベルより高い場合)。

\*2 RTC の割り込みでスタンバイモードの解除ができます。

#### (2) リセットによる解除

$\overline{\text{RESET}}$  端子によるリセット（パワーオン、マニュアル）により、スタンバイモードは解除されます。 $\overline{\text{RESET}}$  端子は、クロックの発振が安定するまで、ローレベルを保持してください。CKIO 端子には、内部のクロックが出力され続けます。

### 9.5.3 クロックポーズ機能

スタンバイモードでは、EXTAL 端子から入力するクロックを停止したり、周波数を変更したりすることができます。この機能は、次のようにして使用します。

- (1) スタンバイモードへの遷移の手順でスタンバイモードに遷移させます。
- (2) スタンバイモードに入り、LSI 内部のクロックが停止すると、STATUS1 端子からローレベル、STATUS0 端子からハイレベルが出力されます。
- (3) STATUS1 端子がローレベル、STATUS0 端子がハイレベルになってから、入力クロックの停止、または周波数の変更を行います。
- (4) 周波数変更の場合、変更後に NMI または IRL の割り込みを入れます。クロック停止の場合、クロックの印加後に同様の割り込みを入れます。
- (5) WDT で設定した時間後に LSI 内部にクロックが印加され始め、STATUS1、STATUS0 端子がどちらもローレベルになって割り込み例外処理から動作を再開します。

## 9.6 モジュールスタンバイ機能

### 9.6.1 モジュールスタンバイ機能への遷移

スタンバイコントロールレジスタの MSTP4～MSTP0 ビットに 1 をセットすることで、それぞれ対応した内蔵周辺モジュールへのクロック供給を停止させることができます。この機能を使用することで、スリープ時の消費電力を低減させることができます。

モジュールスタンバイ状態では、内蔵周辺モジュールの外部端子は、停止前の状態を保持します。レジスタは一部を除いて停止前の状態を保持します。

ビット		説明
MSTP4	0	DMAC は動作します
	1	DMAC へ供給されるクロックが停止します
MSTP3	0	SCIF は動作します
	1	SCIF へ供給されるクロックが停止します
MSTP2	0	TMU は動作します
	1	TMU へ供給されるクロックが停止し、レジスタが初期化されます* <sup>1</sup>
MSTP1	0	RTC は動作します
	1	RTC へ供給されるクロックが停止します* <sup>2</sup>
MSTP0	0	SCI は動作します
	1	SCI へ供給されるクロックが停止します

【注】 \*<sup>1</sup> 初期化されるレジスタはスタンバイモードと同じですが、RTC クロックを使用している場合は初期化されません（ハードウェアマニュアルの「第 12 章 タイマユニット」参照）。

\*<sup>2</sup> RCR2 の START ビットが 1 のとき、カウンタは動作します（ハードウェアマニュアルの「第 11 章 リアルタイムクロック」参照）。

### 9.6.2 モジュールスタンバイ機能の解除

モジュールスタンバイ機能の解除は、MSTP4～MSTP0 ビットを 0 にクリアするか、 $\overline{\text{RESET}}$  端子によるパワーオンリセット、またはウォッチドッグタイマオーバーフローにより発生するパワーオンリセットで解除されます。

# 10. 各命令の説明

## この章の見方

以下の形式でアルファベット順に説明します。

命令の名称 命令の機能	命令の機能（英文）	命令の分類 （遅延分岐命令、または 割り込み禁止命令の表示）
----------------	-----------	--------------------------------------

書式	動作概略	命令コード	実行 ステート	Tビット
アセンブラの入力書式で表示しています。 imm、disp は数値、式またはシンボルになります。	動作の概略を表示しています。	MSB    LSB の順で表示しています。	ノーウェイトのときの値です。	命令実行後の、Tビットの値を表示しています。

(1)      説明

動作の説明を行います。

(2)      注意

命令を使用する上で特に注意が必要なことを説明します。

(3)      動作内容

C で動作内容を表示しています。動作を理解するための参考として記述してあります。ここでは以下の資源の使用を仮定しています。

```
char      8-bit integer
short     16-bit integer
int       32-bit integer
long      64-bit integer
float     single precision floating point number(32 bits)
double    double precision floating point number(64 bits)
```

データのタイプです。

```
unsigned char  Read_Byte(unsigned long Addr);
unsigned short Read_Word(unsigned long Addr);
unsigned long  Read_Long(unsigned long Addr);
```

アドレス Addr のそれぞれのサイズの内容を返します。2n 番地以外からのワード、4n 番地以外からのロングワードの読み込みはアドレスエラーとして検出します。



## 10. 各命令の説明

---

```
unsigned char Write_Byte(unsigned long Addr, unsigned long Data);
unsigned short Write_Word(unsigned long Addr, unsigned long Data);
unsigned long Write_Long(unsigned long Addr, unsigned long Data);
```

アドレス **Addr** にデータ **Data** をそれぞれのサイズで書き込みます。2n 番地以外へのワード、4n 番地以外へのロングワードの書き込みはアドレスエラーとして検出します。

```
Delay_Slot(unsigned long Addr);
```

アドレス (**Addr**) のスロット命令に実行を移します。

```
unsigned long R[16];
unsigned long SR,GBR,VBR;
unsigned long MACH,MACL,PR;
unsigned long PC;
```

各レジスタの本体

```
struct SR0 {
    unsigned long dummy0:22;
    unsigned long    M0:1;
    unsigned long    Q0:1;
    unsigned long    I0:4;
    unsigned long dummy1:2;
    unsigned long    S0:1;
    unsigned long    T0:1;
};
```

SR の構造の定義

```
#define M ((*(struct SR0 *)(&SR)).M0)
#define Q ((*(struct SR0 *)(&SR)).Q0)
#define S ((*(struct SR0 *)(&SR)).S0)
#define T ((*(struct SR0 *)(&SR)).T0)
```

SR 内ビットの定義

```
Error( char *er );
```

エラー表示関数

浮動小数点用の定義文です。

```
#define PZERO      0
#define NZERO      1
#define DENORM     2
#define NORM       3
#define PINF       4
#define NINF       5
#define qNaN       6
#define sNaN       7
#define EQ         0
#define GT         1
#define LT         2
#define UO         3
#define INVALID    4
#define FADD       0
#define FSUB       1

#define CAUSE      0x0003f000 /* FPSCR(bit17-12) */
#define SET_E      0x00020000 /* FPSCR(bit17) */
#define SET_V      0x00010040 /* FPSCR(bit16,6) */
#define SET_Z      0x00008020 /* FPSCR(bit15,5) */
#define SET_O      0x00004010 /* FPSCR(bit14,4) */
#define SET_U      0x00002008 /* FPSCR(bit13,3) */
#define SET_I      0x00001004 /* FPSCR(bit12,2) */
#define ENABLE_VOUI 0x00000b80 /* FPSCR(bit11,9-7) */
#define ENABLE_V    0x00000800 /* FPSCR(bit11) */
#define ENABLE_Z    0x00000400 /* FPSCR(bit10) */
#define ENABLE_OUI  0x00000380 /* FPSCR(bit9-7) */
#define ENABLE_I    0x00000080 /* FPSCR(bit7) */

#define FPSCR_FR    FPSCR>>21&1
#define FPSCR_PR    FPSCR>>19&1
#define FPSCR_DN    FPSCR>>18&1
#define FPSCR_I     FPSCR>>12&1
#define FPSCR_RM    FPSCR&1
#define FR_HEX      frf.1[ FPSCR_FR]
#define FR          frf.f[ FPSCR_FR]
#define DR          frf.d[ FPSCR_FR]
#define XF_HEX      frf.1[~FPSCR_FR]
#define XF          frf.f[~FPSCR_FR]
```

```
#define XD                frf.d[~FPSCR_FR]

union {
    int  l[2][16];
    float f[2][16];
    double d[2][8];
} frf;
int FPSCR,FPUL;

int sign_of(int n)
{
    return(FR_HEX[n]>>31);
}

int data_type_of(int n)
int abs;
    abs = FR_HEX[n] & 0x7fffffff;
    if(FPSCR_PR == 0) { /* 単精度 */
        if(abs < 0x00800000){
            if((FPSCR_DN == 1) || (abs == 0x00000000)){
                if(sign_of(n) == 0) return(PZERO);
                else return(NZERO);
            }
            else return(DENORM);
        }
        else if(abs < 0x7f800000) return(NORM);
        else if(abs == 0x7f800000) {
            if(sign_of(n) == 0) return(PINF);
            else return(NINF);
        }
        else if(abs < 0x7fc00000) return(qNaN);
        else return(sNaN);
    }
    else { /* 倍精度 */
        if(abs < 0x00100000){
            if((FPSCR_DN == 1) || (abs == 0x00000000)){
                if(sign_of(n) == 0) return(PZERO);
                else return(NZERO);
            }
            else return(DENORM);
        }
    }
}
```

```

        else if(abs < 0x7ff00000) return(NORM);
        else if((abs == 0x7ff00000) &&
                (FR_HEX[n+1] == 0x00000000)) {
            if(sign_of(n) == 0) return(PINF);
            else return(NINF);
        }
        else if(abs < 0x7ff80000) return(qNaN);
        else return(sNaN);
    }
}

void register_copy(int m,n)
{
    FR[n] = FR[m];
    if(FPSCR_PR == 1) FR[n+1] = FR[m+1];
}

void normal_faddsub(int m,n,type)
{
    union {
        float f;
        int l;
    } dstf,srcf;
    union {
        double d;
        int l[2];
    } dstd,srcd;
    union {
        /* "long double" のフォーマット: */
        int double x; /* 1-bit 符号 */
        int l[4]; /* 15-bit 指数 */
    } dstx; /* 112-bit 小数 */

    if(FPSCR_PR == 0) {
        if(type == FADD) srcf.f = FR[m];
        else srcf.f = -FR[m];
        dstd.d = FR[n]; /* 単精度から倍精度への変換*/
        dstd.d += srcf.f;
        if(((dstd.d == FR[n]) && (srcf.f != 0.0)) ||
            ((dstd.d == srcf.f) && (FR[n] != 0.0))) {
            set_I();
            if(sign_of(m)^ sign_of(n)) {
                dstd.l[1] -= 1;
                if(dstd.l[1] == 0xffffffff) dstd.l[0] -= 1;
            }
        }
    }
}

```

```
    }
}
if(dst.d.l[1] & 0x1fffffff) set_I();
dstf.f += srcf.f; /* 近傍への丸め */
if(FPSCR_RM == 1) {
    dst.d.l[1] &= 0xe0000000; /* 0への丸め */
    dstf.f = dst.d.d;
}
check_single_exception(&FR[n],dstf.f);
} else {
    if(type == FADD)    srcd.d = DR[m>>1];
    else                srcd.d = -DR[m>>1];
    dstx.x = DR[n>>1]; /* 倍精度から拡張倍精度への変換 */
    dstx.x += srcd.d;
    if(((dstx.x == DR[n>>1]) && (srcd.d != 0.0)) ||
        ((dstx.x == srcd.d) && (DR[n>>1] != 0.0)) ) {
        set_I();
        if(sign_of(m)^ sign_of(n)) {
            dstx.l[3] -= 1;
            if(dstx.l[3] == 0xffffffff) dstx.l[2] -= 1;
            if(dstx.l[2] == 0xffffffff) dstx.l[1] -= 1;
            if(dstx.l[1] == 0xffffffff) dstx.l[0] -= 1;
        }
    }
}
if((dstx.l[2] & 0x0fffffff) || dstx.l[3]) set_I();
dst.d += srcd.d; /*近傍への丸め */
if(FPSCR_RM == 1) {
    dstx.l[2] &= 0xf0000000; /* 0への丸め */
    dstx.l[3] = 0x00000000;
    dst.d = dstx.x;
}
check_double_exception(&DR[n>>1] ,dst.d);
}
}

void normal_fmuls(int m,n)
{
    union {
        float f;
        int l;
    } tmpf;
```

```

union {
    double d;
    int l[2];
} tmpd;
union {
    int double x;
    int l[4];
} tmpx;

if(FPSCR_PR == 0) {
    tmpd.d = FR[n]; /* 単精度から倍精度 */
    tmpd.d *= FR[m]; /* 正確に作成 */
    tmpf.f = FR[m]; /* 近傍への丸め */
    if(tmpf.f != tmpd.d) set_I();
    if((tmpf.f > tmpd.d) && (SPSCR_RM == 1)) {
        tmpf.l -= 1; /* 0への丸め */
    }
    check_single_exception(&FR[n],tmpf.f);
} else {
    tmpx.x = DR[n>>1]; /* 単精度から倍精度 */
    tmpx.x *= DR[m>>1]; /* 正確に作成 */
    tmpd.d = DR[m>>1]; /* 近傍への丸め */
    if(tmpd.d != tmpx.x) set_I();
    if(tmpd.d > tmpx.x) && (SPSCR_RM == 1)) {
        tmpd.l[1] -= 1; /* 0への丸め */
        if(tmpd.l[1] == 0xffffffff) tmpd.l[0] -= 1;
    }
    check_double_exception(&DR[n>>1], tmpd.d);
}
}

void fipr(int m,n)
{
    union {
        double d;
        int l[2];
    } mlt[4];

```

```
float dstf;

if((data_type_of(m) == sNaN) || (data_type_of(n) == sNaN) ||
    (data_type_of(m+1) == sNaN) || (data_type_of(n+1) == sNaN) ||
    (data_type_of(m+2) == sNaN) || (data_type_of(n+2) == sNaN) ||
    (data_type_of(m+3) == sNaN) || (data_type_of(n+3) == sNaN) ||
    (check_product_invalid(m,n)) ||
    (check_product_invalid(m+1,n+1)) ||
    (check_product_invalid(m+2,n+2)) ||
    (check_product_invalid(m+3,n+3)) )    invalid(n+3);
else if((data_type_of(m) == qNaN) || (data_type_of(n) == qNaN) ||
    (data_type_of(m+1) == qNaN) || (data_type_of(n+1) == qNaN) ||
    (data_type_of(m+2) == qNaN) || (data_type_of(n+2) == qNaN) ||
    (data_type_of(m+3) == qNaN) || (data_type_of(n+3) == qNaN)) qnan(n+3);
else if(check_positive_infinity() &&
    (check_negative_infinity()    invalid(n+3);
else if (check_positive_infinity()    inf(n+3,0);
else if (check_negative_infinity()    inf(n+3,1);
else {
    for(i=0;i<4;i++) {
        /* FPSCR_DN == 1 なら、0 にする) */
        if      (data_type_of(m+i) == PZERO) FR[m+i] = +0.0;
        else if (data_type_of(m+i) == NZERO) FR[m+i] = -0.0;
        if      (data_type_of(n+i) == PZERO) FR[n+i] = +0.0;
        else if (data_type_of(n+i) == NZERO) FR[n+i] = -0.0;
        mlt[i].d = FR[m+i];
        mlt[i].d *= FR[n+i];

/* 正確には、FIPR では、下位 18bit を切り捨てているので、ここに記述したものは
ハードウェアとは異なりより単純にしたものです。 */
        mlt[i].l[1] &= 0xff000000;
        mlt[i].l[1] |= 0x00800000;
    }
    mlt[0].d += mlt[1].d + mlt[2].d + mlt[3].d;
    mlt[0].l[1] &= 0xff800000;
    dstf = mlt[0].d;
    set_I();
    check_single_exception(&FR[n+3],dstf);
}
}
```

```

void check_single_exception(float *dst,result)
{
union {
    float f;
    int l;
} tmp;
float abs;

if(result < 0.0) tmp.l = 0xff800000; /* -無限大 */
else tmp.l = 0x7f800000; /* +無限大 */
if(result == tmp.f) {
    set_O();
    if(FPSCR_RM == 1) {
        tmp.l -= 1; /* 正規化数の最大値 */
        result = tmp.f;
    }
}
if(result < 0.0) abs = -result;
else abs = result;
tmp.l = 0x00800000; /* 正規化数の最小値 */
if(abs < tmp.f) {
    if((FPSCR_DN == 1) && (abs != 0.0)) {
        set_I();
        if(result < 0.0) result = -0.0; /* 非正規化数を0にする。 */
        else result = 0.0;
    }
    if(FPSCR_I == 1) set_U();
}
if(FPSCR & ENABLE_OUI) fpu_exception_trap();
else *dst = result;
}

void check_double_exception(double *dst,result)
{
union {
    double d;
    int l[2];
} tmp;
double abs;

if(result < 0.0) tmp.l[0] = 0xffff00000; /* -無限大 */
else tmp.l[0] = 0x7ff00000; /* +無限大 */
tmp.l[1] = 0x00000000;

```



```
if(result == tmp.d)
    set_O();
if(FPSCR_RM == 1) {
    tmp.l[0] -= 1;
    tmp.l[1] = 0xffffffff;
    result = tmp.d; /* 正規化数の最大値 */
}
}
if(result < 0.0) abs = -result;
else abs = result;
tmp.l[0] = 0x00100000; /* 正規化数の最小値 */
tmp.l[1] = 0x00000000;
if(abs < tmp.d) {
    if((FPSCR_DN == 1) && (abs != 0.0)) {
        set_I();
        if(result < 0.0) result = -0.0; /* 非正規化数を 0 にする。 */
        else result = 0.0;
    }
    if(FPSCR_I == 1) set_U();
}
if(FPSCR & ENABLE_OUI) fpu_exception_trap();
else *dst = result;
}

int check_product_invalid(int m,n)
{
    return(check_product_infinity(m,n) &&
        ((data_type_of(m) == PZERO) || (data_type_of(n) == PZERO) ||
        (data_type_of(m) == NZERO) || (data_type_of(n) == NZERO)));
}

int check_product_infinity(int m,n)
{
    return((data_type_of(m) == PINF) || (data_type_of(n) == PINF) ||
        (data_type_of(m) == NINF) || (data_type_of(n) == NINF));
}

int check_positive_infinity(int m,n)
{
    return(((check_product_infinity(m,n) && (~sign_of(m)^ sign_of(n))) ||
        ((check_product_infinity(m+1,n+1) && (~sign_of(m+1)^ sign_of(n+1))) ||
        ((check_product_infinity(m+2,n+2) && (~sign_of(m+2)^ sign_of(n+2))) ||
        ((check_product_infinity(m+3,n+3) && (~sign_of(m+3)^ sign_of(n+3)))));
```

```

}
int check_negative_infinity(int m,n)
{
    return(((check_product_infinity(m,n) && (sign_of(m)^ sign_of(n))) ||
        ((check_product_infinity(m+1,n+1) && (sign_of(m+1)^ sign_of(n+1))) ||
        ((check_product_infinity(m+2,n+2) && (sign_of(m+2)^ sign_of(n+2))) ||
        ((check_product_infinity(m+3,n+3) && (sign_of(m+3)^ sign_of(n+3)))));
}
void clear_cause () {FPSCR &= ~CAUSE;}
void set_E() {FPSCR |= SET_E;}
void set_V() {FPSCR |= SET_V;}
void set_Z() {FPSCR |= SET_Z;}
void set_O() {FPSCR |= SET_O;}
void set_U() {FPSCR |= SET_U;}
void set_I() {FPSCR |= SET_I;}
void invalid(int n)
{
    set_V();
    if((FPSCR & ENABLE_V) == 0 qnan(n);
    else    fpu_exception_trap();
}

void dz(int n,sign)
{
    set_Z();
    if((FPSCR & ENABLE_Z) == 0 inf(n,sign);
    else    fpu_exception_trap();
}
void zero(int n,sign)
{
    if(sign == 0)      FR_HEX [n]   = 0x00000000;
    else               FR_HEX [n]   = 0x80000000;
    if (FPSCR_PR==1)  FR_HEX [n+1] = 0x00000000;
}
void inf(int n,sign) {
    if (FPSCR_PR==0) {
        if(sign == 0) FR_HEX [n]   = 0x7f800000;
        else         FR_HEX [n]   = 0xff800000;
    } else {
        if(sign == 0) FR_HEX [n]   = 0x7ff00000;
        else         FR_HEX [n]   = 0xfff00000;
    }
}

```

```
        FR_HEX [n+1] = 0x00000000;
    }
}
void qnan(int n)
{
    if (FPSCR_PR==0)    FR[n]    = 0x7fbfffff;
    else {              FR[n]    = 0x7ff7ffff;
                    FR[n+1] = 0xffffffff;
    }
}
```

#### (4) 使用例

アセンブラニーモニックで例を示し、命令の実行前後の状態を表示しています。

イタリック字体（例: *.align*）はアセンブラ制御命令であることを示します。アセンブラ制御命令の意味は次のようになります。詳しくは、「クロスアセンブラユーザズマニュアル」を参照してください。

<i>.org</i>	ロケーションカウンタ設定
<i>.data.w</i>	ワード整数データ確保
<i>.data.l</i>	ロングワード整数データ確保
<i>.sdata</i>	文字列データ確保
<i>.align</i> 2	2 バイト境界調整
<i>.align</i> 4	4 バイト境界調整
<i>.align</i> 32	32 バイト境界調整
<i>.arepeat</i> 16	16 回繰り返し展開
<i>.arepeat</i> 32	32 回繰り返し展開
<i>.aendr</i>	回数指定繰り返し展開終了

【注】 SH シリーズクロスアセンブラ Ver 1.0 では、条件付きアセンブラ機能をサポートしておりません。

## 10.1 ADD ADD binary

### 2 進加算

## 算術演算命令

書式	動作概略	命令コード	実行 ステート	Tビット
ADD Rm,Rn	Rn+Rm Rn	0011nnnnnnmmmm1100	1	
ADD #imm,Rn	Rn+imm Rn	0111nnnnnniiiiiii	1	

#### (1) 説明

汎用レジスタ Rn の内容と Rm とを加算し、結果を Rn に格納します。  
 汎用レジスタ Rn と 8 ビットのイミディエイトデータとの加算も可能です。  
 8 ビットのイミディエイトデータは 32 ビットに符号拡張しますので減算との兼用が可能です。

#### (2) 動作内容

```
ADD(long m, long n) /* ADD Rm,Rn */
```

```
{
    R[n]+=R[m];
    PC+=2;
}
```

```
ADDI(long i, long n) /* ADD #imm,Rn */
```

```
{
    if ((i&0x80)==0)
        R[n]+=(0x000000FF & (long)i);
    else R[n]+=(0xFFFFF00 | (long)i);
    PC+=2;
}
```

#### (3) 使用例

```
ADD R0,R1      ;実行前 R0=H'7FFFFFFF,R1=H'00000001
                ;実行後 R1=H'80000000
ADD #H'01,R2    ;実行前 R2=H'00000000
                ;実行後 R2=H'00000001
ADD #H'FE,R3    ;実行前 R3=H'00000001
                ;実行後 R3=H'FFFFFFF
```

## 10.2 ADDC      ADD with Carry

キャリ付き2進加算

## 算術演算命令

書式	動作概略	命令コード	実行 ステート	Tビット
ADDC Rm,Rn	$Rn + Rm + T$ Rn, キャリ    T	0011nnnnnnmmmm1110	1	キャリ

### (1) 説明

汎用レジスタ Rn の内容と Rm と T ビットを加算し、結果を Rn に格納します。演算の結果によってキャリを T ビットに反映します。32 ビットを超える加算を行うとき使用します。

### (2) 動作内容

```
ADDC(long m, long n)    /* ADDC Rm,Rn */
{
    unsigned long tmp0,tmp1;

    tmp1=R[n]+R[m];
    tmp0=R[n];
    R[n]=tmp1+T;
    if (tmp0>tmp1) T=1;
    else T=0;
    if (tmp1>R[n]) T=1;
    PC+=2;
}
```

### (3) 使用例

```
CLRT                    ;R0:R1(64ビット)+R2:R3(64ビット)=R0:R1(64ビット)
ADDC R3,R1            ;実行前 T=0,R1=H'00000001,R3=H'FFFFFFF
                      ;実行後 T=1,R1=H'00000000
ADDC R2,R0            ;実行前 T=1,R0=H'00000000,R2=H'00000000
                      ;実行後 T=0,R0=H'00000001
```

## 10.3 ADDV ADD with (Vflag) overflow check 算術演算命令

オーバーフロー付き

2進加算

書式	動作概略	命令コード	実行 ステート	Tビット
ADDV Rm,Rn	Rn+Rm Rn, オーバーフロー T	0011nnnnnnmmmm1111	1	オーバ フロー

## (1) 説明

汎用レジスタ Rn の内容と Rm とを加算し、結果を Rn に格納します。オーバーフローが発生すると、T ビットをセットします。

## (2) 動作内容

```
ADDV(long m, long n)    /* ADDV Rm,Rn */
{
    long dest,src,ans;

    if ((long)R[n]>=0) dest=0;
    else dest=1;
    if ((long)R[m]>=0) src=0;
    else src=1;
    src+=dest;
    R[n]+=R[m];
    if ((long)R[n]>=0) ans=0;
    else ans=1;
    ans+=dest;
    if (src==0 || src==2) {
        if (ans==1) T=1;
        else T=0;
    }
    else T=0;
    PC+=2;
}
```

## (3) 使用例

```
ADDV R0,R1      ;実行前 R0=H'00000001,R1=H'7FFFFFFE, T=0
                  ;実行後 R1=H'7FFFFFFF, T=0
ADDV R0,R1      ;実行前 R0=H'00000002,R1=H'7FFFFFFE, T=0
                  ;実行後 R1=H'80000000, T=1
```

## 10.4 AND AND logical

### 論理積演算

## 論理演算命令

書式	動作概略	命令コード	実行 ステート	Tビット
AND Rm,Rn	Rm & Rm Rn	0010nnnnmmmm1001	1	
AND #imm,R0	R0 & imm R0	11001001iiiiiii	1	
AND.B #imm,@(R0,GBR)	(R0+GBR) & imm (R0+GBR)	11001101iiiiiii	4	

#### (1) 説明

汎用レジスタ Rn の内容と Rm の論理積をとり、結果を Rn に格納します。

汎用レジスタ R0 とゼロ拡張した 8 ビットのイミディエイトデータとの論理積、もしくはインデックス付き GBR 間接アドレッシングモードで 8 ビットのメモリと 8 ビットのイミディエイトデータとの論理積が可能です。

#### (2) 注意

AND #imm,R0 では演算の結果、R0 の上位 24 ビットは常にクリアされます。

#### (3) 動作内容

```
AND(long m, long n) /* AND Rm,Rn */
```

```
{
    R[n]&=R[m];
    PC+=2;
}
```

```
ANDI(long i) /* AND #imm,R0 */
```

```
{
    R[0]&=(0x000000FF & (long)i);
    PC+=2;
}
```

```
ANDM(long i) /* AND.B #imm,@(R0,GBR) */
```

```
{
    long temp;

    temp=(long)Read_Byte(GBR+R[0]);
    temp&=(0x000000FF & (long)i);
    Write_Byte(GBR+R[0],temp);
    PC+=2;
}
```

## (4) 使用例

```
AND    R0,R1                ;実行前 R0=H'AAAAAAAA,R1=H'55555555
                                ;実行後 R1=H'00000000
AND    #H'0F,R0             ;実行前 R0=H'FFFFFFFF
                                ;実行後 R0=H'0000000F
AND.B   #H'80,@(R0,GBR)     ;実行前 @(R0,GBR)=H'A5
                                ;実行後 @(R0,GBR)=H'80
```



## 10.5 BF Branch if False

### 条件分岐

## 分岐命令

書式	動作概略	命令コード	実行ステート	Tビット
BF label	T=0 のとき PC+4+disp×2 PC, T=1 のとき nop	10001011ddddddd	1	

## (1) 説明

Tビットを参照する条件付き分岐命令です。T=1 のときは分岐しません。逆に T=0 のとき、分岐します。分岐先はアドレス (PC+4+ディスプレースメント×2) です。PC ソース値は BF の命令アドレスです。8 ビットディスプレースメントは符号拡張後 2 倍しますので、分岐先との相対距離は-256 バイトから+254 バイトの範囲になります。

## (2) 注意

分岐先に届かないときは BRA、JMP 命令などとの組み合わせで対応する必要があります。

## (3) 動作内容

```
BF(int d)    /* BF disp */
{
    int disp;

    if ((d&0x80)==0)
        disp=(0x000000FF & d);
    else    disp=(0xFFFFF00 | d);
    if (T==0)
        PC=PC+4+(disp<<1);
    else    PC+=2;
}
```

## (4) 使用例

```
CLRT                ;常に T=0
BT   TRGET_T        ;T=0 のため分岐しません。
BF   TRGET_F        ;T=0 のため TRGET_F へ分岐します。
NOP
NOP
TRGET_F:            ; BF 命令の分岐先
```

## 10.6 BF/S Branch if False with delay Slot

遅延付き条件分岐

## 分岐命令

遅延分岐命令

書式	動作概略	命令コード	実行ステート	T ビット
BF/S label	T=0 のとき PC+4+disp×2 PC, T=1 のとき nop	10001111ddddddd	1	

### (1) 説明

T ビットを参照する遅延付き条件分岐命令です。T=1 のとき、次の命令を実行し、分岐しません。T=0 のとき、次の命令を実行した後で分岐します。

分岐先はアドレス (PC+4+ディスプレースメント×2) です。PC ソース値は BF/S の命令アドレスです。8 ビットディスプレースメントは符号拡張後 2 倍しますので、分岐先との相対距離は-256 バイトから+254 バイトの範囲になります。

### (2) 注意

遅延分岐命令ですので、分岐成立時には本命令の直後の命令を先に実行してから分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。

直後の命令が、分岐命令の場合、それをスロット不当命令として認識します。

遅延分岐命令直後の遅延スロットに本命令が配置されたときには、スロット不当命令として認識します。

分岐先に届かないときは BF、BRA、JMP 命令などとの組み合わせで対応する必要があります。

### (3) 動作内容

```
BFS(int d)    /* BFS disp */
{
    int disp;
    unsigned int temp;

    temp=PC;
    if ((d&0x80)==0)
        disp=(0x000000FF & d);
    else    disp=(0xFFFFFFF0 | d);
    if (T==0)
        PC=PC+4+(disp<<1);
    else PC+=4;
    Delay_Slot(temp+2);
}
```

### (4) 使用例

```
CLRT          ;常に T=0
BT/S TRGET_T  ;T=0 のため分岐しません。
NOP           ;
BF/S TRGET_F  ;T=0 のため TRGET に分岐します。
ADD R0,R1     ;分岐に先立ち実行します。
NOP           ;
TRGET_F:      ; BF/S 命令の分岐先
```

## 10.7 BRA BRAnch

無条件分岐

## 分岐命令

遅延分岐命令

書式	動作概略	命令コード	実行ステート	Tビット
BRA label	PC+4+disp×2 PC	1010ddddddddddd	1	

### (1) 説明

無条件の遅延分岐命令です。分岐先はアドレス (PC+4+ディスプレースメント×2) です。PC ソース値は BRA の命令アドレスです。12 ビットディスプレースメントは符号拡張後 2 倍しますので、分岐先との相対距離は-4096 バイトから+4094 バイトの範囲になります。分岐先に届かないときは、JMP 命令によってこの分岐が可能になります。

### (2) 注意

遅延分岐命令ですので、本命令の直後の命令を先に実行してから、分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

### (3) 動作内容

```
BRA(int d)    /* BRA disp */
{
    int disp;
    unsigned int temp;

    temp=PC;
    if ((d&0x800)==0)
        disp=(0x00000FFF & d);
    else    disp=(0xFFFFF000 | d);
    PC=PC+4+(disp<<1);
    Delay_Slot(temp+2);
}
```

### (4) 使用例

```
BRA    TRGET        ;TRGET へ分岐します。
ADD    R0,R1        ;分岐に先立ち ADD を実行します。
NOP
TRGET:                ; BRA 命令の分岐先
```

## 10.8 BRAF BRAnch Far

無条件分岐

## 分岐命令

遅延分岐命令

書式	動作概略	命令コード	実行ステート	Tビット
BRAF Rn	PC+4+Rn PC	0000nnnn00100011	2	

### (1) 説明

無条件の遅延分岐命令です。分岐先はアドレス（PC+4+Rn）です。分岐先アドレスはPCに4と汎用レジスタRnの内容の32ビットを加えたアドレスです。

### (2) 注意

遅延分岐命令ですので、本命令の直後の命令を先に実行してから、分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

### (3) 動作内容

```
BRAF(int n) /* BRAF Rn */
{
    unsigned int temp;

    temp=PC;
    PC=PC+4+R[n];
    Delay_Slot(temp+2);
}
```

### (4) 使用例

```
MOV.L #(TRGET-BRAF_PC),R0 ;アドレスレジスタを設定します。
BRAF R0                    ;TRGETへ分岐します。
ADD R0,R1                  ;分岐に先立ちADDを実行します。
BRAF_PC:                   ;
NOP
TRGET:                     ; BRAF命令の分岐先
```

## 10.9 BSR Branch to SubRoutine

## 分岐命令

サブルーチンプロ

シージャへの分岐

遅延分岐命令

書式	動作概略	命令コード	実行ステート	Tビット
BSR label	PC+4 PR, PC+4+disp×2 PC	1011ddddddddddd	1	

## (1) 説明

アドレス (PC+4+ディスプレースメント×2) に分岐し、PR にアドレス (PC+4) を格納します。PC ソース値は BSR の命令アドレスです。12 ビットディスプレースメントは符号拡張後 2 倍しますので、分岐先との相対距離は - 4096 バイトから +4094 バイトの範囲になります。分岐先に届かないときは、JSR 命令によってこの分岐が可能になります。

## (2) 注意

遅延分岐命令ですので、本命令の直後の命令を先に実行してから、分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

## (3) 動作内容

```
BSR(int d) /* BSR disp */
{
    int disp;
    unsigned int temp;

    temp=PC;
    if ((d&0x800)==0)
        disp=(0x00000FFF & d);
    else disp=(0xFFFFF000 | d);
    PR=PC+4;
    PC=PC+4+(disp<<1);
    Delay_Slot(temp+2);
}
```

### (4) 使用例

```
BSR  TRGET      ;TRGET へ分岐します。
MOV  R3,R4      ;分岐に先立ち MOV を実行します。
ADD  R0,R1      ;サブ ルーチン プ ロシ ャからの戻り先 (PR の内容) です。
.....
.....

TRGET:          ; プ ロシ ャの入り口
MOV  R2,R3      ;
RTS             ;上記 ADD 命令に戻ります。
MOV  #1,R0      ;分岐に先立ち MOV を実行します。
```

## 10.10 BSRF Branch to SubRoutine Far 分岐命令

サブルーチンプロ

シージャへの分岐

遅延分岐命令

書式	動作概略	命令コード	実行ステート	Tビット
BSRF Rn	PC+4 PR, PC+4+Rn PC	0000nnnn00000011	2	

## (1) 説明

アドレス (PC+4+Rn) に分岐し、PR にアドレス (PC+4) を格納します。PC ソース値は BSRF の命令アドレスです。分岐先は PC+4 に汎用レジスタ Rn の内容の 32 ビットデータを加えたアドレスです。

## (2) 注意

遅延分岐命令ですので、本命令の直後の命令を先に実行してから、分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

## (3) 動作内容

```
BSRF(int n)      /* BSRF Rn */
{
    unsigned int temp;

    temp=PC;
    PR=PC+4;
    PC=PC+4+R[n];
    Delay_Slot(temp+2);
}
```

## (4) 使用例

```
MOV.L #(TRGET-BSRF_PC),R0    ;ディスプレイメントを設定します。
BSRF R0                      ;TRGET へ分岐します。
MOV R3,R4                    ;分岐に先立ち MOV を実行します。
BSRF_PC:                      ;
    ADD R0,R1                 ;
    .....
TRGET:                        ; プログラムの入り口
    MOV R2,R3                 ;
    RTS                       ;上記 ADD 命令に戻ります。
    MOV #1,R0                 ;分岐に先立ち MOV を実行します。
```



## 10.11 BT Branch if True

## 分岐命令

条件分岐

書式	動作概略	命令コード	実行ステート	Tビット
BT label	T=1 のとき PC+4+disp×2 PC, T=0 のとき nop	10001001ddddddd	1	

## (1) 説明

Tビットを参照する条件付き分岐命令です。T=1 のとき、分岐します。逆に T=0 のとき、分岐しません。

分岐先はアドレス (PC+4+ディスプレースメント×2) です。PC ソース値は BT の命令アドレスです。8 ビットディスプレースメントは符号拡張後 2 倍しますので、分岐先との相対距離は - 256 バイトから +254 バイトの範囲になります。

## (2) 注意

分岐先に届かないときは BRA、JMP 命令などとの組み合わせで対応する必要があります。

## (3) 動作内容

```
BT(int d)    /* BT disp */
{
    int disp;

    if ((d&0x80)==0)
        disp=(0x000000FF & d);
    else disp=(0xFFFFF00 | d);
    if (T==1)
        PC=PC+4+(disp<<1);
    else PC+=2;
}
```

## (4) 使用例

```
SETT          ;常に T=1
BF TRGET_F    ;T=1 のため分岐しません。
BT TRGET_T    ;T=1 のため TRGET_T へ分岐します。
NOP           ;
NOP           ;
TRGET_T:      ; BT 命令の分岐先
```

## 10.12 BT/S Branch if True with delay Slot

遅延付き条件分岐

## 分岐命令

遅延分岐命令

書式	動作概略	命令コード	実行ステート	Tビット
BT/S label	T=1 のとき PC+4+disp×2 PC, T=0 のとき nop	10001101ddddddd	1	

### (1) 説明

Tビットを参照する遅延付き条件分岐命令です。T=1 のとき、分岐します。T=0 のとき、分岐しません。

PC ソース値は BT/S の命令アドレスです。8 ビットディスプレースメントは符号拡張後 2 倍しますので、分岐先との相対距離は - 256 バイトから + 254 バイトの範囲になります。分岐先に届かないときは BRA、JMP 命令などとの組み合わせで対応する必要があります。

### (2) 注意

遅延分岐命令ですので、分岐成立時には本命令の直後の命令を先に実行してから分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。

直後の命令が、分岐命令の場合、それをスロット不当命令として認識します。

### (3) 動作内容

```
BTS(int d)    /* BTS disp */
{
    int disp;
    unsigned temp;

    temp=PC;
    if ((d&0x80)==0)
        disp=(0x000000FF & d);
    else disp=(0xFFFFFFF0 | d);
    if (T==1)
        PC=PC+4+(disp<<1);
    else PC+=4;
    Delay_Slot(temp+2);
}
```

### (4) 使用例

```
SETT                ;常に T=1
BF/S TRGET_F        ;T=1 のため分岐しません。
NOP                ;
BT/S TRGET_T        ;T=1 のため TRGET_T に分岐します。
ADD R0,R1           ;分岐に先立ち実行します。
NOP                ;
TRGET_T:            ; BT/S 命令の分岐先
```

## 10.13 CLRMAC CLear MAC register

## システム制御命令

MAC レジスタの  
クリア

書式	動作概略	命令コード	実行 ステート	Tビット
CLRMAC	0 MACH,MACL	0000000000101000	1	

## (1) 説明

MACH、MACL レジスタをクリアします。

## (2) 動作内容

```
CLRMAC( )    /* CLRMAC */
{
    MACH=0;
    MACL=0;
    PC+=2;
}
```

## (3) 使用例

```
CLRMAC          ;MACレジスタをクリアして初期化します。
MAC.W  @R0+,@R1+ ;積和演算
MAC.W  @R0+,@R1+ ;
```

## 10.14 CLRS      CLeaR Sbit

S ビットのクリア

## システム制御命令

---

書式	動作概略	命令コード	実行 ステート	T ビット
CLRS	0 S	0000000001001000	1	0

### (1) 説明

S ビットを 0 にクリアします。

### (2) 動作内容

```
CLRS( )    /* CLRS */  
{  
    S=0;  
    PC+=2;  
}
```

### (3) 使用例

```
CLRS        ;実行前 S=1  
            ;実行後 S=0
```

## 10.15 CLRT CLeaR Tbit

### Tビットのクリア

## システム制御命令

書式	動作概略	命令コード	実行 ステート	Tビット
CLRT	0 T	00000000000001000	1	0

#### (1) 説明

Tビットをクリアします。

#### (2) 動作内容

```
CLRT( ) /* CLRT */
```

```
{
```

```
    T=0;
```

```
    PC+=2;
```

```
}
```

#### (3) 使用例

```
CLRT ;実行前 T=1
```

```
      ;実行後 T=0
```

## 10.16 CMP/cond CoMPare conditionally 比較

## 算術演算命令

書式	動作概略	命令コード	実行 ステート	T ビット
CMP/EQ Rm,Rn	Rn=Rm のとき 1 T	0011nnnnnnnnnn0000	1	比較結果
CMP/GE Rm,Rn	有符号で Rn Rm のとき 1 T	0011nnnnnnnnnn0011	1	比較結果
CMP/GT Rm,Rn	有符号で Rn>Rm のとき 1 T	0011nnnnnnnnnn0111	1	比較結果
CMP/HI Rm,Rn	無符号で Rn>Rm のとき 1 T	0011nnnnnnnnnn0110	1	比較結果
CMP/HS Rm,Rn	無符号で Rn Rm のとき 1 T	0011nnnnnnnnnn0010	1	比較結果
CMP/PL Rn	Rn>0 のとき 1 T	0100nnnn00010101	1	比較結果
CMP/PZ Rn	Rn 0 のとき 1 T	0100nnnn00010001	1	比較結果
CMP/STR Rm,Rn	いずれかのバイトが等しいとき 1 T	0010nnnnnnnnnn1100	1	比較結果
CMP/EQ #imm,R0	R0=imm のとき 1 T	10001000iiiiiiii	1	比較結果

### (1) 説明

汎用レジスタ Rn と Rm とを比較し、その結果、指定された条件(cond)が成立していると T ビットをセットします。条件が不成立のときは T ビットをクリアします。Rn の内容は変化しません。9 条件が指定できます。PZ と PL の 2 条件については Rn と 0 との比較になります。

EQ の条件については符号拡張した 8 ビットのイミディエイトデータと R0 との比較も可能です。R0 の内容は変化しません。

ニーモニック	説明
CMP/EQ Rm,Rn	Rn=Rm のとき T=1
CMP/GE Rm,Rn	有符号値として Rn Rm のとき T=1
CMP/GT Rm,Rn	有符号値として Rn>Rm のとき T=1
CMP/HI Rm,Rn	無符号値として Rn>Rm のとき T=1
CMP/HS Rm,Rn	無符号値として Rn Rm のとき T=1
CMP/PL Rn	Rn>0 のとき T=1
CMP/PZ Rn	Rn 0 のとき T=1
CMP/STR Rm,Rn	いずれかのバイトが等しいとき T=1
CMP/EQ #imm,R0	R0=imm のとき T=1

## (2) 動作内容

```

CMPSEQ(long m, long n)    /* CMP_EQ Rm,Rn */
{
    if (R[n]==R[m]) T=1;
    else T=0;
    PC+=2;
}

CMPGE(long m, long n)    /* CMP_GE Rm,Rn */
{
    if ((long)R[n]>=(long)R[m]) T=1;
    else T=0;
    PC+=2;
}

CMPGT(long m, long n)    /* CMP_GT Rm,Rn */
{
    if ((long)R[n]>(long)R[m]) T=1;
    else T=0;
    PC+=2;
}

CMPHI(long m, long n)    /* CMP_HI Rm,Rn */
{
    if ((unsigned long)R[n]>(unsigned long)R[m]) T=1;
    else T=0;
    PC+=2;
}

CMPHS(long m, long n)    /* CMP_HS Rm,Rn */
{
    if ((unsigned long)R[n]>=(unsigned long)R[m]) T=1;
    else T=0;
    PC+=2;
}

CMPPL(long n)            /* CMP_PL Rn */
{
    if ((long)R[n]>0) T=1;
    else T=0;
    PC+=2;
}

CMPPPZ(long n)           /* CMP_PZ Rn */
{
    if ((long)R[n]>=0) T=1;
    else T=0;
    PC+=2;
}

```



```
}

CMPSTR(long m, long n)    /* CMP_STR Rm,Rn */
{
    unsigned long temp;
    long HH,HL,LH,LL;

    temp=R[n]^R[m];
    HH=(temp&0xFF000000)>>24;
    HL=(temp&0x00FF0000)>>16;
    LH=(temp&0x0000FF00)>>8;
    LL=temp&0x000000FF;
    HH=HH&&HL&&LH&&LL;
    if (HH==0) T=1;
    else T=0;
    PC+=2;
}

CMPIM(long i)    /* CMP_EQ #imm,R0 */
{
    long imm;

    if ((i&0x80)==0) imm=(0x000000FF & (long i));
    else imm=(0xFFFFFFFF00 | (long i));
    if (R[0]==imm) T=1;
    else T=0;
    PC+=2;
}
```

(3) 使用例

```
CMP/GE R0,R1    ;R0=H'7FFFFFFF,R1=H'80000000
BT TRGET_T      ;T=0 なので分岐しません。

CMP/HS R0,R1    ;R0=H'7FFFFFFF,R1=H'80000000
BT TRGET_T      ;T=1 なので分岐します。

CMP/STR R2,R3    ;R2="ABCD",R3="XYZC"
BT TRGET_T      ;T=1 なので分岐します。
```

## 10.17 DIV0S      DIvide(step0) as Signed

## 算術演算命令

符号付き除算の  
初期化

書式	動作概略	命令コード	実行 ステート	Tビット
DIV0S   Rm,Rn	Rn の MSB   Q, Rm の MSB   M, M^Q   T	0010nnnnnnmmmm0111	1	計算結果

## (1)      説明

符号付き除算の初期設定をします。本命令に続けて 1 桁分の除算をする DIV1 命令などを組み合わせて、繰り返し除算を行い商を求めます。詳しくは DIV1 の説明を参照してください。

## (2)      動作内容

```

DIV0S(long m, long n)    /* DIV0S Rm,Rn */
{
    if ((R[n] & 0x80000000)==0) Q=0;
    else Q=1;
    if ((R[m] & 0x80000000)==0) M=0;
    else M=1;
    T=!(M==Q);
    PC+=2;
}

```

## (3)      使用例

DIV1 の使用例を参照してください。

## 10.18 DIV0U      DIVide (step0) as Unsigned      算術演算命令

符号なし除算の  
初期化

---

書式	動作概略	命令コード	実行 ステート	Tビット
DIV0U	0 M/Q/T	00000000000011001	1	0

### (1) 説明

符号なし除算の初期設定をします。本命令に続けて1桁分の除算をするDIV1命令などを組み合わせて、繰り返し除算を行い商を求めます。詳しくはDIV1の説明を参照してください。

### (2) 動作内容

```
DIV0U( )            /* DIV0U */  
{  
    M=Q=T=0;  
    PC+=2;  
}
```

### (3) 使用例

DIV1の使用例を参照してください。

## 10.19 DIV1      DIvIde 1 step

### 除算

## 算術演算命令

書式	動作概略	命令コード	実行 ステート	Tビット
DIV1 Rm,Rn	1 ステップ除算 ( $Rn \div Rm$ )	0011nnnnnnmmmm0100	1	計算結果

### (1) 説明

汎用レジスタ Rn の 32 ビットの内容(被除数)を Rm の内容(除数)で 1 桁分の除算(1 ステップ除算)を実行する命令です。本命令単独でまたは他の命令と組み合わせて繰り返し実行し商を求めます。この繰り返し中は、指定したレジスタと M、Q、T ビットを書き替えないでください。

1 ステップ除算とは、被除数を左に 1 ビットシフトし、それから除数を減算し、結果の正負によって商のビットを Q ビットに反映するという処理を実行します。

割り算で余りを求めるには、DIV1 命令を用いて商を求めた後、

$$(\text{余り}) = (\text{被除数}) - (\text{除数}) \times (\text{商})$$

として求めてください。

ゼロ除算とオーバフローの検出は用意していません。除算の前にゼロ除算とオーバフロー除算をチェックしてください。剰余の演算は用意していません。除数と求められた商との積を求めて、被除数から減算して剰余を求めてください。

最初に、DIV0S または DIV0U で初期設定します。DIV1 を除数のビット数分繰り返します。商が 17 ビット以上必要なとき、ROTCL を DIV1 の前に置きます。詳しい 除算のシーケンスは使用例を参考にしてください。

## (2) 動作内容

```
DIVl(long m, long n)    /* DIVl Rm,Rn */
{
    unsigned long tmp0;
    unsigned char  old_q, tmp1;

    old_q=Q;
    Q=(unsigned char)((0x80000000 & R[n])!=0);
    R[n]<=<=1;
    R[n]|=(unsigned long)T;
    switch(old_q){
    case 0:switch(M){
        case 0:tmp0=R[n];
            R[n]-=R[m];
            tmp1=(R[n]>tmp0);
            switch(Q){
            case 0:Q=tmp1;
                break;
            case 1:Q=(unsigned char)(tmp1==0);
                break;
            }
            break;
        case 1:tmp0=R[n];
            R[n]+=R[m];
            tmp1=(R[n]<tmp0);
            switch(Q){
            case 0:Q=(unsigned char)(tmp1==0);
                break;
            case 1:Q=tmp1;
                break;
            }
            break;
        }
        break;
    case 1:switch(M){
        case 0:tmp0=R[n];
            R[n]+=R[m];
            tmp1=(R[n]<tmp0);
            switch(Q){
            case 0:Q=tmp1;
                break;
            case 1:Q=(unsigned char)(tmp1==0);
                break;
            }
            break;
        }
    }
}
```

```

        case 1:tmp0=R[n];
            R[n]-=R[m];
            tmp1=(R[n]>tmp0);
            switch(Q){
                case 0:Q=(unsigned char)(tmp1==0);
                    break;
                case 1:Q=tmp1;
                    break;
            }
            break;
        }
        break;
    }
    T=(Q==M);
    PC+=2;
}

```

## (3) 使用例 1

```

                                ;R1(32ビット)÷R0(16ビット)=R1(16ビット):符号なし
SHLL16    R0                    ;除数を上位16ビット、下位16ビットを0に設定
TST       R0,R0                 ;ゼロ除算チェック
BT        ZERO_DIV              ;
CMP/HS    R0,R1                 ;オーバーフローチェック
BT        OVER_DIV              ;
DIV0U     ;フラグの初期化
.arepeat  16                    ;
DIV1      R0,R1                 ;16回繰り返し
.aendr
ROTCL     R1                    ;
EXTU.W    R1,R1                 ;R1=商

```

## (4) 使用例 2

```

                                ;R1:R2(64ビット)÷R0(32ビット)=R2(32ビット):符号なし
TST       R0,R0                 ;ゼロ除算チェック
BT        ZERO_DIV              ;
CMP/HS    R0,R1                 ;オーバーフローチェック
BT        OVER_DIV              ;
DIV0U     ;フラグの初期化
.arepeat  32                    ;
ROTCL     R2                    ;32回繰り返し
DIV1      R0,R1                 ;
.aendr
ROTCL     R2                    ;R2=商

```

## (5) 使用例 3

```
SHLL16    R0          ;R1(16ビット)÷R0(16ビット)=R1(16ビット):符号付き
EXTS.W    R1,R1       ;除数を上位16ビット、下位16ビットを0に設定
XOR        R2,R2       ;被除数は符号拡張して32ビット
MOV        R2,R2       ;R2=0
ROTCL     R3           ;
SUBC       R2,R1       ;被除数が負のとき、-1する。
DIV0S      R0,R1       ;フラグの初期化
.arepeat 16           ;
DIV1       R0,R1       ;16回繰り返し
.aendr     ;
EXTS.W     R1,R1       ;
ROTCL      R1          ;R1=商(1の補数表現)
ADDC       R2,R1       ;商のMSBが1のとき、+1して2の補数表現に変換
EXTS.W     R1,R1       ;R1=商(2の補数表現)
```

## (6) 使用例 4

```
MOV        R2,R3       ;R2(32ビット)÷R0(32ビット)=R2(32ビット):符号付き
ROTCL      R3          ;
SUBC       R1,R1       ;被除数は符号拡張して64ビット(R1:R2)
XOR        R3,R3       ;R3=0
SUBC       R3,R2       ;被除数が負のとき、-1して1の補数表現に変換
DIV0S      R0,R1       ;フラグの初期化
.arepeat 32           ;
ROTCL      R2          ;32回繰り返し
DIV1       R0,R1       ;
.aendr     ;
ROTCL      R2          ;R2=商(1の補数表現)
ADDC       R3,R2       ;商のMSBが1のとき、+1して2の補数表現に変換
ROTCL      R2          ;R2=商(2の補数表現)
```

## 10.20 DMULS.L Double-length MULTiply as Signed 算術演算命令

符号付き倍精度乗算

書式	動作概略	命令コード	実行 ステート	Tビット
DMULS.L Rm,Rn	符号付きで $Rn \times Rm$ MACH,MACL	0011nnnnnnmmmm1101	2~5	

### (1) 説明

汎用レジスタ Rn の内容と Rm を 32 ビットで乗算し、結果の 64 ビットを MACH レジスタと MACL レジスタに格納します。演算は符号付き算術演算で行います。

### (2) 動作内容

```
DMULS(long m, long n) /* DMULS.L Rm,Rn */
{
    unsigned long RnL,RnH,RmL,RmH,Res0,Res1,Res2;
    unsigned long temp0,temp1,temp2,temp3;
    long tempm,tempn,fnLmL;

    tempn=(long)R[n];
    tempm=(long)R[m];
    if (tempn<0) tempn=0-tempn;
    if (tempm<0) tempm=0-tempm;
    if ((long)(R[n]^R[m])<0) fnLmL=-1;
    else fnLmL=0;

    temp1=(unsigned long)tempn;
    temp2=(unsigned long)tempm;

    RnL=temp1&0x0000FFFF;
    RnH=(temp1>>16)&0x0000FFFF;
    RmL=temp2&0x0000FFFF;
    RmH=(temp2>>16)&0x0000FFFF;

    temp0=RmL*RnL;
    temp1=RmH*RnL;
    temp2=RmL*RnH;
    temp3=RmH*RnH;

    Res2=0;
    Res1=temp1+temp2;
    if (Res1<temp1) Res2+=0x00010000;
    temp1=(Res1<<16)&0xFFFF0000;
    Res0=temp0+temp1;
    if (Res0<temp0) Res2++;
}
```



```
Res2=Res2+((Res1>>16)&0x0000FFFF)+temp3;
```

```
if (fnLmL<0) {  
    Res2=~Res2;  
    if (Res0==0)  
        Res2++;  
    else  
        Res0=(~Res0)+1;  
}
```

```
MACH=Res2;  
MACL=Res0;  
PC+=2;
```

```
}
```

(3) 使用例

DMULS.L	R0,R1	;実行前 R0=H'FFFFFFFE,R1=H'00005555
		;実行後 MACH=H'FFFFFFFF,MACL=H'FFFF5556
STS	MACH,R0	;演算結果(上位)を得る
STS	MACL,R1	;演算結果(下位)を得る

## 10.21 DMULU.L Double-length MULTiply as Unsigned 算術演算命令

符号なし倍精度乗算

書式	動作概略	命令コード	実行 ステート	Tビット
DMULU.L Rm,Rn	符号なしで $Rn \times Rm$ MACH,MACL	0011nnnnmmmm0101	2~5	

### (1) 説明

汎用レジスタ Rn の内容と Rm を 32 ビットで乗算し、結果の 64 ビットを MACH レジスタと MACL レジスタに格納します。演算は符号なし算術演算で行います。

### (2) 動作内容

```
DMULU(long m, long n) /* DMULU.L Rm,Rn */
{
    unsigned long RnL,RnH,RmL,RmH,Res0,Res1,Res2;
    unsigned long temp0,temp1,temp2,temp3;

    RnL=R[n]&0x0000FFFF;
    RnH=(R[n]>>16)&0x0000FFFF;

    RmL=R[m]&0x0000FFFF;
    RmH=(R[m]>>16)&0x0000FFFF;

    temp0=RmL*RnL;
    temp1=RmH*RnL;
    temp2=RmL*RnH;
    temp3=RmH*RnH;

    Res2=0
    Res1=temp1+temp2;
    if (Res1<temp1) Res2+=0x00010000;

    temp1=(Res1<<16)&0xFFFF0000;
    Res0=temp0+temp1;
    if (Res0<temp0) Res2++;

    Res2=Res2+((Res1>>16)&0x0000FFFF)+temp3;

    MACH=Res2;
    MACL=Res0;
    PC+=2;
}
```

## 10. 各命令の説明

---

### (3) 使用例

```
DMULU.L    R0,R1      ;実行前 R0=H'FFFFFFFE,R1=H'00005555
                                ;実行後 MACH=H'00005554,MACL=H'FFFF5556

STS         MACH,R0     ;演算結果(上位)を得る
STS         MACL,R1     ;演算結果(下位)を得る
```

## 10.22 DT Decrement and Test

## 算術演算命令

デクリメントと  
テスト

書式	動作概略	命令コード	実行 ステート	Tビット
DT Rn	Rn-1 Rn,Rnが0のとき 1 T Rnが0以外のとき 0 T	0100nnnn00010000	1	比較結果

## (1) 説明

汎用レジスタ Rn の内容を 1 デクリメントして、結果を 0(ゼロ)と比較します。結果が 0 のとき T ビットを 1 にセットします。結果が 0 以外のとき、T ビットを 0 にセットします。

## (2) 動作内容

```
DT(long n) /* DT Rn */
{
    R[n]--;
    if (R[n]==0) T=1;
    else T=0;
    PC+=2;
}
```

## (3) 使用例

```
MOV #4,R5      ;ループ回数を設定します。
LOOP:
    ADD R0,R1    ;
    DT R5        ;R5 の値をデクリメントし、0 になったかどうか判定します。
    BF LOOP      ;T=0 なら LOOP へ分岐します(この例では 4 回ループします)。
```

## 10.23 EXTS EXTend as Signed

## 算術演算命令

符号拡張

書式	動作概略	命令コード	実行 ステート	Tビット
EXTS.B Rm,Rn	Rm をバイトから 符号拡張 Rn	0110nnnnnnmmmm1110	1	
EXTS.W Rm,Rn	Rm をワードから 符号拡張 Rn	0110nnnnnnmmmm1111	1	

## (1) 説明

汎用レジスタ Rm の内容を符号拡張して、結果を Rn に格納します。

バイト指定のとき、Rn のビット 8 からビット 31 に Rm のビット 7 の内容を転送します。ワード指定のとき、Rn のビット 16 からビット 31 に Rm のビット 15 の内容を転送します。

## (2) 動作内容

```
EXTSB(long m, long n) /* EXTS.B Rm,Rn */
{
    R[n]=R[m];
    if ((R[m]&0x00000080)==0) R[n]&=0x000000FF;
    else R[n]|=0xFFFFF00;
    PC+=2;
}
```

```
EXTSW(long m, long n) /* EXTS.W Rm,Rn */
{
    R[n]=R[m];
    if ((R[m]&0x00008000)==0) R[n]&=0x0000FFFF;
    else R[n]|=0xFFFF0000;
    PC+=2;
}
```

## (3) 使用例

```
EXTS.B    R0,R1    ;実行前 R0=H'00000080
              ;実行後  R1=H'FFFFFF80
EXTS.W    R0,R1    ;実行前 R0=H'00008000
              ;実行後  R1=H'FFFF8000
```

## 10.24 EXTU      EXTend as Unsigned

### ゼロ拡張

## 算術演算命令

書式	動作概略	命令コード	実行 ステート	Tビット
EXTU.B Rm,Rn	Rm をバイトからゼロ拡張 Rn	0110nnnnnnmmmm1100	1	
EXTU.W Rm,Rn	Rm をワードからゼロ拡張 Rn	0110nnnnnnmmmm1101	1	

### (1) 説明

汎用レジスタ Rm の内容をゼロ拡張して、結果を Rn に格納します。

バイト指定のとき、Rn のビット 8 からビット 31 に 0 を転送します。ワード指定のとき、Rn のビット 16 からビット 31 に 0 を転送します。

### (2) 動作内容

```
EXTUB(long m, long n) /* EXTU.B Rm,Rn */
{
    R[n]=R[m];
    R[n]&=0x000000FF;
    PC+=2;
}
```

```
EXTUW(long m, long n) /* EXTU.W Rm,Rn */
{
    R[n]=R[m];
    R[n]&=0x0000FFFF;
    PC+=2;
}
```

### (3) 使用例

```
EXTU.B    R0,R1    ;実行前 R0=H'FFFFFF80
              ;実行後 R1=H'00000080

EXTU.W    R0,R1    ;実行前 R0=H'FFFF8000
              ;実行後 R1=H'00008000
```

## 10. 各命令の説明

---

### 10.25 FABS Floating Point Absolute Value 浮動小数点命令 浮動小数点絶対値

---

PR	書式	動作概略	命令コード	実行 ステート	Tビット
0	FABS FRn	FRn  FRn	1111nnnn01011101	1	
1	FABS DRn	DRn  DRn	1111nnn001011101	1	

#### (1) 説明

浮動小数点レジスタ FRn/DRn の内容の最上位ビットを 0 にクリアして、結果を FRn/DRn に格納します。

FPSCR の cause/flag 部分は更新されません。

#### (2) 動作内容

```
void FABS (int n){  
    FR[n] = FR[n] & 0x7fffffff;  
    pc += 2;  
}  
/* 精度に依存せず、同じ動作を行います。 */
```

## 10.26 FADD Floating Point Add

## 浮動小数点命令

浮動小数点加算

PR	書式	動作概略	命令コード	実行 ステート	Tビット
0	FADD FRm,FRn	FRn + FRm FRn	1111nnnnmmmm0000	6	
1	FADD DRm,DRn	DRn + DRm DRn	1111nnn0mmm00000	6	

## (1) 説明

FPSCR.PR=0 の場合：FRn と FRm の内容の 2 つの単精度浮動小数点数を算術加算し、結果を FRn に格納します。

FPSCR.PR=1 の場合：DRn と DRm の内容の 2 つの倍精度浮動小数点数を算術加算し、結果を DRn に格納します。

FPSCR.enable.O/U/I がセットされている場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

## (2) 動作内容

```
void FADD (int m,n)
{
    pc += 2;
    clear_cause();
    if((data_type_of(m) == sNaN) ||
        (data_type_of(n) == sNaN)) invalid(n);
    else if((data_type_of(m) == qNaN) ||
        (data_type_of(n) == qNaN)) qnan(n);
    else if((data_type_of(m) == DENORM) ||
        (data_type_of(n) == DENORM)) set_E();
    else switch (data_type_of(m)){
        case NORM: switch (data_type_of(n)){
            case NORM:    normal_faddsub(m,n,ADD); break;
            case PZERO:
            case NZERO: register_copy(m,n); break;
            default:      break;
        }            break;
        case PZERO: switch (data_type_of(n)){
            case NZERO:    zero(n,0); break;
            default:      break;
        }            break;
        case NZERO:      break;
    }
}
```



## 10. 各命令の説明

```

    case PINF: switch (data_type_of(n)){
        case NINF:    invalid(n);    break;
        default:     inf(n,0);       break;
    }              break;
    case NINF: switch (data_type_of(n)){
        case PINF:    invalid(n);    break;
        default:     inf(n,1);       break;
    }              break;
}
}

```

### FADD 特殊

FRm,DRm	FRn,DRn										
	NORM	+0	−0	+INF	−INF	DENORM	qNaN	sNaN			
NORM	ADD				−INF						
+0		+0									
−0			−0								
+INF				+INF	Invalid						
−INF	−INF			Invalid	−INF						
DENORM									Error		
qNaN									qNaN		
sNaN									Invalid		

(注) DN=1 の場合、非正規化数の値は 0 として扱われます。

発生する可能性のある例外

FPU エラー (FPU error)

無効演算 (Invalid operation)

オーバーフロー (Overflow)

アンダフロー (Underflow)

不正確例外 (Inexact)

## 10.27 FCMP Floating Point Compare

### 浮動小数点比較

## 浮動小数点命令

PR	書式	動作概略	命令コード	実行 ステート	Tビット
0	1.FCMP/EQ FRm,FRn	(FRn==FRm)?1:0 T	1111nnnnmmmm0100	1	1/0
1	2.FCMP/EQ DRm,DRn	(DRn==DRm)?1:0 T	1111nnn0mmmm00100	1	1/0
0	3.FCMP/GT FRm,FRn	(FRn>FRm)?1:0 T	1111nnnnmmmm0101	2	1/0
1	4.FCMP/GT DRm,DRn	(DRn>DRm)?1:0 T	1111nnn0mmmm00101	2	1/0

#### (1) 説明

1. FPSCR.PR=0 の場合：FRn と FRm の内容の 2 つの単精度浮動小数点数を算術比較し、等しい場合に T ビットに 1 を、他の場合に 0 を格納します。
2. FPSCR.PR=1 の場合：DRn と DRm の内容の 2 つの倍精度浮動小数点数を算術比較し、等しい場合に T ビットに 1 を、他の場合に 0 を格納します。
3. FPSCR.PR=0 の場合：FRn と FRm の内容の 2 つの単精度浮動小数点数を算術比較し、FRn>FRm の場合に T ビットに 1 を、他の場合に 0 を格納します。
4. FPSCR.PR=1 の場合：DRn と DRm の内容の 2 つの倍精度浮動小数点数を算術比較し、DRn>DRm の場合に T ビットに 1 を、他の場合に 0 を格納します。

#### (2) 動作内容

```
void FCMP_EQ(int m,n) /* FCMP/EQ FRm,FRn */
{
    pc += 2;
    clear_cause();
    if(fcmp_chk (m,n) == INVALID) fcmp_invalid();
    else if(fcmp_chk (m,n) == EQ)      T = 1;
    else                               T = 0;
}

void FCMP_GT(int m,n) /* FCMP/GT FRm,FRn */
{
    pc += 2;
    clear_cause();
    if ((fcmp_chk (m,n) == INVALID) ||
        (fcmp_chk (m,n) == UO)) fcmp_invalid();
    else if(fcmp_chk (m,n) == GT) T = 1;
    else                               T = 0;
}

int fcmp_chk (int m,n)
{
    if((data_type_of(m) == sNaN) ||
```

```
(data_type_of(n) == sNaN))      return(INVALID);
else if((data_type_of(m) == qNaN) ||
        (data_type_of(n) == qNaN))      return(UO);
else switch(data_type_of(m)){
    case NORM:      switch(data_type_of(n)){
        case PINF   :return(GT);   break;
        case NINF   :return(LT);   break;
        default:      break;
    }      break;
    case PZERO:
    case NZERO:      switch(data_type_of(n)){
        case PZERO   :
        case NZERO   :return(EQ);   break;
        default:      break;
    }      break;
    case PINF :      switch(data_type_of(n)){
        case PINF   :return(EQ);   break;
        default:return(LT);       break;
    }      break;
    case NINF :      switch(data_type_of(n)){
        case NINF   :return(EQ);   break;
        default:return(GT);       break;
    }      break;
}
if(FPSCR_PR == 0) {
    if(FR[n] == FR[m]) return(EQ);
    else if(FR[n] > FR[m]) return(GT);
    else
        return(LT);
}
else {
    if(DR[n>>1] == DR[m>>1]) return(EQ);
    else if(DR[n>>1] > DR[m>>1]) return(GT);
    else
        return(LT);
}
}

void fcmp_invalid()
{
    set_V();   if((FPSCR & ENABLE_V) == 0) T = 0;
}
}
```

## FCMP 特殊ケース

FCMP/EQ	FRn,DRn										
FRm,DRm	NORM	DNORM	+0	−0	+INF	−INF	qNaN	sNaN			
NORM	CMP										
DNORM											
+0									EQ		
−0											
+INF									EQ		
−INF					EQ						
qNaN					!EQ						
sNaN											

(注) DN=1 の場合、非正規化数の値は 0 として扱う。

FCMP/GT	FRn,DRn								
FRm,DRm	NORM	DENORM	+0	−0	+INF	−INF	qNaN	sNaN	
NORM	CMP				GT	!GT			
DENORM									
+0									
−0									
+INF				!GT	!GT		UO		
−INF	GT				!GT				
qNaN									
sNaN									Invalid

(注) DN=1 の場合、非正規化数の値は 0 として扱う。

発生する可能性のある例外

無効演算 (Invalid operation)

## 10.28 FCNVDS Floating Point Convert

Double to Single precision

浮動小数点命令

倍精度単精度変換

PR	書式	動作概略	命令コード	実行 ステート	Tビット
0					
1	FCNVDS DRm,FPUL	(float)DRm FPUL	1111mmm010111101	2	

## (1) 説明

FPSCR.PR=1 の場合：DRm 内の倍精度浮動小数点数を単精度浮動小数点数に変換し FPUL に格納します。

FPSCR.enable.O/U/I がセットされている場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FPUL は更新されません。ソフトウェアで適切な処理を行ってください。

## (2) 動作内容

```
void FCNVDS(int m){
    case((FPSCR.PR)){
        0: undefined_operation(); /* reserved */
        1: fcnvds(m); break; /* FCNVDS */
    }
}

void fcnvds(int m)
{
    pc += 2;
    clear_cause();
    case(data_type_of(m)){
        NORM :
        PZERO :
        NZERO :    normal_ fcnvds(m); break;
        DENORM : set_E();
        PINF :     FPUL = 0x7f800000; break;
        NINF :     FPUL = 0xff800000; break;
        qNaN :     FPUL = 0x7fbfffff; break;
        sNaN :     set_V();
                    if((FPSCR & ENABLE_V) == 0) FPUL = 0x7fbfffff;
                    else fpu_exception_trap(); break;
    }
}
```

```

void normal_fcnvds(int m)
{
    int sign;
    float abs;
    union {
        float f;
        int l;
    } dstf,tmpf;
    union {
        double d;
        int l[2];
    } dstd;
    dstd.d = DR[m>>1];
    if(dstd.l[1] & 0x1fffffff) set_I();
    if(FPSCR_RM == 1) dstd.l[1] &= 0xe0000000; /* round toward zero */
    dstf.f = dstd.d;
    check_single_exception(&FPUL, dstf.f);
}

```

## FCNVDS 特殊ケース

FRn	+NORM	-NORM	+0	-0	+INF	-INF	qNaN	sNaN
FCNVDS(FRn FPUL)	FCNVDS	FCNVDS	+0	-0	+INF	-INF	qNaN	Invalid

(注) DN=1 の場合、非正規化数の値は 0 として扱われます。

発生する可能性のある例外

FPU エラー (FPU error)

無効演算 (Invalid operation)

オーバフロー (Overflow)

アンダフロー (Underflow)

不正確例外 (Inexact)

## 10.29 FCNVSD Floating Point Convert Single to Double Precision

浮動小数点命令

単精度倍精度変換

---

PR	書式	動作概略	命令コード	実行 ステート	Tビット
0					
1	FCNVSD FPUL, DRn	Double(FPUL) DRn	1111nnn010101101	2	

### (1) 説明

FPSCR.PR=1 の場合 : FPUL の内容を単精度浮動小数点数と解釈し、倍精度浮動小数点数に変換し、結果を DRn に格納します。

### (2) 動作内容

```
void FCNVSD(int n){
    pc += 2;
    clear_cause();
    case((FPSCR_PR){
        0: undefined_operation(); /* reserved */
        1: fcnvds (n,&FPUL); break; /* FCNVSD */
    }
}

void fcnvds(int n, float *FPUL)
{
    case(fpul_typ()){
        PZERO :
        NZERO :
        PINF :
        NINF :      DR[n>>1] = *FPUL; break;
        DENORM : set_E();      break;
        qNaN :      qnan(n);    break;
        sNaN :      invalid(n);  break;
    }
}

int fpul_type()
{
    int abs;

    abs = FPUL & 0x7fffffff;
    if(abs < 0x00800000){
        if((FPSCR_DN == 1) || (abs == 0x00000000)){
```

```

        if(sign_of(src) == 0) return(PZERO);
        else                    return(NZERO);
    }
    else                    return(DENORM);
}
else if(abs < 0x7f800000) return(NORM);
else if(abs == 0x7f800000) {
    if(sign_of(src) == 0) return(PINF);
    else                    return(NINF);
}
else if(abs < 0x7fc00000) return(qNaN);
else                    return(sNaN);
}

```

## FCNVSD 特殊ケース

FRn	+NORM	-NORM	+0	-0	+INF	-INF	qNaN	sNaN
FCNVSD(FPUL FRn)	+NORM	-NORM	+0	-0	+INF	-INF	qNaN	Invalid

(注) DN=1 の場合、非正規化数の値は 0 として扱われます。

発生する可能性がある例外

FPU エラー (FPU error)

無効演算 (Invalid operation)



### 10.30 FDIV Floating Point Divide

浮動小数点除算

### 浮動小数点命令

PR	書式	動作概略	命令コード	実行 ステート	Tビット
0	FDIV FRm,FRm	FRn/FRm FRn	1111nnnnmmmm0011	10	
1	FDIV DRm,DRn	DRn/DRm DRn	1111nnn0mmm00011	23	

#### (1) 説明

FPSCR.PR=0 の場合：FRn と FRm の内容の 2 つの単精度浮動小数点数を算術除算し、結果を FRn に格納します。

FPSCR.PR=1 の場合：DRn と DRm の内容の 2 つの倍精度浮動小数点数を算術除算し、結果を DRn に格納します。

FPSCR.enable.O/U/I がセットされている場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

#### (2) 動作内容

```
void FDIV(int m,n)    /* FDIV FRm,FRn */
{
    pc += 2;
    clear_cause();
    if((data_type_of(m) == sNaN) ||
        (data_type_of(n) == sNaN)) invalid(n);
    else if((data_type_of(m) == qNaN) ||
        (data_type_of(n) == qNaN)) qnan(n);
    else switch (data_type_of(m)){
        case NORM: switch (data_type_of(n)){
            case PINF:
            case NINF:    inf(n,sign_of(m)^sign_of(n));break;
            case PZERO:
            case NZERO: zero(n,sign_of(m)^sign_of(n));break;
            case DENORM:  set_E();    break;
            default:      normal_fdiv(m,n); break;
        }    break;
        case PZERO: switch (data_type_of(n)){
            case PZERO:
            case NZERO: invalid(n);break;
            case PINF:
            case NINF:    break;
        }
    }
}
```

```
        default:      dz(n,sign_of(m)^sign_of(n));break;
    }    break;
case NZERO: switch (data_type_of(n)){
    case PZERO:
        case NZERO:invalid(n);    break;
        case PINF:    inf(n,1); break;
        case NINF:    inf(n,0); break;
        default:      dz(FR[n],sign_of(m)^sign_of(n)); break;
    }    break;
case DENORM:    set_E();    break;
case PINF :
case NINF : switch (data_type_of(n)){
    case PINF:
        case NINF: invalid(n);    break;
        default:    zero(n,sign_of(m)^sign_of(n));break
    }    break;
}
}
void normal_fdiv(int m,n)
{
union {
    float f;
    int l;
}    dstf,tmpf;
union {
    double d;
    int l[2];
}    dstd,tmpd;
union {
    int double x;
    int l[4];
}    tmpx;
if(FPSCR_PR == 0) {
    tmpf.f = FR[n]; /* save destination value */
    dstf.f /= FR[m]; /* round toward nearest or even */
    tmpd.d = dstf.f; /* convert single to double */
    tmpd.d *= FR[m];
    if(tmpf.f != tmpd.d) set_I();
}
```

## 10. 各命令の説明

```

    if((tmpf.f < tmpd.d) && (SPSCR_RM == 1))
        dstf.l -= 1; /* round toward zero */
    check_single_exception(&FR[n], dstf.f);
} else {
    tmpd.d = DR[n>>1]; /* save destination value */
    dstd.d /= DR[m>>1]; /* round toward nearest or even */
    tmpx.x = dstd.d; /* convert double to int double */
    tmpx.x *= DR[m>>1];
    if(tmpd.d != tmpx.x) set_I();
    if((tmpd.d < tmpx.x) && (SPSCR_RM == 1)) {
        dstd.l[1] -= 1; /* round toward zero */
        if(dstd.l[1] == 0xffffffff) dstd.l[0] -= 1;
    }
    check_double_exception(&DR[n>>1], dstd.d);
}
}

```

### FDIV 特殊ケース

FRm,DRm	FRn,DRn							
	NORM	+0	−0	+INF	−INF	DENORM	qNaN	sNaN
NORM	DIV	0		INF		Error	qNaN	Invalid
+0	DZ	Invalid		+INF	-INF	DZ		
−0				−INF	+INF			
+INF	0	+0	−0	Invalid		Error		
−INF		−0	+0					
DENORM	Error						qNaN	Invalid
qNaN								
sNaN								

(注) DN=1 の場合、非正規化数の値は 0 として扱われます。

発生する可能性がある例外

FPU エラー (FPU error)

無効演算 (Invalid operation)

ゼロ割り (Divide by zero)

オーバフロー (Overflow)

アンダフロー (Underflow)

不正確例外 (Inexact)

## 10.31 FIPR Floating Point Inner Product

## 浮動小数点命令

## 浮動小数点内積

PR	書式	動作概略	命令コード	実行 ステート	Tビット
0 1	FIPR FVm,FVn	FVn・FVm FR[n+3]	1111nnmm11101101	1	

(注) FV0 = {FR0, FR1, FR2, FR3}  
 FV4 = {FR4, FR5, FR6, FR7}  
 FV8 = {FR8, FR9, FR10, FR11}  
 FV12 = {FR12, FR13, FR14, FR15}

## (1) 説明

FPSCR.PR=0 の場合：FVn と FVm で示される 4 次元の単精度浮動小数点数ベクタを算術内積し、結果を FR[n+3] に格納します。

FIPR 命令は正確さよりも高速化のための命令です。そのため、FADD や FMUL を組み合わせて使用した場合と、結果が異なります。FIPR は次の順序で実行します。

- (1) 各項をそれぞれ乗算します。結果は 28 ビットです。
- (2) それらの結果をアライメントします。このとき、30 ビット以内に入るように丸めます。
- (3) アライメントした結果を加算します。
- (4) 正規化と丸めを行います。

以下の場合、特別な処理になります。

- (1) 入力値に sNaN がある場合、無効例外になります。
- (2) 乗算する入力値で 0 と無限大の組み合わせがある場合、無効例外になります。
- (3) 上記以外で、入力値に qNaN が含まれる場合、結果は qNaN になります。
- (4) 上記以外に入力値に無限大がある場合、
  - (a) もしも乗算した結果が 2 つ以上無限大になり、符号が異なる場合、無効例外になります。
  - (b) 上記以外の場合、正しい無限大が格納されます。
- (5) 入力値に sNaN、qNaN、無限大が含まれない場合は、通常と同じ処理を行います。

FPSCR.enable.O/U/I がセットされている場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

## (2) 動作内容

```
void FIPR(int m,n) /* FIPR FVm,FVn */
{
    if(FPSCR_PR == 0) {
        pc += 2;
        clear_cause();
        fipr(m,n);
    }
    else    undefined_operation();
}
```

## 10. 各命令の説明

---

発生する可能性がある例外

無効演算 (Invalid operation)

オーバフロー (Overflow)

アンダフロー (Underflow)

不正確例外 (Inexact)

## 10.32 FLDI0 Floating Point Load 0.0

0.0 ロード

## 浮動小数点命令

PR	書式	動作概略	命令コード	実行 ステート	T ビット
0 1	FLDI0 FRn	0x00000000 FRn	1111nnnn10001101	1	

### (1) 説明

FPSCR.PR=0 の場合、浮動小数点数の 0 ( 0x00000000)を FRn に格納します。

### (2) 動作内容

```
void FLDI0(int n)
{
    FR[n] = 0x00000000;
    pc += 2;
}
```

発生する可能性がある例外

なし

## 10. 各命令の説明

---

### 10.33 FLDI1 Floating Point Load 1.0 1.0 ロード

浮動小数点命令

---

PR	書式	動作概略	命令コード	実行 ステート	Tビット
0 1	FLDI1 FRn	0x3F800000 FRn	1111nnnn10011101	1	

#### (1) 説明

FPSCR.PR=0 の場合、浮動小数点数の 1.0 ( 0x3F800000)を FRn に格納します。

#### (2) 動作内容

```
void FLDI1(int n)
{
    FR[n] = 0x3F800000;
    pc += 2;
}
```

発生する可能性がある例外

なし

### 10.34 FLDS Floating Point Load to System Register

浮動小数点命令

システムレジスタへの転送

書式	動作概略	命令コード	実行 ステート	Tビット
FLDS FRm,FPUL	FRm    FPUL	1111mmmm00011101	1	

## (1) 説明

浮動小数点レジスタ FRm の内容をシステムレジスタである FPUL に格納します。

## (2) 動作内容

```
void FLDS(int n)
```

```
{
    FPUL = FR[m];
    pc += 2;
}
```

発生する可能性がある例外

なし



### 10.35 FLOAT Floating Point Convert from Integer 浮動小数点命令 整数浮動小数点数変換

PR	書式	動作概略	命令コード	実行 ステート	Tビット
0	FLOAT FPUL,FRn	(float)FPUL FRn	1111nnnn00101101	1	
1	FLOAT FPUL,DRn	(double)FPUL DRn	1111nnn000101101	2	

#### (1) 説明

FPSCR.PR=0 の場合：FPUL の内容を 32bit 整数とみなして、単精度浮動小数点数に変換し、結果を FRn に格納します。

FPSCR.PR=1 の場合：FPUL の内容を 32bit 整数とみなして、倍精度浮動小数点数に変換し、結果を DRn に格納します。

FPSCR.enable.I=1 と、FPSCR.PR=0 の場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

#### (2) 動作内容

```
void FLOAT(int n)
{
    union {
        double d;
        int l[2];
    } tmp;
    pc += 2;
    clear_cause();
    if(FPSCR.PR==0){
        FR[n] = FPUL; /* convert from integer to float */
        tmp.d = FPUL;
        if(tmp.l[1] & 0x1fffffff) inexact();
    } else {
        DR[n>>1] = FPUL; /* convert from integer to double */
    }
}
```

発生する可能性がある例外

不正確例外 (Inexact) : FPSCR.PR = 1 の場合、発生しません。

## 10.36 FMAC Floating Point Multiply and Accumulate

### 浮動小数点命令

浮動小数点積和

PR	書式	動作概略	命令コード	実行 ステート	Tビット
0 1	FMAC FR0,FRm,FRn	FR0*FRm + FRn FRn	1111nnnnmmmm1110	1	

#### (1) 説明

FPSCR.PR=0 の場合: FRn と FRm の内容の 2 つの単精度浮動小数点数を算術乗算し、さらに、FRn の内容を算術加算し、結果を FRn に格納します。

FPSCR.enable.O/U/I がセットされている場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

#### (2) 動作内容

```
void FMAC(int m,n)
{
    pc += 2;
    clear_cause();
    if(FPSCR_PR == 1) undefined_operation();
    else if((data_type_of(0) == sNaN) ||
            (data_type_of(m) == sNaN) ||
            (data_type_of(n) == sNaN)) invalid(n);
    else if((data_type_of(0) == qNaN) ||
            (data_type_of(m) == qNaN)) qnan(n);
    else if((data_type_of(0) == DENORM) ||
            (data_type_of(m) == DENORM)) set_E();
    else switch (data_type_of(0)){
        case NORM: switch (data_type_of(m)){
            case PZERO:
            case NZERO: switch (data_type_of(n)){
                case DENORM: set_E(); break;
                case qNaN: qnan(n); break;
                case PZERO:
                case NZERO: zero(n,sign_of(0)^ sign_of(m)^sign_of(n)); break;
                default: break;
            }
        }
        case PINF:
```

```
case NINF: switch (data_type_of(n)){
    case DENORM: set_E();break;
    case qNaN:   qnan(n);   break;
    case PINF:
    case NINF: if(sign_of(0)^ sign_of(m)^sign_of(n)) invalid(n);
               else      inf(n,sign_of(0)^ sign_of(m)); break;
    default:    inf(n,sign_of(0)^ sign_of(m)); break;
}
case NORM: switch (data_type_of(n)){
    case DENORM: set_E();   break;
    case qNaN:   qnan(n);   break;
    case PINF:
    case NINF:   inf(n,sign_of(n)); break;
    case PZERO:
    case NZERO:
    case NORM:   normal_fmac(m,n); break;
}      break;
case PZERO:
case NZERO: switch (data_type_of(m)){
    case PINF:
    case NINF:   invalid(n); break;
    case PZERO:
    case NZERO:
    case NORM: switch (data_type_of(n)){
        case DENORM: set_E(); break;
        case qNaN:   qnan(n); break;
        case PZERO:
        case NZERO:  zero(n,sign_of(0)^ sign_of(m)^sign_of(n)); break;
        default:     break;
    }      break;
}      break;
case PINF :
case NINF : switch (data_type_of(m)){
    case PZERO:
    case NZERO: invalid(n); break;
    default: switch (data_type_of(n)){
        case DENORM: set_E(); break;
        case qNaN:   qnan(n); break;
    }
```

```

        default:      inf(n,sign_of(0)^sign_of(m)^sign_of(n));break
    }      break;
}      break;
}

}

void normal_fmac(int m,n)
{
union {
    int double x;
    int l[4];
}    dstx,tmpx;
float dstf,srcf;

    if((data_type_of(n) == PZERO) || (data_type_of(n) == NZERO))
        srcf = 0.0; /* flush denormalized value */
    else    srcf = FR[n];
    tmpx.x = FR[0]; /* convert single to int double */
    tmpx.x *= FR[m]; /* exact product */
    dstx.x = tmpx.x + srcf;
    if(((dstx.x == srcf) && (tmpx.x != 0.0)) ||
        ((dstx.x == tmpx.x) && (srcf != 0.0))) {
        set_I();
        if(sign_of(0)^ sign_of(m)^ sign_of(n)) {
            dstx.l[3] -= 1; /* correct result */
            if(dstx.l[3] == 0xffffffff) dstx.l[2] -= 1;
            if(dstx.l[2] == 0xffffffff) dstx.l[1] -= 1;
            if(dstx.l[1] == 0xffffffff) dstx.l[0] -= 1;
        }
        else    dstx.l[3] |= 1;
    }
    if((dstx.l[1] & 0x01ffffff) || dstx.l[2] || dstx.l[3]) set_I();
    if(FPSCR_RM == 1) {
        dstx.l[1] &= 0xfe000000; /* round toward zero */
        dstx.l[2]  = 0x00000000;
        dstx.l[3]  = 0x00000000;
    }
    dstf = dstx.x;
    check_single_exception(&FR[n],dstf);
}

```

## 10. 各命令の説明

### FMAC 特殊ケース

FRn	FR0	FRm											
		+Norm	-Norm	+0	−0	+INF	−INF	Denorm	qNaN	sNaN			
Norm	Norm	MAC				INF		Error	qNaN	sNaN			
	0					Invalid							
	INF	INF		Invalid		INF							
+0	Norm	MAC											
	0					+0							
	INF	INF		Invalid		INF							
−0	+Norm	MAC		+0	−0	+INF	−INF						
	-Norm			−0	+0	−INF	+INF						
	+0	+0	−0	+0	−0	Invalid							
	-0	−0	+0	−0	+0								
	INF	INF		Invalid		INF							
+INF	+Norm	+INF				Invalid							
	-Norm					+INF							
	0					Invalid							
	+INF			Invalid		+INF							
	−INF	Invalid	+INF			+INF							
−INF	+Norm	−INF											
	-Norm												
	0												
	+INF	Invalid			Invalid		−INF						
	−INF	−INF			−INF		Invalid						
Denorm	Norm												
	0					Invalid							
	INF					Invalid							
!sNaN	Denorm										Error		
qNaN	0					Invalid							
	INF					Invalid							
	Norm												
!sNaN	qNaN								qNaN				
All types	sNaN												
SNaN	all types									Invalid			

(注) DN=1 の場合、非正規化数の値は 0 として扱われます。

発生する可能性がある例外

FPU エラー (FPU error)

無効演算 (Invalid operation)

オーバフロー (Overflow)

アンダフロー (Underflow)

不正確例外 (Inexact)

## 10.37 FMOV Floating Point Move

### 浮動小数点転送

## 浮動小数点命令

SZ	書式	動作概略	命令コード	実行 ステート	Tビット
0	1.FMOV FRm,FRn	FRm FRn	1111nnnnmmmm1100	1	
1	2.FMOV DRm,DRn	DRm DRn	1111nnn0mmmm01100	1	
0	3.FMOV.S FRm,@Rn	FRm (Rn)	1111nnnnmmmm1010	1	
1	4.FMOV DRm,@Rn	DRm (Rn)	1111nnn0mmmm01010	1	
0	5.FMOV.S @Rm,FRn	(Rm) FRn	1111nnnnmmmm1000	1	
1	6.FMOV @Rm,DRn	(Rm) DRn	1111nnn0mmmm01000	1	
0	7.FMOV.S @Rm+,FRn	(Rm) FRn,Rm+=4	1111nnnnmmmm1001	1	
1	8.FMOV @Rm+,DRn	(Rm) DRn,Rm+=8	1111nnn0mmmm01001	1	
0	9.FMOV.S FRm,@-Rn	Rn-=4,FRm (Rn)	1111nnnnmmmm1011	1	
1	10.FMOV DRm,@-Rn	Rn-=8,DRm (Rn)	1111nnn0mmmm01011	1	
0	11.FMOV.S @(R0,Rm),FRn	(R0+Rm) FRn	1111nnnnmmmm1100	1	
1	12.FMOV @(R0,Rm),DRn	(R0+Rm) DRn	1111nnn0mmmm00110	1	
0	13.FMOV.S FRm,@(R0,Rn)	FRm (R0+Rn)	1111nnnnmmmm0111	1	
1	14.FMOV DRm,@(R0,Rn)	DRm (R0+Rn)	1111nnn0mmmm01101	1	

### (1) 説明

1. FRm の内容を FRn に転送します。
2. DRm の内容を DRn に転送します。
3. FRm の内容を Rn が示すアドレスのメモリに転送します。
4. DRm の内容を Rn が示すアドレスのメモリに転送します。
5. Rm が示すアドレスのメモリの内容を FRn に転送します。
6. Rm が示すアドレスのメモリの内容を DRn に転送します。
7. Rm が示すアドレスのメモリの内容を FRn に転送し、Rm に 4 を加算します。
8. Rm が示すアドレスのメモリの内容を DRn に転送し、Rm に 8 を加算します。
9. Rn に 4 を減算し、FRm の内容をその Rn が示すアドレスのメモリに転送します。
10. Rn に 8 を減算し、DRm の内容をその Rn が示すアドレスのメモリに転送します。
11. (R0+Rm) が示すアドレスのメモリの内容を FRn に転送します。
12. (R0+Rm) が示すアドレスのメモリの内容を DRn に転送します。
13. FRm の内容を (R0+Rn) が示すアドレスのメモリに転送します。
14. DRm の内容を (R0+Rn) が示すアドレスのメモリに転送します。

### (2) 動作内容

```

void FMOV(int m,n)                /* FMOV FRm,FRn */
{
    FR[n] = FR[m];
    pc += 2;
}

void FMOV_DR(int m,n)             /* FMOV DRm,DRn */
{

```

## 10. 各命令の説明

---

```
    DR[n>>1] = DR[m>>1];
    pc += 2;
}

void FMOV_STORE(int m,n)      /* FMOV.S FRm,@Rn */
{
    store_int(FR[m],R[n]);
    pc += 2;
}

void FMOV_STORE_DR(int m,n)   /* FMOV DRm,@Rn */
{
    store_quad(DR[m>>1],R[n]);
    pc += 2;
}

void FMOV_LOAD(int m,n)       /* FMOV.S @Rm,FRn */
{
    load_int(R[m],FR[n]);
    pc += 2;
}

void FMOV_LOAD_DR(int m,n)    /* FMOV @Rm,DRn */
{
    load_quad(R[m],DR[n>>1]);
    pc += 2;
}

void FMOV_RESTORE(int m,n)    /* FMOV.S @Rm+,FRn */
{
    load_int(R[m],FR[n]);
    R[m] += 4;
    pc += 2;
}

void FMOV_RESTORE_DR(int m,n) /* FMOV @Rm+,DRn */
{
    load_quad(R[m],DR[n>>1]) ;
    R[m] += 8;
    pc += 2;
}

void FMOV_SAVE(int m,n)       /* FMOV.S FRm,@-Rn */
{
    store_int(FR[m],R[n]-4);
```

```

    R[n] -= 4;
    pc += 2;
}
void FMOV_SAVE_DR(int m,n) /* FMOV DRm,@-Rn */
{
    store_quad(DR[m>>1],R[n]-8);
    R[n] -= 8;
    pc += 2;
}
void FMOV_INDEX_LOAD(int m,n) /* FMOV.S @(R0,Rm),FRn */
{
    load_int(R[0] + R[m],FR[n]);
    pc += 2;
}
void FMOV_INDEX_LOAD_DR(int m,n) /*FMOV @(R0,Rm),DRn */
{
    load_quad(R[0] + R[m],DR[n>>1]);
    pc += 2;
}
void FMOV_INDEX_STORE(int m,n) /*FMOV.S FRm,@(R0,Rn)*/
{
    store_int(FR[m], R[0] + R[n]);
    pc += 2;
}
void FMOV_INDEX_STORE_DR(int m,n)/*FMOV DRm,@(R0,Rn)*/
{
    store_quad(DR[m>>1], R[0] + R[n]);
    pc += 2;
}

```

発生する可能性がある例外

データ TLB ミス例外 (Data TLB miss exception)

データ保護違反例外 (Data protection violation exception)

初期書き込み例外 (Initial write exception)

アドレスエラー (Address error)



## 10.38 FMOV Floating Point Move Extension

## 浮動小数点命令

## 浮動小数点転送

SZ	書式	動作概略	命令コード	実行 ステート	Tビット
1	1.FMOV XDm,@Rn	XRm (Rn)	1111nnnnmmmm11010	1	
1	2.FMOV @Rm,XDn	(Rm) XDn	1111nnn1mmmm1000	1	
1	3.FMOV @Rm+,XDn	(Rm) XDn,Rm+=8	1111nnn1mmmm1001	1	
1	4.FMOV XDm,@-Rn	Rn-=8,XDm (Rn)	1111nnnnmmmm01010	1	
1	5.FMOV @(R0,Rm),XDn	(R0+Rm) XDn	1111nnnnmmmm11011	1	
1	6.FMOV XDm,@(R0,Rn)	XDm (R0+Rn)	1111nnn1mmmm00110	1	
1	7.FMOV XDm,XDn	XDm XDn	1111nnn1mmmm11100	1	
1	8.FMOV XDm,DRn	XDm DRn	1111nnn0mmmm11100	1	
1	9.FMOV DRm,XDn	DRm XDn	1111nnn1mmmm01100	1	

## (1) 説明

1. XDm の内容を Rn が示すアドレスのメモリに転送します。
2. Rm が示すアドレスのメモリの内容を XDn に転送します。
3. Rm が示すアドレスのメモリの内容を XDn に転送し、Rm に 8 を加算します。
4. Rn に 8 を減算し、XDm の内容をその Rn が示すアドレスのメモリに転送します。
5. (R0+Rm)が示すアドレスのメモリの内容を XDn に転送します。
6. XDm の内容を(R0+Rn)が示すアドレスのメモリに転送します。
7. XDm の内容を XDn に転送します。
8. XDm の内容を DRn に転送します。
9. DRm の内容を XDn に転送します。

## (2) 動作内容

```

void FMOV_STORE_XD(int m,n)      /* FMOV XDm,@Rn */
{
    store_quad(XD[m>>1],R[n]);
    pc += 2;
}

void FMOV_LOAD_XD(int m,n)      /* FMOV @Rm,XDn */
{
    load_quad(R[m],XD[n>>1]);
    pc += 2;
}

void FMOV_RESTORE_XD(int m,n)  /* FMOV @Rm+,DBn */
{
    load_quad(R[m],XD[n>>1]);
    R[m] += 8;
    pc += 2;
}

```

```

}
void FMOV_SAVE_XD(int m,n)      /* FMOV XDm,@-Rn */
{
    store_quad(XD[m>>1],R[n]-8);
    R[n] -= 8;
    pc += 2;
}
void FMOV_INDEX_LOAD_XD(int m,n) /* FMOV @(R0,Rm),XDn */
{
    load_quad(R[0] + R[m],XD[n>>1]);
    pc += 2;
}
void FMOV_INDEX_STORE_XD(int m,n) /* FMOV XDm,@(R0,Rn) */
{
    store_quad(XD[m>>1], R[0] + R[n]);
    pc += 2;
}
void FMOV_XDXD(int m,n)      /* FMOV XDm,XDn */
{
    XD[n>>1] = XD[m>>1];
    pc += 2;
}
void FMOV_XDDR(int m,n)      /* FMOV XDm,DRn */
{
    DR[n>>1] = XD[m>>1];
    pc += 2;
}
void FMOV_DRXD(int m,n)      /* FMOV DRm,XDn */
{
    XD[n>>1] = DR[m>>1];
    pc += 2;
}

```

発生する可能性がある例外

データ TLB ミス例外 (Data TLB miss exception)

データ保護違反例外 (Data protection violation exception)

初期書き込み例外 (Initial write exception)

アドレスエラー (Address error)

## 10.39 FMUL Floating Point Multiply

## 浮動小数点命令

## 浮動小数点乗算

PR	書式	動作概略	命令コード	実行 ステート	Tビット
0	FMUL FRm,FRn	FRn*FRm FRn	1111nnnnmmmm0010	1	
1	FMUL DRm,DRn	DRn*DRm DRn	1111nnn0mmmm00010	6	

## (1) 説明

FPSCR.PR=0 の場合: FRn と FRm の内容の 2 つの単精度浮動小数点数を算術乗算し、結果を FRn に格納します。

FPSCR.PR=1 の場合: DRn と DRm の内容の 2 つの倍精度浮動小数点数を算術乗算し、結果を DRn に格納します。

FPSCR.enable.O/U/I がセットされている場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

## (2) 動作内容

```
void FMUL(int m,n)
{
    pc += 2;

    clear_cause();

    if((data_type_of(m) == sNaN) ||
        (data_type_of(n) == sNaN)) invalid(n);
    else if((data_type_of(m) == qNaN) ||
        (data_type_of(n) == qNaN)) qnan(n);
    else if((data_type_of(m) == DENORM) ||
        (data_type_of(n) == DENORM)) set_E();
    else switch (data_type_of(m)){
        case NORM: switch (data_type_of(n)){
            case PZERO:
            case NZERO: zero(n,sign_of(m)^sign_of(n)); break;
            case PINF:
            case NINF: inf(n,sign_of(m)^sign_of(n)); break;
            default: normal_fmula(m,n); break;
        }
        case PZERO:
        case NZERO: switch (data_type_of(n)){
            case PINF:
            case NINF: invalid(n); break;
        }
    }
}
```

```

        default:      zero(n,sign_of(m)^sign_of(n));break;
    }    break;
    case PINF :
    case NINF : switch (data_type_of(n)){
        case PZERO:
        case NZERO:invalid(n);    break;
        default:      inf(n,sign_of(m)^sign_of(n));break
    }    break;
}
}
}

```

## FMUL 特殊ケース

FRm,DRm	FRn,DRn							
	NORM	+0	−0	+INF	−INF	DENORM	qNaN	sNaN
NORM	MUL	0		INF		Error	qNaN	Invalid
+0	0	+0	−0	Invalid				
−0		−0	+0					
+INF	INF	Invalid		+INF	−INF			
−INF				−INF	+INF			
DENORM	Error							
qNaN	qNaN							
sNaN	Invalid							

(注) DN=1 の場合、非正規化数の値は 0 として扱われます。

発生する可能性がある例外

FPU エラー (FPU error)

無効演算 (Invalid operation)

オーバフロー (Overflow)

アンダフロー (Underflow)

不正確例外 (Inexact)

## 10. 各命令の説明

---

### 10.40 FNEG Floating Point Negate Value 浮動小数点命令 浮動小数点符号反転

---

PR	書式	動作概略	命令コード	実行 ステート	Tビット
0	FNEG FRn	-FRn FRn	1111nnnn01001101	1	
1	FNEG DRn	-DRn DRn	1111nnn001001101	1	

#### (1) 説明

浮動小数点レジスタ FRn/DRn の内容の最上位ビット(符号ビット)を反転して、結果を FRn/DRn に格納します。

FPSCR の cause/flag 部分は更新されません。

#### (2) 動作内容

```
void FNEG (int n){  
    FR[n] = -FR[n];  
    pc += 2;  
}
```

/\* 精度に依存せず、同じ動作を行います。 \*/

発生する可能性がある例外

なし

## 10.41 FRCHG FR-Bit Change

## 浮動小数点命令

FR ビット反転

PR	書式	動作概略	命令コード	実行 ステート	T ビット
0 1	FRCHG	FRSCR.FR=~FRSCR.FR	1111101111111101	1	

## (1) 説明

浮動小数点状態レジスタ FPSCR の FR ビットを反転します。FPSCR の FR ビットを換えると、FPPR0 から FPPR31 の中の、FR0 から FR15 が、XR0 から XR15 になり、XR0 から XR15 が、FR0 から FR15 になります。FPSCR.FR=0 のとき、FPPR0 から FPPR15 が FR0 から FR15 に対応し、FPPR16 から FPPR31 が XR0 から XR15 に対応します。FPSCR.FR=1 のとき、FPPR16 から FPPR31 が FR0 から FR15 に対応し、FPPR0 から FPPR15 が XR0 から XR15 に対応します。

## (2) 動作内容

```
void FRCHG() /* FRCHG */
{
    if(FPSCR_PR == 0){
        FPSCR ^= 0x00200000; /* bit 21 */
        PC += 2;
    }
    else undefined_operation();
}
```

発生する可能性がある例外

なし

## 10.42 FSCHG SZ-Bit Change

## 浮動小数点命令

## SZ ビット反転

PR	書式	動作概略	命令コード	実行 ステート	T ビット
0 1	FSCHG	FRSCR.SZ=~FRSCR.SZ	1111001111111101	1	

## (1) 説明

浮動小数点状態レジスタ FPSCR の SZ ビットを反転します。FPSCR の SZ ビットを換えると、FMOV 命令のデータ転送が、単精度データ 1 つか、ペアかが切り替わります。FPSCR.SZ=0 のとき、FMOV 命令は単精度データを一つ転送します。FPSCR.SZ=1 のとき、FMOV 命令は単精度データをペアで 2 つ転送します。

## (2) 動作内容

```
void FSCHG() /* FSCHG */
{
    if(FPSCR_PR == 0){
        FPSCR ^= 0x00100000; /* bit 20 */
        PC += 2;
    }
    else undefined_operation();
}
```

発生する可能性がある例外

なし

## 10.43 FSQRT Floating Point Square Root

## 浮動小数点命令

## 浮動小数点平方根

PR	書式	動作概略	命令コード	実行 ステート	Tビット
0	FSQRT FRn	FRn FDn	1111nnnn01101101	9	
1	FSQRT DRn	DRn DRn	1111nnnn01101101	22	

## (1) 説明

FPSCR.PR=0 の場合：FRn の内容の単精度浮動小数点の算術平方根を求め、結果を FRn に格納します。

FPSCR.PR=1 の場合：DRn の内容の単精度浮動小数点の算術平方根を求め、結果を DRn に格納します。

FPSCR.enable.I がセットされている場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

## (2) 動作内容

```
void FSQRT(int n){
    pc += 2;
    clear_cause();
    switch(data_type_of(n)){
        case NORM :    if(sign_of(n) == 0) normal_fsqrtn(n);
                        else        invalid(n); break;
        case DENORM:    if(sign_of(n) == 0) set_E();
                        else        invalid(n); break;
        case PZERO :
        case NZERO :
        case PINF :     break;
        case NINF :     invalid(n); break;
        case qNaN :     qnan(n);    break;
        case sNaN :     invalid(n); break;
    }
}

void normal_fsqrtn(int n)
{
    union {
        float f;
        int i;
    };
}
```



```
}    dstf,tmpf;
union {
    double d;
    int l[2];
}    dstd,tmpd;
union {
    int double x;
    int l[4];
}    tmpx;
if(FPSCR_PR == 0) {
    tmpf.f = FR[n]; /* save destination value */
    dstf.f = sqrt(FR[n]); /* round toward nearest or even */
    tmpd.d = dstf.f; /* convert single to double */
    tmpd.d *= dstf.f;
    if(tmpf.f != tmpd.d) set_I();
    if((tmpf.f < tmpd.d) && (SPSCR_RM == 1))
        dstf.l -= 1; /* round toward zero */
    if(FPSCR & ENABLE_I) fpu_exception_trap();
    else
        FR[n] = dstf.f;
} else {
    tmpd.d = DR[n>>1]; /* save destination value */
    dstd.d = sqrt(DR[n>>1]); /* round toward nearest or even */
    tmpx.x = dstd.d; /* convert double to int double */
    tmpx.x *= dstd.d;
    if(tmpd.d != tmpx.x) set_I();
    if((tmpd.d < tmpx.x) && (SPSCR_RM == 1)) {
        dstd.l[1] -= 1; /* round toward zero */
        if(dstd.l[1] == 0xffffffff) dstd.l[0] -= 1;
    }
    if(FPSCR & ENABLE_I) fpu_exception_trap();
    else
        DR[n>>1] = dstd.d;
}
}
```

## FSQRT 特殊ケース

FRn	+NORM	−NORM	+0	−0	+INF	−INF	qNaN	sNaN
FSQRT(FRn)	SQRT	Invalid	+0	−0	+INF	Invalid	qNaN	Invalid

（注）DN=1 の場合、非正規化数の値は 0 として扱われます。

発生する可能性がある例外

**FPU** エラー ( **FPU error** )

無効演算 ( **Invalid operation** )

不正確例外 ( **Inexact** )

## 10. 各命令の説明

---

### 10.44 FSTS Floating Point Store System Register 浮動小数点命令 システムレジスタからの転送

---

書式	動作概略	命令コード	実行 ステート	Tビット
FSTS FPUL,FRn	FPUL FRn	1111nnnn00001101	1	

#### (1) 説明

システムレジスタ FPUL の内容を浮動小数点レジスタ FRn に転送します。

#### (2) 動作内容

```
void FSTS(int n)
{
    FR[n] = FPUL;
    pc += 2;
}
```

発生する可能性がある例外

なし

## 10.45 FSUB Floating Point Subtract

## 浮動小数点命令

浮動小数点減算

PR	書式	動作概略	命令コード	実行 ステート	Tビット
0	FSUB FRm,FRn	FRn-FRm FRn	1111nnnnmmmm0001	1	
1	FSUB DRm,DRn	DRn-DRm DRn	1111nnn0mmm00001	6	

## (1) 説明

FPSCR.PR=0 の場合：FRn と FRm の内容の 2 つの単精度浮動小数点数を算術減算し、結果を FRn に格納します。

FPSCR.PR=1 の場合：DRn と DRm の内容の 2 つの倍精度浮動小数点数を算術減算し、結果を DRn に格納します。

FPSCR.enable.O/U/I がセットされている場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

## (2) 動作内容

```
void FSUB (int m,n)
{
    pc += 2;
    clear_cause();
    if((data_type_of(m) == sNaN) ||
        (data_type_of(n) == sNaN)) invalid(n);
    else if((data_type_of(m) == qNaN) ||
        (data_type_of(n) == qNaN)) qnan(n);
    else if((data_type_of(m) == DENORM) ||
        (data_type_of(n) == DENORM)) set_E();
    else switch (data_type_of(m)){
        case NORM: switch
            case NORM:    normal_faddsub(m,n,SUB); break;
            case PZERO:
            case NZERO: register_copy(m,n); FR[n] = -FR[n];break;
            default:      break;
        }          break;
    case PZERO:    break;
    case NZERO: switch
        case NZERO: zero(n,0); break;
        default:      break;
    }          break;
}
```

## 10. 各命令の説明

```

    case PINF: switch (data_type_of(n)){
        case PINF:    invalid(n);    break;
        default:     inf(n,1);      break;
    }                break;
    case NINF: switch (data_type_of(n)){
        case NINF:    invalid(n);    break;
        default:     inf(n,0);      break;
    }                break;
}
}

```

### FSUB 特殊ケース

FRm,DRm	FRn,DRn								
	NORM	+0	−0	+INF	−INF	DENORM	qNaN	sNaN	
NORM	SUB			+INF	−INF	Error	qNaN	Invalid	
+0			−0						
−0	+0								
+INF	−INF			Invalid					
−INF	+INF				Invalid				
DENORM									Error
qNaN						qNaN			
sNaN						Invalid			

(注) DN=1 の場合、非正規化数の値は 0 として扱われます。

発生する可能性がある例外

FPU エラー (FPU error)

無効演算 (Invalid operation)

オーバフロー (Overflow)

アンダフロー (Underflow)

不正確例外 (Inexact)

## 10.46 FTRC Floating Point Truncate and Convert to Integer

### 浮動小数点命令

整数への変換

PR	書式	動作概略	命令コード	実行ステート	Tビット
0	FTRC FRm,FPUL	(long)FRm FPUL	1111mmmm00111101	1	
1	FTRC DRm,FPUL	(long)DRm FPUL	1111mmmm00111101	2	

#### (1) 説明

FPSCR.PR=0 の場合:FRm の内容の単精度浮動小数点数を 32 ビット整数に変換し、結果を FPUL に格納します。

FPSCR.PR=1 の場合: DRm の内容の倍精度浮動小数点数を 32 ビット整数に変換し、結果を FPUL に格納します。

丸めモードは常に切り捨てになります。

FPSCR.enable.I がセットされている場合、命令実行の前に FPU 例外が発生しますので、ソフトウェアで適切な処理を行ってください。

#### (2) 動作内容

```
#define N_INT_SINGLE_RANGE 0xcf000000 /* -1.000000 * 2^31 */
#define P_INT_SINGLE_RANGE 0x4effffff /* 1.fffffe * 2^30 */
#define N_INT_DOUBLE_RANGE 0xc1e00000 /* higher of -1.0000000000000 * 2^31 */
#define P_INT_DOUBLE_RANGE 0x41dfffff /* higher of 1.ffffffffffffff * 2^30 */
```

```
void FTRC(int m)
{
    pc += 2;
    clear_cause();
    if(FPSCR.PR==0){
        case(ftrc_single_type_of(m)){
            NORM:    FPUL = FR[m];    break;
            PINF:    ftrc_invalid(0); break;
            NINF:    ftrc_invalid(1); break;
        }
    }
    else{
        /* case FPSCR.PR=1 */
        case(ftrc_double_type_of(m)){
            NORM:    FPUL = DR[m>>1]; break;
            PINF:    ftrc_invalid(0); break;
            NINF:    ftrc_invalid(1); break;
        }
    }
}
```

```
}
int ftrc_signle_type_of(int m)
{
    if(sign_of(m) == 0){
        if(FR_HEX[m] > 0x7f800000)    return(NINF);    /* NaN */
        else if(FR_HEX[m] > P_INT_SINGLE_RANGE)
            return(PINF);    /* out of range,+INF */
        else    return(NORM);    /* +0,+NORM */
    } else {
        if(FR_HEX[m]< N_INT_SINGLE_RANGE)
            return(NINF);    /* out of range ,+INF,NaN*/
        else    return(NORM);    /* -0,-NORM */
    }
}

int ftrc_double_type_of(int m)
{
    if(sign_of(m) == {
        if((FR_HEX[m] > 0x7ff00000) ||
            ((FR_HEX[m] == 0x7ff00000) &&
             (FR_HEX[m+1] != 0x00000000)))    return(NINF);    /* NaN */
        else if(FR_HEX[m] > P_INT_DOUBLE_RANGE)
            return(PINF);    /* out of range,+INF */
        else    return(NORM);    /* +0,+NORM */
    } else {
        if(FR_HEX[m] < N_INT_DOUBLE_RANGE)
            return(NINF);    /* out of range ,+INF,NaN*/
        else    return(NORM);    /* -0,-NORM */
    }
}

void ftrc_invalid(int sign)
{
    set_V();
    if((FPSCR & ENABLE_V) == 0){
        if(sign == 0)    FPUL = 0x7fffffff;
        else    FPUL = 0x80000000;
    }
    else fpu_exception_trap();
}
```

## FTRC 特殊ケース

FRn,DRn	NORM	+0	−0	Positive Out of Range	Negative Out of Range	+INF	−INF	qNaN	sNaN
FTRC (FRn,DRn)	TRC	0	0	Invalid +MAX	Invalid −MAX	Invalid +MAX	Invalid −MAX	Invalid −MAX	Invalid −MAX

（注）DN=1 の場合、非正規化数の値は 0 として扱われます。

発生する可能性がある例外

FPU エラー (FPU error)

無効演算 (Invalid operation)

不正確例外 (Inexact)



## 10.47 FTRV Floating Point Transform Vector 浮動小数点命令 ベクトル変換

PR	書式	動作概略	命令コード	実行 ステート	Tビット
0 1	FTRV XMTRX,FVn	XMTRX*FVn FVn	1111nn0111111101	4	

### (1) 説明

FPSCR.PR=0 の場合：XMTRX で示される、浮動小数点レジスタ XF0 から XF15 の内容を 4 行 4 列の行列とし、FVn で示される、浮動小数点レジスタ FR[n]から FR[n+3]の内容を 4 次元のベクトルとして、行列とベクトルの乗算を行い、結果を FVn に格納します。

$$\begin{matrix} \text{XMTRX} \\ \left[ \begin{array}{cccc} \text{XF}[0] & \text{XF}[4] & \text{XF}[8] & \text{XF}[12] \\ \text{XF}[1] & \text{XF}[5] & \text{XF}[9] & \text{XF}[13] \\ \text{XF}[2] & \text{XF}[6] & \text{XF}[10] & \text{XF}[14] \\ \text{XF}[3] & \text{XF}[7] & \text{XF}[11] & \text{XF}[15] \end{array} \right] \end{matrix} \times \begin{matrix} \text{FVn} \\ \left[ \begin{array}{c} \text{FR}[n] \\ \text{FR}[n+1] \\ \text{FR}[n+2] \\ \text{FR}[n+3] \end{array} \right] \end{matrix} = \begin{matrix} \text{FVn} \\ \left[ \begin{array}{c} \text{FR}[n] \\ \text{FR}[n+1] \\ \text{FR}[n+2] \\ \text{FR}[n+3] \end{array} \right] \end{matrix}$$

FTRV 命令は正確さよりも高速化のための命令です。そのため、FADD や FMUL を組み合わせて使用した場合と、結果が異なります。FTRV は次の順序で実行します。

1. 各項をそれぞれ乗算し、結果は 30 ビットです。
2. それらの結果をアライメントします。このとき、28 ビット以内に入るように丸めます。
3. アライメントした結果を加算します。
4. 正規化と丸めを行います。

以下の場合、特別な処理になります。

1. 入力値に sNaN がある場合、無効例外になります。
2. 乗算する入力値で 0 と無限大の組み合わせがある場合、無効演算例外になります。
3. 上記以外で、入力値に qNaN が含まれる場合、結果は qNaN になります。
4. 上記以外に入力値に無限大がある場合、
  - a)もしも乗算した結果が 2 つ以上無限大になり、符号が異なる場合、無効演算例外になります。
  - b)上記以外の場合、正しい無限大が格納されます。
5. 入力値に sNaN、qNaN、無限大が含まれない場合は、通常と同じ処理を行います。

FPSCR.enable.V/O/U/I がセットされている場合、FPU 例外トラップが、例外の発生如何に関わらず発生します。例外発生時は、FPSCR.cause FPSCR.flag には、例外の正しい情報が反映され、FRn/DRn は更新されません。ソフトウェアで適切な処理を行ってください。

### (2) 動作内容

```
void FTRV (int n)      /* FTRV FVn */
{
    float saved_vec[4],result_vec[4];
    int saved_fpscr;
    int dst,i;
```

```

if(FPSCR_PR == 0) {
    PC += 2;
    clear_cause();
    saved_fpscr = FPSCR;
    FPSCR &= ~ENABLE_VOUI; /* mask VOUI enable */
    dst = 12 - n; /* select other vector than FVn */
    for(i=0;i<4;i++) saved_vec [i] = FR[dst+i];
    for(i=0;i<4;i++) {
        for(j=0;j<4;j++) FR[dst+j] = XF[i+4j];
        fipr(n,dst);
        result_vec [i] = FR[dst+3];
    }
    for(i=0;i<4;i++) FR[dst+i] = saved_vec [i];
    FPSCR = saved_fpscr;
    if(FPSCR & ENABLE_VOUI)fpu_exception_trap();
    else for(i=0;i<4;i++) FR[n+i] = result_vec [i];
}
else undefined_operation();
}

```

発生する可能性がある例外

無効演算 (Invalid operation)

オーバーフロー (Overflow)

アンダフロー (Underflow)

不正確例外 (Inexact)

## 10.48 JMP JuMP

無条件分岐

## 分岐命令

遅延分岐命令

書式	動作概略	命令コード	実行ステート	Tビット
JMP @Rn	Rn PC	0100nnnn00101011	2	

### (1) 説明

Rn で指定したアドレスへ無条件に遅延分岐します。

### (2) 注意

遅延分岐命令ですので、本命令の直後の命令を先に実行してから、分岐先の命令を実行します。  
本命令と直後の命令との間には、割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

### (3) 動作内容

```
JMP(int n) /* JMP @Rn */
{
    unsigned int temp;

    temp=PC;
    PC=R[n];
    Delay_Slot(temp+2);
}
```

### (4) 使用例

```
MOV.L    JMP_TABLE,R0    ;R0=TARGET のアドレス
JMP      @R0              ;TARGET へ分岐します。
MOV      R0,R1            ;分岐に先立ち MOV を実行します。
.align   4
JMP_TABLE: .data.l    TARGET    ;ジャンプテーブル
.....
TARGET:   ADD      #1,R1    ; 分岐先
```

## 10.49 JSR Jump to SubRoutine 分岐命令

サブルーチンプロ  
シージャへの分岐

遅延分岐命令

書式	動作概略	命令コード	実行ステート	Tビット
JSR @Rn	PC+4 PR, Rn PC	0100nnnn00001011	2	

## (1) 説明

本命令の直後の命令の実行後指定したアドレスのサブルーチンプロシージャへ遅延分岐します。戻り先アドレス (PC+4) を PR に退避し、汎用レジスタ Rn で表されるアドレスへ分岐します。RTS と組み合わせて、サブルーチンプロシージャコールに使用します。

## (2) 注意

遅延分岐命令ですので、本命令の直後の命令を先に実行してから、分岐先の命令を実行します。本命令と直後の命令との間には、割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

## (3) 動作内容

```
JSR(int n) /* JSR @Rn */
{
    unsigned int temp;

    temp=PC;
    PR=PC+4;
    PC=R[n];
    Delay_Slot(temp+2);
}
```

## (4) 使用例

```
MOV.L    JSR_TABLE,R0    ;R0=TRGET のアドレス
JSR      @R0              ;TRGET へ分岐します。
XOR      R1,R1            ;分岐に先立ち XOR を実行します。
ADD      R0,R1            ; プログラムからの戻り先
.....                (PR の内容)です。
.align   4
JSR_TABLE: .data.l    TRGET    ;ジャンプテーブル
TRGET:    NOP          ; プログラムの入り口
MOV      R2,R3          ;
RTS      ;上記 ADD 命令に戻ります。
MOV      #70,R1         ;RTS に先立ち MOV を実行します。
```

## 10.50 LDC      LoaD to Control register      システム制御命令

コントロール  
レジスタへのロード

書式	動作概略	命令コード	実行 ステート	Tビット
LDC    Rm,DBR	Rm   DBR	0100mmmm11111010	1	
LDC    @Rm+, DBR	Rm   DBR, Rm+4   Rm	0100mmmm11110110	1	

## (1) 説明

ソースオペランドをコントロールレジスタ DBR に格納します。

## (2) 注意

本命令は特権命令です。ユーザモードで使用すると、不当命令例外が発生します。

## (3) 動作内容

```

LDCDBR(int m)      /* LDC Rm,DBR : Privileged */
{
    DBR=R[m];
    PC+=2;
}

LDCMDBR(int m)      /* LDC.L @Rm+,DBR : Privileged */
{
    DBR=Read_Long(R[m]);
    R[m]+=4;
    PC+=2;
}

```

発生する可能性がある例外

General illegal instruction exception

Illegal slot instruction exception

Data TLB miss exception

Data TLB protection violation exception

Address error

## 10.51 LDS Load to FPU System register システム制御命令

FPU システム  
レジスタへのロード

書式	動作概略	命令コード	実行 ステート	Tビット
LDS Rm,FPUL	Rm FPUL	0100mmmm01011010	1	
LDS.L @Rm+,FPUL	(Rm) FPUL, Rm + 4 Rm	0100mmmm01010110	1	
LDS Rm,FPSCR	Rm FPSCR	0100mmmm01101010	1	
LDS.L @Rm+,FPSCR	(Rm) FPSCR, Rm + 4 Rm	0100mmmm01100110	1	

## (1) 説明

ソースオペランドを FPU システムレジスタ FPUL、FPSCR に格納します。

## (2) 動作内容

```
#define FPSCR_MASK 0x003FFFFFF
```

```
LDSFPUL(int m)      /* LDS Rm,FPUL */
{
    FPUL=R[m];
    PC+=2;
}

LDSMFPUL(int m)     /* LDS.L @Rm+,FPUL */
{
    FPUL=Read_Long(R[m]);
    R[m]+=4;
    PC+=2;
}

LDSFPSCR(int m)     /* LDS Rm,FPSCR */
{
    FPSCR=R[m] & FPSCR_MASK;
    PC+=2;
}

LDSMFPSCR(int m)    /* LDS.L @Rm+,FPSCR */
{
    FPSCR=Read_Long(R[m]) & FPSCR_MASK;
    R[m]+=4;
    PC+=2;
}
```

## 10. 各命令の説明

---

発生する可能性がある例外

Data TLB miss exception

Data Access Protection exception

Address Error

## 10.52 LDTLB    LoaD PTEH/PTEL/PTEA to TLB    システム制御命令

TLB へのロード (特権命令)

書式	動作概略	命令コード	実行ステート	T ビット
LDTLB	PTEH/PTEL/PTEA TLB	0000000000111000	1	

### (1) 説明

PTEH/PTEL/PTEA レジスタ内容を MMUCR.URC (MMC 制御レジスタのランダムカウンタフィールド) で指定した TLB (Translation Lookaside Buffer) に格納します。

LDTLB 命令は特権命令であり、特権モードでだけ使われます。もしユーザモードで使われた場合は不当命令例外が発生します。

### (2) 注意

本命令は PTEH/PTEL/PTEA レジスタを TLB にロードする命令なので、MMU がディスエーブル状態か、MMU がイネーブル状態で論理空間の P1 または P2 空間で本命令を使用するようにしてください (詳しくは「第 3 章 メモリマネジメントユニット」を参照してください)。本命令の発行後、LDTLB 命令と領域 P0、U0、P3 へのアクセスに関わる命令、すなわち BRAF、BSRF、JMP、JSR、RTS、RTE の発行の間には少なくとも 1 つの命令がなければなりません。

### (3) 動作内容

```

LDTLB( )    /*LDTLB */
{
    TLB[MMUCR.URC] .ASID=PTEH & 0x000000FF;
    TLB[MMUCR.URC] .VPN=(PTEH & 0xFFFFFC00)>>10;
    TLB[MMUCR.URC] .PPN=(PTEH & 0x1FFFFC00)>>10;
    TLB[MMUCR.URC] .SZ=(PTEL & 0x00000080)>>6 | (PTEL & 0x00000010)>>4;
    TLB[MMUCR.URC] .SH=(PTEH & 0x00000002)>>1;
    TLB[MMUCR.URC] .PR=(PTEH & 0x00000060)>>5;
    TLB[MMUCR.URC] .WT=(PTEH & 0x00000001);
    TLB[MMUCR.URC] .C=(PTEH & 0x00000008)>>3;
    TLB[MMUCR.URC] .D=(PTEH & 0x00000004)>>2;
    TLB[MMUCR.URC] .V=(PTEH & 0x00000100)>>8;
    TLB[MMUCR.URC] .SA=(PTEA & 0x00000007);
    TLB[MMUCR.URC] .TC=(PTEA & 0x00000008)>>3;

    PC+=2;
}

```

### (4) 使用例

```

MOV @R0,R1                    ;PTEH レジスタの上位を R1 にロード
MOV R1,@R2                    ;R1 を PTEH にロード, R2 は PTEH のアドレス(0xFFFFFFF0)
LDTLB                         ;PTEH,PTEL,PTEA レジスタを TLB にロード

```



## 10.53 MAC.L Multiply and ACcumulate Long 算術演算命令

倍精度積和演算

書式	動作概略	命令コード	実行 ステート	Tビット
MAC.L @Rm+,@Rn+	符号付きで (Rn) × (Rm)+MAC MAC Rn+4 Rn, Rm+4 Rm	0000nnnnnnmmmm1111	2~5	

### (1) 説明

汎用レジスタ Rm と Rn の内容をアドレスとする 32 ビットオペランドを符号付きで乗算し、結果の 64 ビットと MAC レジスタの内容とを加算し、結果を MAC レジスタに格納します。各オペランドを読み出す毎にそれぞれ、Rm を+4、Rn を+4 します。

S ビットが 0 のときは、連結した MACH、MACL レジスタに結果の 64 ビットを格納します。

S ビットが 1 のときは、MAC レジスタとの加算は LSB から 48 番目のビットで飽和演算になります。飽和演算では、MAC レジスタの下位 48 ビットのみが有効となり結果の範囲を H'FFFF800000000000(最小値)から H'00007FFFFFFFFFFFFF(最大値)までに制限します。

### (2) 動作内容

```
MACL(long m, long n) /* MAC.L @Rm+,@Rn+ */
{
    unsigned long RnL,RnH,RmL,RmH,Res0,Res1,Res2;
    unsigned long temp0,temp1,temp2,temp3;
    long tempm,tempn,fnLmL;

    tempn=(long)Read_Long(R[n]);
    R[n]+=4;
    tempm=(long)Read_Long(R[m]);
    R[m]+=4;

    if ((long)(tempn^tempm)<0) fnLmL=-1;
    else fnLmL=0;
    if (tempn<0) tempn=0-tempn;
    if (tempm<0) tempm=0-tempm;

    temp1=(unsigned long)tempn;
    temp2=(unsigned long)tempm;
```

```

    RnL=temp1&0x0000FFFF;
    RnH=(temp1>>16)&0x0000FFFF;
    RmL=temp2&0x0000FFFF;
    RmH=(temp2>>16)&0x0000FFFF;
    temp0=RmL*RnL;
    temp1=RmH*RnL;
    temp2=RmL*RnH;
    temp3=RmH*RnH;

    Res2=0;

    Res1=temp1+temp2;
    if (Res1<temp1) Res2+=0x00010000;

    temp1=(Res1<<16)&0xFFFF0000;
    Res0=temp0+temp1;
    if (Res0<temp0) Res2++;

    Res2=Res2+((Res1>>16)&0x0000FFFF)+temp3;

    if(fnLmL<0){
        Res2=`Res2;
        if (Res0==0) Res2++;
        else Res0=(`Res0)+1;
    }
    if(S==1){
        Res0=MACL+Res0;
        if (MACL>Res0) Res2++;
        if (MACH&0x00008000);
        else Res2+=MACH|0xFFFF0000;
            Res2+=MACH&0x00007FFF;

        if(((long)Res2<0)&&(Res2<0xFFFF8000)){
            Res2=0xFFFF8000;
            Res0=0x00000000;
        }
        if(((long)Res2>0)&&(Res2>0x00007FFF)){
            Res2=0x00007FFF;
            Res0=0xFFFFFFFF;
        };

        MACH=(Res2&0x0000FFFF)|(MACH&0xFFFF0000);
        MACL=Res0;
    }

```

```
    else {  
        Res0=MACL+Res0;  
        if (MACL>Res0) Res2++;  
        Res2+=MACH;  
  
        MACH=Res2;  
        MACL=Res0;  
    }  
    PC+=2;  
}  
(3)  使用例
```

	MOVA	TBLM,R0	;テーブルのアドレスを得る
	MOV	R0,R1	;
	MOVA	TBLN,R0	;テーブルのアドレスを得る
	CLRMAC		;MACレジスタの初期化
	MAC.L	@R0+,@R1+	;
	MAC.L	@R0+,@R1+	;
	STS	MACL,R0	;結果を R0 に得る
	.....		
	.align	2	;
TBLM	.data.l	H'1234ABCD	;
	.data.l	H'5678EF01	;
TBLN	.data.l	H'0123ABCD	;
	.data.l	H'4567DEF0	;

## 10.54 MAC.W Multiply and ACcumulate Word 算術演算命令

### 積和演算

書式	動作概略	命令コード	実行 ステート	Tビット
MAC.W @Rm+, @Rn+ MAC @Rm+, @Rn+	符号付きで (Rn) × (Rm) + MAC MAC Rn+2 Rn, Rm+2 Rm	0100nnnnnnmmmm1111	2~5	

#### (1) 説明

汎用レジスタ Rm と Rn の内容をアドレスとする 16 ビットオペランドを符号付きで乗算し、結果の 32 ビットと MAC レジスタの内容とを加算し、結果を MAC レジスタに格納します。各オペランドを読み出す毎にそれぞれ、Rm を+2、Rn を+2 します。

S ビットが 0 のとき、 $16 \times 16 + 64$  64 ビットの積和演算となり、連結した MACH、MACL レジスタに結果の 64 ビットを格納します。

S ビットが 1 のとき、 $16 \times 16 + 32$  32 ビットの積和演算となり、MAC レジスタとの加算は飽和演算になります。飽和演算では、MACL レジスタのみが有効となり結果の範囲を H'80000000(最小値)から H'7FFFFFFF(最大値)までに制限します。オーバーフローが発生すると、MACH レジスタの LSB を 1 にセットします。結果が負の方向にオーバーフローしたときは H'80000000(最小値)を、正の方向にオーバーフローしたときは H'7FFFFFFF(最大値)を、MACL レジスタに格納します。

#### (2) 注意

S ビットが 0 のとき、 $16 \times 16 + 64$  64 ビットの積和演算を行います。

#### (3) 動作内容

```
MACW(long m, long n) /* MAC.W @Rm+,@Rn+ */
{
    long tempm,tempn,dest,src,ans;
    unsigned long templ;
    tempn=(long)Read_Word(R[n]);
    R[n]+=2;
    tempm=(long)Read_Word(R[m]);
    R[m]+=2;
    templ=MACL;
    tempm=((long)(short)tempn*((long)(short)tempm);
    if ((long)MACL>=0) dest=0;
    else dest=1;
    if ((long)tempm>=0) {
        src=0;
        tempn=0;
    }
    else {
        src=1;
        tempn=0xFFFFFFFF;
    }
}
```

```
    }
    src+=dest;
    MACL+=tempm;
    if ((long)MACL>=0) ans=0;
    else ans=1;
    ans+=dest;
    if (S==1) {
        if (ans==1) {
            if (src==0) MACL=0x7FFFFFFF;
            if (src==2) MACL=0x80000000;
        }
    }
    else {
        MACH+=tempn;
        if (templ>MACL) MACH+=1;
    }
    PC+=2;
}
(4)  使用例

        MOVA        TBLM,R0          ;テーブルのアドレスを得る
        MOV         R0,R1            ;
        MOVA        TBLN,R0          ;テーブルのアドレスを得る
        CLRMAC      ;MACレジスタの初期化
        MAC.W       @R0+,@R1+        ;
        MAC.W       @R0+,@R1+        ;
        STS         MACL,R0          ;結果を R0 に得る
        .....
        .align      2                ;
TBLM     .data.w    H'1234            ;
        .data.w     H'5678            ;
TBLN     .data.w    H'0123            ;
        .data.w     H'4567            ;
```

## 10.55 MOV MOVE data

### データ転送

## データ転送命令

書式		動作概略	命令コード	実行 ステート	Tビット
MOV	Rm,Rn	Rm Rn	0110nnnnnnmm0011	1	
MOV.B	Rm,@Rn	Rm (Rn)	0010nnnnnnmm0000	1	
MOV.W	Rm,@Rn	Rm (Rn)	0010nnnnnnmm0001	1	
MOV.L	Rm,@Rn	Rm (Rn)	0010nnnnnnmm0010	1	
MOV.B	@Rm,Rn	(Rm) 符号拡張 Rn	0110nnnnnnmm0000	1	
MOV.W	@Rm,Rn	(Rm) 符号拡張 Rn	0110nnnnnnmm0001	1	
MOV.L	@Rm,Rn	(Rm) Rn	0110nnnnnnmm0010	1	
MOV.B	Rm,@-Rn	Rn-1 Rn, Rm (Rn)	0010nnnnnnmm0100	1	
MOV.W	Rm,@-Rn	Rn-2 Rn, Rm (Rn)	0010nnnnnnmm0101	1	
MOV.L	Rm,@-Rn	Rn-4 Rn, Rm (Rn)	0010nnnnnnmm0110	1	
MOV.B	@Rm+,Rn	(Rm) 符号拡張 Rn, Rm+1 Rm	0110nnnnnnmm0100	1	
MOV.W	@Rm+,Rn	(Rm) 符号拡張 Rn, Rm+2 Rm	0110nnnnnnmm0101	1	
MOV.L	@Rm+,Rn	(Rm) Rn, Rm+4 Rm	0110nnnnnnmm0110	1	
MOV.B	Rm,@(R0,Rn)	Rm (R0+Rn)	0000nnnnnnmm0100	1	
MOV.W	Rm,@(R0,Rn)	Rm (R0+Rn)	0000nnnnnnmm0101	1	
MOV.L	Rm,@(R0,Rn)	Rm (R0+Rn)	0000nnnnnnmm0110	1	
MOV.B	@(R0,Rm),Rn	(R0+Rm) 符号拡張 Rn	0000nnnnnnmm1100	1	
MOV.W	@(R0,Rm),Rn	(R0+Rm) 符号拡張 Rn	0000nnnnnnmm1101	1	
MOV.L	@(R0,Rm),Rn	(R0+Rm) Rn	0000nnnnnnmm1110	1	

### (1) 説明

ソースオペランドをデスティネーションへ転送します。オペランドがメモリのときは転送するデータサイズをバイト/ワード/ロングワードの範囲で指定できます。ソースオペランドがメモリのときは、ロードされたデータをロングワードに符号拡張後レジスタに格納します。

### (2) 動作内容

```
MOV(long m, long n) /* MOV Rm,Rn */
{
    R[n]=R[m];
    PC+=2;
}

MOVBS(long m, long n) /* MOV.B Rm,@Rn */
{
    Write_Byte(R[n],R[m]);
    PC+=2;
}

MOVWS(long m, long n) /* MOV.W Rm,@Rn */
```

```
{
    Write_Word(R[n],R[m]);
    PC+=2;
}

MOVLS(long m, long n) /* MOV.L Rm,@Rn */
{
    Write_Long(R[n],R[m]);
    PC+=2;
}

MOVBL(long m, long n) /* MOV.B @Rm,Rn */
{
    R[n]=(long)Read_Byte(R[m]);
    if ((R[n]&0x80)==0) R[n]&=0x000000FF;
    else R[n]|=0xFFFFFF00;
    PC+=2;
}

MOVWL(long m, long n) /* MOV.W @Rm,Rn */
{
    R[n]=(long)Read_Word(R[m]);
    if ((R[n]&0x8000)==0) R[n]&=0x0000FFFF;
    else R[n]|=0xFFFF0000;
    PC+=2;
}

MOVLl(long m, long n) /* MOV.L @Rm,Rn */
{
    R[n]=Read_Long(R[m]);
    PC+=2;
}

MOVBM(long m, long n) /* MOV.B Rm,@-Rn */
{
    Write_Byte(R[n]-1,R[m]);
    R[n]-=1;
    PC+=2;
}

MOVWM(long m, long n) /* MOV.W Rm,@-Rn */
{
    Write_Word(R[n]-2,R[m]);
    R[n]-=2;
    PC+=2;
}
```

```

MOVLm(long m, long n) /* MOV.L Rm,@-Rn */
{
    Write_Long(R[n]-4,R[m]);
    R[n]-=4;
    PC+=2;
}

MOVBP(long m, long n) /* MOV.B @Rm+,Rn */
{
    R[n]=(long)Read_Byte(R[m]);
    if ((R[n]&0x80)==0) R[n]&=0x000000FF;
    else R[n]|=0xFFFFF00;
    if (n!=m) R[m]+=1;
    PC+=2;
}

MOVWP(long m, long n) /* MOV.W @Rm+,Rn */
{
    R[n]=(long)Read_Word(R[m]);
    if ((R[n]&0x8000)==0) R[n]&=0x0000FFFF;
    else R[n]|=0xFFFF0000;
    if (n!=m) R[m]+=2;
    PC+=2;
}

MOVLp(long m, long n) /* MOV.L @Rm+,Rn */
{
    R[n]=Read_Long(R[m]);
    if (n!=m) R[m]+=4;
    PC+=2;
}

MOVBS0(long m, long n) /* MOV.B Rm,@(R0,Rn) */
{
    Write_Byte(R[n]+R[0],R[m]);
    PC+=2;
}

MOVWS0(long m, long n) /* MOV.W Rm,@(R0,Rn) */
{
    Write_Word(R[n]+R[0],R[m]);
    PC+=2;
}

```



```
MOVLS0(long m, long n) /* MOV.L Rm,@(R0,Rn) */
```

```
{
    Write_Long(R[n]+R[0],R[m]);
    PC+=2;
}
```

```
MOVBL0(long m, long n) /* MOV.B @(R0,Rm),Rn */
```

```
{
    R[n]=(long)Read_Byte(R[m]+R[0]);
    if ((R[n]&0x80)==0) R[n]&=0x000000FF;
    else R[n]|=0xFFFFF00;
    PC+=2;
}
```

```
MOVWL0(long m, long n) /* MOV.W @(R0,Rm),Rn */
```

```
{
    R[n]=(long)Read_Word(R[m]+R[0]);
    if ((R[n]&0x8000)==0) R[n]&=0x0000FFFF;
    else R[n]|=0xFFFF0000;
    PC+=2;
}
```

```
MOVLL0(long m, long n) /* MOV.L @(R0,Rm),Rn */
```

```
{
    R[n]=Read_Long(R[m]+R[0]);
    PC+=2;
}
```

(3) 使用例

MOV	R0,R1	;実行前 R0=H'FFFFFFFF,R1=H'00000000 ;実行後 R1=H'FFFFFFFF
MOV.W	R0,@R1	;実行前 R0=H'FFFF7F80 ;実行後 (R1)=H'7F80
MOV.B	@R0,R1	;実行前 (R0)=H'80,R1=H'00000000 ;実行後 R1=H'FFFFFF80
MOV.W	R0,@-R1	;実行前 R0=H'AAAAAAA,(R1)=H'FFFF7F80 ;実行後 R1=H'FFFF7F7E,(R1)=H'AAAA
MOV.L	@R0+,R1	;実行前 R0=H'12345670 ;実行後 R0=H'12345674,R1=(H'12345670)
MOV.B	R1,@(R0,R2)	;実行前 R2=H'00000004,R0=H'10000000 ;実行後 R1=(H'10000004)
MOV.W	@(R0,R2),R1	;実行前 R2=H'00000004,R0=H'10000000 ;実行後 R1=(H'10000004)

## 10.56 MOV MOVE Constant Value

## データ転送命令

イミディエイト  
データの転送

書式	動作概略	命令コード	実行ステート	Tビット
MOV #imm,Rn	imm 符号拡張 Rn	1110nnnniiiiiii	1	
MOV.W @(disp,PC),Rn	(disp×2+PC+4) 符号拡張 Rn	1001nnnnddddddd	1	
MOV.L @(disp,PC),Rn	(disp×4+PC+4) Rn	1101nnnnddddddd	1	

## (1) 説明

ロングワードに符号拡張したイミディエイトデータを汎用レジスタ Rn に格納します。データがワードまたはロングワードのときは、データはアドレス (PC+4+ディスプレースメント×2) または (PC+4+ディスプレースメント×4) のメモリ位置から格納されます。

データがワードのとき、8 ビットディスプレースメントはゼロ拡張後 2 倍しますので、テーブルとの相対距離は PC+4+510 バイトまでの範囲になります。PC は本命令の命令アドレスです。

データがロングワードのとき、8 ビットディスプレースメントはゼロ拡張後 4 倍しますので、オペランドとの相対距離は PC+4+1020 バイトまでの範囲になります。PC は本命令の命令アドレスですが、下位 2 ビットを B'00 に補正した値をアドレス計算に使用します。

## (2) 注意

PC 相対ロード命令を遅延スロットで実行すると不当スロット命令例外が発生します。

## (3) 動作内容

```

MOVI(int i, int n) /* MOV #imm,Rn */
{
    if ((i&0x80)==0) R[n]=(0x000000FF & i);
    else R[n]=(0xFFFFF000 | i);
    PC+=2;
}

MOVWI(d, n) /* MOV.W @(disp,PC),Rn */
{
    unsigned int disp;

    disp=(unsigned int)(0x000000FF & d);
    R[n]=(int)Read_Word(PC+4+(disp<<1));
    if ((R[n]&0x8000)==0) R[n]&=0x0000FFFF;
    else R[n]|=0xFFFF0000;
    PC+=2;
}

MOVLI(int d, int n)/* MOV.L @(disp,PC),Rn */
{
    unsigned int disp;

```

## 10. 各命令の説明

---

```
disp=(unsigned int)(0x000000FF & (int)d);
R[n]=Read_Long( (PC&0xFFFFFFFF)+4+(disp<<2));
PC+=2;
```

```
}
```

### (4) 使用例

アドレス

1000	MOV	#H'80,R1	;R1=H'FFFFFF80
1002	MOV.W	IMM,R2	;R2=H'FFFF9ABC IMM は(PC+4+H'08) の意味
1004	ADD	#-1,R0	;
1006	TST	R0,R0	;
1008	MOV.L	@(3,PC),R3	;R3=H'12345678
100A	BRA	NEXT	;遅延分岐命令
100C	NOP		
100E IMM	.data.w	H'9ABC	;
1010	.data.w	H'1234	;
1012 NEXT	JMP	@R3	;BRA の分岐先
1014	CMP/EQ	#0,R0	;
	.align	4	;
1018	.data.l	H'12345678	;
101C	.data.l	H'9ABCDEF0	;

## 10.57 MOV MOVE Global data

## データ転送命令

グローバル  
データの転送

書式	動作概略	命令コード	実行ステート	Tビット
MOV.B @(disp,GBR),R0	(disp+GBR) 符号拡張 R0	11000100dddddddd	1	
MOV.W @(disp,GBR), R0	(disp×2+GBR) 符号拡張 R0	11000101dddddddd	1	
MOV.L @(disp,GBR),R0	(disp×4+GBR) R0	11000110dddddddd	1	
MOV.B R0,@(disp,GBR)	R0 (disp+GBR)	11000000dddddddd	1	
MOV.W R0,@(disp,GBR)	R0 (disp×2+GBR)	11000001dddddddd	1	
MOV.L R0,@(disp,GBR)	R0 (disp×4+GBR)	11000010dddddddd	1	

## (1) 説明

ソースオペランドをデスティネーションへ転送します。データサイズをバイト、ワード、またはロングワードの範囲で指定できますが、レジスタが R0 固定になります。転送データがバイトサイズるとき 8 ビットディスプレースメントはゼロ拡張するだけですので、+255 バイトまでの範囲が指定できます。ワードサイズるとき 8 ビットディスプレースメントはゼロ拡張後 2 倍しますので、+510 バイトまでの範囲が指定できます。ロングワードサイズるとき 8 ビットディスプレースメントはゼロ拡張後 4 倍しますので、+1020 バイトまでの範囲が指定できます。

ソースオペランドがメモリのときは、ロードされたデータをロングワードに符号拡張後レジスタへ格納します。

## (2) 注意

ロードするときデスティネーションレジスタが R0 固定です。

## (3) 動作内容

```

MOVBLG(int d /* MOV.B @(disp,GBR),R0 */
{
    unsigned int disp;

    disp=(unsigned int)(0x000000FF & d);
    R[0]=(int)Read_Byte(GBR+disp);
    if ((R[0]&0x80)==0) R[0]&=0x000000FF;
    else R[0]|=0xFFFFF00;
    PC+=2;
}

MOVWLG(int d) /* MOV.W @(disp,GBR),R0 */
{
    unsigned int disp;

    disp=(unsigned int)(0x000000FF & d);

    R[0]=(int)Read_Word(GBR+(disp<<1));
    if ((R[0]&0x8000)==0) R[0]&=0x0000FFFF;
}

```

## 10. 各命令の説明

---

```
        else R[0] |= 0xFFFF0000;
        PC+=2;
    }

    MOVLLG(int d) /* MOV.L @(disp,GBR),R0 */
    {
        unsigned int disp;

        disp=(unsigned int)(0x000000FF & d);
        R[0]=Read_Long(GBR+(disp<<2));
        PC+=2;
    }

    MOVBSG(int d) /* MOV.B R0,@(disp,GBR) */
    {
        unsigned int disp;

        disp=(unsigned int)(0x000000FF & d);
        Write_Byte(GBR+disp,R[0]);
        PC+=2;
    }

    MOVWSG(int d) /* MOV.W R0,@(disp,GBR) */
    {
        unsigned int disp;

        disp=(unsigned int)(0x000000FF & d);
        Write_Word(GBR+(disp<<1),R[0]);
        PC+=2;
    }

    MOVLSG(int d) /* MOV.L R0,@(disp,GBR) */
    {
        unsigned int disp;

        disp=(unsigned int)(0x000000FF & (long)d);
        Write_Long(GBR+(disp<<2),R[0]);
        PC+=2;
    }
}

(4) 使用例
MOV.L  @(2,GBR),R0      ;実行前 @(GBR+8)=H'12345670
                        ;実行後 R0=@H'12345670
MOV.B   R0,@(1,GBR)     ;実行前 R0=H'FFFF7F80
                        ;実行後 @(GBR+1)=H'FFFF7F80
```

## 10.58 MOV MOVE structure data

### 構造体データの転送

## データ転送命令

書式	動作概略	命令コード	実行 ステート	Tビット
MOV.B R0,@(disp,Rn)	R0 (disp+Rn)	10000000nnnnndddd	1	
MOV.W R0,@(disp,Rn)	R0 (disp × 2+Rn)	10000001nnnnndddd	1	
MOV.L Rm,@(disp,Rn)	Rm (disp × 4+Rn)	0001nnnnnnmmmmndddd	1	
MOV.B @(disp,Rm),R0	(disp+Rm) 符号拡張 R0	10000100mmmmndddd	1	
MOV.W @(disp,Rm),R0	(disp × 2+Rm) 符号拡張 R0	10000101mmmmndddd	1	
MOV.L @(disp,Rm),Rn	(disp × 4+Rm) Rn	0101nnnnnnmmmmndddd	1	

### (1) 説明

ソースオペランドをデスティネーションへ転送します。構造体、スタック内のデータアクセスに最適です。データサイズをバイト、ワード、またはロングワードの範囲で指定できますが、バイトまたはワードのときはレジスタが R0 固定になります。

データがバイトサイズるとき 4 ビットディスプレースメントはゼロ拡張するだけですので、+15 バイトまでの範囲が指定できます。ワードサイズるとき 4 ビットディスプレースメントはゼロ拡張後 2 倍しますので、+30 バイトまでの範囲が指定できます。ロングワードサイズるとき 4 ビットディスプレースメントはゼロ拡張後 4 倍しますので、+60 バイトまでの範囲が指定できます。メモリオペランドに届かないときは前述の @(R0,Rn) モードを使う必要があります。

ソースオペランドがメモリのときは、ロードされたデータをロングワードに符号拡張後レジスタへ格納します。

### (2) 注意

バイト/ワードデータをロードするときデスティネーションレジスタが R0 固定です。したがって、直後の命令で R0 を参照しようとしてもロード命令の実行完了まで待たされます。これは命令の順序を替えることによって最適化が可能です。

MOV.B	@(2,R1),R0		MOV.B	@(2,R1),R0
AND	#80,R0	→	ADD	#20,R1
ADD	#20,R1	→	AND	#80,R0

### (3) 動作内容

```
MOVBS4(long d, long n /* MOV.B R0,@(disp,Rn) */
{
    long disp;
    disp=(0x0000000F & (long)d);
    Write_Byte(R[n]+disp,R[0]);
    PC+=2;
}
```

```
MOVWS4(long d, long n) /* MOV.W R0,@(disp,Rn) */
{
    long disp;

    disp=(0x0000000F & (long)d);
    Write_Word(R[n]+(disp<<1),R[0]);
    PC+=2;
}

MOVLS4(long m, long d, long n) /* MOV.L Rm,@(disp,Rn) */
{
    long disp;

    disp=(0x0000000F & (long)d);
    Write_Long(R[n]+(disp<<2),R[m]);
    PC+=2;
}

MOVBL4(long m, long d) /* MOV.B @(disp,Rm),R0 */
{
    long disp;

    disp=(0x0000000F & (long)d);
    R[0]=Read_Byte(R[m]+disp);
    if ((R[0]&0x80)==0) R[0]&=0x000000FF;
    else R[0]|=0xFFFFF00;
    PC+=2;
}

MOVWL4(long m, long d) /* MOV.W @(disp,Rm),R0 */
{
    long disp;

    disp=(0x0000000F & (long)d);
    R[0]=Read_Word(R[m]+(disp<<1));
    if ((R[0]&0x8000)==0) R[0]&=0x0000FFFF;
    else R[0]|=0xFFFF0000;
    PC+=2;
}
```

```
MOVL4(long m, long d, long n) /* MOV.L @(disp,Rm),Rn */
{
    long disp;

    disp=(0x0000000F & (long)d);
    R[n]=Read_Long(R[m]+(disp<<2));
    PC+=2;
}
(4) 使用例

MOV.L      @(2,R0),R1          ;実行前 @(R0+8)=H'12345670
                                ;実行後 R1=@H'12345670
MOV.L      R0,@(H'F,R1)       ;実行前 R0=H'FFFF7F80
                                ;実行後@(R1+60)=H'FFFF7F80
```



## 10.59 MOVA MOVE effective Address

## データ転送命令

実効アドレスの転送

書式	動作概略	命令コード	実行ステート	Tビット
MOVA @(disp,PC),R0	disp×4+PC+4 R0	11000111ddddddd	1	

## (1) 説明

汎用レジスタ R0 にソースオペランドの実効アドレスを格納します。8 ビットディスプレースメントはゼロ拡張後 4 倍します。PC は本命令の命令アドレスですが、下位 2 ビットを B'00 に補正した値をアドレス計算に使用します。

## (2) 注意

本命令を遅延スロットで実行すると、不当スロット命令が発生します。

## (3) 動作内容

```
MOVA(int d)                /* MOVA @(disp,PC),R0 */
{
    unsigned int disp;

    disp=(unsigned int)(0x000000FF & d);
    R[0]=(PC&0xFFFFF0)+4+(disp<<2);
    PC+=2;
}
```

## (4) 使用例

```
アドレス .org H'1006
1006 MOVA STR,R0          ;STR のアドレス R0
1008 MOV.B @R0,R1         ;R1="X" PC 下位 2 ビット補正後の位置
100A ADD R4,R5            ; MOVA 命令のアドレス計算時、PC の本来の位置
      .align 4
100C STR:..sdata "XYZP12"
```

## 10.60 MOVCA.L Move with Cache Block Allocation データ転送命令

### キャッシュブロックの確保

書式	動作概略	命令コード	実行 ステート	Tビット
MOVCA.L R0,@Rn	R0 (Rn)	0000nnnn11000011	1	

#### (1) 説明

汎用レジスタ R0 を、実効アドレス Rn で示されているメモリに格納します。この命令は他の格納命令とは以下の点で異なります。

アクセスされたメモリがライトバック方式を選択していた場合で、かつキャッシュミスが起きた場合、キャッシュブロックは確保されますが、ブロックリードは行わず、R0 のデータの書き込みを、そのキャッシュブロックに行います。他のキャッシュブロックの内容は未定義です。

#### (2) 動作内容

```
MOVCA.L(int n)      /*MOVCA.L  R0,@Rn */
{
    if ((is_write_back_memory(R[n]))
        && (look_up_in_operand_cache(R[n]) == MISS))
        allocate_operand_cache_block(R[n]);
    Write_Long(R[n], R[0]);
    PC+=2;
}
```

発生する可能性がある例外

Data TLB miss exception

Data TLB protection violation exception

Initial page write exception

Address error

10.61    **MOVT      MOVE Tbit**  
          T ビットの転送**データ転送命令**

---

書式	動作概略	命令コード	実行 ステート	T ビット
MOVT Rn	T Rn	0000nnnn00101001	1	

## (1)    説明

T ビットを汎用レジスタ Rn に格納します。T=1 のとき Rn=1、T=0 のとき Rn=0 になります。

## (2)    動作内容

```
MOVT(long n)                    /* MOVT Rn */
{
    R[n]=(0x00000001 & SR);
    PC+=2;
}
```

## (3)    使用例

```
XOR                    R2,R2    ;R2=0
CMP/PZ                R2       ;T=1
MOVT                  R0       ;R0=1
CLRT                            ;T=0
MOVT                  R1       ;R1=0
```

## 10.62 MUL.L MULtiplY Long

倍精度乗算

## 算術演算命令

書式	動作概略	命令コード	実行 ステート	Tビット
MUL.L Rm,Rn	$R_n \times R_m$ MACL	0000nnnnnnmmmm0111	2~5	

### (1) 説明

汎用レジスタ  $R_n$  の内容と  $R_m$  を 32 ビットで乗算し、結果の下位側 32 ビットを MACL レジスタに格納します。MACL の内容は変化しません。

### (2) 動作内容

```
MULL(long m, long n) /* MUL.L Rm,Rn */
{
    MACL=R[n]*R[m];
    PC+=2;
}
```

### (3) 使用例

```
MUL.L    R0,R1          ;実行前 R0=H'FFFFFFFE,R1=H'00005555
                        ;実行後  MACL=H'FFFF5556
STS      MACL,R0        ;演算結果を得る
```

## 10.63 MULS.W MULTiply as Signed Word

## 算術演算命令

符号付き乗算

書式	動作概略	命令コード	実行 ステート	Tビット
MULS.W Rm,Rn MULS Rm,Rn	符号付きで $R_n \times R_m$ MACL	0010nnnnnnmmmm1111	2~5	

## (1) 説明

汎用レジスタ  $R_n$  の内容と  $R_m$  を 16 ビットで乗算し、結果の 32 ビットを MACL レジスタに格納します。演算は符号付き算術演算で行います。MACL の内容は変化しません。

## (2) 動作内容

```

MULS(long m, long n) /* MULS Rm,Rn */
{
    MACL=((long)(short)R[n]*(long)(short)R[m]);
    PC+=2;
}

```

## (3) 使用例

```

MULS.W      R0,R 1      ;実行前 R0=H'FFFFFFFE,R1=H'00005555
              ;実行後 MACL=H'FFFF5556
STS          MACL,R0     ;演算結果を得る

```

## 10.64 MULU.W MULTiply as Unsigned Word

## 算術演算命令

符号なし乗算

書式	動作概略	命令コード	実行 ステート	Tビット
MULU.W Rm,Rn MULU Rm,Rn	符号なしで $R_n \times R_m$ MACL	0010nnnnnnmmmm1110	2~5	

## (1) 説明

汎用レジスタ  $R_n$  の内容と  $R_m$  を 16 ビットで乗算し、結果の 32 ビットを MACL レジスタに格納します。演算は符号なし算術演算で行います。MACL の内容は変化しません。

## (2) 動作内容

```
MULU(long m, long n) /* MULU Rm,Rn */
{
    MACL=((unsigned long)(unsigned short)R[n]*
    (unsigned long)(unsigned short)R[m];
    PC+=2;
}
```

## (3) 使用例

```
MULU.W      R0,R1      ;実行前 R0=H'00000002,R1=H'FFFFAAAA
               ;実行後 MACL=H'00015554
STS          MACL,R0    ;演算結果を得る
```

## 10.65    NEG            NEGate

符号反転

## 算術演算命令

---

書式	動作概略	命令コード	実行 ステート	Tビット
NEG Rm,Rn	0-Rm   Rn	0110nnnnnnmm1011	1	

### (1)    説明

汎用レジスタ Rm の内容の 2 の補数を取り、結果を Rn に格納します。即ち 0 から Rm を減算し、結果を Rn に格納します。

### (2)    動作内容

```
NEG(long m, long n) /* NEG Rm,Rn */
{
    R[n]=0-R[m];
    PC+=2;
}
```

### (3)    使用例

```
NEG R0,R1                            ;実行前 R0=H'00000001
                                     ;実行後 R1=H'FFFFFFFF
```

## 10.66 NEGC NEGate with Carry

## 算術演算命令

ボロー付き符号反転

書式	動作概略	命令コード	実行 ステート	Tビット
NEGC Rm,Rn	0-Rm-T Rn, ボロー T	0110nnnnnnmmmm1010	1	ボロー

## (1) 説明

0 から汎用レジスタ Rm の内容と T ビットを減算し、結果を Rn に格納します。演算の結果によってボローを T ビットに反映します。32 ビットを超える値の符号反転を行うとき使用します。

## (2) 動作内容

```
NEGC(long m, long n) /* NEGC Rm,Rn */
{
    unsigned long temp;

    temp=0-R[m];
    R[n]=temp-T;
    if (0<temp) T=1;
    else T=0;
    if (temp<R[n]) T=1;
    PC+=2;
}
```

## (3) 使用例

```
CLRT          ;R0:R1(64ビット)の符号反転
NEGC R1,R1    ;実行前 R1=H'00000001,T=0
              ;実行後 R1=H'FFFFFFFF,T=1
NEGC R0,R0    ;実行前 R0=H'00000000,T=1
              ;実行後 R0=H'FFFFFFFF,T=1
```



## システム制御命令

PC のインクリメントのみを行い、次の命令に実行を移します。

```

NOP( ) /* NOP */
{
    PC+=2;
}

```

**NOP** ;1 実行ステート分の時間が過ぎます。

## 10.68 NOT NOT-logical complement

ビット反転

## 論理演算命令

書式	動作概略	命令コード	実行 ステート	Tビット
NOT Rm,Rn	~Rm Rn	0110nnnnnnmm0111	1	

### (1) 説明

汎用レジスタ Rm の内容の 1 の補数を取り、結果を Rn に格納します。即ち Rm のビットを反転して Rn に格納します。

### (2) 動作内容

```
NOT(long m, long n) /* NOT Rm,Rn */
{
    R[n]=~R[m];
    PC+=2;
}
```

### (3) 使用例

```
NOT R0,R1          ;実行前 R0=H'AAAAAAAA
                   ;実行後 R1=H'55555555
```

## 10.69 OCBI Operand Cache Block Invalidate データ転送命令

### キャッシュブロックの無効化

書式	動作概略	命令コード	実行 ステート	Tビット
OCBI @Rn	オペランドキャッシュ ブロックの無効化	0000nnnn10010011	1	

#### (1) 説明

実効アドレス Rn で示されている内容を使用して、データをアクセスします。キャッシュにヒットした場合、対応するキャッシュブロックを無効(Vbit=0)にします。このとき、たとえライトバック方式で、未書き込み情報あり(U bit=1)の場合でも、書き戻しはしません。キャッシュミスの場合や、非キャッシュ領域へのアクセスの場合は、動作しません。

#### (2) 動作内容

```
OCBI(int n)          /* OCBI @Rn */
{
    invalidate_operand_cache_block(R[n]);
    PC+=2;
}
```

発生する可能性がある例外

```
Data TLB miss exception
Data TLB protection violation exception
Initial page write exception
Address error
```

OCBI が動作しない場合でも、上記例外が発生しますので注意してください。

## 10.70 OCBP Operand Cache Block Purge

### キャッシュブロックのパージ

## データ転送命令

書式	動作概略	命令コード	実行 ステート	Tビット
OCBP @Rn	オペランドキャッシュ の待避	0000nnnn10100011	1	

### (1) 説明

実効アドレス Rn で示されている内容を使用して、データをアクセスします。キャッシュにヒットして未書き込み情報あり(U bit=1)の場合、対応するキャッシュブロックを外部メモリに書き戻して、そのブロックを無効(Vbit=0)にします。このとき、未書き込み情報 無し(U bit=0)の場合、単にそのブロックを無効にします。キャッシュミスの場合や、非キャッシュ領域へのアクセスの場合は、動作しません。

### (2) 動作内容

```
OCBP(int n)          /* OCBP @Rn */
{
    if(is_dirty_block(R[n])) write_back(R[n])
    invalidate_operand_cache_block(R[n]);
    PC+=2;
}
```

発生する可能性がある例外

Data TLB miss exception

Data TLB protection violation exception

Address error

OCBP が動作しない場合でも、上記例外が発生しますので注意してください。

## 10.71 OCBWB Operand Cache Block Write Back データ転送命令

### キャッシュブロックの書き戻し

---

書式	動作概略	命令コード	実行 ステート	Tビット
OCBWB @Rn	オペランドキャッシュ ブロックの書き戻し	0000nnnn11010011	1	

#### (1) 説明

実効アドレス Rn で示されている内容を使用して、データをアクセスします。キャッシュにヒットして未書き込み情報あり(U bit=1)の場合、対応するキャッシュブロックを外部メモリに書き戻して、そのブロックをクリーン(Ubit=0)にします。そのほかの場合つまり、キャッシュミスの場合や、すでにクリーンな場合、非キャッシュ領域へのアクセスの場合などは動作しません。

#### (2) 動作内容

```
OCBWB(int n)      /* OCBWB @Rn */  
{  
    if(is_dirty_block(R[n])) write_back(R[n]);  
    PC+=2;  
}
```

発生する可能性がある例外

Data TLB miss exception

Data TLB protection violation exception

Address error

OCBWB が動作しない場合でも、上記例外が発生しますので注意してください。

## 10.72 OR OR logical

論理和演算

## 論理演算命令

書式	動作概略	命令コード	実行 ステート	Tビット
OR Rm,Rn	Rn   Rm Rn	0010nnnnmmmm1011	1	
OR #imm,R0	R0   imm R0	11001011iiiiiii	1	
OR.B #imm,@(R0,GBR)	(R0+GBR)   imm (R0+GBR)	11001111iiiiiii	4	

### (1) 説明

汎用レジスタ Rn の内容と Rm の論理和をとり、結果を Rn に格納します。

汎用レジスタ R0 とゼロ拡張した 8 ビットのイミディエイトデータとの論理和、もしくはインデックス付き GBR 間接アドレッシングモードで 8 ビットのメモリと 8 ビットのイミディエイトデータとの論理和が可能です。

### (2) 動作内容

```

OR(long m, long n) /* OR Rm,Rn */
{
    R[n] |= R[m];
    PC += 2;
}

ORI(long i) /* OR #imm,R0 */
{
    R[0] |= (0x000000FF & (long)i);
    PC += 2;
}

ORM(long i) /* OR.B #imm,@(R0,GBR) */
{
    long temp;

    temp = (long)Read_Byte(GBR + R[0]);
    temp |= (0x000000FF & (long)i);
    Write_Byte(GBR + R[0], temp);
    PC += 2;
}

```

### (3) 使用例

```

OR      R0,R1          ;実行前 R0=H'AAAA5555,R1=H'55550000
                        ;実行後 R1=H'FFFF5555
OR      #H'F0,R0       ;実行前 R0=H'00000008
                        ;実行後 R0=H'000000F8
OR.B    #H'50,@(R0,GBR) ;実行前 @(R0,GBR)=H'A5
                        ;実行後 @(R0,GBR)=H'F5

```

## 10.73 PREF PREFetch data to cache

## データ転送命令

データキャッシュ  
へのプリフェッチ

書式	動作概略	命令コード	実行ステート	Tビット
PREF @Rn	prefetch cache lock	0000nnnn10000011	1	

## (1) 説明

32 バイト境界で始まる 32 バイトのデータブロックをオペランドキャッシュに読み込みます。Rn で指定したアドレスの下位 5 ビットはゼロにマスクされます。

この命令でアドレスに関するエラーは発生しません。エラーの場合には、この命令は NOP (無操作) 命令として取り扱われます。

## (2) 動作内容

```
PREF(int n) /* PREF */
{
    PC+=2;
}
```

## (3) 使用例

```
MOV.L SOFT_PF,R1          ;R1 のアドレスは SOFT_PF
PREF @R1                  ;SOFT_PF のデータを内蔵キャッシュへロード

SOFT_PF: .align 32
        .data.l H'12345678
        .data.l H'9ABCDEF0
        .data.l H'AAAA5555
        .data.l H'5555AAAA
        .data.l H'11111111
        .data.l H'22222222
        .data.l H'33333333
        .data.l H'44444444
```

## 10.74 ROTCL ROTate with Carry Left

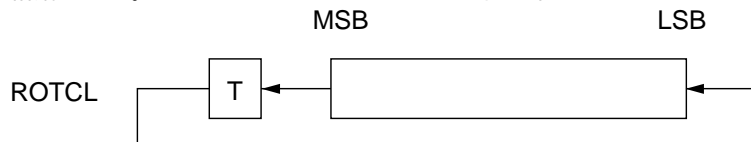
## シフト命令

T ビット付き  
1 ビット左回転

書式	動作概略	命令コード	実行 ステート	T ビット
ROTCL Rn	T Rn T	0100nnnn00100100	1	MSB

## (1) 説明

汎用レジスタ Rn の内容を左方向に T ビットを含めて 1 ビットローテート(回転)し、結果を Rn に格納します。ローテートしてオペランドの外に出てしまったビットは、T ビットへ転送します。



## (2) 動作内容

```

ROTCL(long n) /* ROTCL Rn */
{
    long temp;

    if ((R[n]&0x80000000)==0) temp=0;
    else temp=1;
    R[n]<<=1;
    if (T==1) R[n]|=0x00000001;
    else R[n]&=0xFFFFFFF0;
    if (temp==1) T=1;
    else T=0;
    PC+=2;
}

```

## (3) 使用例

```

ROTCL R0          ;実行前 R0=H'80000000,T=0
                  ;実行後 R0=H'00000000,T=1

```



## 10.75 ROTCR ROTate with Carry Right

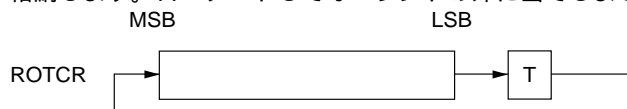
## シフト命令

T ビット付き  
1 ビット右回転

書式	動作概略	命令コード	実行 ステート	T ビット
ROTCR Rn	T Rn T	0100nnnn00100101	1	LSB

## (1) 説明

汎用レジスタ Rn の内容を右方向に T ビットを含めて 1 ビットローテート(回転)し、結果を Rn に格納します。ローテートしてオペランドの外に出てしまったビットは、T ビットへ転送します。



## (2) 動作内容

```
ROTCR(long n) /* ROTCR Rn */
{
    long temp;

    if ((R[n]&0x00000001)==0) temp=0;
    else temp=1;
    R[n]>>=1;
    if (T==1) R[n]|=0x80000000;
    else R[n]&=0x7FFFFFFF;
    if (temp==1) T=1;
    else T=0;
    PC+=2;
}
```

## (3) 使用例

```
ROTCR R0          ;実行前 R0=H'00000001,T=1
                  ;実行後 R0=H'80000000,T=1
```

## 10.76 ROTL ROTate Left

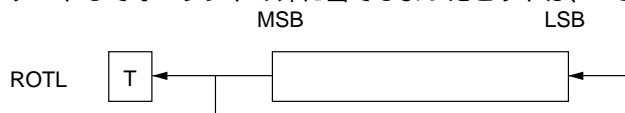
## シフト命令

1 ビット左回転

書式	動作概略	命令コード	実行 ステート	Tビット
ROTL Rn	T Rn MSB	0100nnnn00000100	1	MSB

## (1) 説明

汎用レジスタ Rn の内容を左方向に 1 ビットローテート(回転)し、結果を Rn に格納します。ローテートしてオペランドの外に出てしまったビットは、T ビットへ転送します。



## (2) 動作内容

```

ROTL(long n) /* ROTL Rn */
{
    if ((R[n]&0x80000000)==0) T=0;
    else T=1;
    R[n]<<=1;
    if (T==1) R[n]|=0x00000001;
    else R[n]&=0xFFFFFFF0;
    PC+=2;
}

```

## (3) 使用例

```

ROTL R0 ;実行前 R0=H'80000000,T=0

;実行後 R0=H'00000001,T=1

```

## 10.77 ROTR ROTate Right

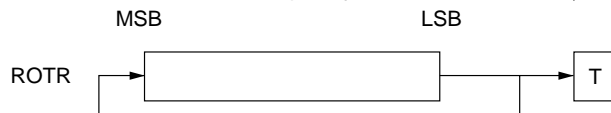
### 1ビット右回転

## シフト命令

書式	動作概略	命令コード	実行 ステート	Tビット
ROTR Rn	LSB Rn T	0100nnnn00000101	1	LSB

## (1) 説明

汎用レジスタ Rn の内容を右方向に 1 ビットローテート(回転)し、結果を Rn に格納します。ローテートしてオペランドの外に出てしまったビットは、T ビットへ転送します。



## (2) 動作内容

```
ROTR(long n) /* ROTR Rn */
{
    if ((R[n]&0x00000001)==0) T=0;
    else T=1;
    R[n]>>=1;
    if (T==1) R[n]|=0x80000000;
    else R[n]&=0x7FFFFFFF;
    PC+=2;
}
```

## (3) 使用例

```
ROTR R0          ;実行前 R0=H'00000001,T=0
                  ;実行後 R0=H'80000000,T=1
```

10.78	RTE	ReTurn from Exception	システム制御命令
	例外処理から の復帰		(特権命令) 遅延分岐命令

書式	動作概略	命令コード	実行ステート	Tビット
RTE	SSR    SR,SPC    PC	0000000000101011	5	

## (1) 説明

例外、割り込み処理ルーチンから復帰します。PC と SR の値を SPC と SSR から回復させます。プログラムは回復された PC の値で指定されるアドレスから続行されます。

RTE 命令は特権命令なので特権モードでだけ使うことができます。もしユーザモードで使われた場合は不当命令例外が発生します。

## (2) 注意

遅延分岐命令なので、この RTE 命令の次の命令を先に実行してから、分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。本命令の遅延スロット内の命令によって例外が発生してはなりません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

遅延分岐命令直後の遅延スロットに本命令が配置されたときは、スロット不当命令として認識します。

RTE の遅延スロット中の命令によってアクセスした SR の内容は、RTE によって SSR から復帰した値です。ただし、RTE の実行前に定義済みの SR、MD の値は RTE の遅延スロット内の命令をフェッチするために使用します。

## (3) 動作内容

```
RTE( ) /* RTE */
{
    unsigned int temp;
    temp=PC;
    SR=SSR;
    PC=SPC;
    Delay_Slot(temp+2);
}
```

## (4) 使用例

```
RTE           ;元のルーチンへ復帰します。
ADD #8,R14    ;分岐前に先立ち実行します。
```

【注】 遅延分岐においては分岐という動作そのものは、スロット命令の実行後に発生しますが、命令の実行（レジスタの更新など）は、あくまでも遅延分岐命令 遅延スロット命令の順に行われます。例えば遅延スロットで分岐先アドレスが格納されたレジスタを変更しても、変更前のレジスタ内容が分岐先アドレスとなります。

## 10.79 RTS ReTurn from SubRoutine 分岐命令

サブルーチンプロ

シージャからの復帰

遅延分岐命令

書式	動作概略	命令コード	実行ステート	Tビット
RTS	PR PC	0000000000001011	2	

## (1) 説明

サブルーチンプロシージャから復帰します。すなわち、PC を PR から復帰し、復帰した PC の示すアドレスから処理を続行します。本命令によって、BSR および JSR 命令でコールされたサブルーチンプロシージャからコール元へ戻ることが出来ます。

## (2) 注意

遅延分岐命令ですので、本命令の直後の命令を先に実行してから、分岐先の命令を実行します。

本命令と直後の命令との間には、割り込みを受け付けません。直後の命令が分岐命令のときは、それをスロット不当命令として認識します。

PR を復帰する命令は RTS 命令に先行しなければなりません。この復帰命令は RTS の遅延スロットであってはなりません。

## (3) 動作内容

```
RTS( ) /* RTS */
{
    unsigned int temp;

    temp=PC;
    PC=PR;
    Delay_Slot(temp+2);
}
```

## (4) 使用例

```
MOV.L    TABLE,R3 ;R3=TARGET のアドレス
JSR      @R3       ;TARGET へ分岐します。
NOP      ;分岐前に NOP を実行します。
ADD      R0,R1     ; サブルーチンプロシージャからの戻り先 (PR の内容)
.....

TABLE:   .data.1   TARGET ;ジャンプテーブル
.....

TARGET:  MOV      R1,R0 ; プログラムの入り口
          RTS      ;PR の内容 PC
          MOV      #12,R0 ;分岐に先立ち MOV を実行します。
```

## 10.80 SETS SET Sbit

S ビットのセット

## システム制御命令

書式	動作概略	命令コード	実行 ステート	T ビット
SETS	1 S	0000000001011000	1	1

### (1) 説明

S ビットを 1 にセットします。

### (2) 動作内容

```
SETS( ) /* SETS */
{
    S=1;
    PC+=2;
}
```

### (3) 使用例

```
SETS ;実行前 S=0
      ;実行後 S=1
```

## 10. 各命令の説明

---

### 10.81    SETT        SET Tbit           T ビットのセット

### システム制御命令

---

書式	動作概略	命令コード	実行 ステート	T ビット
SETT	1 T	00000000000011000	1	1

(1)    説明

T ビットをセットします。

(2)    動作内容

```
SETT( ) /* SETT */  
{  
    T=1;  
    PC+=2;  
}
```

(3)    使用例

```
SETT                    ;実行前 T=0  
                        ;実行後 T=1
```

## 10.82 SHAD SHift Arithmetic Dynamically シフト命令

ダイナミック算術  
シフト

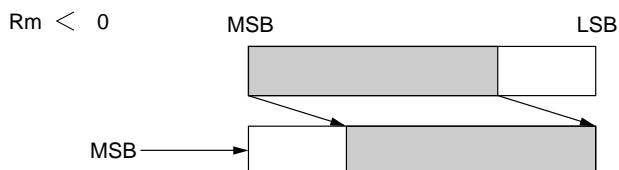
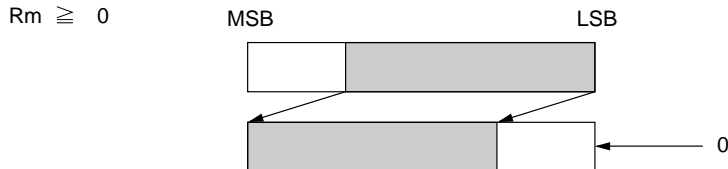
書式	動作概略	命令コード	実行 ステート	Tビット
SHAD Rm, Rn	Rm ≥ 0 のとき、 Rn<<Rm Rn Rm<0 のとき、Rn>>Rm [ MSB Rn ]	0100nnnnnnmmmm1100	1	

## (1) 説明

汎用レジスタ Rn の内容を算術的にシフトします。汎用レジスタ Rm がシフトの方向とシフトするビット数を指定します。

Rm レジスタの値が正のとき左へシフトし、負のとき右へシフトします。右にシフトするとき上位に MSB が追加されます。

シフトするビット数は Rm レジスタの下位 5 ビット (ビット 4 ~ 0) で指定されます。値が負 (MSB=1) のときは Rm レジスタは 2 の補数で表されています。左シフトのシフト量は 0 ~ 31 で、右シフトのシフト量は 1 ~ 32 です。



## (2) 動作内容

```
SHAD(long m,n) /*SHAD Rm,Rn */
{
    long cont,sgn;
    sgn=R[m]&0x80000000;
    cnt=R[m]&0x0000001F;
    if (sgn==0) R[n]<=cnt;
    else R[n]=(long)R[n]>>((~cnt+1)&0x1F);
    PC+=2;
}
```



### (3) 使用例

SHAD R1,R2

;実行前 R1=H'FFFFFFEC,R2=H'80180000

;実行後 R1=H'FFFFFFEC,R2=H'FFFFFF801

SHAD R3,R4

;実行前 R3=H'00000014,R4=H'FFFFFF801

;実行後 R3=H'00000014,R4=H'80100000

## 10.83 SHAL SHift Arithmetic Left

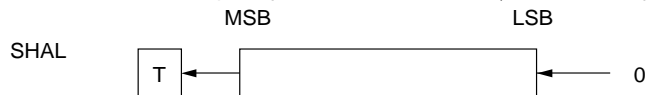
## シフト命令

1 ビット左算術  
シフト

書式	動作概略	命令コード	実行 ステート	T ビット
SHAL Rn	T Rn 0	0100nnnn00100000	1	MSB

## (1) 説明

汎用レジスタ Rn の内容を左方向に算術的に 1 ビットシフトし、結果を Rn に格納します。シフトしてオペランドの外に出てしまったビットは、T ビットへ転送します。



## (2) 動作内容

```
SHAL(long n) /* SHAL Rn (Same as SHLL) */
{
    if ((R[n]&0x80000000)==0) T=0;
    else T=1;
    R[n]<<=1;
    PC+=2;
}
```

## (3) 使用例

```
SHAL R0 ;実行前 R0=H'80000001,T=0
          ;実行後 R0=H'00000002,T=1
```

## 10.84 SHAR SHift Arithmetic Right

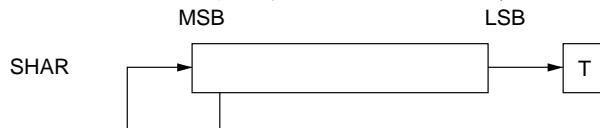
## シフト命令

1 ビット右算術  
シフト

書式	動作概略	命令コード	実行 ステート	T ビット
SHAR Rn	MSB Rn T	0100nnnn00100001	1	LSB

## (1) 説明

汎用レジスタ Rn の内容を右方向に算術的に 1 ビットシフトし、結果を Rn に格納します。シフトしてオペランドの外に出てしまったビットは、T ビットへ転送します。



## (2) 動作内容

```
SHAR(long n) /* SHAR Rn */
{
    long temp;

    if ((R[n]&0x00000001)==0) T=0;
    else T=1;
    if ((R[n]&0x80000000)==0) temp=0;
    else temp=1;
    R[n]>>=1;
    if (temp==1) R[n]|=0x80000000;
    else R[n]&=0x7FFFFFFF;
    PC+=2;
}
```

## (3) 使用例

```
SHAR R0 ;実行前 R0=H'80000001,T=0
          ;実行後 R0=H'C0000000,T=1
```

## 10.85 SHLD SHift Logical Dynamically

## シフト命令

ダイナミック論理  
シフト

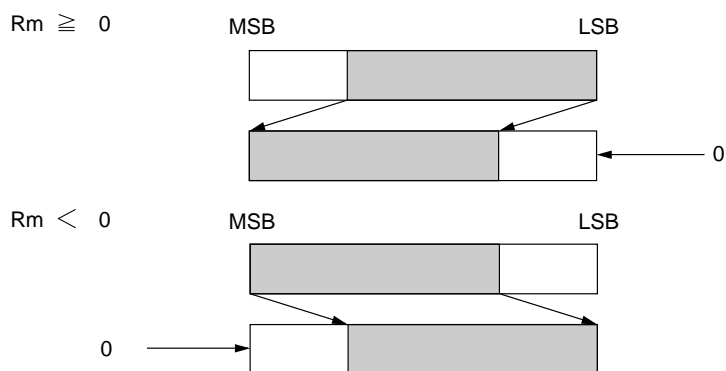
書式	動作概略	命令コード	実行 ステート	Tビット
SHLD Rm, Rn	Rm ≥ 0 のとき、 Rn<<Rm Rn Rm<0 のとき、 Rn>>Rm [0 Rn]	0100nnnnnnmmmm1101	1	

## (1) 説明

汎用レジスタ Rn の内容を論理的にシフトします。汎用レジスタ Rm がシフトの方向とシフトするビット数を指定します。

Rm レジスタの値が正のとき左へシフトし、負のとき右へシフトします。右にシフトするときは上位に 0 が追加されます。

シフトするビット数は Rm レジスタの下位 5 ビット (ビット 4~0) で指定されます。値が負 (MSB=1) のときは Rm レジスタは 2 の補数で表されています。左シフトのシフト量は 0 ~ 31 で、右シフトのシフト量は 1 ~ 32 です。



## (2) 動作内容

```
SHLD(long m,n)/*SHLD Rm,Rn */
```

```
{  
    long cont,sgn;  
    sgn=R[m]&0x80000000;  
    cnt=R[m]&0x0000001F;  
    if (sgn==0) R[n]<=<cnt;  
    else R[n]=(unsigned long)R[n]>>((~cnt+1)&0x1F);  
    PC+=2;  
}
```

## (3) 使用例

SHLD R1, R2	;実行前	R1=H'FFFFFFEC, R2=H'80180000
	;実行後	R1=H'FFFFFFEC, R2=H'00000801
SHLD R3, R4	;実行前	R3=H'00000014, R4=H'FFFFFF801
	;実行後	R3=H'00000014, R4=H'80100000

## 10.86 SHLL SHift Logical Left

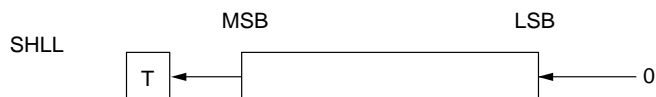
## シフト命令

1 ビット左論理  
シフト

書式	動作概略	命令コード	実行 ステート	Tビット
SHLL Rn	T Rn 0	0100nnnn00000000	1	MSB

## (1) 説明

汎用レジスタ Rn の内容を左方向に論理的に 1 ビットシフトし、結果を Rn に格納します。シフトしてオペランドの外に出てしまったビットは、T ビットへ転送します。



## (2) 動作内容

```
SHLL(long n) /* SHLL Rn (Same as SHAL) */
```

```
{
    if ((R[n]&0x80000000)==0) T=0;
    else T=1;
    R[n]<<=1;
    PC+=2;
}
```

## (3) 使用例

```
SHLL R0          ;実行前    R0=H'80000001,T=0
                  ;実行後    R0=H'00000002,T=1
```

## 10.87 SHLLn n bits SHift Logical Left

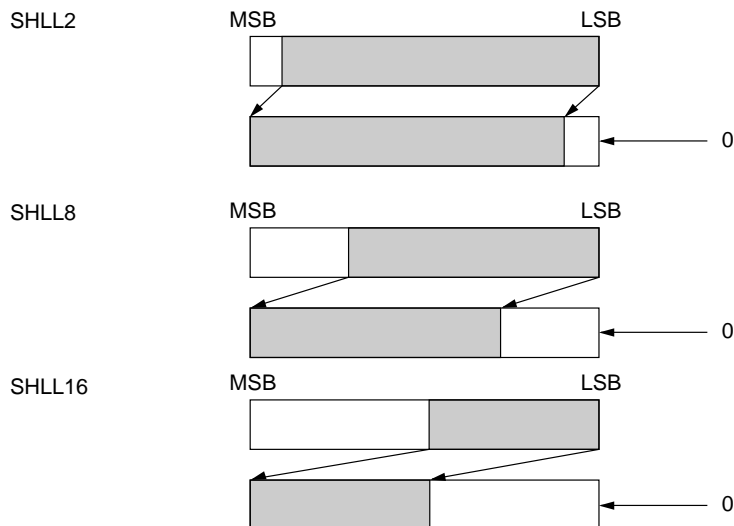
## シフト命令

n ビット左論理  
シフト

書式	動作概略	命令コード	実行 ステート	Tビット
SHLL2 Rn	$Rn \ll 2$ Rn	0100nnnn00001000	1	
SHLL8 Rn	$Rn \ll 8$ Rn	0100nnnn00011000	1	
SHLL16 Rn	$Rn \ll 16$ Rn	0100nnnn00101000	1	

## (1) 説明

汎用レジスタ Rn の内容を左方向に論理的に 2/8/16 ビットシフトし、結果を Rn に格納します。シフトしてオペランドの外に出てしまったビットは捨てます。



## (2) 動作内容

```
SHLL2(long n) /* SHLL2 Rn */
{
    R[n] <<= 2;
    PC += 2;
}
```

```
SHLL8(long n) /* SHLL8 Rn */
{
    R[n] <<= 8;
    PC += 2;
}
```

```
SHLL16(long n) /* SHLL16 Rn */  
{  
    R[n]<=<=16;  
    PC+=2;  
}
```

(3) 使用例

SHLL2 R0	;実行前	R0=H'12345678
	;実行後	R0=H'48D159E0
SHLL8 R0	;実行前	R0=H'12345678
	;実行後	R0=H'34567800
SHLL16 R0	;実行前	R0=H'12345678
	;実行後	R0=H'56780000



## 10.88 SHLR SHift Logical Right

## シフト命令

1 ビット右論理  
シフト

書式	動作概略	命令コード	実行 ステート	T ビット
SHLR Rn	0 Rn T	0100nnnn00000001	1	LSB

## (1) 説明

汎用レジスタ Rn の内容を右方向に論理的に 1 ビットシフトし、結果を Rn に格納します。シフトしてオペランドの外に出てしまったビットは、T ビットへ転送します。



## (2) 動作内容

```
SHLR(long n) /* SHLR Rn */
{
    if ((R[n]&0x00000001)==0) T=0;
    else T=1;
    R[n]>>=1;
    R[n]&=0x7FFFFFFF;
    PC+=2;
}
```

## (3) 使用例

```
SHLR R0 ;実行前 R0=H'80000001,T=0
          ;実行後 R0=H'40000000,T=1
```

## 10.89 SHLRn n bits SHift Logical Right

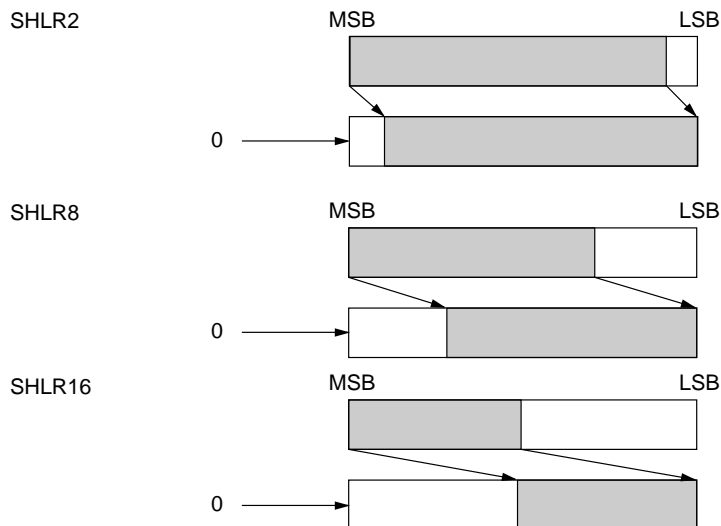
## シフト命令

n ビット右論理  
シフト

書式	動作概略	命令コード	実行 ステート	Tビット
SHLR2 Rn	$Rn \gg 2$ Rn	0100nnnn00001001	1	
SHLR8 Rn	$Rn \gg 8$ Rn	0100nnnn00011001	1	
SHLR16 Rn	$Rn \gg 16$ Rn	0100nnnn00101001	1	

## (1) 説明

汎用レジスタ Rn の内容を右方向に論理的に 2/8/16 ビットシフトし、結果を Rn に格納します。シフトしてオペランドの外に出てしまったビットは捨てます。



## (2) 動作内容

```

SHLR2(long n)          /* SHLR2 Rn */
{
    R[n]>>=2;
    R[n]&=0x3FFFFFFF;
    PC+=2;
}

```

## 10. 各命令の説明

---

```
SHLR8(long n)          /* SHLR8 Rn */
{
    R[n]>>=8;
    R[n]&=0x00FFFFFF;
    PC+=2;
}
```

```
SHLR16(long n)         /* SHLR16 Rn */
{
    R[n]>>=16;
    R[n]&=0x0000FFFF;
    PC+=2;
}
```

### (3) 使用例

SHLR2 R0	;実行前	R0=H'12345678
	;実行後	R0=H'048D159E
SHLR8 R0	;実行前	R0=H'12345678
	;実行後	R0=H'00123456
SHLR16 R0	;実行前	R0=H'12345678
	;実行後	R0=H'00001234

## 10.90 SLEEP SLEEP

低消費電力モード  
への遷移

## システム制御命令

(特権命令)

書式	動作概略	命令コード	実行ステート	Tビット
SLEEP	スリープ	00000000000011011	4	

## (1) 説明

CPU を低消費電力状態にします。

低消費電力モードでは、CPU の内部状態を保持し、直後の命令の実行を停止し、割り込み要求の発生を待ちます。要求が発生すると、低消費電力状態から抜けます。

SLEEP 命令は特権命令なので、特権モードでだけ使うことができます。もしユーザモードで使われた場合は、不当命令例外が発生します。

## (2) 注意

SLEEP の性能は STBCR (スタンバイコントロールレジスタ) に存在します。SH4 ハードウェアマニュアルの「第 9 章 低消費電力モード」を参照してください。

## (3) 動作内容

```
SLEEP( ) /* SLEEP */
{
    sleep_standby();
}
```

## (4) 使用例

```
SLEEP ;低消費電力モードへの遷移
```

## 10.91 STC STore Control register

コントロールレジスタからのストア

## システム制御命令

書式	動作概略	命令コード	実行 ステート	Tビット
STC SGR,Rn	SGR Rn	0000nnnn001111010	3	
STC DBR,Rn	DBR Rn	0000nnnn111111010	2	
STC.L SGR,@-Rn	Rn - 4 Rn, SGR (Rn)	0100nnnn001110010	3	
STC.L DBR,@-Rn	Rn - 4 Rn, DBR (Rn)	0100nnnn111110010	2	

### (1) 説明

コントロールレジスタ SGR、DBR をデスティネーションに格納します。

### (2) 注意

本命令は特権モードの場合だけ使用可能です。ユーザモードで使用すると、不当命令例外が発生します。

### (3) 動作内容

```

STCSGR(int n)      /* STC SGR,Rn : Privileged */
{
    R[n]=SGR;
    PC+=2;
}

STCDBR(int n)      /* STC DBR,Rn : Privileged */
{
    R[n]=DBR;
    PC+=2;
}

STCMSGR(int n)     /* STC.L SGR,@-Rn : Privileged */
{
    R[n]-=4;
    Write_Long(R[n],SGR);
    PC+=2;
}

STCMDDBR(int n)    /* STC.L DBR,@-Rn : Privileged */
{
    R[n]-=4;
    Write_Long(R[n],DBR);
    PC+=2;
}

```

発生する可能性がある例外

General illegal instruction exception

Slot illegal instruction exception

Data TLB miss exception

Data TLB protection violation exception

Address error

## 10.92 STS Store from FPU System register システム制御命令

FPU システムレジスタからのストア

書式	動作概略	命令コード	実行 ステート	Tビット
STS FPUL,Rn	FPUL Rn	0000nnnn01011010	1	
STS FPSCR,Rn	FPSCR Rn	0000nnnn01101010	1	
STS.L FPUL,@-Rn	Rn - 4 Rn, FPUL (Rn)	0100nnnn01011010	1	
STS.L FPSCR,@-Rn	Rn - 4 Rn, FPSCR (Rn)	0100nnnn01100010	1	

### (1) 説明

FPU システムレジスタ FPUL、FPSCR をデスティネーションに格納します。

### (2) 動作内容

```

STS(int n)                /* STS FPUL,Rn */
{
    R[n]= FPUL;
    PC+=2;
}

STS_SAVE(int n)           /* STS.L FPUL,@-Rn */
{
    R[n]--4;
    Write_Long(R[n],FPUL) ;
    PC+=2;
}

STS(int n)                /* STS FPSCR,Rn */
{
    R[n]=FPSCR&0x003FFFFFF;
    PC+=2;
}

STS_RESTORE(int n)        /* STS.L FPSCR,@-Rn */
{
    R[n]--4;
    Write_Long(R[n],FPSCR&0x003FFFFFF)
    PC+=2;
}

```

発生する可能性がある例外:

Address Error.

(3) 使用例:

- STS

Example 1:

```
MOV.L #H'12ABCDEF, R12
```

```
LDS R12, FPUL
```

```
STS FPUL, R13
```

```
; After executing the STS instruction:
```

```
; R13 = 12ABCDEF
```

Example 2:

```
STS FPSCR, R2
```

```
; After executing the STS instruction:
```

```
; The current content of FPSCR is stored in register R2
```

- STS.L

Example 1:

```
MOV.L #H'0C700148, R7
```

```
STS.L FPUL, @-R7
```

```
; Before executing the STS.L instruction:
```

```
; R7 = 0C700148
```

```
; After executing the STS.L instruction:
```

```
; R7 = 0C700144, and the content of FPUL is saved at memory
```

```
; location 0C700144.
```

Example 2:

```
MOV.L #H'0C700154, R8
```

```
STS.L FPSCR, @-R8
```

```
; After executing the STS.L instruction:
```

```
; The content of FPSCR is saved at memory location 0C700150.
```

### 10.93 SUB SUBtract binary

2進減算

### 算術演算命令

書式	動作概略	命令コード	実行 ステート	Tビット
SUB Rm,Rn	Rn-Rm Rn	0011nnnnnnmm1000	1	

#### (1) 説明

汎用レジスタ Rn の内容から Rm を減算し、結果を Rn に格納します。イミディエイトデータとの減算は ADD #imm,Rn を使います。

#### (2) 動作内容

```
SUB(long m, long n) /* SUB Rm,Rn */
{
    R[n] -= R[m];
    PC += 2;
}
```

#### (3) 使用例

```
SUB    R0,R1           ;実行前 R0=H'00000001,R1=H'80000000
                        ;実行後  R1=H'7FFFFFFF
```



## 10.94 SUBC SUBtract with Carry

## 算術演算命令

ボロー付き 2 進減算

書式	動作概略	命令コード	実行 ステート	Tビット
SUBC Rm,Rn	Rn-Rm-T Rn, ボロー T	0011nnnnnnmmmm1010	1	ボロー

## (1) 説明

汎用レジスタ Rn の内容から Rm と T ビットを減算し、結果を Rn に格納します。演算の結果によってボローを T ビットに反映します。32 ビットを超える減算を行うとき使用します。

## (2) 動作内容

```

SUBC(long m, long n) /* SUBC Rm,Rn */
{
    unsigned long tmp0,tmp1;

    tmp1=R[n]-R[m];
    tmp0=R[n];
    R[n]=tmp1-T;
    if (tmp0<tmp1) T=1;
    else T=0;
    if (tmp1<R[n]) T=1;
    PC+=2;
}

```

## (3) 使用例

```

CLRT          ; R0:R1(64ビット)-R2:R3(64ビット)=R0:R1(64ビット)
SUBC  R3,R1   ;実行前  T=0,R1=H'00000000,R3=H'00000001
              ;実行後  T=1,R1=H'FFFFFFFF
SUBC  R2,R0   ;実行前  T=1,R0=H'00000000,R2=H'00000000
              ;実行後  T=1,R0=H'FFFFFFFF

```

## 10.95 SUBV SUBtract with (Vflag)underflow check 算術演算命令

アンダフロー付き

2進減算

書式	動作概略	命令コード	実行 ステート	Tビット
SUBV Rm,Rn	Rn-Rm Rn, アンダフロー T	0011nnnnnnmmmm1011	1	アンダフロー

## (1) 説明

汎用レジスタ Rn の内容から Rm を減算し、結果を Rn に格納します。アンダフローが発生すると、T ビットをセットします。

## (2) 動作内容

```

SUBV(long m, long n) /* SUBV Rm,Rn */
{
    long dest,src,ans;

    if ((long)R[n]>=0) dest=0;
    else dest=1;
    if ((long)R[m]>=0) src=0;
    else src=1;
    src+=dest;
    R[n]-=R[m];
    if ((long)R[n]>=0) ans=0;
    else ans=1;
    ans+=dest;
    if (src==1) {
        if (ans==1) T=1;
        else T=0;
    }
    else T=0;
    PC+=2;
}

```

## (3) 使用例

```

SUBV R0,R1          ;実行前 R0=H'00000002,R1=H'80000001
                    ;実行後  R1=H'7FFFFFFF,T=1
SUBV R2,R3          ;実行前 R2=H'FFFFFFFE,R3=H'7FFFFFFFE
                    ;実行後  R3=H'80000000,T=1

```

## 10.96 SWAP SWAP register halves

## データ転送命令

上位と下位の交換

書式	動作概略	命令コード	実行 ステート	Tビット
SWAP.B Rm,Rn	Rm 下位 2 バイトの 上下バイト交換 Rn	0110nnnnnnmm1000	1	
SWAP.W Rm,Rn	Rm 上下ワード交換 Rn	0110nnnnnnmm1001	1	

## (1) 説明

汎用レジスタ Rm の内容の上位と下位を交換して、結果を Rn に格納します。

バイト指定のとき、Rm のビット 15 からビット 8 の 8 ビットと、ビット 7 からビット 0 の 8 ビットを交換します。Rn の上位 16 ビットには Rm の上位 16 ビットをそのまま転送します。

ワード指定のとき、Rm のビット 31 からビット 16 の 16 ビットと、ビット 15 からビット 0 の 16 ビットを交換します。

## (2) 動作内容

```
SWAPB(long m, long n)      /* SWAP.B Rm,Rn */
{
    unsigned long temp0,temp1;

    temp0=R[m]&0xFFFF0000;
    temp1=(R[m]&0x000000FF)<<8;
    R[n]=(R[m]&0x0000FF00)>>8;
    R[n]=R[n]|temp1|temp0;
    PC+=2;
}
```

```
SWAPW(long m, long n)      /* SWAP.W Rm,Rn */
{
    unsigned long temp;

    temp=(R[m]>>16)&0x0000FFFF;
    R[n]=R[m]<<16;
    R[n]|=temp;
    PC+=2;
}
```

## (3) 使用例

```
SWAP.B R0,R1      ;実行前 R0=H'12345678
                   ;実行後 R1=H'12347856
SWAP.W R0,R1      ;実行前 R0=H'12345678
                   ;実行後 R1=H'56781234
```

## 10.97 TAS Test And Set

## 論理演算命令

メモリテストと  
ビットセット

書式	動作概略	命令コード	実行ステート	Tビット
TAS.B @Rn	(Rn)が0のとき1 T,それ以外0 下 1 MSBof(Rn)	0100nnnn00011011	5	テスト結果

## (1) 説明

汎用レジスタ Rn の内容で指定したメモリ領域に対し、本命令は該当するキャッシュブロックをパージし、そのアドレスの示すバイトデータを読み込み、そのデータがゼロのとき T=1、ゼロでないとき T=0 とします。その後、ビット 7 を 1 にセットして同じアドレスへ書き込みます。この間、バス権は解放しません。

この場合、パージ動作は次のように実行します。

パージ動作は実効アドレスとして汎用レジスタ Rn の内容によりデータにアクセスします。キャッシュヒットが存在し、該当するキャッシュブロックがダーティ (U ビット=1) の場合、そのキャッシュブロックの内容は外部メモリにライトバックされた後、キャッシュブロックは無効になります (V ビット=0)。キャッシュヒットが存在し、該当するキャッシュブロックがクリーン (U ビット=0) の場合、キャッシュブロックは無効になるだけです (V ビット=0)。キャッシュミスが発生した場合、またはアクセスするメモリ位置がキャッシュ不可の場合、パージは実行されません。

TAS.B の 2 つのメモリアクセスは自動的に実行されます。TAS.B の 2 つのアクセスの間では他のメモリアクセスは実行されません。

## (2) 動作内容

```
TAS(int n) /* TAS.B @Rn */
{
    int temp;

    temp=(int)Read_Byte(R[n]); /* Bus Lock */
    if (temp==0) T=1;
    else T=0;
    temp|=0x00000080;
    Write_Byte(R[n],temp); /* Bus unlock */
    PC+=2;
}
```

## (3) 使用例

データ TLB ミス例外

データ TLB 保護違反例外

初期ページ書込例外

アドレスエラー

例外は本命令によりデータアクセスをバイトストアとして調べます。

## 10.98 TRAPA TRAP Always

### トラップ例外処理

## システム制御命令

書式	動作概略	命令コード	実行ステート	Tビット
TRAPA #imm	imm TRA,PC+2 SPC,SR SSR,1 SR.MD/BL/RB, 0x160 EXPEVT, VBR+H'00000100 PC	11000011iiiiiii	7	

### (1) 説明

トラップ例外処理を開始します。(PC+2)とSRの値がSPCとSSRに退避され、8ビットイミディエートデータがTRAレジスタ(ビット9~2)に格納されます。処理モードは特権モード(SRのMDビットが1)に切り替わり、SRのBLビットとRBビットが1になります。これにより、例外と割り込みの要求をマスクして受け付けず、BANK1レジスタ(R0\_BANK1~R7\_BANK1)が選択されます。例外コード0x160がEXPEVTレジスタ(ビット11~0)に書き込まれます。プログラムはVBRレジスタとオフセットH'00000100の和で表わされるアドレス(VBR+H'00000100)に分岐します。

### (2) 動作内容

```
TRAPA(int i) /* TRAPA #imm */
{
    int imm;

    imm=(0x000000FF & i);
    TRA=imm<<2;
    SSR=SR;
    SPC=PC+2;
    SR.MD=1;
    SR.BL=1;
    SR.RB=1;
    EXPEVT=0x00000160;
    PC=VBR+H'00000100;
}
```

## 10.99 TST TeST logical

## 論理演算命令

論理積演算の  
Tビットセット

書式	動作概略	命令コード	実行 ステート	Tビット
TST Rm,Rn	Rn & Rm,結果が0のとき 1 T その他 0 T	0010nnnnmmmm1000	1	テスト結果
TST #imm,R0	R0 & imm,結果が0のとき 1 T その他 0 T	11001000iiiiiii	1	テスト結果
TST.B #imm,@(R0,GBR)	(R0+GBR)&imm,結果が0の とき 1 T その他 0 T	11001100iiiiiii	3	テスト結果

## (1) 説明

汎用レジスタ Rn の内容と Rm の論理積をとり、結果がゼロのとき T ビットをセットします。結果がゼロでないとき T ビットをクリアします。Rn の内容は変更しません。

汎用レジスタ R0 とゼロ拡張した 8 ビットのイミディエイトデータとの論理積、もしくはインデックス付き GBR 間接アドレッシングモードで 8 ビットのメモリと 8 ビットのイミディエイトデータとの論理積が可能です。R0、もしくはメモリの内容は変更しません。

## (2) 動作内容

```
TST(long m, long n) /* TST Rm,Rn */
{
    if ((R[n]&R[m])==0) T=1;
    else T=0;
    PC+=2;
}

TSTI(long i) /* TST #imm,R0 */
{
    long temp;

    temp=R[0]&(0x000000FF & (long)i);
    if (temp==0) T=1;
    else T=0;
    PC+=2;
}
```

## 10. 各命令の説明

---

```
TSTM(long i) /* TST.B #imm,@(R0,GBR) */
{
    long temp;

    temp=(long)Read_Byte(GBR+R[0]);
    temp&=(0x000000FF & (long)i);
    if (temp==0) T=1;
    else T=0;
    PC+=2;
}
```

### (3) 使用例

TST	R0,R0	;実行前	R0=H'00000000
		;実行後	T=1
TST	#H'80,R0	;実行前	R0=H'FFFFFF7F
		;実行後	T=1
TST.B	#H'A5,@(R0,GBR)	;実行前	@(R0,GBR)=H'A5
		;実行後	T=0

## 10.100 XOR eXclusive OR logical

排他的論理和演算

## 論理演算命令

書式	動作概略	命令コード	実行ステップ	Tビット
XOR Rm,Rn	$Rn \wedge Rm$ Rn	0010nnnnmmmm1010	1	
XOR #imm,R0	$R0 \wedge imm$ R0	11001010iiiiiii	1	
XOR.B #imm,@(R0,GBR)	$(R0+GBR) \wedge imm$ (R0+GBR)	11001110iiiiiii	4	

### (1) 説明

汎用レジスタ Rn の内容と Rm の排他的論理和をとり、結果を Rn に格納します。

汎用レジスタ R0 とゼロ拡張した 8 ビットのイミディエイトデータとの排他的論理和、もしくはインデックス付き GBR 間接アドレッシングモードで 8 ビットのメモリと 8 ビットのイミディエイトデータとの排他的論理和が可能です。

### (2) 動作内容

```
XOR(long m, long n) /* XOR Rm,Rn */
{
    R[n]^=R[m];
    PC+=2;
}

XORI(long i) /* XOR #imm,R0 */
{
    R[0]^=(0x000000FF & (long)i);
    PC+=2;
}

XORM(long i) /* XOR.B #imm,@(R0,GBR) */
{
    int temp;

    temp=(long)Read_Byte(GBR+R[0]);
    temp^=(0x000000FF & (long)i);
    Write_Byte(GBR+R[0],temp);
    PC+=2;
}
```

### (3) 使用例

```
XOR R0,R1          ;実行前 R0=H'AAAAAAA,R1=H'55555555
                   ;実行後 R1=H'FFFFFFFF
XOR #H'F0,R0        ;実行前 R0=H'FFFFFFFF
                   ;実行後 R0=H'FFFFFFF0F
XOR.B #H'A5,@(R0,GBR) ;実行前 @(R0,GBR)=H'A5
                   ;実行後 @(R0,GBR)=H'00
```



## 10.101 XTRCT eXTRaCT

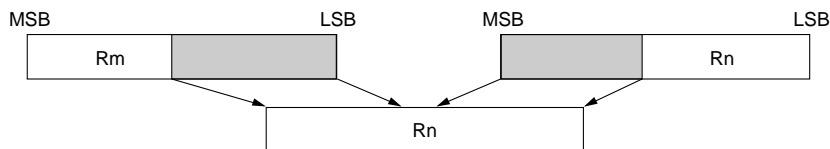
## データ転送命令

連結レジスタの  
中央切り出し

書式	動作概略	命令コード	実行 ステート	Tビット
XTRCT Rm,Rn	Rm:Rn の中央 32 ビット Rn	0010nnnnnnmmmm1101	1	

## (1) 説明

汎用レジスタ Rm と Rn とを連結した 64 ビットの内容から中央の 32 ビットを切り出し、結果を Rn に格納します。



## (2) 動作内容

```
XTRCT(long m, long n)    /* XTRCT Rm,Rn */
{
    unsigned long temp;

    temp=(R[m]<<16)&0xFFFF0000;
    R[n]=(R[n]>>16)&0x0000FFFF;
    R[n]|=temp;
    PC+=2;
}
```

## (3) 使用例

```
XTRCT R0,R1                ;実行前  R0=H'01234567,R1=H'89ABCDEF
                           ;実行後  R1=H'456789AB
```

## 付録

### A. アドレス一覧

表 A.1 アドレス一覧 (1)

モジュール	レジスタ	P4 アドレス	エリア7 アドレス*1	サイズ	パワーオン リセット	マニュアル リセット	スリープ	スタンバイ	同期 クロック
CCN	PTEH	H'FF00 0000	H'1F00 0000	32	不定	不定	保持	保持	lclk
CCN	PTL	H'FF00 0004	H'1F00 0004	32	不定	不定	保持	保持	lclk
CCN	TTB	H'FF00 0008	H'1F00 0008	32	不定	不定	保持	保持	lclk
CCN	TEA	H'FF00 000C	H'1F00 000C	32	不定	保持	保持	保持	lclk
CCN	MMUCR	H'FF00 0010	H'1F00 0010	32	H'0000 0000	H'0000 0000	保持	保持	lclk
CCN	BASRA	H'FF00 0014	H'1F00 0014	8	不定	保持	保持	保持	lclk
CCN	BASRB	H'FF00 0018	H'1F00 0018	8	不定	保持	保持	保持	lclk
CCN	CCR	H'FF00 001C	H'1F00 001C	32	H'0000 0000	H'0000 0000	保持	保持	lclk
CCN	TRA	H'FF00 0020	H'1F00 0020	32	不定	不定	保持	保持	lclk
CCN	EXPEVT	H'FF00 0024	H'1F00 0024	32	H'0000 0000	H'0000 0020	保持	保持	lclk
CCN	INTEVT	H'FF00 0028	H'1F00 0028	32	不定	不定	保持	保持	lclk
CCN	PTEA	H'FF00 0034	H'1F00 0034	32	不定	不定	保持	保持	lclk
CCN	QACR0	H'FF00 0038	H'1F00 0038	32	不定	不定	保持	保持	lclk
CCN	QACR1	H'FF00 003C	H'1F00 003C	32	不定	不定	保持	保持	lclk
UBC	BARA	H'FF20 0000	H'1F20 0000	32	不定	保持	保持	保持	lclk
UBC	BAMRA	H'FF20 0004	H'1F20 0004	8	不定	保持	保持	保持	lclk
UBC	BBRA	H'FF20 0008	H'1F20 0008	16	H'0000	保持	保持	保持	lclk
UBC	BARB	H'FF20 000C	H'1F20 000C	32	不定	保持	保持	保持	lclk
UBC	BAMRB	H'FF20 0010	H'1F20 0010	8	不定	保持	保持	保持	lclk
UBC	BBRB	H'FF20 0014	H'1F20 0014	16	H'0000	保持	保持	保持	lclk
UBC	BDRB	H'FF20 0018	H'1F20 0018	32	不定	保持	保持	保持	lclk
UBC	BDMRB	H'FF20 001C	H'1F20 001C	32	不定	保持	保持	保持	lclk
UBC	BRCR	H'FF20 0020	H'1F20 0020	16	H'0000 *2	保持	保持	保持	lclk

【注】 \*1 コントロールレジスタは物理ページ番号フィールドにおける上記アドレスを TLB セットアップすることでアクセスできます。これらのアドレスが TLB を使わずに直接参照すると、動作は限定されます。

\*2 不定ビットが含まれています。各モジュールの説明を参照してください。

表 A.1 アドレス一覧 (2)

モジュール	レジスタ	P4 アドレス	エリア7 アドレス*1	サイズ	パワーオンリセット	マニュアルリセット	スリープ	スタンバイ	同期クロック
BSC	BCR1	H'FF80 0000	H'1F80 0000	32	H'0000 0000*2	保持	保持	保持	Bclk
BSC	BCR2	H'FF80 0004	H'1F80 0004	16	H'3FFC*2	保持	保持	保持	Bclk
BSC	WCR1	H'FF80 0008	H'1F80 0008	32	H'7777 7777	保持	保持	保持	Bclk
BSC	WCR2	H'FF80 000C	H'1F80 000C	32	H'FFFE EFFF	保持	保持	保持	Bclk
BSC	WCR3	H'FF80 0010	H'1F80 0010	32	H'0777 7777	保持	保持	保持	Bclk
BSC	MCR	H'FF80 0014	H'1F80 0014	32	H'0000 0000	保持	保持	保持	Bclk
BSC	PCR	H'FF80 0018	H'1F80 0018	16	H'0000	保持	保持	保持	Bclk
BSC	RTCSR	H'FF80 001C	H'1F80 001C	16	H'0000	保持	保持	保持	Bclk
BSC	RTCNT	H'FF80 0020	H'1F80 0020	16	H'0000	保持	保持	保持	Bclk
BSC	RTCOR	H'FF80 0024	H'1F80 0024	16	H'0000	保持	保持	保持	Bclk
BSC	RFCR	H'FF80 0028	H'1F80 0028	16	H'0000	保持	保持	保持	Bclk
BSC	PCTRA	H'FF80 002C	H'1F80 002C	32	H'0000 0000	保持	保持	保持	Bclk
BSC	PDTRA	H'FF80 0030	H'1F80 0030	16	不定	保持	保持	保持	Bclk
BSC	PCTRB	H'FF80 0040	H'1F80 0040	32	H'0000 0000	保持	保持	保持	Bclk
BSC	PDTRB	H'FF80 0044	H'1F80 0044	16	不定	保持	保持	保持	Bclk
BSC	GPIOIC	H'FF80 0048	H'1F80 0048	16	H'0000 0000	保持	保持	保持	Bclk
BSC	SDMR2	H'FF90 xxxx	H'1F90 xxxx	8	ライトオンリー				Bclk
BSC	SDMR3	H'FF94 xxxx	H'1F94 xxxx	8					Bclk
DMAC	SAR0	H'FFA0 0000	H'1FA0 0000	32	不定	不定	保持	保持	Bclk
DMAC	DAR0	H'FFA0 0004	H'1FA0 0004	32	不定	不定	保持	保持	Bclk
DMAC	DMATCR0	H'FFA0 0008	H'1FA0 0008	32	不定	不定	保持	保持	Bclk
DMAC	CHCR0	H'FFA0 000C	H'1FA0 000C	32	H'0000 0000	H'0000 0000	保持	保持	Bclk
DMAC	SAR1	H'FFA0 0010	H'1FA0 0010	32	不定	不定	保持	保持	Bclk
DMAC	DAR1	H'FFA0 0014	H'1FA0 0014	32	不定	不定	保持	保持	Bclk
DMAC	DMATCR1	H'FFA0 0018	H'1FA0 0018	32	不定	不定	保持	保持	Bclk
DMAC	CHCR1	H'FFA0 001C	H'1FA0 001C	32	H'0000 0000	H'0000 0000	保持	保持	Bclk
DMAC	SAR2	H'FFA0 0020	H'1FA0 0020	32	不定	不定	保持	保持	Bclk
DMAC	DAR2	H'FFA0 0024	H'1FA0 0024	32	不定	不定	保持	保持	Bclk
DMAC	DMATCR2	H'FFA0 0028	H'1FA0 0028	32	不定	不定	保持	保持	Bclk
DMAC	CHCR2	H'FFA0 002C	H'1FA0 002C	32	H'0000 0000	H'0000 0000	保持	保持	Bclk
DMAC	SAR3	H'FFA0 0030	H'1FA0 0030	32	不定	不定	保持	保持	Bclk
DMAC	DAR3	H'FFA0 0034	H'1FA0 0034	32	不定	不定	保持	保持	Bclk
DMAC	DMATCR3	H'FFA0 0038	H'1FA0 0038	32	不定	不定	保持	保持	Bclk
DMAC	CHCR3	H'FFA0 003C	H'1FA0 003C	32	H'0000 0000	H'0000 0000	保持	保持	Bclk
DMAC	DMAOR	H'FFA0 0040	H'1FA0 0040	32	H'0000 0000	H'0000 0000	保持	保持	Bclk

【注】 \*1 コントロールレジスタは物理ページ番号フィールドにおける上記アドレスを TLB セットアップすることでアクセスできます。これらのアドレスが TLB を使わずに直接参照すると、動作は限定されます。

\*2 不定ビットが含まれています。各モジュールの説明を参照してください。

表 A.1 アドレス一覧 (3)

モジュール	レジスタ	P4 アドレス	エリア7 アドレス*1	サイズ	パワーオン リセット	マニュアル リセット	スリープ	スタンバイ	同期 クロック
CPG	FRQCR	H'FFC0 0000	H'1FC0 0000	16	*2	保持	保持	保持	Pclk
CPG	STBCR	H'FFC0 0004	H'1FC0 0004	8	H'00	保持	保持	保持	Pclk
CPG	WTCNT	H'FFC0 0008	H'1FC0 0008	8	H'00	保持	保持	保持	Pclk
CPG	WTCSR	H'FFC0 000C	H'1FC0 000C	8	H'00	保持	保持	保持	Pclk
CPG	STBCR2	H'FFC0 0010	H'1FC0 0010	8	H'00	保持	保持	保持	Pclk
RTC	R64CNT	H'FFC8 0000	H'1FC8 0000	8	保持	保持	保持	保持	Pclk
RTC	RSECCNT	H'FFC8 0004	H'1FC8 0004	8	保持	保持	保持	保持	Pclk
RTC	RMINCNT	H'FFC8 0008	H'1FC8 0008	8	保持	保持	保持	保持	Pclk
RTC	RHRCNT	H'FFC8 000C	H'1FC8 000C	8	保持	保持	保持	保持	Pclk
RTC	RWKCNT	H'FFC8 0010	H'1FC8 0010	8	保持	保持	保持	保持	Pclk
RTC	RDAYCNT	H'FFC8 0014	H'1FC8 0014	8	保持	保持	保持	保持	Pclk
RTC	RMONCNT	H'FFC8 0018	H'1FC8 0018	8	保持	保持	保持	保持	Pclk
RTC	RYRCNT	H'FFC8 001C	H'1FC8 001C	16	保持	保持	保持	保持	Pclk
RTC	RSECAR	H'FFC8 0020	H'1FC8 0020	8	保持*2	保持	保持	保持	Pclk
RTC	RMINAR	H'FFC8 0024	H'1FC8 0024	8	保持*2	保持	保持	保持	Pclk
RTC	RHRAR	H'FFC8 0028	H'1FC8 0028	8	保持*2	保持	保持	保持	Pclk
RTC	RWKAR	H'FFC8 002C	H'1FC8 002C	8	保持*2	保持	保持	保持	Pclk
RTC	RDAYAR	H'FFC8 0030	H'1FC8 0030	8	保持*2	保持	保持	保持	Pclk
RTC	RMONAR	H'FFC8 0034	H'1FC8 0034	8	保持*2	保持	保持	保持	Pclk
RTC	RCR1	H'FFC8 0038	H'1FC8 0038	8	H'00*2	H'00*2	保持	保持	Pclk
RTC	RCR2	H'FFC8 003C	H'1FC8 003C	8	H'09*2	H'00*2	保持	保持	Pclk
INTC	ICR	H'FFD0 0000	H'1FD0 0000	16	H'0000*2	H'0000*2	保持	保持	Pclk
INTC	IPRA	H'FFD0 0004	H'1FD0 0004	16	H'0000	H'0000	保持	保持	Pclk
INTC	IPRB	H'FFD0 0008	H'1FD0 0008	16	H'0000	H'0000	保持	保持	Pclk
INTC	IPRC	H'FFD0 000C	H'1FD0 000C	16	H'0000	H'0000	保持	保持	Pclk

【注】 \*1 コントロールレジスタは物理ページ番号フィールドにおける上記アドレスを TLB セットアップすることでアクセスできます。これらのアドレスが TLB を使わずに直接参照すると、動作は限定されます。

\*2 不定ビットが含まれています。各モジュールの説明を参照してください。

表 A.1 アドレス一覧 (4)

モジュール	レジスタ	P4 アドレス	エリア7 アドレス*1	サイズ	パワーオン リセット	マニュアル リセット	スリープ	スタンバイ	同期 クロック
TMU	TOCR	H'FFD8 0000	H'1FD8 0000	8	H'00	H'00	保持	保持	Pclk
TMU	TSTR	H'FFD8 0004	H'1FD8 0004	8	H'00	H'00	保持	H'00*2	Pclk
TMU	TCOR0	H'FFD8 0008	H'1FD8 0008	32	H'FFFF FFFF	H'FFFF FFFF	保持	保持	Pclk
TMU	TCNT0	H'FFD8 000C	H'1FD8 000C	32	H'FFFF FFFF	H'FFFF FFFF	保持	保持	Pclk
TMU	TCR0	H'FFD8 0010	H'1FD8 0010	16	H'0000	H'0000	保持	保持	Pclk
TMU	TCOR1	H'FFD8 0014	H'1FD8 0014	32	H'FFFF FFFF	H'FFFF FFFF	保持	保持	Pclk
TMU	TCNT1	H'FFD8 0018	H'1FD8 0018	32	H'FFFF FFFF	H'FFFF FFFF	保持	保持	Pclk
TMU	TCR1	H'FFD8 001C	H'1FD8 001C	16	H'0000	H'0000	保持	保持	Pclk
TMU	TCOR2	H'FFD8 0020	H'1FD8 0020	32	H'FFFF FFFF	H'FFFF FFFF	保持	保持	Pclk
TMU	TCNT2	H'FFD8 0024	H'1FD8 0024	32	H'FFFF FFFF	H'FFFF FFFF	保持	保持	Pclk
TMU	TCR2	H'FFD8 0028	H'1FD8 0028	16	H'0000	H'0000	保持	保持	Pclk
TMU	TCPR2	H'FFD8 002C	H'1FD8 002C	32	不定	不定	保持	保持	Pclk
SCI	SCSMR1	H'FFE0 0000	H'1FE0 0000	8	H'00	H'00	保持	H'00	Pclk
SCI	SCBRR1	H'FFE0 0004	H'1FE0 0004	8	H'FF	H'FF	保持	H'FF	Pclk
SCI	SCSCR1	H'FFE0 0008	H'1FE0 0008	8	H'00	H'00	保持	H'00	Pclk
SCI	SCTDR1	H'FFE0 000C	H'1FE0 000C	8	H'FF	H'FF	保持	H'FF	Pclk
SCI	SCSSR1	H'FFE0 0010	H'1FE0 0010	8	H'84	H'84	保持	H'84	Pclk
SCI	SCRDR1	H'FFE0 0014	H'1FE0 0014	8	H'00	H'00	保持	H'00	Pclk
SCI	SCSCMR1	H'FFE0 0018	H'1FE0 0018	8	H'00	H'00	保持	H'00	Pclk
SCI	SCSPTR1	H'FFE0 001C	H'1FE0 001C	8	H'00*2	H'00*2	保持	H'00*2	Pclk
SCI	SCSMR2	H'FFE8 0000	H'1FE8 0000	16	H'0000	H'0000	保持	保持	Pclk
SCI	SCBRR2	H'FFE8 0004	H'1FE8 0004	8	H'FF	H'FF	保持	保持	Pclk
SCI	SCSCR2	H'FFE8 0008	H'1FE8 0008	16	H'0000	H'0000	保持	保持	Pclk
SCI	SCFTDR2	H'FFE8 000C	H'1FE8 000C	8	不定	不定	保持	保持	Pclk
SCI	SCFSR2	H'FFE8 0010	H'1FE8 0010	16	H'0060	H'0060	保持	保持	Pclk
SCI	SCFRDR2	H'FFE8 0014	H'1FE8 0014	8	不定	不定	保持	保持	Pclk
SCI	SCFCR2	H'FFE8 0018	H'1FE8 0018	16	H'0000	H'0000	保持	保持	Pclk
SCI	SCFDR2	H'FFE8 001C	H'1FE8 001C	16	H'0000	H'0000	保持	保持	Pclk
SCI	SCSPTR2	H'FFE8 0020	H'1FE8 0020	16	H'0000*2	H'0000*2	保持	保持	Pclk
SCI	SCLSR2	H'FFE8 0024	H'1FE8 0024	16	H'0000	H'0000	保持	保持	Pclk
Hitachi-UDI	SDIR	H'FFF0 0000	H'1FF0 0000	16	H'FFFF*2	保持	保持	保持	Pclk
Hitachi-UDI	SDDR	H'FFF0 0008	H'1FF0 0008	32	保持	保持	保持	保持	Pclk

【注】 \*1    コントロールレジスタは物理ページ番号フィールドにおける上記アドレスを TLB セットアップすることでアクセスできます。これらのアドレスが TLB を使わずに直接参照すると、動作は限定されます。

\*2    不定ビットが含まれています。各モジュールの説明を参照してください。

## B. 命令のプリフェッチとその副作用について

SH4 は、先読みした命令を保持するためのバッファを内部に設けており、常に命令の先読みを行っています。したがって、各メモリ空間の最終 20byte 領域にプログラムを配置しないでください。もし、その領域にプログラムを配置した場合、メモリエリアを超えて、命令の先読みのためのバスアクセスが発生する場合があります。以下にこれが問題となるケースを示します。

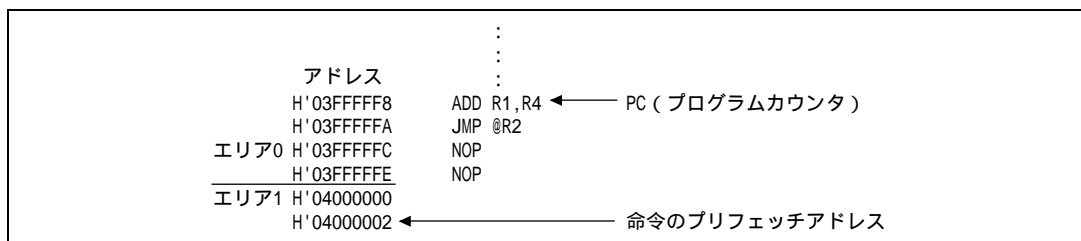


図 B.1 命令のプリフェッチ

図 B.1 では、PC (プログラムカウンタ) が指し示す命令 (ADD) と、H'04000002 番地の命令フェッチが同時に行われるケースを想定しています。また、プログラムは、後続の JMP 命令、ディレイスロット命令の実行後、エリア 1 以外の領域に分岐するものと仮定します。

この場合、プログラムのフローから想定しえない、エリア 1 へのバスアクセス (命令のプリフェッチ) が発生する可能性があります。

### (1) 命令のプリフェッチの副作用

- (1) 命令プリフェッチが引き起こす外部バスアクセスが原因でその領域に接続された FIFO などの外部デバイスが誤動作する場合があります。
- (2) 命令プリフェッチが引き起こす外部バス要求に応答するデバイスが存在しない場合、ハングアップの原因になります。

### (2) 回避方法

- (1) MMU を用いることで、これら不当な命令フェッチを回避することが可能です。
- (2) 各エリア最終 20Byte の領域にプログラムを配置しないことで、回避することが可能です。